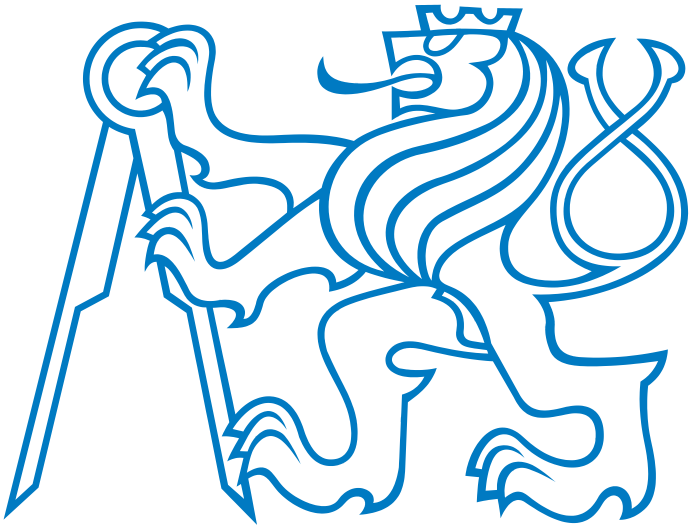
Czech Technical University in Prague
Faculty of Electrical Engineering



Habilitation Thesis

February, 2018

Tomáš Pevný

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

CHALLENGES AND OPEN QUESTIONS OF MACHINE LEARNING IN COMPUTER SECURITY

Habilitation Thesis

Tomáš Pevný

Prague, February, 2018

Copyright

Abstract

This habilitation thesis presents advancements in machine learning for computer security, arising from problems in network intrusion detection and steganography.

The thesis put an emphasis on explanation of traits shared by steganalysis, network intrusion detection, and other security domains, which makes these domains different from computer vision, speech recognition, and other fields where machine learning is typically studied. Then, the thesis presents methods developed to at least partially solve the identified problems with an overall goal to make machine learning based intrusion detection system viable. Most of them are general in the sense that they can be used outside intrusion detection and steganalysis on problems with similar constraints.

A common feature of all methods is that they are generally simple, yet surprisingly effective. According to large-scale experiments they almost always improve the prior art, which is likely caused by being tailored to security problems and designed for large volumes of data.

Specifically, the thesis addresses following problems:

- anomaly detection with low computational and memory complexity such that efficient processing of large data is possible;
- multiple-instance anomaly detection improving signal-to-noise ration by classifying larger group of samples;
- supervised classification of tree-structured data simplifying their encoding in neural networks;
- clustering of structured data;
- supervised training with the emphasis on the precision in top $p\%$ of returned data;
- and finally explanation of anomalies to help humans understand the nature of anomaly and speed-up their decision.

Many algorithms and method presented in this thesis are deployed in the real intrusion detection system protecting millions of computers around the globe.

Anotace

Cílem této habilitační práce je uceleně prezentovat aplikaci a zejména adaptaci metod strojového učení v detekci útoků na počítače přes počítačovou síť a ve steganografii.

Práce se snaží shrnout vlastnosti společné steganalýze a detekci útoků na počítače přes počítačovou síť, které tyto dvě aplikace odlišují od aplikací typicky řešených ve strojovém učení, jakými jsou například počítačové vidění, rozponávání řeči, či jiné problémy učení s učitelem. Práce souhrně představuje metody, které se snaží identifikované problémy řešit s důrazem na rychlé zpracování velkých objemů dat. Většina prezentovaných metod je obecná ve smyslu jejich použití mimo zamýšlenou doménu počítačové bezpečnosti.

Společnou vlastností téměř všech metod je důraz na jejich praktičnost (nízké výpočetní nároky). Přestože je většina z nich jednoduchá, při experimentálních ověření na velkých datech často porážejí doposud známé a často mnohem komplikovanější metody.

Tato práce konkrétně reší následující problémy:

- detekce anomalií s důrazem na nízkou výpočetní náročnost tak, aby velké objemy dat mohly být efektivně zpracovány;
- multi-instanční detekce anomálií zvyšující odstup signálu od šumu aggregací vzorků do větších skupin;
- učení s učitelem strukturovaných dat zjednodušující reprezentaci domény v neuronových sítích;
- shlukování strukturovaných dat;
- učení s důrazem na přesnost v horním kvantilu;
- a vysvětlování nalezených anomalií pro lepší pochopení jejich odlišností lidmi.

Většina algoritmů popsaných v této práci je použita ve skutečném systému chránícím milliony počítačů na celém světě.

Acknowledgments

I would like to express my thanks to my colleagues for a great time during working on the topics presented in this thesis; namely to Martin Grill, Jan Kohout, Jan Stiborek, Martin Kopp, Jan Jusko, and Martin Rehák.

Thanks go also to Jan Faigl for help with collecting and formatting the habilitation.

My special thanks go to Prof. Michal Pěchouček for constant encouragement and support not only in submitting the habilitation.

Lastly, but not least, I would like to thank to my wife Jana for fun I experience with her in our life.

Contents

1 Cybersecurity: Importance and challenges

Computers and related infrastructure have in recent years penetrated our daily life. Most of us now carry a mobile phone, tablets, or personal computers equipped with general-purpose operating systems and processors so powerful that a few years back they would be considered a super-computer. Our homes feature wireless routers to provide internet connectivity, some of us are getting used to personal assistants such as Amazon Alexa or Apple Siri, heating can be controlled electronically by a thermostat connected to the internet (e.g. Google's Nest), televisions stream movies from the internet (through e.g. Netflix service), kids play with game consoles, light system can be connected to internet as well (e.g. Phillips Hue), etc.. Some people trust electronics so much that they replaced analog locks on their front doors with electronic counterparts connected to the internet (e.g. August Smart Lock Pro). Houses can be equipped with cameras connected to the internet storing images remotely in the cloud. Our cars are filled with computers and newspapers report about improvements in their self-driving capability.

This ubiquity of connectivity and availability of computational power represents threat surface of such size, that developers of individual devices cannot anticipate, what happens when their devices or software are connected into the network and how miscreants can exploit it. A prototypical example is a compromise of home routers. Even though their CPUs are of relatively low computational power, large number of poorly protected routers have been recently used in largest coordinated denial of service (DOS) [1]. Moreover, routers are gateways to the home intranet. If a geeky house is equipped with Sonos speaker, Amazon Alexa home assistant, and a Smart lock mounted on the main door, voice-synthesis of Sonos speaker can be to remotely ordered to pronounce "Alexa: open the main door", and the main door magically opens? How? Alexa assistant is not tied to the voice of any person and it simply obeys commands of any voice. From the point of view of individual vendors of devices in the attack vector, they probably did not do anything wrong, but the chain of devices can have unexpected consequences.

The above example is not common at the moment, but other forms of network attacks are widespread [2]. For example, personal and sensitive data of large enterprises and their customers stored in big data centers (or in the cloud) are frequently stolen either by individuals or by professional hackers supported by governments. Personal computers are hacked in order to: be remotely operated to serve as proxies to launch attacks against other computers; steal bank, e-mail and other credentials of owners; harvest virtual currencies; send spam; trick owners to pay for fake antivirus; or to encrypt the data stored on the computer and demand ransom for decryption.

Few above examples sufficiently demonstrate the need for keeping computers and the related infrastructures (computer network) secure. This can be achieved either by improving the security of individual devices or by improving defense mechanisms for detecting and preventing general classes of attacks. An example of the former approach is a decision of Apple to forbid users (even those with super-user privileges) to modify certain parts of the filesystem in order to protect the integrity of core part of the OS. An example of the latter approaches is the anti-virus scanning filesystem and files downloaded from

[1] Interestingly, these attacks have been carried by provider of DOS defense solutions to attract more customers [35].

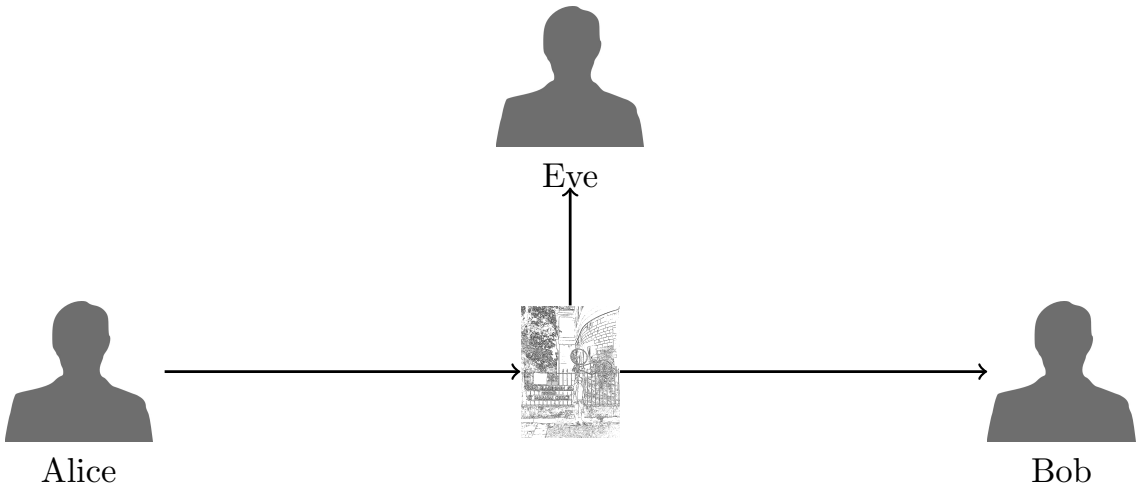


Figure 1: Prisoner's problem in steganography and steganalysis.

the internet (or elsewhere) for known viruses or intrusion detection systems deployed on the network perimeter to detect traces of attacks.

This thesis summarizes the work of the candidate in detection of attacks carried over the network (network intrusion detection) and in steganalysis. Although many presented methods rely on machine learning, the thesis aims to demonstrate that to achieve practically usable results, methods cannot be used as is and they need to be heavily adapted to fit the specifics of the domain.

2 Problem definition

Figure 1 depicts a *prisoner's problem* [14], which is a scenario used to introduce Steganography and Steganalysis. Alice and Bob, two prisoners in solitary cells want to agree on an escape plan. Despite being in solitary cells, they can exchange messages, but all of them are intercepted by a warden Eve, who scans them for any illicit content, such as the escape plan. It is obvious that in this case Alice and Bob cannot use cryptography to prevent Eve seeing their plans because Eve would immediately see that they are trying to hide something and forbid them to write to each other. Consequently, their plan will be ruined. Since Alice and Bob are smart, they hide their messages into an innocuous looking object by slightly perturbing it, for example slightly changing pixels in an innocuous looking image. To a non-educated Eve, such image would look perfectly normal and let it pass to Bob. Educated Eve would scrutinize the image for traces of hidden messages and in case of a positive alarm (detector outputs image contains a message, further called *stego* image), she stops the communication.

Proper mathematical formulation of prisoner's problem is outlined in Figure 2. Steganographic algorithm consists of two functions: f_{emb} takes as an input image $x \in \mathcal{X}$ (or other cover objects), message $m \in \mathcal{M}$, and key $k \in \mathcal{K}$ shared by Alice and Bob and produces stego image $y \in \mathcal{X}$; f_{ext} takes as an input stego object y and key k and outputs the message m. Notice that Bob is generally not interested in the original image, which is just a message carrier used as a decoy. Probability distributions on cover objects P_x, keys P_k,

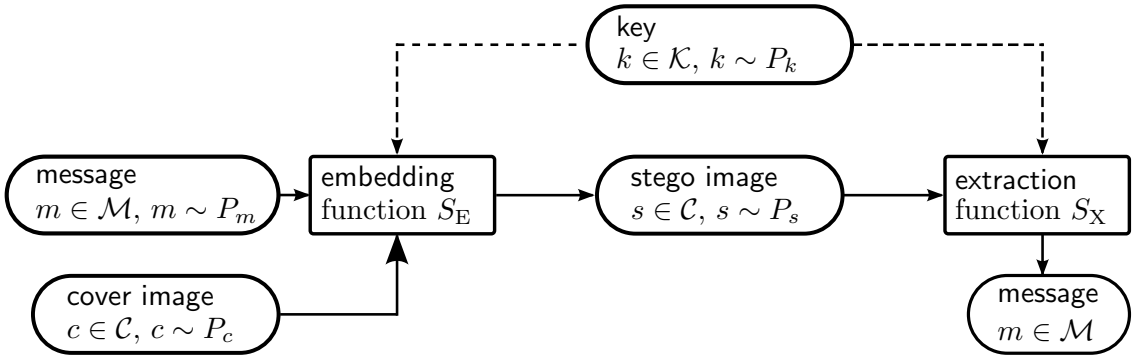


Figure 2: Steganographic channel.

and message P_m together with embedding function f_{emb} implicitly defines probability distribution on stego objects P_y.

The steganographic algorithm is considered secure, if probability distribution of cover and stego objects are identical, i.e. $P_x = P_y$. The rationale behind is that in this case, the best detector Eve can ever have is equal to random guessing. Cachin [14] defines steganographic algorithm to be ϵ-secure, if

$$\mathrm{D}_{\mathrm{KL}}(P_x, P_y) \leq \epsilon.$$

The rationale behind this definition is that Kullback-Leibler (KL) divergence provide upperbound on the best detector Eve can obtain. Recently, Ker [28] has shown that KL divergence exhibits pathological behaviors in special type of covers and suggest to replace it by total variation

$$\sup_{\mathcal{A} \subset \mathcal{X}} |P_x(\mathcal{A}) - P_y(\mathcal{A})|,$$

which is a tight bound on the best detector Eve can ever obtain.

How does intrusion detection relate to the steganographic scenario? In NIDS, observed legitimate traffic can be viewed as a cover object x, the attack can be viewed as a message m, and the specifics of attack execution as an embedding function f_{emb} (the steganographic key is not used in this likening). The traffic modified by the attack can be therefore viewed as a stego object y. Naturally, the attacker wants to be as little detectable as possible while the goal if NIDS is to detect as many attacks as possible. Similarly to steganography, the attack would be theoretically undetectable if probability distributions of the network traffic with and without the attack would be equal, i.e. $P_x = P_y$ with P_x/P_y being a probability distribution of the traffic without/with an attack. Therefore the definition of the steganographic security can be readily used to define undetectable attacks. Moreover, a vast prior art in scaling of steganographic capacity with the size of the cover objects can be used as well [19, 27]. Noteworthy to say that in steganalysis scenario the receiver is used only to impose constraints on the embedding functions (message has to be recoverable from the stego image). In the network intrusion detection, this condition is made implicit by the attack meeting criterions qualifying it to be an attack of a particular type.

It is important to realize that in both above scenarios probability distributions are defined directly on the set of objects of interest. When objects are projected to a new space

through some set of features, this projection can alter the probability distributions and decrease its entropy. This means that if two probability distributions are equal in the projected space, there might not be equal in the original space. Therefore conclusions about security done in projected spaces can be misleading.

Let's identify important properties shared by both and other security scenarios

1. Whichever party (Alice/attacker or Eve/NIDS) have a better model of the environment (images/network traffic)[2] should win in the long term (or in the expectation). This is a consequence of the fact that if the attacker has a better model than the defender, he can modify the attack to be invisible and vice versa [26].
2. If attackers are rational, the attack signal to be detected (image modification or traces of the attack) is weak. Although in other domains such as in astronomy detected signals are weak as well, in security domains the weak signal is actively trying to avoid being detected.
3. Practical algorithms should be lightweight, as volumes of data that needs to be processed are enormous. For example, the number of images uploaded daily to Facebook is 350 millions [4]. The numbers of logs from network traffic are even one magnitude higher, as according to [3] the number of HTTP requests processed by Cisco's CTA cloud service is more than 10 billion a day.
4. The enormous amount of data makes the detection problem highly imbalanced in the sense that the number of benign objects is much higher than that of malicious ones. This means that any practically usable detector should have extremely low false positive rate, 10^{-4} in case of NIDS when the subject of the classification is a network host[3] to 10^{-10},[4] otherwise the operators performing further investigations would be flooded by false positives. This would render the system useless because the trust in its decisions will be lost.
5. In both scenarios entities (players) have antagonistic goals, therefore both scenarios should be formulated by means of Game Theory. Although this has been identified more than twenty years ago, its use is rare due to the computational complexity and domains lacking clear structure.

The network intrusion detection domain posses further properties making application of machine learning methods more difficult.

1. Unlike computer images, the conversion of a traffic into a fixed dimensional vector can be complicated, as the data does not have a fixed size. An example is HTTP request

   `http://www.example.com/res/index.php?action=welcome.`

   The request has three parts (hostname, path, and query) each consisting of several tokens and their number can be anywhere from zero to n, where n is such that the URL has at most 2000 characters. Another example can be a chain of certification authorities provided during TLS handshake.

[2] In the case of the steganography the model of the noise within the image is more important than the model of the image content

[3] This number stems from the internal estimate that in corporate networks, approximately one out of 1000 network hosts are infected by malware or it is under some kind of attack.

[4] Apart from the malware recently starting to use steganograhy, uses of steganography is covered by information embargo imposed by government agencies.

2. The network traffic is non-stationary for example due to the differences between day and night, or due to applications (frameworks) becoming popular and phasing out.
3. The ground truth is expensive and difficult to obtain, sometimes impossible even for experienced security investigator. Imagine for example HTTP request to `google.com`, which can be due to a legitimate user searching for something on the web, or due to malware verifying connectivity to the internet. Therefore any methods to acquire ground truth at lower costs or creating learning algorithms that require less training samples is of great need.
4. Since detections are typically investigated further by humans (network operators), the ability to explain the decision made by the IDS is valuable since it can cut down the time to investigate the incident.

In the rest, contributions of the candidate to solve some of the above problems are briefly described, while a detailed description is left to the appendix with reproduction of original papers. Contributions are presented in logical blocks reflecting the above problems, though the division is not strict, as solutions one can easily span more than one block due to their entwining.

3 Non-stationarity and large volume of data

To process a large volume of data many real-world NIDS are implemented like a funnel [50]. Upon ingesting the data set of lightweight detectors are used to remove 95–99% of the samples, by which the system decreases its computational and storage requirements. These detectors are required (i) to be continuously updated due to non-stationarity of the traffic and (ii) to be general in the sense that they are able to detect novel attacks because only filtered data are stored.

Anomaly detectors are popular choice meeting the above requirements [38, 36, 46]. They do not need labeled data since they presume attacks will be different from the majority of the traffic and therefore they should be anomalous. This makes them the darling of security researchers due to their ability to detect new types of attack (zero-day attacked). Yet, their deployment in NIDS is rare mainly due to their excessive false positive rate caused by the fact that every anomaly does not need to be the violation of a security policy.

To make anomaly detectors practically useful, single all-purpose anomaly detector is frequently replaced by a large number of relatively simple anomaly detectors [50]. This construction has a many-fold benefit and meets requirements identified in the first paragraph: (i) it decreases the computational complexity, since many of these detectors are implemented as a simple histograms over discrete values or use only a few features; (ii) it facilitates inclusion of the domain knowledge, because attacks tend to be anomalous in a few features;[5] (iii) it allows to combine features with widely different semantics, e.g. visiting certain server hosted in certain autonomous system with a fraction of dns requests

[5] This is related to the problem of anomaly detection in subspaces, where anomalies are visible in a particular sub-space and they are shadowed by the noise in the full space.

to newly observed IP addresses.[6]

In [42](see Appendix 1 for details) we have investigated an extreme case, where an ensemble of very weak detectors implemented as a histogram of data projected on a single vector has been constructed. Similarly to supervised cases [10, 20], the ensemble leads to a strong anomaly detector with a performance equal to or better than state of the art methods. It also allows to ingest data with missing variables (practical in domains with sensor outages) and identification of features in which the scrutinized sample deviates from the majority.

The main drawback of the use of an ensemble is that it requires a function aggregating their outputs. This is typically solved using unsupervised methods, such as taking an average of outputs of detectors for a given sample [37]. Due to the constraint on the low false positive rate and the fact that not every anomaly is interesting the candidate believes that it is better to solve this using supervised learning as detailed in Section 7.

To further cope with non-stationarity of the data, Ref. [51] has proposed so-called "trust models". Although the original publication has convincing experimental results, the theoretical justifications were not provided. Our analysis [22](see Appendix 2 for details) revealed that "trust models" are in its essence local adaptive multivariate smoothing (LAMS) . In the same same work we have identified two types of false positives (alarms) in network traffic analysis. *Unstructured false positives* are short-term events distributed uniformly over all network hosts proportionally to the traffic volume typically triggered by widespread, uniformly distributed behaviors (such as web browsing). *Structured false positives* are caused by a (long-term) legitimate behavior of a small number of network hosts different from the background. Because they are found only at a very small portion of network hosts, they are reported as anomalies. Typical examples of these false positives are domain name servers (DNS) , licensing servers, etc. Our work has shown how LAMS decreases the number of unstructured false positive while leaves the structured false positives mostly intact, which is optimistic since structured false positives can be easily white-listed as they occur in small numbers.

4 Weak signal

The change of the probability distribution of observed data caused by the attack made by a rational attacker is small. In steganography Ker [26] proposed to deal with this problem by aggregating the evidence from multiple measurements (this corresponds to the number of images in steganalysis and the length of the observation window in network intrusion detection). This line of research led to "square-root-law" stating that if the length of message hidden in images increases faster than the square root of the number of pixels (proxy-measure for capacity which is more understandable) then the attacker will be in the limit detectable with arbitrary precision.

The practical application of the above assumption has steered our attention to multi-instance learning (MIL) and to the Maximum mean discrepancy [21] (MMD), which is a measure of discrepancy between two probability distributions observed through a set of samples.

[6] This fraction is a good indicator of malware that uses domain-name generation algorithm to identify the valid address of command and control server for a given day.

In steganalysis this means that the subject of the detection is an *actor* sending multiple images instead of a single image as is the common practice. In [29, 30] (see Appendix 3 for further details) we have proposed to calculate the distance between actors using an MMD between images they have communicated, and use this distance in the local outlier factor [11] (LOF) anomaly detector. It has been therefore assumed that actors using steganography will be anomalies in the space of observed actors. The resulting detector is universal in the sense that it can detect any steganographic algorithm. With respect to the prior art [44, 43, 39] the proposed multi-instance learning detector significantly improves the accuracy in identifying actor guilty by using steganography.

In its essence, we have proposed in the previous paragraph (references [29, 30]) multiple-instance anomaly detector. To that date, the only work explicitly solving this problem was [40], where MMD has been used in Gaussian kernel (instead of the usual L_2 distance) in one-class Support Vector Machines (SVM) as was proposed in [16]. The resulting algorithm, called support-measure machines has unfortunately high computational requirements due to the $O(b^2)$ complexity of calculating MMD distances between two sets each containing b vectors, and $O(l^3)$ complexity of training one-class SVM with l samples.

As mentioned in the previous paragraph, the calculation of MMD has a quadratic complexity with respect to the number of measurements (each measurement corresponds to one vector and is called an instance) of each sample (called bag). This prevents to use MMD in domains, where the number of instances per bag is high. To alleviate this complexity, in [33] (see Appendix 4 for further details) we have proposed to approximate MMD by representing each bag by a single vector equal to the mean of instances projected by an inverse of a Cholesky decomposition of the kernel matrix. This representation offers several advantages for practical applications:

1. it decreases the storage requirements and makes it predictable. Each sample is represented by a vector of a fixed size, which is sparse;
2. mean vectors representing bags can be calculated and updated online and the exponential moving window allows to deal with concept drift;
3. the MMD distance is then equal to L_2 distance between sparse vectors representing bags.

While the first two advantages have consequences for practical use, that of the third property implies that any algorithm (supervised, unsupervised, clustering, etc.) can be converted to an algorithm for multi-instance learning problems. The main limitation is that the method is practical for problems of small intrinsic dimensions because it is based on an explicit representation of the kernel space.

This approach has been practically verified in [32, 32, 33] on three problems: inferring the infrastructure of the network service, anomaly detection of command and control servers of malware campaigns, and supervised detection of computers infected by malware on basis of their communication with external servers. In all these problems the proposed approach exceeded the prior art.

$$\mathbf{b} = \left\{ \begin{matrix} (x_{1,1}, \ldots, x_{1,d}) \\ (x_{2,1}, \ldots, x_{2,d}) \\ \vdots \\ (x_{b,1}, \ldots, x_{b,d}) \end{matrix} \right\} \xrightarrow[\text{classifier}]{\text{send to}} f(b) = \begin{cases} +1 \\ -1 \end{cases}$$

Figure 3: Multi-instance learning problem. Each sample $\mathbf{b}$ is called *bag* and individual vectors $\{x_i\}_{i=1}^{b}$ are called *instances*.

5 Structured models

In the previous section it has been demonstrated that multi-instance learning (see Figure 3) is suitable for domains, where subjects of classification are described by a set of vectors (the terminology of multi-instance learning call each sample a *bag* and individual vectors *instances*). Although the technique described in previous section is suitable for unsupervised and supervised variants of MIL problems [33], it was mainly developed for unsupervised learning problems with instances of low intrinsic dimension. Moreover, the author does not believe isotropic Gaussian to be well suited for all applications.

To tackle these challenges, we have written the embedding function of a bag $b = \{x_i\}_{i=1}^{l}$, $x_i \in \mathbb{R}^d$ as an aggregation of instances projected by some function $h(x_i; \theta_h) : \mathbb{R}^d \mapsto \mathbb{R}^k$ with θ_h being vector of parameters. By writing a classifier taking the vector representation of a bag as an input as $f(b; \theta_f) : \mathbb{R}^k \mapsto \{-1, +1\}$, where θ_f is again vector of parameters, the complete classifier can be written as

$$f\left(\{x_i\}_{i=1}^{l}\right) = f\left(\frac{1}{l}\sum_{i=1}^{l} h(x_i; \theta_h); \theta_f\right),$$

where mean is used as an aggregation function. If f and h are chosen such that gradient with respect to θ_f and θ_h exists, than all parameters can be optimized using gradient descend techniques popular in neural networks (see Appendix 5 for further details).

An important aspect of the above solution is that parameters of function h are optimized using labels, whereas most prior art uses unsupervised techniques [6]. Scenarios for which this approach excels is outlined in Figure 4, where probability density functions (pdf) of instances of bags from two different classes are similar on most part the space and differ only little on areas of low probability. The unsupervised methods used in the prior art will mostly fail here because they will model areas where pdfs are high, which is precisely where pdfs of both classes are equal. This scenario seems to be relevant in security, where the attack signal cluttered by the signal of legitimate use.

The prior art on MIL [6] contains many different algorithms stemming from diverse applications. On the suite of 20 benchmark problems from [5], 13 different algorithms are the best at least on one problem. This means that one algorithm from prior art typically works well one to two problems. The approach described in previous section [48] has excelled on 10 problems, which demonstrates its generality and superiority. Needless to say that on problems, where the prior art was better, the algorithm described here has typically over-fitted.

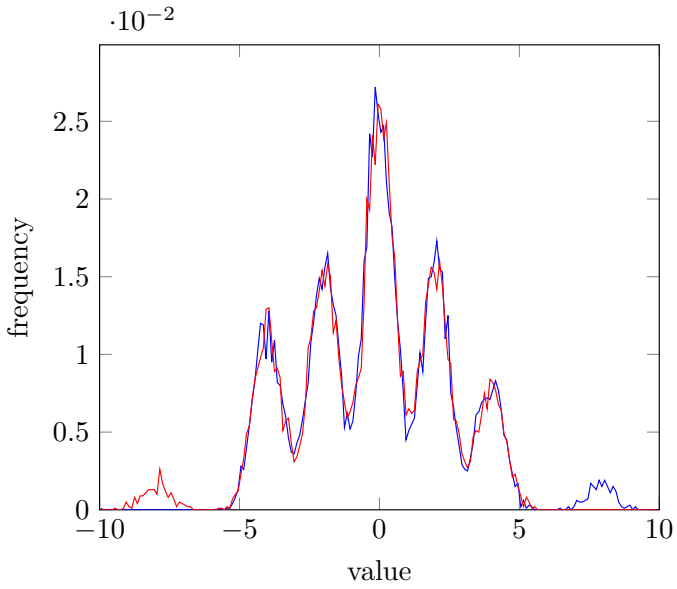


Figure 4: Probability density functions of instances of bags from two classes.

The above approach can be viewed as a building block and applied recursively. This has been demonstrated in [47] (see Appendix 6 for details), where the host has been modeled by servers it has visited and each server has been modeled by messages it has exchanged with the host (see Figure 5). Again, the important part of the approach is that function ψ projecting vectors describing messages is optimized using labels on the level of the host. The consequence is that ψ is sensitive in parts of the message space, where infected hosts behave differently to the normal ones. This knowledge can be utilized further to speed-up forensic analysis, as messages typical of infected hosts can be identified faster.

Also, the fact that the classifier is trained from labels on the level of hosts rather than on the level of individual messages [7] is important in network security, where labels are expensive to obtain a can be ambiguous. Consider for example HTTP request to `google.com`. This request can be caused by a user visiting a search engine or by a malware checking network connection and it is impossible to attach a correct label to this single flow.

The above framework is general and allows to model any tree-structured data, potentially any data stored in JSON format. [7] To assess the generality of the proposed models, the above user traffic has been extended by a hierarchical model of URL in HTTP request shown in Figure 6. The URL is viewed as having three separate MIL blocks, where the first MIL corresponds to a hostname (each part separated by a dot of the hostname is one instance), second corresponds to a path and the third corresponds to the query.

[7] JSON stands for Javascript object notation and it is a popular method to store structured data).

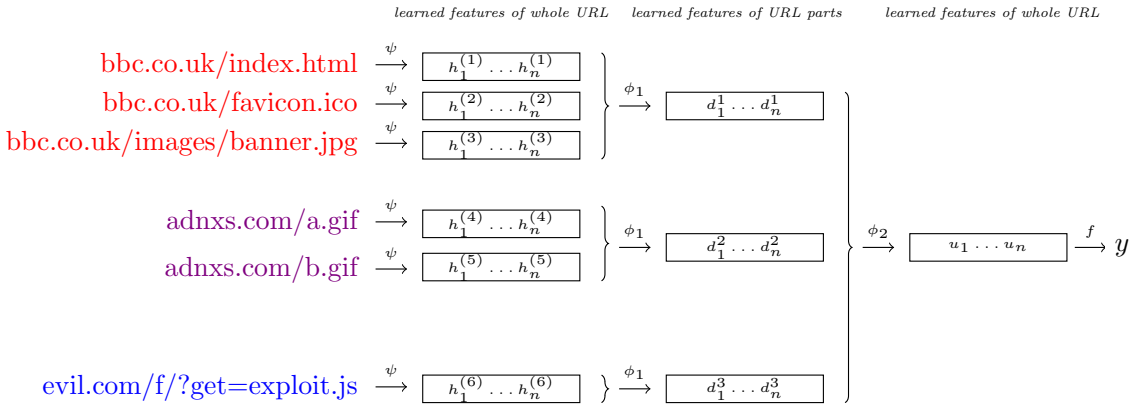


Figure 5: Hierarchical model of user behavior on HTTP servers using two stacked MIL problems. Each user is modeled on basis of visited servers (top MIL problem) and each server is modeled on basis of messages (HTTP requests) the user has exchanged with it (bottom MIL problem).

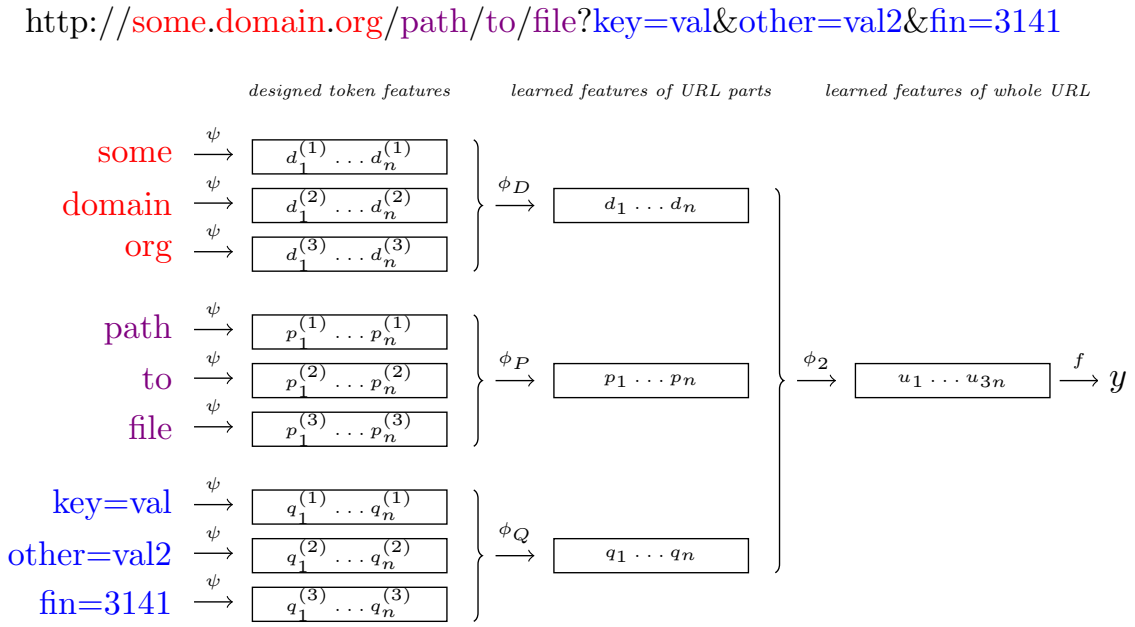


Figure 6: Hierarchical model of the HTTP request.

6 Clustering of structured data

Despite the progress in decreasing the false positive rates of anomaly detection methods, accurately labeled data are still needed, be it for evaluating the accuracy of detectors, determining detection thresholds, finding the right combination of detectors in an ensemble, or finally for assigning anomalies to classes of known malware to shorten the incident response time.

Nevertheless, labeling of samples by humans requires trained experts and even for them, it is challenging due to ambiguities listed at the end of the previous section. Therefore any improvement in simplifying the labeling or making it more effective is important.

A natural approach to increase the number of labeled samples is to first cluster them and then to present the expert with a cluster of samples. Besides increasing their number, a large number of similar samples helps to estimate similarities and variations in malware communication, which eases the writing of indicators of compromise (IOC) rules .

The efficacy of clustering depends on the used metric, which needs to be correlated with human expectations in the sense that samples in a cluster exhibit similarities the human expect. In [24] (see Appendix 7 for details) we have proposed to cluster hostnames by means of three metrics: first based on timing and sizes of network requests [32] described in Section 4, second based on a Jaccard index of a sets of tokens in queries, and finally the third measuring similarity of directory trees of domains as observed from URLs in HTTP requests [24]. The last metric has been inspired by kernels over graphs, but it has embraced the domain knowledge. Note that here the problem can be viewed as a multi-instance multi-domain clustering, where each observed HTTP request is one instance and the clustering is done in three sets of features (domain).

Once clusters are created, they need to be ordered and presented to analysts. The rank is based on the consistency of clusters in all three metrics (consistent clusters are easier to label) and relation to already known malicious hostnames with the relation estimated by probabilistic threat propagation [15].

A similar approach has been used to cluster malware binaries on basis of their interaction with resources during execution in sandboxed environment [56](see Appendix 8 for further detail), such that each cluster will contain one malware family. The assumption is that while it is relatively easy to change the structure of URL in HTTP requests or hide it using HTTPS, masking the intent is difficult due to the required interaction with resources. For example, if ransomware wants to encrypt the hard-drive, it needs to interact with the file system. Similarly, if malware wants to prevent multiple infections of the same host, it needs a mark indicating the host being already infected, which is typically done using windows registry or mutexes.

Since the malware typically uses resources multiple times and it uses multiple different types of resources (external servers, file-system, mutexes, and windows registry), the problem can be again framed as a multi-instance multi-domain clustering. Due to the unsupervised nature of the problem and instances being strings rather than vectors of fixed length, none of the above-described approaches were usable.

The core problem was therefore how to convert strings from each domain (URL, file path, registry keys, and mutexes) to a vector of fixed length. The approach proposed in [55] (see Appendix 9 for further detail) starts by defining similarity function for each

domain and optimize its parameters using Centered Kernel Target Alignment [17] from a couple examples of strings showing what should be similar and dissimilar. Using this function, strings from each domain were clustered and these cluster centers were used for one-hot encoding of each string from each domain. Note that typically used distances over strings like edit distance do not work well here since they do not exploit specifics of domains.

Once instances were represented by a binary vector, binaries were represented using a hierarchical Bayesian model with parameters optimized using expectation maximization algorithm. The main advantage of the used Bayesian model was that it has enabled to estimate the purity of founded clusters. This was used to rank founded clusters such that large and pure clusters will be presented to the analyst first.

Experimentally comparison has used a corpus of 130 000 malware binaries and compared to the state-of-the-art approaches for behavior clustering. The statistical tests of significance deemed our solution based on Bayesian models and tuned string similarities better.

7 Low false positive rate

The imbalance in benign and malicious samples, which in reality can span from 10^{-3} for the intrusion detection to 10^{-10} for steganography means that any detector for practical purposes has to have a very low false positive rate, ideally controlled by hyper-parameters with a clear meaning to the user. Needless to say that training of classifiers when the false positive is bounded (called Neyman-Pearson classification) remains an unresolved problem due to the high complexity of the problem [52, 53]. Therefore practical algorithms have to relax conditions such that the solution is found efficiently yet it remains usable.

Neyman-Pearson classification is not only important for bounding the false positive rate, but also for positive-unlabeled classification [8], where available samples are divided into two parts: positive set containing samples certainly being positives and unknown set with mostly negative samples, but some of them can be positive.

In [45](see Appendix 10 for details) we have postulated that the steganographer will send more than one message. This means that the attack rarely consists from a single malicious event, but more likely it will be repeated. We have therefore decided that it should be sufficient to detect 50% of malicious events (samples) as the overall performance measured on the level of attackers should not be decreased.

Fixing the detection threshold at 50% means that for sufficiently symmetric probability distributions the estimate of median can be safely replaced by a mean, and constraining the set of classifiers to linear and using convex surrogates (exponential, hinge, logistic regression) of indicator loss, the resulting optimization problem solved during training is convex. The proposed solution has been experimentally compared on a large dataset of almost 4.5 million images to an established prior art, which is to put different weights on errors on positive and negative classes. The experimental results have confirmed superiority of the proposed solution. Yet, recall that the solution is restricted only to the operating point at 50% detection accuracy.

In Section 4 it has been mentioned that combining outputs of anomaly detectors is non-trivial. When tight restrictions on false positive rate need to be met, we believe that the right combination cannot be learned unsupervisedly. Since anomaly-detectors can be viewed as feature extractors, the problem of learning the combination function using labeled samples can be framed as an accuracy at the top problem, where the goal is to learn classifier maximizing accuracy in top 1% of samples.

The accuracy at the top problem has been approached by modifying the approach in [9]. The modification consists from the use of gradient descend method, where the step in gradient descend algorithm is alternated with a shift of the threshold to be exactly at 1% quantile, which enforces the desired operating point. Although this algorithm [22] (see Appendix 11 for details) seems to be very naive, an extensive comparison revealed this solution to perform better than the state of the art, especially in conditions where samples are not accurately labeled. This aspect is important for network intrusion detection, where samples are rarely fully labeled. These findings are on par with the theoretical analysis in [8]. Compared to the original solution [9] with computationally complexity $O(n^4)$, the proposed one scales gracefully to large data with millions of samples.

8 Explainability

Almost all samples (events) considered suspicious in network security are passed to a human operator, who investigates them and performs an appropriate action (not investigating and doing nothing is considered as an action). To help her to decide, what to do (in the parlance of network security *to act*), she needs to understand causes of the alarm and to estimate the risk. This is even more important in the case of a large system with multiple alarms, where she needs to prioritize which alarms to resolve first. Therefore any additional information about alarms is valuable, as it can cut down the incident response time.

It is rather surprising that despite the practical implications of explanations of classifier's decisions, the prior art is scarce. In [34] (see Appendix 9 for details) we have proposed a general method to provide the user with an explanation of the anomaly in the form:

"The alarm has occurred because i^{th} feature is bigger than t_1, j^{th} feature is smaller than t_2, etc."

If this simple explanation takes as an input features anomaly detectors and is combined with a sub-system grouping hosts and servers into logical units, then the resulting explanations for the network officer can be meaningful. They can look like

"Computer `aaaa` belonging to a group of administrating staff uploaded more than 1000Mb to `box.com` service.",

which is well understood by a human. If use of `box.com` service violates company's security policy, the owner of the device can be immediately contacted.

To explain the anomaly, the system trains a large number of trees, where the set of positive samples has size one containing only the to be explained anomaly and the set of

negative samples contain remaining unknown samples or their randomly selected subset. From all trained trees, rules along paths from roots to lists with positive samples are extracted and the most frequent ones are assembled into the explanation.

The system, albeit being conceptually simple, provides shorter and sounder explanations than the prior art, which has been demonstrated a large set of 36 problems for anomaly detection. Moreover, it is general in the sense that it can be used to explain anomalies of a wide range of anomaly detectors operating over Euclidean space.

9 Future work

Although above sections have presented some solutions to problems identified in Introduction, they are not solved at all. Quite contrary, many questions remained unanswered and new question have been raised.

Structured domains and learning from unlabeled data

Section 5 have described a practical and general approach to reflect the structure of the data in an architecture of a neural network. The main advantage is its simplicity and generality, as almost any data with a tree structure (e.g. JSON documents) can be without much thinking encoded in the architecture of a neural network. Moreover, any progress within the field of neural networks, such as new optimization methods, layers, transfer functions, etc. can be readily applied.

Presently, the solution is applicable only to supervised training problems. For simple data represented by vectors[8] there exist works trying to decrease dependency on labeled samples, e.g. *semi-supervised learning* [31, 49] or *one-shot learning* [54, 58, 25]. At the moment the candidate is not aware of any similar solution for structured data. Similarly, prior art on anomaly detection for structured data is missing, except works restricted to the simplest case of multi-instance learning [40, 33, 30].

The simplicity with which the presented framework allows to encode the structured data is so impressive that it motivates us to further work in this area, mainly along the lines of decreasing the number of needed labeled samples. But there are also interesting theoretical questions. It is not known, how general is the framework and what are its limitations. For example, the selection of aggregation function (mean, maximum, learned function) is not entirely clear. From the theory behind MMD it seems like that mean should be sufficiently general to be able to differentiate any probability distribution function, yet for some applications, maximum is clearly more efficient in the finite sample and finite computational resources setting. Similarly it is not known if the solution would not suffer from vanishing gradients or forgetting similarly to recurrent neural networks when the tree grows taller. Some of these questions can be answered by applying the model to as many problems as possible, but some theoretical justifications would be in place.

[8] The candidate does not see a conceptual difference between vectors, matrices, and tensors.

Game theory in security domains

Game theory (GT) is by no means a correct mathematical tool to describe the adversarial nature of intrusion detection systems, but it is rarely used. Candidate's own experience suggests that it is because GT is computationally expensive, an improvement over non-GT models is hard to measure, GT frequently uses simplified and inaccurate models of the environment, and the attackers not being always rational.

Why is it difficult to assess improvement of GT models? Supervised classifiers are developed over data collected and labeled in the past and these data are frequently used for the evaluation. This corresponds to the scenario, where defender (detector) knows attacker's strategy, which seems to be bizarre, yet it is used in all antivirus and intrusion detection solutions. The reason for this is the presence of many attackers employing similar tools on many different targets. Therefore detecting already known attacks is a good strategy, because a detector not detecting them would be considered useless. Moreover, the detector will be evaluated on known attacks by third-party evaluators and if it fails to detect known attacks, its vendor will quickly go out of the business. This implies that the success of GT solutions have to be measured in the long term mainly on attack of new type, which is rarely done.

Solving the problem using Game Theory means that the detector has to be optimized with respect to all possible attacks. Putting for a moment aside, how these attacks can be found, the detector has to make a trade-off between false positive rate and detection accuracy. Increasing detection accuracy on unknown attacks causes either increasing the false positive rate or decreasing detection accuracy on known attacks, which is again against the usual evaluation framework.

Although solving GT models can be complex, the complexity depends on the assumptions and the chosen type equilibrium. Experimental results in [18] implies that if the attacker has a full knowledge about the domain, the solution can be found quickly and it works well enough for the case when an attacker does not have a full knowledge, where the solution is expensive to find. Similarly, finding Stackelberg equilibrium [12] might be easier than finding other types of equilibria while the loss of performance might not be dramatic. Finally, there remains open question, if it is better to have sub-optimal solution of a precise model or optimal solution to an imprecise one. Superiority of neural networks over support vector machines (or Gaussian processes) suggests the former approach to lead to more interesting solutions.

Finding a game-theoretic optimum is also interesting from the point of view of anomaly detectors, as it converts the problem of anomaly detection to that of supervised classification, which is a more researched area. Moreover, it would change the paradigm of the intrusion detection system, as instead of detecting attacks similar to already seen in the wild and identified by a security analyst, it would be able to detect never seen attacks.

Finding new attacks

An important aspect limiting the use of Game-Theory in computer security is that the space of all attacker's strategies is not fully known. This problem can be tackled by methods automatically finding new attacks undetectable by the current detector. The importance of this problem is demonstrated by Darpa's grand cybersecurity challenge [1].

Although in recent years several methods to create adversarial samples have been published [13, 41, 57], they are restricted to simple domains without complicated constraints. These constraints can be for example practical feasibility of the attack and satisfaction of requirements on the attack (e.g. the number of tried passwords in brute-force password cracking). Solutions to this problem will probably combine tools from many fields, such as planning, reinforcement learning, supervised learning, constraint satisfaction, automation of test development. Benefits of automatically finding new attacks go beyond Game Theoretic optimization. It can help to secure critical systems by identifying security holes, or it can guide the representation of the application domain, as it can reveal, which parts are not modeled yet they are important for the security.

Neyman-Pearson classification

Neyman-Pearson classification paradigm [53, 52] seems to be more appropriate for security domains than the usual Bayesian approach because it is easier to limit the number of false alarm (or the total number of alarms) than to define costs for all types of error and know the class ratio between malicious and benign use. Moreover, as identified in [8] Neyman-Pearson classification is important for learning from positive-unlabeled data, which is found in security domains.

Solutions presented in Section 7 have been developed and experimentally evaluated with linear classifiers. It is not known yet, how to extend them to non-linear classifiers. Generally, there seem to be few works dealing with this problem especially in conjunction with non-linear classifiers such as neural networks. The optimization problem there is not convex, therefore there will be no guarantees on optimality, yet the solution might lead to more understandable hyper-parameters and more precise classification around the operation point of interest.

References

[1] https://www.darpa.mil/program/cyber-grand-challenge.

[2] http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/.

[3] https://www.cisco.com/c/en/us/products/collateral/security/cognitive-threat-analytics/datasheet-c78-736557.html.

[4] https://www.omnicoreagency.com/facebook-statistics/.

[5] miproblems.org.

[6] Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence*, 201:81–105, 2013.

[7] Karel Bartos and Michal Sofka. Robust representation for domain adaptation in network security. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 116–132. Springer, 2015.

[8] Gilles Blanchard, Gyemin Lee, and Clayton Scott. Semi-supervised novelty detection. *Journal of Machine Learning Research*, 11(Nov):2973–3009, 2010.

[9] Stephen Boyd, Corinna Cortes, Mehryar Mohri, and Ana Radovanovic. Accuracy at the top. In *Advances in neural information processing systems*, pages 953–961, 2012.

[10] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[11] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[12] Michael Brückner and Tobias Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 547–555. ACM, 2011.

[13] S. Rota Bulò, B. Biggio, I. Pillai, M. Pelillo, and F. Roli. Randomized prediction games for adversarial machine learning. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2466–2478, Nov 2017.

[14] Christian Cachin. An information-theoretic model for steganography. In *International Workshop on Information Hiding*, pages 306–318. Springer, 1998.

[15] Kevin M Carter, Nwokedi Idika, and William W Streilein. Probabilistic threat propagation for network security. *IEEE Transactions on Information Forensics and Security*, 9(9):1394–1405, 2014.

[16] Andreas Christmann and Ingo Steinwart. Universal kernels on non-standard input spaces. In *Advances in neural information processing systems*, pages 406–414, 2010.

[17] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *Journal of Machine Learning Research*, 13(Mar):795–828, 2012.

[18] Karel Durkota, Viliam Lisý, Christopher Kiekintveld, Karel Horák, Branislav Bošanský, and Tomáš Pevný. Optimal strategies for detecting data exfiltration by internal and external attackers. In Stefan Rass, Bo An, Christopher Kiekintveld, Fei Fang, and Stefan Schauer, editors, *Decision and Game Theory for Security*, pages 171–192, Cham, 2017. Springer International Publishing.

[19] Tomáš Filler, Andrew D Ker, and Jessica Fridrich. The square root law of steganographic capacity for markov covers. In *Media Forensics and Security*, volume 7254, page 725408. International Society for Optics and Photonics, 2009.

[20] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy, 1996.

[21] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[22] M. Grill and T. Pevný. Learning combination of anomaly detectors for security domain. *Computer Networks*, 107:55 – 63, 2016. Machine learning, data mining and Big Data frameworks for network monitoring and troubleshooting.

[23] M. Grill, T. Pevný, and M. Rehák. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences*, 83(1):43 – 57, 2017.

[24] Jan Jusko, Martin Rehak, Jan Stiborek, Jan Kohout, and Tomas Pevny. Using behavioral similarity for botnet command-and-control discovery. *IEEE Intelligent Systems*, 31(5):16–22, 2016.

[25] L. Kaiserz, O. Nachum, A. Roy, and A. Bengio. Learning to remember rare events. In *5st International Conference on Learning Representations (ICLR) (workshop poster)*, May 2017.

[26] Andrew D Ker. Batch steganography and pooled steganalysis. In *International Workshop on Information Hiding*, pages 265–281. Springer, 2006.

[27] Andrew D Ker. A curiosity regarding steganographic capacity of pathologically non-stationary sources. In *Media Watermarking, Security, and Forensics III*, volume 7880, page 78800E. International Society for Optics and Photonics, 2011.

[28] Andrew D. Ker. The square root law of steganography: Bringing theory closer to practice. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '17, pages 33–44, New York, NY, USA, 2017. ACM.

[29] Andrew D Ker and Tomás Pevný. A new paradigm for steganalysis via clustering. In *Media Watermarking, Security, and Forensics III*, volume 7880, page 78800U. International Society for Optics and Photonics, 2011.

[30] Andrew D Ker and Tomáš Pevný. The steganographer is the outlier: realistic large-scale steganalysis. *IEEE Transactions on information forensics and security*, 9(9):1424–1435, 2014.

[31] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[32] J. Kohout and T. Pevný. Automatic discovery of web servers hosting similar applications. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1310–1315, May 2015.

[33] Jan Kohout and Tomáš Pevný. Network traffic fingerprinting based on approximated kernel two-sample test. *IEEE Transactions on Information Forensics and Security*, 13(3):788–801, 2018.

[34] M. Kopp, Holeňa M., and T. Pevný. Anomaly explanation with random forests. *IEEE Transactions on Knowledge and Data Engineering*, submitted.

[35] Brian Krebs. https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/.

[36] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[37] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.

[38] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 147–152. ACM, 2006.

[39] Siwei Lyu and Hany Farid. Steganalysis using higher-order image statistics. *IEEE transactions on Information Forensics and Security*, 1(1):111–119, 2006.

[40] Krikamol Muandet, Kenji Fukumizu, Francesco Dinuzzo, and Bernhard Schölkopf. Learning from distributions via support measure machines. In *Advances in neural information processing systems*, pages 10–18, 2012.

[41] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.

[42] Tomáš Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

[43] Tomáš Pevný and Jessica Fridrich. Novelty detection in blind steganalysis. In *Proceedings of the 10th ACM workshop on Multimedia and security*, pages 167–176. ACM, 2008.

[44] Tomáš Pevný and Andrew D Ker. The challenges of rich features in universal steganalysis. In *Media Watermarking, Security, and Forensics 2013*, volume 8665, page 86650M. International Society for Optics and Photonics, 2013.

[45] Tomáš Pevný and Andrew D Ker. Towards dependable steganalysis. In *Media Watermarking, Security, and Forensics 2015*, volume 9409, page 94090I. International Society for Optics and Photonics, 2015.

[46] Tomáš Pevný, Martin Rehák, and Martin Grill. Identifying suspicious users in corporate networks. In *Proceedings of workshop on information forensics and security*, pages 1–6, 2012.

[47] Tomas Pevny and Petr Somol. Discriminative models for multi-instance problems with tree structure. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, pages 83–91, New York, NY, USA, 2016. ACM.

[48] Tomáš Pevný and Petr Somol. Using neural network formalism to solve multiple-instance problems. In Fengyu Cong, Andrew Leung, and Qinglai Wei, editors, *Advances in Neural Networks - ISNN 2017*, pages 135–142, Cham, 2017. Springer International Publishing.

[49] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[50] M. Rehák, M. Pěchoucek, M. Grill, J. Stiborek, K. Bartoš, and P. Čeleda. Adaptive multiagent system for network traffic monitoring. *IEEE Intelligent Systems*, 24(3):16–25, May 2009.

[51] Martin Rehak, Michal Pechoucek, Karel Bartos, Martin Grill, and Pavel Celeda. Network intrusion detection by means of community of trusting agents. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '07, pages 498–504, Washington, DC, USA, 2007. IEEE Computer Society.

[52] Clayton Scott. Performance measures for neyman–pearson classification. *IEEE Transactions on Information Theory*, 53(8):2852–2863, 2007.

[53] Clayton Scott and Robert Nowak. A neyman-pearson approach to statistical learning. *IEEE Transactions on Information Theory*, 51(11):3806–3819, 2005.

[54] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943, 2013.

[55] Jan Stiborek, T. Pevný, and Martin Rehák. Multiple instance learning for malware classification. *Expert Systems with Applications*, 93:346–357, 2018.

[56] Jan Stiborek, Tomáš Pevný, and Martin Rehák. Probabilistic analysis of dynamic malware traces. *Computers & Security*, 2018.

[57] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 12 2013.

[58] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

Appendices

p. 23 — Pevný, T. Loda: *Lightweight on-line detector of anomalies. Machine Learning*. 2016, 102(2), s. 275-304, IF = 1.848. Authorship = 100%.

p. 53 — Grill, M., Pevný, T., Rehák, M.: *Reducing False Positives of Network Anomaly Detection by Local Adaptive Multivariate Smoothing*. Journal of Computer and System Sciences. 2017, 83(1), s. 43-57, IF = 1.678 Authorship = 20%.

p. 69 — KER, A. D. a Pevný, T.: *The Steganographer is the Outlier: Realistic Large-Scale Steganalysis*. IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. 2014, 9(9), s. 1424-1435, IF = 4.332, Authorship = 50%.

p. 81 — Kohout, J. a Pevný, T.: *Network traffic fingerprinting based on approximated kernel two-sample test*. IEEE Transactions on Information Forensics and Security. 2017, PP(99), IF = 4.332 Authorship = 50%.

p. 95 — Pevný, T. a SOMOL, P.: *Using Neural Network Formalism to Solve Multiple-Instance Problems*. In: Advances in Neural Networks - ISNN2017. 2017, s. 135–142. Authorship = 50%.

p. 103 — Pevný, T. a SOMOL, P.: *Discriminative Models for Multi-instance Problems with Tree Structure*. In: Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security. 2016, s. 83-91. Authorship = 50%.

p. 113 — Jusko, J. and Rehák, M. and Stiborek, J. and Kohout, J. and Pevný, T.: *Using Behavioral Similarity for Botnet Command-and-Control Discovery*. IEEE Intelligent Systems. 2016, 31(5), s. 16-22, IF = 2.374 Authorship = 5%.

p. 121 — Stiborek, J., Pevný, T., Rehák, M.: *Probabilistic analysis of dynamic malware traces*. Computer & Security. to appear, IF = 3.928 Authorship = 40%.

p. 139 — Stiborek, J., Pevný, T., Rehák, M.: *Multiple instance learning for malware classification*. Expert Systems with Applications. 2018, 93, s. 346–357, IF = 3.928 Authorship = 40%.

p. 151 — Pevný, T. a KER, A. D.: *Towards dependable steganalysis*. In: Proceedings of SPIE Media Watermarking, Security, and Forensics 2015. SPIE Photonics West 2015, 07.02.2015 - 12.02.2015. Authorship = 67%.

p. 165 — Grill, M. a Pevný, T.: *Learning Combination of Anomaly Detectors for Security Domain*. Computer Networks. 2016, 107(1), s. 55-63, IF = 2.516 Authorship = 20%.

p. 175 — Kopp, M., Holeňa, M., Pevný: *Anomaly Explanation with Random Forests*. Submitted to IEEE Transactions on Knowledge and Data Engineering, Authorship = 40%.

Mach Learn (2016) 102:275–304
DOI 10.1007/s10994-015-5521-0

Loda: Lightweight on-line detector of anomalies

Tomáš Pevný[1,2]

Received: 2 November 2014 / Accepted: 25 June 2015 / Published online: 21 July 2015

Abstract In supervised learning it has been shown that a collection of weak classifiers can result in a strong classifier with error rates similar to those of more sophisticated methods. In unsupervised learning, namely in anomaly detection such a paradigm has not yet been demonstrated despite the fact that many methods have been devised as counterparts to supervised binary classifiers. This work partially fills the gap by showing that an ensemble of very weak detectors can lead to a strong anomaly detector with a performance equal to or better than state of the art methods. The simplicity of the proposed ensemble system (to be called Loda) is particularly useful in domains where a large number of samples need to be processed in real-time or in domains where the data stream is subject to concept drift and the detector needs to be updated on-line. Besides being fast and accurate, Loda is also able to operate and update itself on data with missing variables. Loda is thus practical in domains with sensor outages. Moreover, Loda can identify features in which the scrutinized sample deviates from the majority. This capability is useful when the goal is to find out what has caused the anomaly. It should be noted that none of these favorable properties increase Loda's low time and space complexity. We compare Loda to several state of the art anomaly detectors in two settings: batch training and on-line training on data streams. The results on 36 datasets from UCI repository illustrate the strengths of the proposed system, but also provide more insight into the more general questions regarding batch-vs-on-line anomaly detection.

Electronic supplementary material The online version of this article (doi:10.1007/s10994-015-5521-0) contains supplementary material, which is available to authorized users.

Editor: Joao Gama.

✉ Tomáš Pevný
pevnak@gmail.com

[1] Department of Computers and Engineering, Czech Technical University in Prague, Karlovo naměstí 13, 121 35 Prague, Czech Republic

[2] Cisco Systems, Inc., Cognitive Research Team in Prague, Prague, Czech Republic

1 Introduction

Imagine identifying anomalous users in a social network (Šourek et al. 2013), where user's behavior constantly changes and their numbers are enormous, detecting weirdly behaving computers (frequently an indication of an infection by malware) in a network of large corporation with hundreds of thousands of computers, whose traffic constantly changes (Pevný et al. 2012), or identification of fraudulent card transactions (Akhilomen 2013) realized thorough big credit providers. These domains share similar features, which is processing of enormous number of samples with constantly changing characteristics. Most fast versions of existing anomaly detection methods, especially those based on indexing techniques, require the data to be available in one single batch and to fit in the memory, which is in aforementioned domains clearly impossible. Moreover, data's non-stationarity forces detector's models to be continuously updated, which is again very difficult with indexing techniques, as created indexes would need to be recalculated, which is usually expensive. Other methods, such as Bay and Schwabacher (2003) assumes some additional knowledge which might not be available. The presented anomaly detector has been designed with respect to these constraints, and it has been shown to achieve state of the art accuracy measured by area under ROC curve.

The definition of an anomaly, outlier, or novelty is not unified in the literature. Recent book (Aggarwal 2013a) considers all these terms to be equivalent, and defines them as follows: *"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism."* According to this definition, *outliers* are generated by the same probability distribution as normal samples, but they are very rare. In this text, we define *anomalies* to be samples generated by a different probability distribution than normal ones, and their presence in the data to be rare, less than 1–10 % of the data. One-class problem is similar to the anomaly-detection problem with the difference that training set contains samples exclusively from the normal class and testing set contains may contain samples from other classes at any rate.

An ideal detector would model joint probability of data-generating processes. Although this goes against the principle of never solving a more difficult process than is needed (density estimation vs. classification), the knowledge of the probability of observed samples is useful[1] information in making a decision about their anomalousness. Modeling joint probability is generally difficult in spaces of many dimensions and in practice some simplifications have to be made. The detector presented here, further called Loda, approximates the joint probability by using a collection of one-dimensional histograms, where every one-dimensional histogram is constructed on an input space projected onto a randomly generated vector. The rationale behind the use of one-dimensional histograms is that they can be efficiently constructed in one pass over data and the query operation needed during classification is simple. Consequently, Loda's complexity is linear with respect to the number of training samples n and the dimension of the input space d.

Although one one-dimensional histogram is a very weak anomaly detector, their collection yields to a strong detector. This phenomenon (collection of weak classifiers result in a strong classifier) is already a well established paradigm in supervised classification (Kuncheva 2004; Freund and Schapire 1996), but has not been demonstrated in unsupervised anomaly detection, as most ensemble systems used in anomaly detection (Aggarwal 2013b; Lazarevic and Kumar 2005; Tan et al. 2011; Liu et al. 2008) use individual detectors of much higher complexity than that of a one-dimensional histogram. The comprehensive experimental section

[1] The hypothetical knowledge of the probability function generating the data would allow to formulate the problem as a hypothesis test.

(see Sect. 4) demonstrates that the proposed ensemble of one-dimensional histograms has accuracy measured by the area under ROC curve competitive with established solutions of much higher complexities. Loda therefore achieves a very good accuracy to complexity ratio and therefore it is well suited for processing large data.

Besides being fast, Loda is able to deal with missing variables and can rank features according to their contribution to sample's anomalousness. Both of these capabilities are important for practitioners. For example, the knowledge of which features caused the anomaly can serve as guidance for further investigation and can decrease overall cost of anomaly investigation. This knowledge can also be used to cluster similar anomalies together which is again useful during investigation. Interestingly, none of these abilities substantially increase Loda's computational complexity. Note that both abilities can be achieved with any homogenous ensemble-based anomaly detector, where individual detectors within are diversified by sub-space sampling. Detectors with sub-space sampling were recently proposed (Keller et al. 2012; Nguyen et al. 2010; Muller et al. 2011) but neither the robustness against missing features or the explanation of the cause of a sample's anomalousness had been mentioned.

Loda is controlled by two hyper-parameters: number of one dimensional histograms and number of histogram bins. Because the hyper-parameter setting is particularly cumbersome in most anomaly detection methods where few or no anomalous samples are available (most anomaly detection), a method to determine these parameters solely on the basis of observed samples is presented. This makes Loda hyper-parameter free.

Loda's accuracy is firstly extensively compared to the prior art trained in the batch mode on 36 problems downloaded from the UCI database (Frank and Asuncion 2010) in the category of classification problems with numerical attributes. The reason, why Loda is compared to detectors processing only data presented in one batch is that we wanted to compare Loda to state of the art detectors, even though they are not applicable for intended scenario. The experimental results reveal that Loda's measured by area under ROC curve is equal to and many times better than that of other algorithms with higher computational complexity requiring batch training (we expect algorithms trained in batch to perform better than on-line algorithms). Consequently Loda provides a good alternative to the already established solutions on problems with a large number of features and samples that need to be processed efficiently. Secondly, Loda is compared to the prior art on streaming problems and it is shown when it is better to continuously update histogram and when to do so in batches. Finally, Loda's capability to efficiently process big datasets is demonstrated on the WEB UCI dataset (Ma et al. 2009) with 2.4 million samples and 3.2 million features.

The contribution of this paper is threefold: (i) it demonstrates that an ensemble of very weak anomaly detectors can lead to a strong anomaly detector; (ii) it presents a lightweight anomaly detector suitable for data-streams robust to missing variables and identifying causes of anomalies; (iii) it extensively compares existing and the proposed anomaly detector, which sheds some light on conditions in which a particular anomaly detector excels.

This paper is organized as follows: The next section reviews relevant prior art, shows its computational complexity, and discusses issues related to on-line learning and classification. Loda is presented in Sect. 3. In Sect. 4, Loda is experimentally compared to the prior art, its low computational requirements are demonstrated on artificial data, and its efficiency is demonstrated on a large-scale dataset (Ma et al. 2009). The same section also demonstrates Loda's ability to handle data with missing values and explain a sample's anomalousness. Finally Sect. 5 concludes the paper.

For better reproducibility, all source code, results, datasets, and their creation is available at http://agents.cz/~pevnak.

2 Related work

The recent book (Aggarwal 2013a) and survey (Chandola et al. 2009) contain many methods for anomaly and outlier detection. Below, those relevant to this work are reviewed.

2.1 Model-centered detectors

Basic model-centered anomaly detectors assume that data follow a known distribution. Detectors (Shyu et al. 2003) based on principal component analysis (PCA) assume that the data fit a multi-variate normal distribution. Despite this being rarely true in practice, they frequently outperform more complicated detectors. The complexity of their training is $O(nd^3)$, where d is the dimension of the input space and n is the number of samples used for training.

A One-Class Support Vector Machine (Schölkopf et al. 2001) (1-SVM) does not make any assumptions about the distribution of data. It finds the smallest area where $1-\nu$ fraction of data are located (ν is parameter of the method specifying desired false-positive rate). This is achieved by projecting the data to a high-dimensional (feature) space and then finding a hyperplane best separating the data from the origin. It has been noted that when 1-SVM is used with a linear kernel it introduces a bias to the origin (Tax and Duin 2004), which can be removed by using a Gaussian kernel. The Support Vector Data Description algorithm (Tax and Duin 2004) (SVDD) removes the bias in 1-SVM by replacing the separation hyperplane by a sphere encapsulating most of the data. The complexity of both methods is super-linear with respect to the number of samples, n, being $O(n^3 d)$ in the worst-case.

Isolation Forest (Liu et al. 2008) (IForest) relies on a collection of Isolation Trees grown by using unlabeled data. During growth of a tree, its internal nodes are added until the terminal leafs contain one sample or the maximum depth is reached. The anomaly score is proportional to the level of leaf reached by the sample, as the idea is that anomalies will reach leafs at the base of the tree (close to the root), while legitimate samples reach leafs closer to the root. The complexity of training one tree is $O(n \log n)$ and of classification $O(\log n)$, where the authors recommend subsampling of training samples to increase diversity in the ensemble, robustness against anomalies within the data, and simultaneously decrease complexity of the training and classification.

The recently proposed FRAC (Noto et al. 2012) aimed to bridge the gap between supervised and unsupervised learning. FRAC is an ensemble of models, each estimating one feature based on other features (for data of a dimension d, FRAC uses d different models). The rationale behind this is that anomalous samples exhibit different dependencies among features, which can be detected from prediction errors modeled by histograms. FRAC's complexity depends on the algorithm used to implement individual models, which can be large, considering that a search for possible hyper-parameters needs to be undertaken. Because of this, an ordinary linear least-square regression is used here leading to the complexity $O(nd^4)$. FRAC is not well suited for on-line learning since an update of models changes the distributions of errors that cannot be estimated from one sample.

The on-line training of model-centered detectors is generally difficult as the algorithms used to create the model have non-trivial complexity with respect to the dimension, or models cannot be updated incrementally (e.g. principal component analysis). The on-line adaptation of 1-SVM is discussed in Kivinen et al. (2004), but the solution is an approximation of the solution returned by the batch version. The exact on-line version of SVDD is described in Tax and Laskov (2003), but the algorithm requires substantial bookkeeping thereby increasing the complexity. The proposed Loda is also model-based detector and as will be shown in

Sect. 4, its one dimensional histograms can be easily updated based on upcoming samples which makes the whole algorithm friendly to on-line learning.

2.2 Data-centered detectors

Data-centered detectors do not have any built-in model. A sample's anomalousness is determined according to its distance to all previously observed samples. Consequently, there is no training phase as new samples are just added to the set of already observed samples. However, this increases the complexity of the classification phase, which is a linear function of the number of samples n.

A k-nearest neighbor (Knorr and Ng 1999) (KNN) is a popular method to identify outliers inspired by the corresponding method from classification. It ranks samples according to their distance to kth-nearest neighbor. It has been recently shown that a variant of KNN (Sricharan and Hero (2011)) converges to the optimal density-level detector for a given false positive rate. Nevertheless KNN has been criticized for not being able to detect outliers in data with clusters of different densities (Breunig et al. 2000). The local outlier factor (Breunig et al. 2000) (LOF) solves this problem by defining the outlier score as a fraction of sample's distance to its kth-nearest neighbor and the average of the same distance of all its k nearest neighbors. True inliers have a score around one while outliers have much greater score. We refer to Zimek et al. (2012) for a comprehensive review of the prior art.

The complexity of the classification phase of nearest-neighbor based detectors is driven by the nearest-neighbor search, which is an $O(n)$ operation in the worst case. More efficient approaches based on bookkeeping (Pokrajac et al. 2007), better search structures like KD-trees (Bentley 1975), or approximate search (Andoni and Indyk 2006) have been adopted. Nevertheless, the complexity of all methods depends in some way on the number of training samples n, and better search structures are usually useful only in low dimensions.

2.3 On-line anomaly detectors

There are few works in on-line anomaly detection. The closest work to this is Half-Space trees (Tan et al. 2011) (HS-trees), which is a method similar to Isolation Forest with the difference that decision rules within tree-nodes are generated randomly. The output of each HS-tree can be interpreted as a crude estimate of the probability density function, which is further refined by taking a sufficient number of them. It is worth noting that HS-trees assume that the data is scaled such that values of features are bounded in [0, 1]. This is in a sharp contrast to Loda only requiring features to have approximately the same scale, which is a more relaxed condition, especially if concept drift occurs. HS-trees handle concept-drift by dividing data-streams into sample batches of size 256, where HS-trees trained on a previous batch are used to scrutinize samples in a given batch. Simultaneously new HS-trees are learned on a current batch and once all samples from the current batch are processed, new HS-trees replaces the old one.

OLINDDA (Spinosa et al. 2009) uses standard k-means clustering to group previously observed samples into clusters called concepts. Anomalies not belonging to any cluster are grouped into candidate clusters, which based on cohesiveness criteria are either promoted as a new cluster and novel concept is reported, or evicted. Thus OLINDDA focuses on detection of a novel concepts in streams rather than on identification of anomalies.

2.4 Ensembles and random projections

Ensembles have so far been underutilized in anomaly detection. A significant portion of the prior art focuses on a unification of anomaly scores (Gao and Tan 2006; Schubert 2012), because different detectors within one ensemble may provide anomaly scores of different magnitudes (Nguyen et al. 2010). Diversifying detectors by random subsets of features was presented (Lazarevic and Kumar 2005; Keller et al. 2012) to improve the accuracy, especially on high dimensional problems.

Projections of the input space onto the arbitrary sub-space have been utilized mainly in distance-based outlier detection schemes to speedup the search of nearest neighbors. de Vries et al. (2010) performs the kth-NN search in the LOF first in the random sub-space on a larger neighborhood and then it is refined by the search in the original input space. Similarly, Pham and Pagh (2012) estimates the distribution of angles between samples in the input space from the distribution of angles in the sub-space (distribution of angles has been proposed in Kriegel and Zimek (2008) as a good anomaly criterion). In experiments in Sect. 4.3 the similar approach is used on prior art to compare it to Loda.

Most prior art employs ensembles and random projections with data-centered detectors, which have high computational complexity and are not well suited for real-time processing. Exceptions are Isolation Forest (Liu et al. 2008) and Half-Space Trees (Tan et al. 2011) relying on an ensemble of trees, each having a relatively low accuracy. This is similar to Loda which uses even simpler model (one-dimensional histogram) with even lower complexity, but their combination is similarly powerful. Random projections in Loda are used to project the input space into a single dimension, which simplifies the complexity of all operations over it. Loda's sparse random projections can be considered as a sub-space sampling method. Unlike the prior art, the sub-space method is not used only to increase efficacy, but also to gain robustness against missing variables and the ability to find causes of anomalousness of a given sample.

Interpreting the decision in anomaly detection is very important in practice since it can reduce the cost of subsequent analysis and increases trust of the detector. The work of Knorr and Ng (1999) has focused on finding a minimal sub-space in which a given sample is an outlier. Algorithms presented therein are data-centered and suitable only for low-dimensions. HiCS algorithm (Keller et al. 2012) and the algorithm of Dang et al. (2013) both aim to identify a sub-space in which a scrutinized sample is an outlier. Both are data-centered algorithms and consequently their computational complexity is prohibitive for processing data-streams.

Loda's method to identify relevant features bears similarity to HiCS algorithm. But unlike HiCS, Loda's method is general and can be used on all ensembles diversified by random sub-space sampling. Finally, contrary to all prior art, the identification of relevant features does not increase Loda's computational complexity in big O notation.

3 Description of Loda

Loda is comprised of a collection of k one-dimensional histograms $\{h_i\}_{i=1}^k$, each approximating the probability density of input data projected onto a single projection vector $\{w_i \in \mathbb{R}^d\}_{i=1}^k$. Projection vectors $\{w_i\}_{i=1}^k$ diversify individual histograms, which is an essential requirement for ensemble systems to improve performance of a single classifier / detector. Their initialization is described in detail below in Sect. 3.1.

Loda's output, $f(x)$, on a sample, x, is an average of the logarithm of probabilities estimated on individual projection vectors. Adopting $\hat{p}_i$ to denote the probability estimated by

Mach Learn (2016) 102:275–304

Algorithm 1: Loda's training (update) routine.

input: data samples $\{x_i \in \mathbb{R}^d\}_{i=1}^n$;
output: histograms $\{h_1, \ldots, h_n\}$, projection vectors $\{w_i\}_{i=1}^k$. ;
initialize projection vectors with $\left[d^{-\frac{1}{2}}\right]$ non zero elements $\{w_i\}_{i=1}^k$;
initialize histograms $\{h_i\}_{i=1}^k$;
for $j \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** k **do**
    $z_i = x_j^{\mathrm{T}} w_i$;
    update histogram h_i by z_i;
  end
end
return $\{h_i\}_{i=1}^k$ and $\{w_i\}_{i=1}^k$.

ith histogram and w_i to denote the corresponding projection vector, Loda's output $f(x)$ can be written as

$$f(x) = -\frac{1}{k}\sum_{i=1}^{k} \log \hat{p}_i(x^{\mathrm{T}} w_i), \tag{1}$$

which can be reformulated as

$$\begin{aligned} f(x) &= -\log\left(\prod_{i=1}^{k} \hat{p}_i(x^{\mathrm{T}} w_i)\right)^{\frac{1}{k}} \\ &\overset{(a)}{\sim} -\log p(x^{\mathrm{T}} w_1, x^{\mathrm{T}} w_2, \ldots, x^{\mathrm{T}} w_k), \end{aligned} \tag{2}$$

where $p(x^{\mathrm{T}} w_1, x^{\mathrm{T}} w_2, \ldots, x^{\mathrm{T}} w_k)$ denotes the joint probability of projections. Equation (2) shows that Loda's output is proportional to the negative log-likelihood of the sample, which means that the less likely a sample is, the higher the anomaly value it receives. This holds under a strong assumption (a) in (2) that probability distributions on projection vectors w_i and w_j are independent $\forall i, j \in k,\ i \neq j$. Even though this is almost never true in practice, Loda still delivers very good results. We believe that the reasons are similar to those in naïve Bayes classifiers theoretically studied by Zhang (2004), which give conditions under which the effects of conditional dependencies cancel out. These conditions depend on the probability distribution of both classes and they are difficult to be verified in practice as they require an exact knowledge of conditional dependencies among features. Due to Loda's similarity to the Parzen window detector (Yeung and Chow 2002), the similar argumentation might explain Loda's surprisingly good performance.

In high-dimensional spaces Loda can be related to a PCA based detector (Shyu et al. 2003), because projection vectors w_i and w_j, $i \neq j$ will be approximately orthogonal (this is due to their random construction described below). Assuming again the independence of $x^{\mathrm{T}} w_i$ and $x^{\mathrm{T}} w_j$, the projected data are orthogonal and uncorrelated, which are properties of Principal Component Analysis (PCA). Loda's histogram on random projections are therefore similar to histograms on principal components.

Loda's training and classification routines are summarized in Algorithms 1 and 2, respectively. Loda is initialized by generating a set of sparse random projections $\{w_i\}_{i=1}^k$, $w_i \sim N(0, \mathbf{1}^d)$ with $d^{\frac{1}{2}}$ non-zero components and initializing the corresponding set of histograms

Algorithm 2: Loda's classification routine on sample x.

input: sample x, set of histograms $\{h_i\}_{i=1}^k$ and projection vectores $\{w_i\}_{i=1}^k$.;
output: anomaly value $f(x)$;
for $i \leftarrow 1$ **to** k **do**
  $z_i = x^{\mathrm{T}} w_i$;
  obtain $\hat{p}_i = \hat{p}_i(z_i)$ from h_i ;
end
return $f(x) = -\frac{1}{k}\sum_{i=1}^k \log \hat{p}_i(z_i)$;

$\{h_i\}_{i=1}^k$. Each histogram is updated by a training sample by projecting the sample onto a vector and then the corresponding histogram bin is updated. The classification procedure follows similar steps, but instead of updating histograms, they return probabilities whose logarithms are averaged and returned. Notice that the construction requires only one pass over the data and can be used on data-streams by first classifying a new sample and then updating all histograms.

The rest of this section describes creation of and rationale behind sparse projection vectors w_i, presents how sparse projections enable robustness against missing variables and allow explanation of sample's anomalousness, and closes by commenting issues related to building on-line histograms.

3.1 Random projections

Each sparse projection vector $\{w_i\}_{i=1}^k$ is created during initialization of the corresponding histogram by first randomly selecting $d^{-\frac{1}{2}}$ non-zero features (d is the dimension of the input space) and then randomly generating non-zero items according to $N(0, 1)$. The choice of normal distribution of non-zero items comes from the Johnson–Lindenstrauss (1984) lemma showing that with this choice, L_2 distances between points in the *projected space* approximate the same quantity in the *input space*. The $d^{-\frac{1}{2}}$ sparsity is due to Li (2007) showing that L_2 distances can be preserved with random projections with only $d^{-\frac{1}{2}}$ non-zero elements. Another justification for the use of sparse random projections is to increase diversity among histograms by making them work on different sub-subspaces. This is a popular diversification technique used for example in Random Forest (Ho 1998).

One can ask a question, are random projections actually needed? Would a detector based on an ensemble of histograms of individual features (in this section called per-feature detector) have similar accuracy? According to the experimental results on problems used throughout this paper (described in Sect. 4 in detail), random projections consistently improve the performance (see Fig. 10 in "Appendix"). On four out of five rate of anomalies Loda was better than the per-feature detector. Wilcoxon signed-rank test assessing if both detectors delivers the same performance was rejected with a p-value 0.028. Which means that the Loda is better and the difference is statistically significant. For more details see comments under Fig. 10.

3.2 Missing variables

Sparse projections enable Loda to handle missing variables by calculating the output only from those histograms whose projection vector has a zero on the place of missing variables. To formalize the approach, assume that $x \in \mathbb{R}^d$ is a sample with missing variables, and $\mathcal{J}$ is the index-set of missing variables. Let $\mathcal{I}(x)$ be a set of indices of histograms whose

projections have all entries in $\mathcal{J}$ zero, i.e. $(\forall i \in \mathcal{I}(x))\,(\forall j \in \mathcal{J})\,\big(w_{ij} = 0\big)$, where w_{ij} is jth element of the projection vector of ith histogram. Then, the anomaly score for the sample x is calculated as

$$f(x) = -\frac{1}{|\mathcal{I}(x)|}\sum_{i\in\mathcal{I}(x)} \log h_i(x). \tag{3}$$

Since the output of all histograms within Loda have the same meaning there is no need for calibration as in Nguyen et al. (2010). The final output is reliable even if several histograms are omitted. Notice that by using the same mechanism, Loda can be also trained on data with missing variables, as only detectors from $\mathcal{I}(x)$ are updated upon observing sample x.

3.3 Explaining the cause of an anomaly

Two independent works (Rondina et al. 2014; Somol et al. 2011) propose a feature selection method for a binary classification based on the comparison of scores (e.g. classification of error) of classifiers on randomly generated sub-spaces. Recognizing that each histogram with *sparse* projections in Loda provide an anomaly score on a randomly generated sub-space, this method can be used to rank features according to theirs contribution to sample's anomalousness.

Let $\hat{p}_i$ denote the probability estimated by ith histogram on a sample x, and $\mathcal{I}_j/\overline{\mathcal{I}}_j$ denote index sets of histograms that use/do not use jth feature. Formally $(\forall i \in \mathcal{I}(x))\,\big(w_{ij} = 0\big)$ and $\big(\forall i \in \overline{\mathcal{I}}(x)\big)\,\big(w_{ij} \neq 0\big)$, where w_i is the projection vector of ith histogram. The score function proposed in the prior art calculates the difference of means of $-\log \hat{p}_i$ over $\mathcal{I}_j$ and $\bar{\mathcal{I}}_j$, which means that if an anomaly score with a feature being present is much higher than that of without, the feature is important for identification of outliers. We improve this score function by recognizing that it's main goal is to assert if the contribution of jth feature is statistically significant, for which we use one-tailed two-sample t test with a test statistic

$$t_j = \frac{\mu_j - \bar{\mu}_j}{\sqrt{\frac{s_j^2}{|\mathcal{I}_j|} + \frac{\bar{s}_j^2}{|\bar{\mathcal{I}}_j|}}}, \tag{4}$$

where $\mu_j/\bar{\mu_j}$ is the mean and $s_j^2/\bar{s}_j^2$ variance of $-\log \hat{p}_i$ calculated with $i \in \mathcal{I}_j/i \in \bar{\mathcal{I}}_j$. The higher the t_j the more important the j^{th} feature is. Since the complexity of the contrast function (4) is linear with respect to the number of projections k and number of features d, the calculation of a feature's contrast increases Loda's complexity by a constant in big O notation. Experiments demonstrating the feature selection are presented in Sect. 4.6.

3.4 Histogram

The one-dimensional histogram on random projections is an important design element in Loda, as it determines its learning mode (batch vs. on-line). The prior art on histogram construction and the optimal number of bins is very rich. In the following only the tiny subset relevant to on-line construction on streams is recapitulated.

The most common approach for batch data are equi-width or equi-depth histograms with bins having either the same length or containing the same number of samples. In database research, V-Optimal histograms (Poosala et al. 1996) minimizing weighted variance of estimated frequency are popular, but their construction has prohibitive complexity $O(n^2)$. Moving to data-steams, approximations of V-Optimal histograms have been studied in Guha et al. (2006), but the algorithms therein are quite complex. Simpler Partition Incremental

Discretization (Gama and Pinto 2006) constructs a histogram in two steps: (i) create an equi-width histogram with small bins in a single iteration through the data; (ii) use this fine histogram to return either equi-width or equi-depth histogram with a given number of bins. The advantage is that user-provided partition in step (i) is only indicative, as bins can be split if they already contain too many samples. The disadvantage for Loda is that the second step needs to be triggered before classifying a sample if the fine histogram was updated. DADO algorithm (Donjerkovic et al. 2000) constructs incremental histograms close in quality to V-optimal histograms. The key idea behind this technique is to internally represent every bin by two sub-bins. Counts in sub-bins are used for bin-split and bin-merge operations and for their triggering. DADO's biggest advantage for Loda is a fixed number of buckets which implies fixed memory requirements. An interesting alternative was proposed in Ben-Haim and Tom-Tov (2010) originally for determining splitting points in decision trees. Unlike all previous approaches it does not require any knowledge about the range of values in histogram (see Appendix "On-line histogram" for its recapitulation), but according to our results Loda with equi-width histogram is better.

With the exception of the last algorithm, all above approaches require knowledge of the range of modeled values in advance. This can be usually estimated from a sample of data, but in the case of severe non-stationarity, the histogram's support can shift outside the initial range. A simple solution for equi-width histograms is to specify bin width and store bin counts in a hash-map structure with keys $\kappa = \left\lfloor \frac{x}{\Delta} \right\rfloor$, where κ is an integer key, Δ is the bin width, and x is the value to be inserted. This approach has $O(1)$ time complexity and if coupled with count-min-sketch (Cormode and Muthukrishnan 2005) its space complexity can be upper bounded. It also allows the modeling fixed length windows in a stream by keeping samples (indexes of bins) to be later removed in memory. This can have potentially large memory footprint, which can be decreased by using exponential buckets proposed in Datar et al. (2002). Alternatively, two histograms as in HS-Trees (Tan et al. 2011) can be used, where the older one is used for classification while the newer one is being constructed on newly arrived samples. Once the construction of the new one is finished, it replaces the old one and the construction of the new one is started. Both constructions are experimentally compared in Sect. 4.2 with interesting conclusions.

The determination of an optimal number of histogram bins b is an important design parameter. For an *equi-width histogram*, there exists a simple method to determine optimal number of histograms (Birgé and Rozenholc 2006), which is used in Loda. The method of Birgé and Rozenholc (2006) maximizes penalized maximum likelihood in the form

$$\sum_{i=1}^{b} n_i \log \frac{b n_i}{N} - \left[b - 1 + (\log b)^{2.5} \right],$$

where n_i is the number of samples that falls in ith bin and $N = \sum_{i=1}^{b} n_i$ is the total number of samples. Penalization factor $b - 1 + (\log b)^{2.5}$ penalizes histograms with too many bins.

3.5 Computational and storage complexity

Loda's complexity is mainly determined by the type of the histogram. Assuming equi-width histogram and sparse random projections the time complexity of learning is $O(nkd^{-\frac{1}{2}})$, where n is the number of training samples, d is the dimensionality of the input space, and k is the number of histograms. The time complexity of classification is $O(kd^{-\frac{1}{2}})$. As discussed

Table 1 Time and space complexity of the anomaly detection algorithms compared in this paper

	Time complexity		Space complexity
	Batch training	Classification	
Batch			
FRAC	$n \cdot d^4$	$O(d^2)$	$d^2 b$
KNN	$O(1)$	$O(nd)$	$O(nd)$
LOF	$O(1)$	$O(knd)$	$O(nd)$
PCA	$O(nd^4)$	$O(kd)$	$O(kd)$
1-SVM	$O(n^2 d) \sim O(n^3 d)$	$O(nd)$	$O(nd)$
IForest	$O(kl \log l)$	$O(k \log l)$	$O(kl)$
On-line			
HS-Trees	–	$O(k(h+l))$	$O(kh^2)$
Loda two hist.	$O(nkd^{-\frac{1}{2}})$	$O(k(d^{-\frac{1}{2}}+b))$	$O(k(d^{-\frac{1}{2}}+b))$
Loda cont.	$O(nkd^{-\frac{1}{2}})$	$O(kd^{-\frac{1}{2}})$	$O(k(d^{-\frac{1}{2}}+b+l))$

The classification times for on-line detectors (HS-Trees, Loda with continuous histograms and Loda with two alternating histograms) include time to update the detector. The time to train Loda on batch data is included for comparison purposes. n is the number of samples in the training set, d is the number of features, k in LOF is the number of nearest-neighbor points, k in PCA is the number of components retained after the projection, k in IForest, HS-Trees, and Loda is the number of trees or histograms, h in HS-Trees is a maximal height of the tree, l in IForest is the number of samples used to construct one tree, l in HS-Trees and Loda with continuous histogram is the length of observation window, and finally b in FRAC and Loda is the number of histogram bins. The FRAC implementation assumes ordinary least-square regression as features predictors. "Loda cont." denotes Loda with naïve implementation of continuously updated histogram. "Loda two hist." denotes Loda with two alternating histograms

in the previous sub-section the space complexity can be made $O(k(d^{-\frac{1}{2}}+b))$ with b being number of histogram bins.

For a floating window histogram over a length of l the space complexity can be made $O(k(d^{-\frac{1}{2}}+b\log l))$ by using the algorithm of Datar et al. (2002) if space is constrained, otherwise $O(k(d^{-\frac{1}{2}}+b+l))$ for a naïve approach storing all l values for discounting them on expiration.

Alternatively, on-line histogram can be implemented by using two alternating histograms as in Tan et al. (2011), where the older histogram is used for the classification while the new one is constructed. Once the construction of the new one is finished (it has accommodated l samples), it replaces the older one for the classification and the new construction of the new histogram one is started. The advantage is possibly smaller space complexity $O(k(d^{-\frac{1}{2}}+b))$ and according to the experimental results in Sect. 4.2 better robustness to clustered anomalies. Table 1 shows the time and space complexity of Loda with continuous and two equi-width alternating histograms and the prior art used in the experimental section.

4 Experiments

Below Loda is compared to relevant state of the art in three different scenarios. The first scenario simulates anomaly detection on stationary problems, where all detectors are first

trained on a training set and then evaluated on a separate testing set. This scenario is included because it enables the comparison of Loda to other anomaly detectors, which are designed for batch training and their accuracy should be superior due to more sophisticated training algorithms. The second scenario mimics streamed data, where anomaly detectors first classify a sample and then use it to update their model. The third scenario uses a dataset with millions of samples and features to demonstrate that Loda is able to efficiently handle big data. The section is concluded by an experimental comparison of detectors time complexity in classify and update scenarios and by the demonstration of robustness against missing variables and the identification of causes of anomalies.

4.1 Stationary data

Loda has been compared to the following anomaly detectors chosen to represent different approaches to anomaly detection: PCA based anomaly detector (Shyu et al. 2003), 1-SVM (Schölkopf et al. 2001), FRAC (Noto et al. 2012), δ-version of kth-nearest neighbor detector (Harmeling et al. 2006) (KNN) which is equivalent to Sricharan and Hero (2011) with $\gamma = 1$ and $s = d$ (in notation of the referenced work) if the area under ROC curve (AUC) is used for evaluation, LOF (Breunig et al. 2000), and IForest (Liu et al. 2008).

Benchmarking data were constructed by the methodology proposed in Emmott et al. (2013) converting real-world datasets (Frank and Asuncion 2010) to anomaly detection problems. The methodology produces set of normal and anomalous samples, where samples from different classes have different probability distribution, which is aligned with our definition of anomalous samples. Created problems are divided according to (i) difficulty of finding an anomaly (*easy*, *medium*, *hard*, *very hard*), (ii) scatter / clusteredness of the anomalies (*high scatter*, *medium scatter*, *low scatter*, *low clusteredness*, *medium clusteredness*, *high clusteredness*), (iii) and finally with respect to the rate of anomalies within the data $\{0, 0.005, 0.01, 0.05, 0.1\}$. The training and testing set were created such that they have the same properties (difficulty, scatter/clusteredness, and rate of anomalies)[2] with clustered anomalies located around the same point. Note that the total number of unique combinations of problem's properties is up to 120 for each dataset. The construction of individual problems is recapitulated in Appendix "Construction of datasets".

The quality of the detection is measured by the usual area under ROC curve (AUC). Since from every dataset (out of 36) up to 120 problems with different combinations of problem's properties can be derived, it is impossible to present all 7×4200 AUC values. Tables with AUCs averaged over difficulty of detecting anomaly are in the supplemental and cover 15 pages. They are therefore presented in an aggregated form by using critical difference diagrams (Demšar 2006) which show average rank of detectors together with an interval in which Bonferroni–Dunn correction of Wilcoxon signed ranks test cannot reject the hypothesis that the detectors within the interval have the same performance as the best one. The average rank comparing AUC is calculated over all datasets on which all detectors provided the output, following Demšar (2006) lower rank is better. AUCs used for ranking on each dataset are an average over all combinations of problem parameters with fixed parameter(s) of interest. For example, to compare detectors on different rates of anomalies within the data, for each rate of anomalies the average AUC is calculated over all combinations of difficulty and scatter. Average AUCs of different detectors on the same dataset are ranked, and the average of ranks over datasets is the final average rank of a detector on one rate of anomalies within the data.

[2] The dataset with no anomalies contained no anomalies in the training data and 10 % of anomalies in the evaluation data, which captures the case the case when the anomaly detector is trained on clean data.

Table 2 Summary of detectors hyper-parameters used in experiments on stationary data in Sect. 4.1.2

Detector	Hyper-parameters
FRAC	–
PCA	Used all components with eigenvalues greater than 0.01
KNN	$k = \max\{10, 0.03 \times n\}$
LOF	$k = \max\{10, 0.03 \times n\}$
1-SVM	$\nu = 0.05$, $\gamma =$ inverse of median of L_2 distances of samples
IForest	100 trees each constructed with 256 samples
Loda	All hyper-parameters optimized automatically

To decrease the noise in the data caused by statistical variations, experiments for every combination of problem properties were repeated 100 times. This means that every detector was evaluated on up to $100 \times 120 \times 35 = 4.32 \times 10^5$ problems. Benchmark datasets created as described in Appendix "Construction of datasets" were varied between repetitions by randomly selecting 50 % of normal samples (but maximum of 10,000) to be in the training set and putting the remaining data to the set. Similarly, anomalous samples were first selected randomly from all anomalous samples to reach the desired fraction of outliers within the data, and then randomly divided between training and testing set. The data in the training and testing set have the same properties in terms of difficulty, clusteredness, and fraction of anomalies within.

4.1.1 Settings of hyper-parameters

Setting hyper-parameters in anomaly detection is generally a difficult unsolved problem unless there is a validation ground truth available for the parameter tuning. Wrong parameters can cause an otherwise well designed anomaly detector to fail miserably. Nevertheless, in this context we do not aim to solve the problem as our intention here is primarily to compare multiple detectors against each other. For this purpose we follow hyper-parameter setting guidelines given by the authors of the respective methods. Note that our proposed method does not require manual hyper-parameter setting. Employed parameter settings are detailed below and in Table 2.

Settings of the number of nearest neighbors in our implementations of LOF and KNN algorithms followed (Emmott et al. 2013; Breunig et al. 2000) and was set to $\max\{10, 0.03 \times n\}$, where n is the number of samples in the training data. 1-SVM with a Gaussian kernel used $\nu = 0.05$ and width of the kernel γ equal to an inverse of median L_2 distance between training data. SVM implementation has been taken from the libSVM library (Chang and Lin 2011). Our implementation of the FRAC detector used ordinary linear least-square estimators, which in this setting does not have any hyper-parameters. Our implementation of PCA detector based on principal component transformation used top k components capturing more than 95 % variance. IForest [implementation taken from Jonathan et al. (2014)] used parameters recommended in Liu et al. (2008): 100 trees each constructed with 256 randomly selected samples.

The number of histograms in Loda, k, was determined similarly to the number of probes in feature selection in Somol et al. (2013). Denoting $f_k(x)$ Loda's output with k histograms, the reduction of variance after adding another histogram can be estimated as

$$\hat{\sigma}_k = \frac{1}{n}\sum_{i=1}^{n} |f_{k+1}(x_i) - f_k(x_i)| .$$

Although $\hat{\sigma}_k \to 0$ as $k \to \infty$, its magnitude is problem-dependent making it difficult to set a threshold on this quantity. Therefore $\hat{\sigma}_k$ is normalized by $\hat{\sigma}_1$, and the k is determined as

$$\arg\min_k \frac{\hat{\sigma}_k}{\hat{\sigma}_1} \geq \tau,$$

where τ is the threshold. In all experiments presented in this paper, τ was set to 0.01. Unless said otherwise, Loda used equi-width histogram with the number of histogram bins, b, determined for each histogram separately by maximizing penalized maximum likelihood method of Birgé and Rozenholc (2006) described briefly in Sect. 3.4. With b being set, width of histogram bins in equi-width histograms was set to $\frac{1}{b}(x_{\max} - x_{\min})$.

Setting of hyper-parameters is summarized in Table 2.

4.1.2 Experimental results

Figure 1a–e shows the average rank of detectors plotted against rate of anomalies within the training set for different clusteredness of anomalies (AUCs were averaged only with respect to the difficulty of anomalies). Since highly scattered anomalies could not be produced from the used datasets, the corresponding plot is omitted. The number of datasets creating problems with medium scattered anomalies is also low, and we should not draw any conclusions as they would not be statistically sound.

For problems with low number of low-scattered and low-clustered anomalies the data-centered detectors (KNN and LOF) are good. With increasing number of anomalies in data and with their increasing clusteredness, however, the performance of data-centered detectors appears to deteriorate (note LOF in particular). Additional investigation revealed that this effect can be mitigated by hyper-parameter tweaking, specifically by increasing the number of neighbours k. In practice this is a significant drawback as there is no way to optimize k unless labeled validation data is available.

The model-centered detectors (Loda, IForest as well as 1-SVM) appear to be comparatively more robust with respect to the increase in number of anomalies or their clusteredness, even if used with fixed parameters (see Sect. 4.1.1.).

In terms of statistical hypothesis testing the experiments do not show marked differences between the detectors in question. Note that in isolated cases FRAC and 1-SVM perform statistically worse than the respective best detector.

Figure 1f shows the detectors time to train and classify samples with respect to the size of the problem measured by the dimension multiplied by the number of samples.[3] On small problems KNN, LOF, and 1-SVM are very fast, as their complexity is mainly determined by the number of samples, but Loda is not left behind too much. As the size of problems gets bigger Loda quickly starts to dominate all detectors followed by PCA. Surprisingly IForest with low theoretical complexity had the highest running times. Since its running time is almost independent of the problem size, it is probably due to large multiplication constant absorbed in big O notation.

Comparing detectors by the performance/time complexity ratio, we recommend KNN for small problems with scattered anomalies, while for bigger problems and problems with

[3] Because all algorithms have not used any search for hyper-parameters, Loda used fixed number of histogram being equal to $\sqrt{n}$, where n is the number samples.

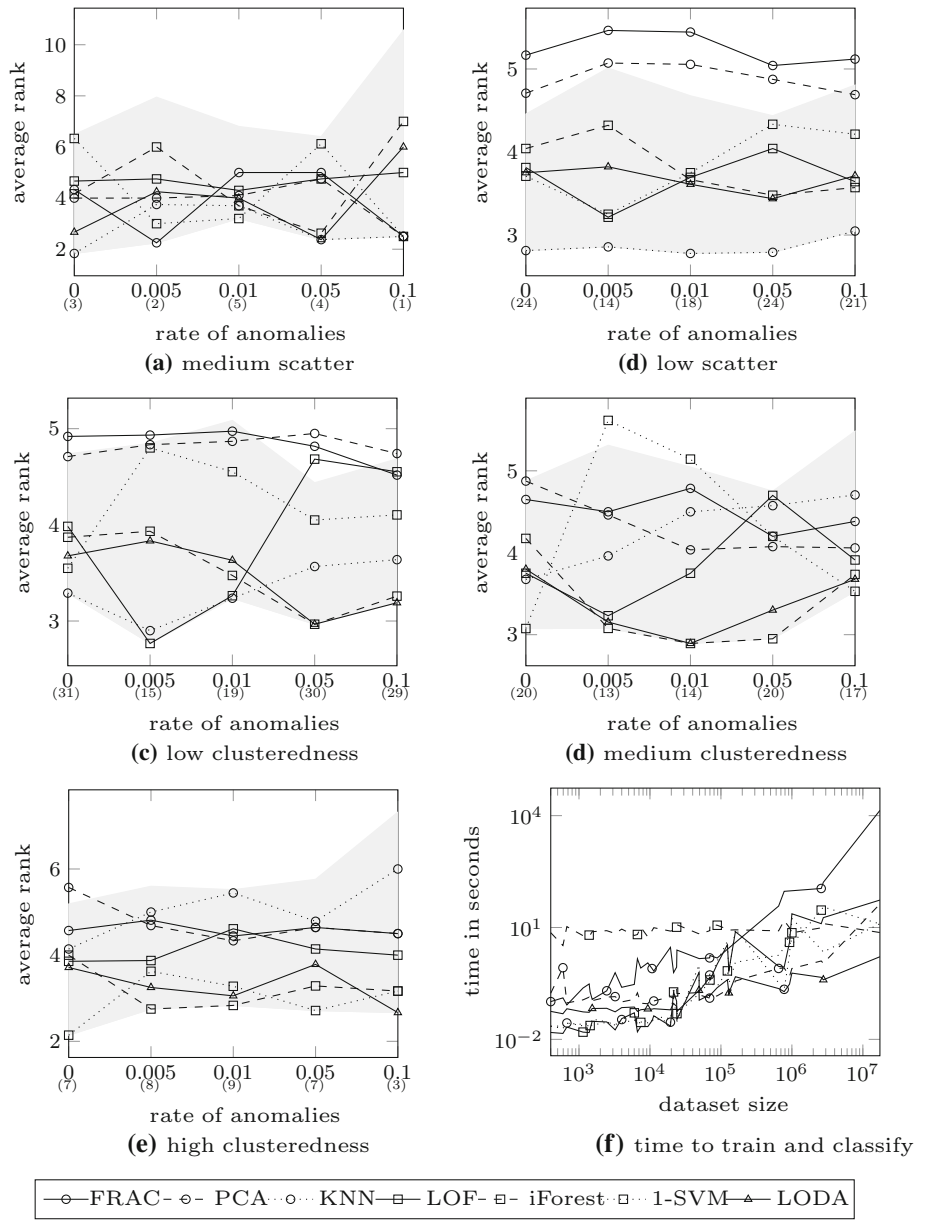
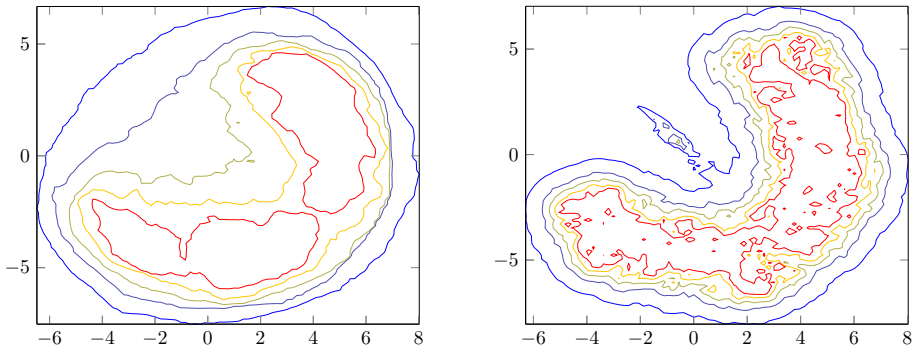


Fig. 1 **a–e** The average rank of detectors with respect to the rate of anomalies within the training data for various levels of clusteredness. The average rank is calculated by averaging rank of detectors over datasets, where the rank of detectors on a single dataset compares AUCs. The *small number in parentheses* shows the number of datasets from which benchmarking problems with a given combination of clusteredness and rate of anomalies were created. Critical difference is shown as a *light grey*—it is the area in which a in which Bonferroni–Dunn correction of Wilcoxon signed ranks test cannot reject the hypothesis that the detectors within has the same performance as the best detector. **f** The time to train the detector and classify all testing samples



(a) Loda **(b)** k-nearest neighbor

Fig. 2 Isolines at false positive rates {0.01, 0.05, 0.1, 0.2, 0.4} of Loda and k-nearest neighbor detector on a banana dataset, **a** Loda, **b** k-nearest neighbor

clustered anomalies we recommend the proposed Loda, as its performance is similar to IForest but it is much faster.

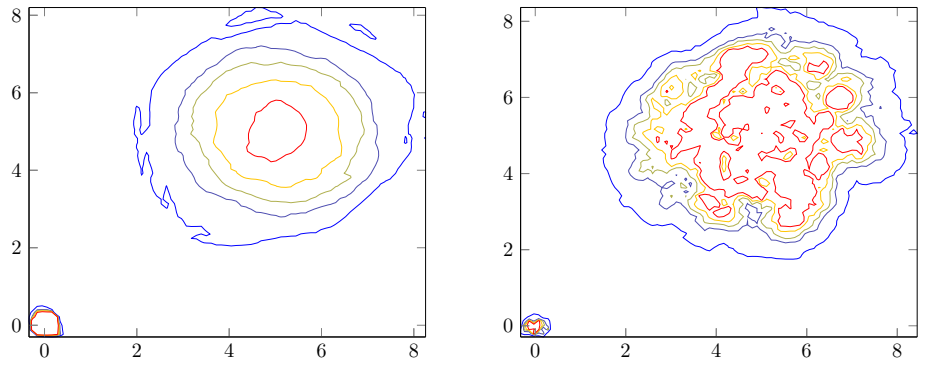
The investigation of problems on which Loda is markedly worse than KNN revealed that Loda performs poorly in cases where the support of the probability distribution of nominal class is not convex and it encapsulates the support of the probability distribution of the anomalous class. A typical example is a "banana" dataset in Fig. 2 showing isolines at false positive rates {0.01, 0.05, 0.1, 0.2, 0.4} of Loda and KNN. Isolines show that Loda has difficulty modeling the bay inside the arc. Although the adopted construction of problems from UCI datasets (see Appendix "Construction of datasets" for details) aimed to create these difficult datasets, Loda's performance was overall competitive to other, more difficult detectors.

Breunig et al. (2000) has criticized KNN, which has performed the best on clean training data, for its inability to detect local outliers occurring in datasets with areas of very different densities. Figure 3 shows again isolines at false positive rates {0.01, 0.05, 0.1, 0.2, 0.4} of Loda and LOF (designed to detect local outliers) on a data generated by two Gaussians with very different standard deviations (a typical example of such data). Loda's isolines around the small compact cluster (lower left) are very close to each other, whereas isolines of the loose cluster (upper right) are further away which demonstrates that Loda is able do detect local outliers similarly to LOF. LOF's wiggly isolines also demonstrates the over-fitting to the training data.

Figure 4 visualizes the average rank of Loda with equi-width, equi-depth, and on-line (Ben-Haim and Tom-Tov 2010) histograms on different rates of anomalies within the data. Although on almost all rates Friedman's statistical test accepted the hypothesis that all three versions of Loda performs the same, Loda with equi-width histograms is obviously better. This is important because equi-width histograms can be efficiently constructed from a stream of data if the bin-width is determined, for example by using sample data.

4.2 Streamed and non-stationary data

This section compares Loda with floating-window histogram and two alternating histograms (see Sect. 3.4) to Half-Space Trees (HS-Trees) (Tan et al. 2011) [implementation taken from Tan (2014)]. On-line version of 1-SVM was omitted from the comparison due to the difficulties with hyper-parameter setting and inferior performance when algorithms were



(a) Loda on Two Gaussians **(b)** Local outlier factor on Two Gaussians

Fig. 3 Isolines at false positive rates {0.01, 0.05, 0.1, 0.2, 0.4} of Loda and Local Outlier Factor on a dataset composed of two Gaussians with different SD, **a** Loda on Two Gaussians, **b** Local outlier factor on Two Gaussians

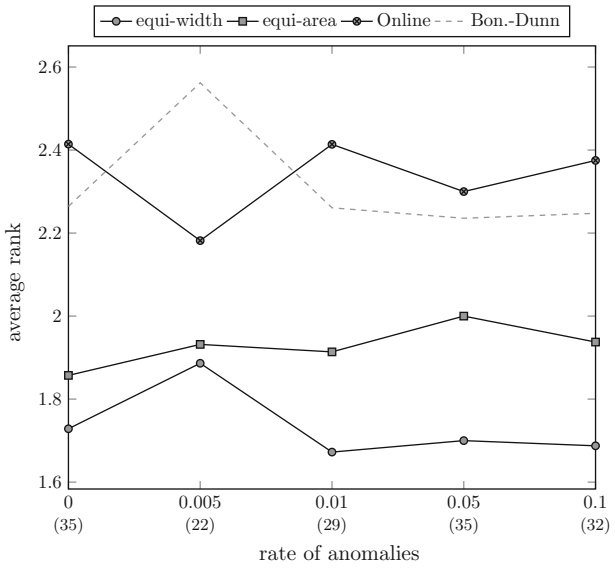


Fig. 4 Average rank of Loda with different types of histograms with respect to the rate of anomalies within the training data. The rank of three compared detectors is calculated for each problem separately, and then the average over all problems with a given rate of anomalies in the training data is shown in the *graph*

compared in the batch learning. The comparison is made on seven datasets: Shuttle, Covertype, HTTP and SMTP data from kdd-cup 99. HTTP and SMTP data are used in two variants: (i) with all 41 features (called HTTP-full and SMTP-full) and (ii) with 3 features (called HTTP-3 and SMTP-3) used in Tan et al. (2011).[4]

In every repetition of an experiment the testing data were created by replacing maximum of 1 % of randomly selected normal samples by randomly selected anomalous samples. Order

[4] The original publication (Tan et al. 2011) does not specify which three features from kdd-cup dataset were used, but datasets used in the publication are available at Tan (2014).

Table 3 AUCs of Loda and HS-Trees (Tan et al. 2011) on streaming problems with various levels of clusteredness

	Dataset	Continuous		Two histograms		HS-Trees	
		AUC	Time	AUC	Time	AUC	Time
Clusteredness = 1	Covertype	**0.987**	4.36	0.985	3.04	0.980	19.00
	http-3	**0.995**	7.42	0.989	5.91	0.993	37.82
	http	**0.998**	7.84	0.983	5.68	0.995	35.30
	shuttle	0.992	0.54	0.992	0.39	**0.999**	4.90
	smtp	0.994	1.25	0.989	1.01	**0.998**	8.25
	smtp-3	0.890	1.52	0.883	1.27	**0.915**	8.38
	smtp+http	0.992	6.51	0.987	5.88	**0.998**	42.56
Clusteredness = 4	Covertype	0.972	4.42	**0.989**	3.00	0.980	19.05
	http-3	0.992	7.51	**0.994**	5.24	0.983	40.17
	http	0.991	8.40	**0.993**	6.00	0.990	34.87
	shuttle	0.980	0.49	0.994	0.41	**0.999**	5.02
	smtp	0.970	1.34	0.994	1.06	**0.996**	8.58
	smtp-3	0.871	1.35	0.886	1.11	**0.914**	8.31
	smtp+http	0.989	9.65	0.993	7.99	**0.998**	42.70
Clusteredness = 16	Covertype	0.937	4.42	**0.989**	3.01	0.980	19.33
	http-3	0.987	6.71	**0.996**	4.95	0.980	40.84
	http	0.982	8.43	**0.993**	6.27	0.987	34.52
	shuttle	0.943	0.51	0.994	0.42	**0.999**	5.12
	smtp	0.947	1.34	0.993	1.06	**0.995**	8.63
	smtp-3	0.863	1.38	0.882	1.17	**0.915**	8.35
	smtp+http	0.983	7.99	0.995	4.80	**0.998**	42.20

Bold-faced values are the best values for a given dataset and clusterdness

of normal samples has not been permuted in order to preserve data continuity. Thus, the variation between repetitions comes from (i) the position to where anomalous samples were inserted and (ii) from the selection of anomalous samples to be mixed in. To study the effect of clustered anomalies, anomalies were first selected to form clusters in space (as in the previous section) and then they replaced sequences of normal data of the same size. By this process data-streams contained anomalies clustered in time and space of sizes {1, 4, 16} (this quantity is hereafter called clusteredness). Experiments on each dataset for every value of clusteredness of anomalies were repeated 100 times.

HS-Trees used recommended configuration of 50 trees of depth 15 with minimum of 20 samples in leaf node. Loda's configuration, namely number of histograms and bins were optimized on first 256 samples in every repetition, thus there were no parameters to be set manually. Both algorithms used a window of length 256 samples, which is the hard-coded length in implementation of HS-Trees provided by authors.

Table 3 shows AUCs and average time to process samples of Loda and that of HS-Trees. We can see that both algorithms deliver nearly the same performance, while Loda is on average 7–8 times faster (the time complexity is treated in more detail in the next subsection). Figure 5 shows AUCs on segments of 120,000 continuous samples on datasets Covertype, HTTP, HTTP-3, and SMTP+ HTTP with clusters of 16 anomalies (omitted datasets did not

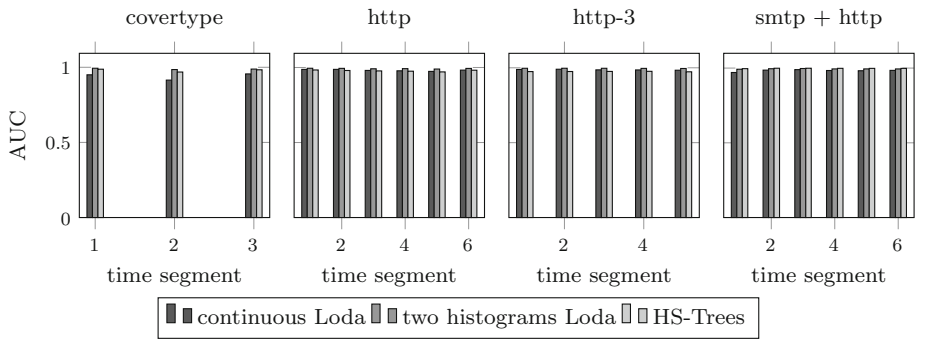


Fig. 5 AUCs of Loda with continuous histogram, Loda with two alternating histograms, and HS-Trees on segments of continuous 12,000 samples from the stream. AUCs in every segment is an average from 100 repetitions

contained enough samples). Since all AUCs of all three detectors remains stable, it can be concluded that all detectors equally well combat the concept drift in datasets.

Further investigation of results reveals that Loda with two alternating histograms (construction similar to that in HS-Trees) is more robust to clustered anomalies than Loda with continuous histogram. Our explanation of this phenomenon is that clustered anomalies are classified most of the time by histograms built on clean data without anomalies, while polluted histograms classify most of the time clean data, which is not influenced by anomalies in training data. These results demonstrate that the right choice of histogram for non-stationary data should depend on the type of anomalies (time and space clustered vs. scattered).

4.3 Big data

The demonstration of Loda's ease of handling big data is done on a URL dataset (Ma et al. 2009) containing 2.4 million samples with 3.2 million features. Normal samples contain features extracted from random URLs, while anomalous samples have features extracted from links in spam e-mails during 120 days at rate 20,000 samples per day (see Ma et al. 2009 for dataset details). Due to the high number of "anomalous" samples and cleanliness of normal samples, the dataset belongs to a category of one-class problems where the goal is to separate class of legitimate URLs from others.

Since the data are presented in batches of 20,000 samples per day, all methods evaluated in Sect. 4.1 can be theoretically applied by training them on normal samples from the previous day, i.e. detectors classifying samples from lth day are trained on normal samples from the $(l-1)$th day. The problem is that the computational complexity effectively prevents a direct application of FRAC, 1-SVMs, PCA, LOF, and KNN, as methods would not finish in reasonable time. The only methods that can be used without modification is Loda and IForest. Their settings were same as in Sect. 4.1 with the difference that (i) Loda used 500 projections and histograms bins were optimized on data from day zero and then stay fixed; (ii) number of trees within IForest was increased to 1000. Rows labeled 3.2×10^6 in Table 4 show AUCs, average rank over 120 days, and the average time to classify new samples and retrain the detectors. We can see that Loda was not only significantly better (AUC of IForest is close to the detector answering randomly), but also more than 15 times faster.

To reduce the computational complexity such that other algorithms can be used, the original data of dimension 3.2 million were projected onto 500 dimensional space with a randomly

Table 4 Average rank (caption rank), average AUC (caption AUC), and average time to classify and update detectors on 20 000 samples (caption time) on the full problem ($d = 3.2 \times 10^6$) and on the reduced problem ($d = 500$)

d	Detector	Rank	AUC	Time
500	KNN	1.596	0.818	140.5s
	PCA	2.758	0.803	1.4s
	Loda	3.067	0.795	1.16s
	FRAC	3.958	0.789	283.8s
	IForest	4.733	0.776	6.4s
	LOF	4.987	0.771	307.5s
	1-SVM	6.900	0.528	711.4s
3.2×10^6	Loda	1.025	0.795	21.6s
	IForest	1.975	0.488	369s

The average rank is an average of detector's rank over 120 days

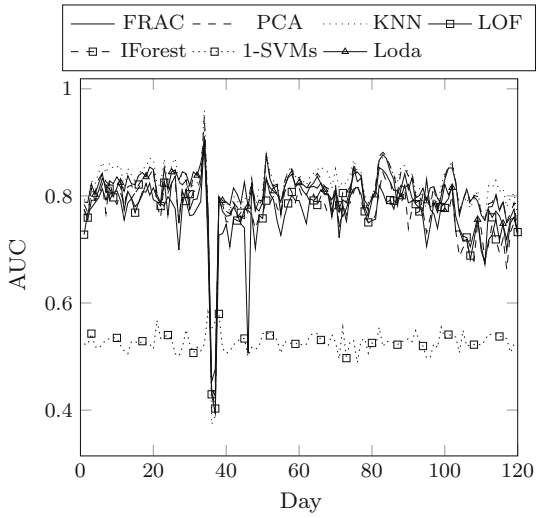


Fig. 6 AUCs of detectors on URL dataset with respect to time. AUCs of PCA, KNN, IForest, and 1-SVMs is on dataset projected to 500 dimension. AUCs of Loda are on the full dataset of dimension 3.2 million

generated matrix $W \in \mathbb{R}^{3.2\times10^6,500}$, $W_{ij} \sim N(0, 1)$. According to the Johnson-Lindenstrauss lemma, L_2 distances between points should be approximately preserved, therefore most detectors should work. Hyper-parameters of all detectors were set as in Sect. 4.1.1. AUCs of all detectors on every day are shown in Fig. 6, and their summaries again in Table 4. On this new dataset the best algorithm was the KNN algorithm followed by PCA and Loda. These results are consistent with those in Sect. 4.1.2, where KNN detector was generally good if trained on clean data. The good performance of PCA can be explained by projected data being close to multivariate normal distribution, although the Lilliefors' test has rejected the hypothesis that marginals follow the normal distribution with 85 % rate. Notice that Loda's execution time was more than 100 times faster than KNN, as its time to classify and update on samples from one day was 1.16s on average, while that of KNN was 140s.

4.4 Time complexity

Loda was primarily designed to have low computational complexity in order to efficiently process large data and data-streams. This feature is demonstrated below, where times to

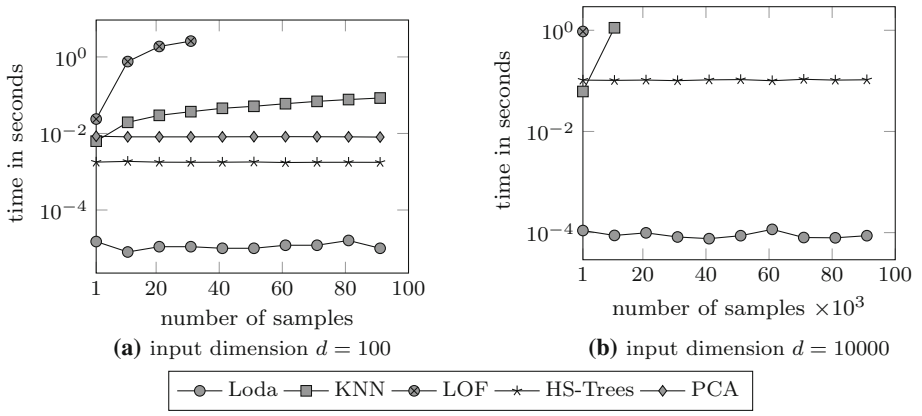


Fig. 7 Average time of classification and update of PCA, KNN, LOF, and Loda detectors against number of observed samples. *Left* and *right* figures show times for the problem of dimension 100 and 10,000 respectively

classify one sample with simultaneous update of the detector is compared for PCA, KNN, LOF, HS-Trees, and Loda with continuous histogram.[5] All settings of individual algorithms were kept the same as in previous experiments. Loda's time includes optimization of hyper-parameters on the first 256 samples of the stream as described in Sect. 4.1.1 (Loda used on average 140 projections with histograms having 16 bins and sliding window of length 256).

The comparison was made on 100,000 artificially generated samples of dimensions 100 and 10,000. As the interest here is on throughput, the details of data generation are skipped as not being relevant for the experiment. The results are summarized in Fig. 7 showing the average time to classify and update one sample against the number of observed samples. The experiment was stopped if the processing of 1000 samples took more than half an hour causing some graphs to be incomplete. The graph for the PCA detector is missing entirely in the left Fig. 7, because processing of 1000 samples took more than 15 h (109s per one sample). Both graphs demonstrate that Loda's efficiency is superior to all other detectors, as it is two order of magnitudes faster than the fastest algorithms from the prior art. This experiment highlights Loda's suitability for efficiently handling data-streams.

4.5 Robustness to missing variables

As mentioned in Sect. 3, Loda with sparse projections can classify and learn from samples with missing variables. This feature was evaluated under the "missing at random" scenario, where missing variables are independent of the class membership. In every repetition of an experiment, before the data was divided into training and testing sets, $\{0, 0.01, 0.02, 0.04, \ldots, 0.20\}$ fraction of all variables in data matrices were made missing (set to `NaN`). Consequently, missing variables were equally present in sets on which Loda were trained and evaluated. The robustness was evaluated on datasets created in the same way as in Sect. 4.1 with clean training sets and 10 % of anomalies in the test sets. Loda's hyper-parameters were determined automatically as described in Sect. 4.1.1.

The effect of missing features on detector's AUC is summarized in Fig. 8 showing the highest missing rate at which Loda had the AUC higher than 99 % of that on the data without

[5] SVM and FRAC were omitted, as their on-line adaptation is not straight-forward. Although the implementation of HS-Trees is in Java, the binary (jar) reports time to process samples, which was used in this comparison.

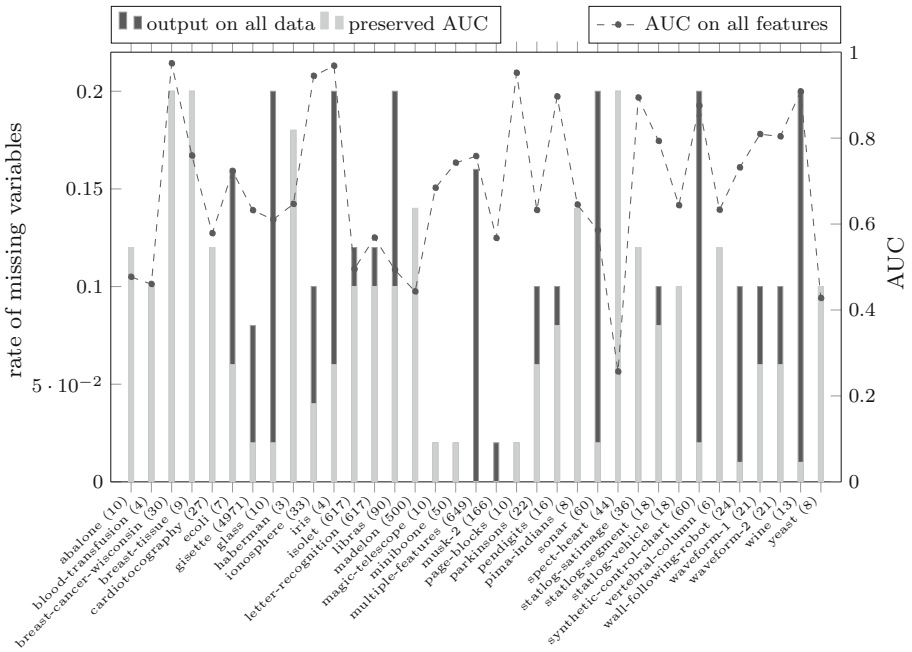


Fig. 8 *Light grey bars* show the maximum rate of missing variables at which Loda had AUC greater than $0.99 \times$ AUC on data without missing variables. *Dark grey bars* show the maximum rate of missing variables at which Loda provided answers for all samples. If *dark grey* are covered by *light grey bars*, then both maximum rates were equal. The *dashed line* shows AUC of Loda on data without missing variables

missing variables (light grey bars). For a large set of problems the tolerance is high and more than 10 % of missing variables are tolerated with negligible impact on the AUC. There are also exceptions, e.g. magic telescope, page-blocks, wine, etc, where the robustness is very small, but we have failed to find a single cause for this fragility. Interestingly, the robustness is independent on the dimension of the problem and the success with which outliers are detected. The exception is datasets on which the detection of outliers is poor (AUC is around 0.5), which is caused by the fact that missing some variables cannot generally improve or decrease already bad AUC. The same figure also shows the rate of missing variables at which Loda stops to provide output (dark grey bars). We can see that this rate is usually much higher than the rate at which Loda retains its AUC.

4.6 Identification of features responsible for an outlier

Explaining the causes of an anomaly detection is relatively new field and there is not yet an established methodology to evaluate and compare the quality of algorithms, hence, the following is adopted here. It starts by calculating the average of Loda's feature scores according to (4) over all anomalous samples in the testing set. Once the scores of individual features are known, they are sorted in decreasing order, which means that features in which anomalous samples deviates the most should be the first. Then, Loda with dense projections utilizing only first d' features is used and its AUC is recorded (d' has varied from 2 to $\min(100, d)$). If the feature relevance score (4) is meaningful, then Loda that uses first couple features (low d') should have the same or better AUC than Loda that uses all features. This is because all

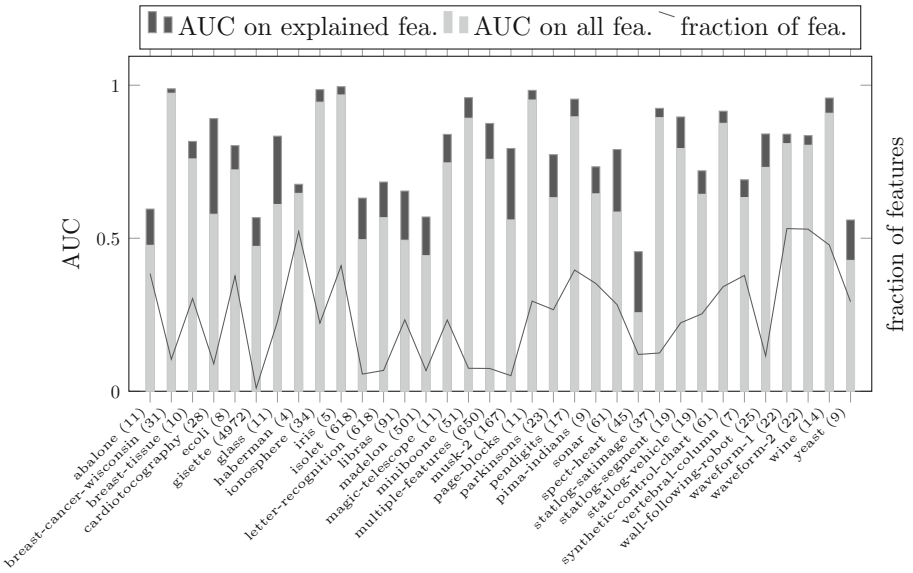


Fig. 9 AUC of Loda with dense projections on features selected according to relevance provided by Loda (captioned "AUC on explained fea.") and AUC of Loda with sparse projections on all features (caption "all"). The *curve* traversing *bars* shows the relative number of features selected according to Loda's relevance

testing problems were originally classification problems which means that anomalous samples were generated by the same probability and hence similar features should be responsible for their anomalousness. Experiments were performed on clean training datasets and all other experimental settings were kept same as in Sect. 4.1.

Figure 9 shows the average AUC of Loda with dense projections using the most important $d_{\min}$ features, where $d_{\min}$ is the least such that AUC by using only first $d_{\min}$ features is higher than 0.99 times the best AUC. On average only 25 % of features are needed to identify outliers with AUC comparable or better than that of Loda with all features. This result confirms that Loda is able to identify features causing the anomaly.

It is interesting that for some problems (e.g. cardiotocography, glass, spect-heart, yeast, etc.) the improvement in the AUC is significant. This can be explained by the curse of dimensionality as anomaly detectors are, by their nature, more sensitive to noise generated by non-informative features.

5 Conclusion

This paper has focused on creating a detector of anomalous samples being able to quickly process enormous amounts of data produced in many contemporary domains. Although the presented detector (Loda) was aimed to process streams of data with a constantly changing behavior, it has been compared to the state of the art in settings where all data are available at once and they can fit into the memory. The rationale behind was to compare Loda under various conditions differing by the rate of anomalies within the data, their clusteredness/scatterness to state of the art detectors, even though they are not applicable for intended scenario. This comparison, which scale is to our knowledge the biggest so far published has

shown that Loda's accuracy measured by area under ROC curve does not lag behind more sophisticated detectors. Moreover, it has revealed conditions under which particular type of detectors with the recommended setting of its parameters excels. Based on it we can conclude that data-centered detectors such as kth-nearest neighbor or Local Outlier Factor with recommended setting for k are good for problems with a few number of scattered anomalies. Contrary, for problems with clustered anomalies and/or with higher rate of anomalies within the data (more than 1 %), the model-centered detectors such as the proposed Loda, 1-SVM, and Isolation Forest are more suitable. Loda was also the only evaluated method capable to process data with 3.2 millions features in a reasonable time without any modification. Last but not least, it was shown that Loda is well suited for intended domains, where data are in the form of non-stationary streams with samples observed just once, classified, and the detector is updated. The comparison to state of the art Half-Space Trees method on seven problems of this type showed that both detectors have nearly the same detection performance, but Loda is on average 7–8 times faster.

Loda has practically demonstrated that an ensemble of anomaly detectors as weak as one-dimensional histograms yields to a strong anomaly detector. Loda's diversification of weak anomaly detectors relies on sparse random projections, which can be viewed as a random sub-space sampling. It is shown that this type of diversification gives the ensemble capability to explain why a scrutinized sample is anomalous and to be robust against missing values. A thorough evaluation showed that detectors employing only features explaining anomalies have equal or better performance than detectors using all features, which shows that features explaining anomalies were selected correctly (on average 25 % of features explained the anomaly). The similar evaluation showed that the ensemble keeps on most problems its detection performance even if more than 10 % of features are missing.

To conclude, this paper has shown that there is no single detector excelling on all type of problems, as different detectors are suitable for different types of problems. The biggest advantage of the proposed Loda with respect to others is its overall good performance and very good speed to detection performance ratio, as it is order of magnitude faster than prior art on big problems while its detection performance is comparable to them. Besides, it can explain causes of anomalies and it is robust to missing variables in data.

Appendix: On-line histogram

The on-line histogram from Ben-Haim and Tom-Tov (2010) approximates the distribution of data by using a set of pairs $\mathbf{H} = \{(z_1, m_1), \ldots, (z_b, m_b)\}$, where $z_i \in \mathbb{R}$ and $m_i \in \mathbb{N}$, where b is an upper bound on the number of histogram bins. It is assumed that every point z_i is surrounded by m_i points, of which half is to the left and half is to the right to z_i. Consequently, the number of points in the interval $[z_i, z_{i+1}]$ is equal to $\frac{m_i+m_{i+1}}{2}$, and the probability of point $z \in (z_i, z_{i+1})$ is estimated as a weighted average.

The construction of the set $\mathbf{H}$ is described in Algorithm 3. It starts with $\mathbf{H} = \{\}$ being an empty set. Upon receiving a sample, $z = x^{\mathrm{T}}w$, it looks if there is a pair (z_i, m_i) in $\mathbf{H}$ such that z is equal to z_i. If so, the corresponding count m_i is increased by one. If not, a new pair $(z, 1)$ is added to $\mathbf{H}$. If the size of $\mathbf{H}$ exceeds the maximal number of bins b, the algorithm finds the two closest pairs (z_i, m_i), (z_{i+1}, m_{i+1}), and replaces them with an interpolated pair $\left(\frac{z_i m_i + z_{i+1} m_{i+1}}{m_i + m_{i+1}}, m_i + m_{i+1}\right)$. Keeping z_i sorted makes all the above operations efficient.

The estimation of the probability density in point $z = x^{\mathrm{T}}w$ is described in Algorithm 4. Assuming the pairs in $\mathbf{H}$ are sorted according to z_i, the algorithm starts by finding i such

Mach Learn (2016) 102:275–304

Algorithm 3: Algorithm constructing approximation of the probability distribution of the data $\{x_i \in \mathbb{R}^d\}_{i=1}^n$ projected on the vector $w \in \mathbb{R}^d$.

Input: sample $x \in \mathbb{R}^d$, $w \in \mathbb{R}^d$;
Output: set of pairs $\mathbf{H} = \{(z_1, m_1), \ldots, (z_b, m_b)\}$,
initialize $\mathbf{H} = \{\}$, $z_{\min} = +\infty$, $z_{\max} = -\infty$. ;
for $j \leftarrow 1$ **to** n **do**
  $z = x_j^{\mathrm{T}} w$;
  $z_{\min} = \min\{z_{\min}, z\}$;
  $z_{\max} = \min\{z_{\max}, z\}$;
  if $\exists(z_i == z)$ **then**
    $m_i = m_i + 1$;
    continue
  else
    $\mathbf{H} = \mathbf{H} \cup \{z, 1\}$
  end
  if $|\mathbf{H}| > b$ **then**
    Sort pairs in $\mathbf{H}$ such that $z_1 < z_2 < \ldots < z_{b+1}$;
    Find i minimizing $z_{i+1} - z_i$;
    Replace pairs (z_i, m_i), (z_{i+1}, m_{i+1}), by the pair

$$\left(\frac{z_i m_i + z_{i+1} m_{i+1}}{m_i + m_{i+1}}, m_i + m_{i+1}\right)$$

end
$\mathbf{H} = \mathbf{H} \cup \{(z_{\min}, 0), (z_{\max}, 0)\}$;
Sort pairs in $\mathbf{H}$ such that $z_{\min} < z_1 < z_2 < \ldots < z_{\max}$;

Algorithm 4: Algorithm returning approximate of probability density in point x projected on the vector w.

Input: sample $x \in \mathbb{R}^d$, $w \in \mathbb{R}^d$, set of pairs $\mathbf{H} = \{(z_1, m_1), \ldots, (z_b, m_b)\}$;
Output: return estimate of the probability density $p(x)$.;
$\mathbf{H} = \mathbf{H} \cup \{(z_{\min}, 0), (z_{\max}, 0)\}$;
Sort pairs in $\mathbf{H}$ such that $z_{\min} < z_1 < z_2 < \ldots < z_{\max}$;
$z = x^{\mathrm{T}} w$;
if $\exists(i | z_i < z \leq z_{i+1})$ **then**
  return $p(x) = \frac{z_i m_i + z_{i+1} m_{i+1}}{2M(z_{i+1} - z_i)}$;
else
  return invalid;
end

that $z_i < z \leq z_{i+1}$. If such i exists, then the density in z is estimated as $\frac{z_i m_i + z_{i+1} m_{i+1}}{2M(z_{i+1} - z_i)}$, where $M = \sum_{i=1}^b m_i$. Otherwise it is assumed that z is outside the estimated region.

Construction of datasets

Benchmark problems for the evaluation of batch anomaly detectors were constructed following the process in Emmott et al. (2013). It advocates the creation of anomaly detection benchmarks from real data, since artificial problems can be too far from realistic problems. 36 source datasets from which all benchmark problems have been created were downloaded

from Frank and Asuncion (2010) from a category of classification problems with numerical attributes and without missing variables. The list of downloaded datasets together with informations about the number of normal / anomalous samples is in Table 5

If the source dataset was a binary problem, the larger class was used as a normal class and the smaller as the anomaly class. Multi-class datasets were converted to binary ones as follows:

1. Train a random forest classifier to solve the multi-class problem in the original source dataset using all samples.
2. Estimate the confusion matrix $\mathbf{C}$ with $\mathbf{C}_{k,j} = P(j|x_i, k)$, where the conditional probability is the probability returned by random forest that the sample x_i belongs to the class j when k is true.
3. Create a complete graph with vertices being classes from the datasets and edges having weights from confusion matrix as $\mathbf{C}_{k,j} + \mathbf{C}_{j,k}$.
4. Compute the maximum weight spanning tree of the graph to identify "most-confusable" pairs of classes.
5. Two-color the maximum spanning tree such that no adjacent vertices has the same color. Each color determines set of classes that make the normal and anomalous class.

Notice that the above construction aims to maximize the difficulty (confusion) between normal and anomalous class.

Anomalous samples were divided into four groups according to the probability that a sample from the anomaly class is assigned to an anomaly class estimated the by the kernel logistic regression with a Gaussian kernel (Zhu and Hastie 2005). Based on this score, all anomalous points were assigned a difficulty category as: *easy* ([0.84, 1), *medium* ([0.7, 0.84), *hard* ([0.5, 0.7), *very hard* (0, 0.5].

A concrete benchmark problem with a given fraction of anomalies (experiments used fractions 0.005, 0.01, 0.05, 0.1) and difficulty (easy, medium, hard, very hard) was created as follows:

1. Randomly divide samples into training a testing set allowing maximum of 10000 samples in each set.
2. If anomalies should be clustered, select randomly a pivot and find sufficient number of nearest points in chosen difficulty category such that the final training and testing sets have the desired number of anomalous samples. If anomalies are not clustered, select anomalous samples randomly.
3. Divide the selected anomaly samples into training a testing set and mix with normal samples.

The final benchmark problem is attached a clusteredness category according to a fraction of sample variance of normal samples to the sample variance of anomalous samples as: *high scatter* (0, 0.25), *medium scatter* [0.25, 0.5), *low scatter* [0.5, 1), *low clusteredness* [1, 2), *medium clusteredness* [2, 4), and *high clusteredness* [4, ∞).

Created benchmark problems have three basic properties (rate of anomalies within data, difficulty of anomalies, and clusteredness) according to which they can be divided.

Experimental results

See Fig. 10.

Table 5 Datasets used to create benchmarking problems

Dataset	Dimension	Number of samples				
		Normal	Easy	Medium	Hard	Very hard
Abalone	10	2153	7	44	955	1018
Blood-transfusion	4	384	7	9	49	84
Breast-cancer-wisconsin	30	357	188	18	5	–
Breast-tissue	9	66	17	5	3	15
Cardiotocography	27	1831	143	86	48	18
Ecoli	7	205	81	27	9	14
Gisette	4971	3500	–	1471	2029	–
Glass	10	114	75	20	3	2
Haberman	3	225	4	11	19	47
Ionosphere	33	225	36	86	3	–
Iris	4	100	44	2	2	2
Isolet	617	4497	40	3260	–	–
Letter-recognition	617	4197	35	3565	–	–
Libras	90	216	115	28	–	–
Madelon	500	1300	–	1300	–	–
Magic-telescope	10	12, 332	2808	1074	1079	1727
Miniboone	50	93,565	17,744	6179	5703	6873
Multiple-features	649	1200	63	737	–	–
Musk-2	166	5581	604	212	105	96
Page-blocks	10	4913	315	69	77	99
Parkinsons	22	147	31	13	4	–
Pendigits	16	5539	5286	99	36	32
Pima-indians	8	500	101	76	45	46
Sonar	60	111	55	42	–	–
Spect-heart	44	212	7	46	2	–
Statlog-satimage	36	3594	2520	111	84	126
Statlog-segment	18	1320	866	73	30	21
Statlog-shuttle	8	57,769	10	19	26	176
Statlog-vehicle	18	629	46	86	65	20
Synthetic-control-chart	60	400	197	3	–	–
Vertebral-column	6	410	–	–	68	142
Wall-following-robot	24	2923	1841	380	170	142
Waveform-1	21	3304	1204	279	147	66
Waveform-2	21	3304	1203	269	150	74
Wine	13	107	65	6	–	–
Yeast	8	752	177	214	211	130

The number of samples is after the dataset has been converted to a two-class problem as described in Appendix "Construction of datasets". Columns captioned easy, medium, hard, very-hard shows the number of anomalous samples with a given level of difficulty. Column captioned normal shows the number of normal samples

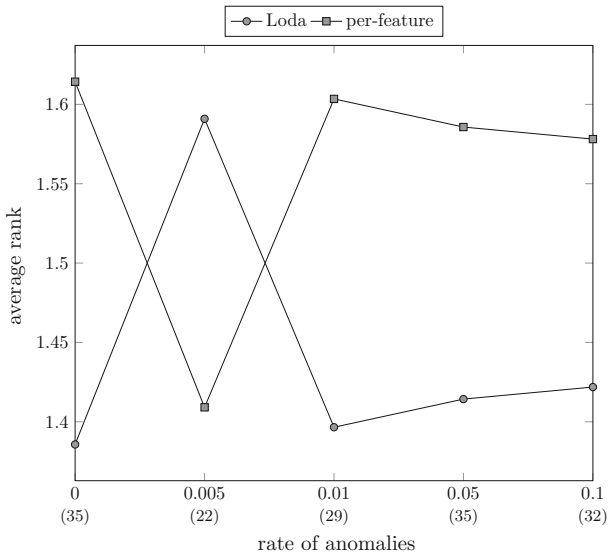


Fig. 10 The figure shows average rank of Loda and detector based on an ensemble of histograms modeling individual features on different rates of anomalies within data. The *small number in parentheses* shows the number of datasets used to create benchmarking problems. Notice that only 22 datasets had enough normal samples such that the problems with rate of anomalies 0.005 can be created. Wilcoxon signed rank test assessing that both detectors delivers the same performance on each rate of anomalies separately accepted hypothesis with p values 0.053, 0.44, 0.11, 0.16, and 0.31 in order of increasing rate of anomalies. The same test on results from all experiments rejected the hypothesis with a p value 0.028, which means that Loda's performance is statistically better

References

Aggarwal, C. C. (2013a). *Outlier analysis*. New York: Springer.

Aggarwal, C. C. (2013b). Outlier ensembles: Position paper. *ACM SIGKDD Explorations Newsletter*, *14*(2), 49–58.

Akhilomen, J. (2013). Data mining application for cyber credit-card fraud detection system. In P. Perner (Ed.), *Advances in data mining. Applications and theoretical aspects, LNCS* (Vol. 7987, pp. 218–228). Berlin: Springer.

Andoni, A., & Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th annual IEEE symposium on foundations of computer science* (pp 459–468)

Bay, S. D., & Schwabacher, M. (2003). Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*, ACM, New York, NY, USA, KDD '03, pp 29–38. doi:10.1145/956750.956758

Ben-Haim, Y., & Tom-Tov, E. (2010). A streaming parallel decision tree algorithm. *The Journal of Machine Learning Research*, *11*, 849–872.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communication of ACM*, *18*(9), 509–517.

Birgé, L., & Rozenholc, Y. (2006). How many bins should be put in a regular histogram. *ESAIM: Probability and Statistics*, *10*, 24–45.

Breunig, M. M., Kriegel, H., Ng, R., & Sander, J. (2000). Lof: Identifying density-based local outliers. *SIGMOD Record*, *29*(2), 93–104.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*(3), 1–58.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, *55*(1), 58–75.
Dang, X., Micenková, B., Assent, I., & Ng, R. (2013). Local outlier detection with interpretation. In *Machine learning and knowledge discovery in databases* LNCS, Vol. 8190, pp. 304–320, Springer.
Datar, M., Gionis, A., Indyk, P., & Motwani, R. (2002). Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, *31*(6), 1794–1813.
de Vries, T., Chawla, S., & Houle, M. E. (2010). Finding local anomalies in very high dimensional space. In *10th international conference on data mining*, IEEE, pp. 128–137.
Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.
Donjerkovic, D., Ioannidis, Y.E., & Ramakrishnan, R. (2000). Dynamic histograms: Capturing evolving data sets. In *Proceedings of the international conference on data engineering*, IEEE Computer Society Press, 1998, pp. 86–86.
Emmott, A. F., Das, S., Dietterich, T., Fern, A., & Wong, W. K. (2013). Systematic construction of anomaly detection benchmarks from real data. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, ACM, ODD '13, pp. 16–21.
Frank, A., & Asuncion, A. (2010). UCI machine learning repository. http://archive.ics.uci.edu/ml
Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. *ICML*, *96*, 148–156.
Gama, J., & Pinto, C. (2006). Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on applied computing*, pp. 662–667.
Gao, J., & Tan, P. N. (2006). Converting output scores from outlier detection algorithms into probability estimates. In *Sixth international conference on data mining*, IEEE, pp. 212–221.
Guha, S., Koudas, N., & Shim, K. (2006). Approximation and streaming algorithms for histogram construction problems. *ACM Transactions on Database Systems, 31*(1), 396–438.
Harmeling, S., Dornhege, G., Tax, D., Meinecke, F., & Muller, K. (2006). From outliers to prototypes: Ordering data. *Neurocomputing*, *69*(13–15), 1608–1618.
Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(8), 832–844.
Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, *26*(189–206), 1.
Jonathan, W., Aryal, S., & Zhou, G. T. (2014). Implementation of isolation forest. http://sourceforge.net/projects/mass-estimation/
Keller, F., Muller, E., & Bohm, K. (2012). HiCS: High contrast subspaces for density-based outlier ranking. In *Proceedings of international conference on data engineering (ICDE)*, IEEE, pp. 1037–1048.
Kivinen, J., Smola, A. J., & Williamson, R. (2004). Online learning with kernels. *IEEE Transactions on Signal Processing*, *52*(8), 2165–2176.
Knorr, E., & Ng, R. T. (1999). Finding intensional knowledge of distance-based outliers. In *Proceedings of the international conference on very large data bases*, pp. 211–222.
Kriegel, H. P., & Zimek, A. (2008). Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th international conference on Knowledge discovery and data mining*, ACM, pp. 444–452.
Kuncheva, L. (2004). *Combining pattern classifiers: Methods and algorithms*. New York: John Wiley & Sons.
Lazarevic, A., & Kumar, V. (2005). Feature bagging for outlier detection. In *Proceedings of the 11th international conference on Knowledge discovery and data mining*, Vol. 21, pp. 157–166.
Li, P. (2007). Very sparse stable random projections for dimension reduction in l $\alpha(0<\alpha<2)$) norm. In *Proceedings of the 13th international conference on Knowledge discovery and data mining*, p. 440.
Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In *Eighth IEEE international conference on data mining*, IEEE, pp. 413–422.
Ma, J., Saul, L. K., Savage, S., & Voelker, G. M. (2009). Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, ACM, pp. 681–688.
Muller, E., Schiffer, M., & Seidl, T. (2011). Statistical selection of relevant subspace projections for outlier ranking. In *27th international conference on data engineering (ICDE)*, IEEE, pp. 434–445.
Nguyen, H. V., Ang, H. H., & Gopalkrishnan, V. (2010). Mining outliers with ensemble of heterogeneous detectors on random subspaces. In H. Kitagawa, Y. Ishikawa, Q. Li, & C. Watanabe (Eds.), *Database systems for advanced applications. Lecture notes in computer science* (Vol. 5981, pp. 368–383). Berlin, Heidelberg: Springer.
Noto, K., Brodley, C., & Slonim, D. (2012). FRaC: A feature-modeling approach for semi-supervised and unsupervised anomaly detection. *Data Mining and Knowledge Discovery*, *25*, 109–133.
Pevný, T., Rehák, M., & Grill, M. (2012). Detecting anomalous network hosts by means of pca. In *IEEE international workshop on information forensics and security*, pp. 103–108.

Pham, N., & Pagh, R. (2012). A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. In *Proceedings of the 18th international conference on knowledge discovery and data mining*, ACM, pp. 877–885.

Pokrajac, D., Lazarevic, A., & Latecki, J. L. (2007). Incremental local outlier detection for data streams. In *2007 IEEE symposium on computational intelligence and data mining*, IEEE, pp. 504–515.

Poosala, V., Haas, P. J., Ioannidis, Y. E., & Shekita, E. J. (1996). Improved histograms for selectivity estimation of range predicates. *SIGMOD Records*, *25*(2), 294–305.

Rondina, J., Hahn, T., de Oliveira, L., Marquand, A., Dresler, T., Leitner, T., et al. (2014). SCoRS—A method based on stability for feature selection and mapping in neuroimaging. *IEEE Transactions on Medical Imaging*, *33*(1), 85–98.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural Computation*, *13*(7), 1443–1471.

Schubert, E., Wojdanowski, R., Zimek, A., & Kriegel, H. P. (2012). On evaluation of outlier rankings and outlier scores. In *Proceedings of the 12th SIAM international conference on data mining*, pp. 1047–1058.

Shyu, M. l., Chen, S. C., Sarinnapakorn, K., & Chang, L. (2003). A novel anomaly detection scheme based on principal component classifier. In *Proceedings of the IEEE foundations and new directions of data mining workshop*, pp. 172–179.

Somol, P., Grim, J., & Pudil, P. (2011). Fast dependency-aware feature selection in very-high-dimensional pattern recognition. In *International conference on systems, man, and cybernetics*, IEEE, pp. 502–509.

Somol, P., Grim, J., Filip, J., & Pudil, P. (2013). On stopping rules in dependency-aware feature ranking. In *Progress in pattern recognition, image analysis, computer vision, and applications*, Springer, pp. 286–293.

Šourek, G., Kuzelka, O., & Železnỳ, F. (2013). Predicting top-k trends on twitter using graphlets and time features. ILP 2013 Late Breaking Papers, pp. 52–57.

Spinosa, E. J., Carvalho, L. F., & Gama, J. (2009). Novelty detection with application to data streams. *Intelligent Data Analysis*, *13*(3), 405–422.

Sricharan, K., & Hero, A. O. (2011). Efficient anomaly detection using bipartite k-nn graphs. *Advances in Neural Information Processing Systems*, *24*, 478–486.

Tan, S. C. (2014). *Implementation of half-space trees*. https://sites.google.com/site/analyticsofthings/recentwork-fast-anomaly-detection-for-streaming-data

Tan, S. C., Ting, K. M., & Liu, T. F. (2011). Fast anomaly detection for streaming data. In *Proceedings of the twenty-second international joint conference on artificial intelligence-volume volume two*, AAAI Press, pp. 1511–1516.

Tax, D. M. J., & Duin, R. P. W. (2004). Support vector data description. *Machine Learning*, *54*(1), 45–66.

Tax, D. M. J., & Laskov, P. (2003). Online svm learning: From classification to data description and back. In *13th workshop on neural networks for signal processing*, IEEE, pp. 499–508.

Yeung, D. Y., & Chow, C. (2002). Parzen-window network intrusion detectors. In *Proceedings of 16th international conference on pattern recognition*, Vol. 4, pp. 385–388.

Zhang, H. (2004). The optimality of naive bayes. In *Proceedings of the seventeenth international Florida artificial intelligence research society conference*.

Zhu, J., & Hastie, T. (2005). Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics*, *14*(1), 185–205.

Zimek, A., Schubert, E., & Kriegel, H. P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, *5*(5), 363–387.

Appendix 2 - Grill, M. and Pevný, T. and Rehák, M. : *Reducing false positives of network anomaly detection by local adaptive multivariate smoothing* [23]

Journal of Computer and System Sciences 83 (2017) 43–57

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Reducing false positives of network anomaly detection by local adaptive multivariate smoothing

Martin Grill [a,b,*], Tomáš Pevný [a,b], Martin Rehak [a,b]

[a] Czech Technical University in Prague, Faculty of Electrical Engineering, Czech Republic
[b] Cisco Systems, Inc., United States

ARTICLE INFO

Article history:
Received 31 August 2015
Received in revised form 6 March 2016
Accepted 17 March 2016
Available online 4 April 2016

Keywords:
Network anomaly detection
Regression smoothing
False positive rate reduction

ABSTRACT

Network intrusion detection systems based on the anomaly detection paradigm have high false alarm rate making them difficult to use. To address this weakness, we propose to smooth the outputs of anomaly detectors by online Local Adaptive Multivariate Smoothing (LAMS). LAMS can reduce a large portion of false positives introduced by the anomaly detection by replacing the anomaly detector's output on a network event with an aggregate of its output on all similar network events observed previously. The arguments are supported by extensive experimental evaluation involving several anomaly detectors in two domains: NetFlow and proxy logs. Finally, we show how the proposed solution can be efficiently implemented to process large streams of non-stationary data.

1. Introduction

Increasing number and sophistication of attacks against critical enterprise computing infrastructure drives the need to deploy increasingly more sophisticated defense solutions. An essential component of the defense is Intrusion Detection System (IDS) [1] analyzing network traffic that crosses the defense perimeter and looking for evidence of ongoing malicious activities (network attacks). When such activity is detected, an alarm is raised and then analyzed by network administrator who determines the existence and scope of the damage, repairs it and improves the defense infrastructure against future attacks.

IDS can be categorized according to different criteria: by the location (on a host, a wired network, or a wireless network), detection methodology (signature matching, anomaly detection, or stateful protocol analysis), or capability (simple detection or active attack prevention) [1]. In context of our work, the most important criterion is the categorization by detection methodology, which is described in more detail below.

Signature matching techniques identify attacks by matching packet contents against specific attack signatures. The signatures are created using already identified and well-described attack samples, which are time consuming and can take from couple of hours up to several days, which gives attackers plenty of time for their criminal activities. The biggest weakness of this solution is that it detects only known attacks, which can be due to smart evasion techniques used by malware limiting. With the growing proportion of encrypted traffic, use of self-modifying malware and other evasion techniques, the use of detection techniques tailored to catch predefined known set of attacks is becoming increasingly irrelevant.

* Corresponding author at: Czech Technical University in Prague, Faculty of Electrical Engineering, Zikova 1903/4, 166 36 Prague 6, Czech Republic.
E-mail addresses: magrill@cisco.com (M. Grill), tpevny@cisco.com (T. Pevný), marrehak@cisco.com (M. Rehak).

http://dx.doi.org/10.1016/j.jcss.2016.03.007
0022-0000/

Anomaly-based detection tries to decrease the human work (e.g. manual creation of signatures) by building statistical model of a normal behavior and detect all deviations from it. This enables to detect new, previously unknown attacks provided that their statistical behavior is different from that of the normal traffic. While anomaly-based methods are attractive conceptually, they have not been widely adopted. This is because they typically suffer from high false alarm rate (not every anomaly is related to the attack) rendering them useless in practice, since network operator can analyze only few incidents per day [2,3]. Decreasing the false positive rate is therefore important to make anomaly-based IDS competitive with signature matching based solutions.

Rehak [4] divides false positives of anomaly-based IDS into two classes: *unstructured* false positives are essentially a random noise caused by the stochasticity of the network traffic and *structured* false positives caused by a persistent but a very different behavior of a small number of network hosts, for example mail or DNS servers (the precise definition is left to Section 2).

This work proposes a method designed to decrease the rate of unstructured false positives by smoothing anomaly values with respect to time. This causes similar anomalies[1] occurring at different times to receive similar anomaly score, even though one would be otherwise flagged as anomaly. The rate of structured false positives is either decreased or remains the same, which depends on circumstances discussed in detail in Section 3. The method is evaluated with two different anomaly detection based IDSs, in both cases improving their accuracy.

The contribution of the paper is threefold.

- First, it mathematically formulates structured and unstructured false positives and argues why unstructured false positives are more difficult to white-list or remove.
- Second, it proposes a theoretically sound method to decrease the rate of unstructured false positives.
- Third, it shows how the method can be implemented to efficiently process data-streams.

This paper is organized as follows. The next section first properly defines classes of false positives, describe the proposed solution and mathematically proofs its correctness under mild assumptions. The same section also discusses the modification to efficiently process data-streams. Section 4 shows, how the method was deployed in two Intrusion detection systems. The related work is discussed in Section 5 while Section 6 concludes the paper.

2. Classification of false positives

Hereafter it is assumed that the network anomaly detection system (AD) observes a stream of network events (e.g., NetFlow [5], HTTP connections, etc.) produced by a set of hosts within the network. Anomaly detection system maintains internal model(s) to assign a score in range $[0, 1]$ to each observed event with zero indicating the normal event and one indicating possible attack, as it is assumed that malicious activities have statistical characteristics different from the normal ones [6–8] making them rare. As mentioned in the introduction the anomaly detection systems produce false alarms since an overwhelming majority of rare events are not caused by any attack. Rehak [4] divides these false positives into following two classes:

- *Unstructured false positives* are short-term events distributed uniformly over all the network hosts proportionally to the traffic volume. They are typically triggered by widespread, uniformly distributed behaviors (such as web browsing) and we model them as white noise (zero mean and finite variance) added to the anomaly detector's output. Therefore, the observed anomaly score y_i of an event x_i is equal to
$$y_i = g(x_i) + \eta_i, \tag{1}$$
where $g(x_i)$ is the true anomaly score on event x_i and η_i is the additive white noise. The η_i therefore hides the true value $g(x_i)$.
- *Structured false positives* are caused by a (long-term) legitimate behaviors of a small number of network hosts. These behaviors are different from the background, and because they are found only at a very small portion of network hosts, they are reported as anomalies. Examples are rare applications performing software update, regular calls of unusual network APIs, etc. Since this type of false positive is typically limited to a small number of network hosts and its behavior is very regular, it can be quickly identified and filtered out using white-lists. Nevertheless, these white-lists are specific for a given network and are difficult to create before deployment. We define the structured false positives to be generated by a mixture of distributions
$$x_{sfp} \sim \frac{\sum_{j=1}^{m} \beta_j \epsilon_j(x_i)}{\sum_{j=1}^{m} \beta_j}, \tag{2}$$
where ϵ_j represent structured false positive with weight β_j. Each component ϵ_j has small variance comparing to that of the unstructured false positives, but means of the components are typically far from each other.

[1] Similarity can be an arbitrary function $k : \mathcal{X} \times \mathcal{X} \mapsto [0, 1]$.

Appendix 2 - Grill, M. and Pevný, T. and Rehák, M. : *Reducing false positives of network anomaly detection by local adaptive multivariate smoothing* [23]

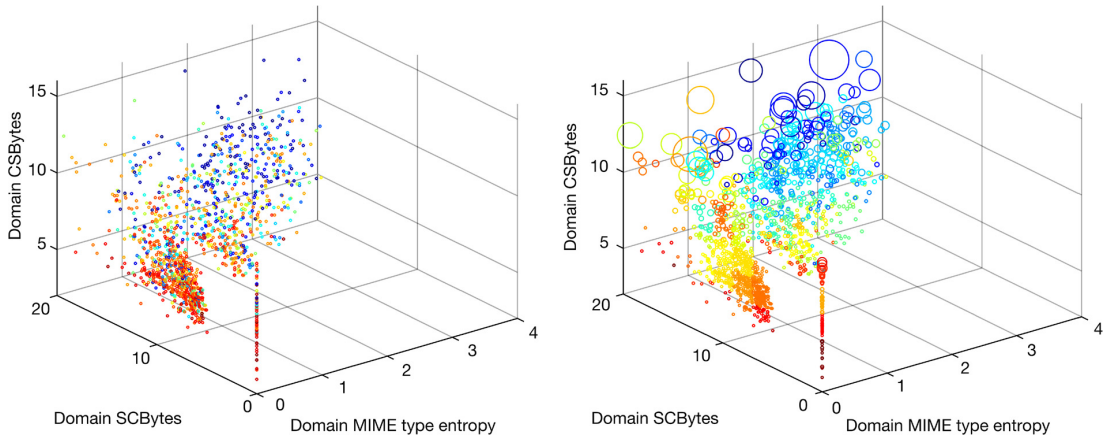


Fig. 1. Visualization of score provided by HTTP anomaly detection engine (described in Section 4.2) without (left) and with (right) LAMS models in LAMS 3 context space (see Table 4 for details). Each point represents one HTTP request observed in the network, where its position is defined by the context features (defined in Table 4) and the color denotes the anomaly score, red being the most anomalous and blue the less. The larger the circle the more events share the same context position and anomaly score. The anomaly scores of the anomaly detection engine are shown in the left figure. As can be seen there are regions of the context space where most of the points have high anomaly scores and regions of low anomaly scores with additional noise. The right figure shows the same points but the anomaly score is smoothed by the LAMS model effectively reducing the unstructured false positives via smoothing. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. Proposed method

The idea behind the proposed local adaptive multivariate smoothing (LAMS) method is to replace the output of the anomaly detector by average anomaly score of similar events observed in the past, where the similarity between two events is defined as $K_h : \mathcal{X} \times \mathcal{X} \mapsto [0, 1]$ (further also called *context*). This effectively smooths the output of the anomaly detector and therefore reduces unstructured false positives. This smoothing can be mathematically formulated as

$$\hat{g}_{\mathrm{nw}}(x) = \frac{\sum_{i=1}^{n} K_h(x, x_i) y_i}{\sum_{i=1}^{n} K_h(x, x_i)}, \tag{3}$$

where $\{x_i\}_{i=1}^{n}$ is the set of observed network events, $\hat{g}_{\mathrm{nw}}(x)$ is the expected anomaly of event x, and $\{y_i\}_{i=1}^{n}$ is the set of corresponding AD outputs. The space $\mathcal{X}$ on which it is defined can be arbitrary (e.g., space of all strings, graphs, etc.), but Gaussian kernel $K_h(x, x') = \exp\left(-\frac{\|x-x'\|^2}{h}\right)$ on Euclidean space is the most common combination (h parameterizes the width of the kernel). Estimator (3) is known as a Nadaraya–Watson estimator [9,10] which is a non-parametric estimator of the conditional expectation of a random variable.

How does the above smoothing remove false positives according to the division introduced in Section 2?

- *Unstructured false positives* are reduced by the Nadaraya–Watson estimator that performs local averaging of the events. Since the unstructured false positives are of the form $y_i = g(x_i) + \eta_i$, it is proven by Devroye et al. [11] that it converges to the true value $g(x_i)$ under fairly general assumptions. More formally, Devroye has shown

  $$E|\hat{g}_{\mathrm{nw}}(x) - g(x)| \leq \frac{c}{\sqrt{nh}}, \tag{4}$$

  assuming h is chosen in terms of n, with $h \to 0$, $nh \to \infty$ as $n \to \infty$; $c > 0$ is a constant; the kernel K is absolutely continuous and differentiable, with $K' \in L^1$; g is differentiable and the $\mathrm{Var}(y_i)$ is bounded. Since we are using a Gaussian kernel and the anomaly scores y_i are bounded to $[0, 1]$ the estimate converges to the underlying true anomaly score $g(x)$ with a rate of $(nh)^{-1/2}$, when only unstructured false positives are present. However, setting the $h \to 0$ requires infinite computation and memory resources, therefore we restrict the estimator to fixed h.
  The smoothing effect is shown in the Fig. 1 where the left figure shows the input to LAMS and right figure the output. It is apparent that the left figure is noisier, as points with different colors are close to each other.
- *Structured false positives* are long-term events confined to subset of network hosts without direct relationship to the background in the sense that AD's output on them does not change with the time. LAMS can remove them in the following situations.
  If LAMS' similarity measure of alerts K_h is different from that used in the anomaly detector, large number of events with normal AD score can be similar to structured false positives, which decreases the scores output by LAMS on false positives. Example of this type of behavior can be seen on the Fig. 2, where we see that the amplitude of LAMS input is higher than that of the output.

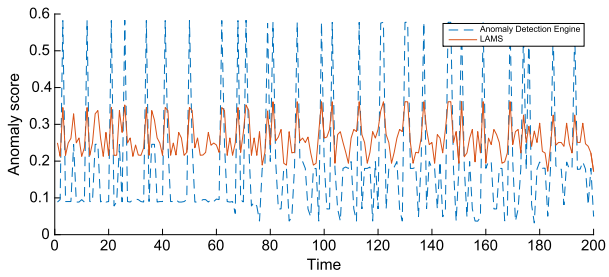


Fig. 2. Anomaly score of the HTTP request to google web API in time. There are sudden spikes in AD's anomaly score (blue dashed line) that are reduced by the LAMS model (red solid line).

If LAMS aggregates outputs of the same anomaly detector deployed on many networks, structured false positives can be normal in other networks. Then, LAMS aggregation eliminates the false positives by aggregating them with these normal activities.

In other situations, structured false positives are confined and there are no similar events receiving lower anomaly score. In this case LAMS fails to remove them, but does not increase their number.

3.1. Complexity considerations

The complexity of the smoothing of AD's output described by Equation (3) is linear with respect to the number of observed events. This, together with the fact that all observed events need to be stored, makes LAMS useless for practical deployment, since number of network events can be as high as millions per second. We therefore resort to common approach to approximate (3) from values maintained in a set of pivots $\Phi = \{\phi_j\}_{j=1}^{J}$, $\phi_j \in \mathcal{X}$ as

$$\hat{g}_{\text{lams}}(x) = \frac{\sum_{j=1}^{J} K_h(x, \phi_j)\hat{y}_j}{\sum_{j=1}^{J} K_h(x, \phi_j)}, \tag{5}$$

where $\{\hat{y}_j\}_{j=1}^{J}$ are estimates of $\hat{g}_{\text{nw}}(\phi_j)$ calculated according to (3). This reduces the computational complexity of the estimate in arbitrary x to $O(J)$, which is linear with the number of pivots and independent of the number of observed alerts.[2] The same holds for space complexity, because only positions ϕ_j of finite number of pivots and relevant estimates $\hat{y}_k$ need to be kept. The set of pivots Φ is updated using the modified Leader–Follower algorithm [13], where new pivot is added to the set Φ when an event not similar to any pivot in Φ is received. The update process is described in more detail in Section 3.2. Contrary to the standard definition of Leader–Follower [13,14] pivots are not allowed to move, because moving pivots towards areas of higher density causes forgetting of rare events, which we want to remember.

3.2. Incremental update of the LAMS model

Keeping LAMS model up to date on data-streams requires maintaining estimates $\{\hat{y}_j\}_{j=1}^{J}$ in $\{\phi_j\}_{j=1}^{J}$ and alternatively adding new pivot if needed. Upon arrival of a new observation of a network event (y_t, x_t), estimates $\hat{y}_j$ stored in pivots ϕ_j should be updated using the Equation (3) as

$$\hat{y}_j^{\text{new}} = \frac{\sum_{i=1}^{n} K_h(\phi_j, x_i)y_i + K_h(\phi_j, x_t)y_t}{\sum_{i=1}^{n} K_h(\phi_j, x_i) + K_h(\phi_j, x_t)}. \tag{6}$$

If we denote

$$w_j^{\text{old}} = \sum_{i=1}^{n} K_h(\phi_j, x_i), \tag{7}$$

$$\hat{y}_j^{\text{old}} = \frac{\sum_{i=1}^{n} K_h(\phi_j, x_i)y_i}{\sum_{i=1}^{n} K_h(\phi_j, x_i)}, \tag{8}$$

the Equation (6) can be rewritten as

$$\hat{y}_j^{\text{new}} = \frac{w_j^{\text{old}}\hat{y}_j^{\text{old}} + K_h(x_t, \phi_j)y_t}{w_j^{\text{old}} + K_h(x_t, \phi_j)y_t}. \tag{9}$$

[2] This statement holds only partially, since with increasing number of observed samples the number of pivots will increase as well [12].

Appendix 2 - Grill, M. and Pevný, T. and Rehák, M. : *Reducing false positives of network anomaly detection by local adaptive multivariate smoothing* [23]

Therefore, for efficient incremental update of the model only $\hat{y}_j$ and w_j need to be stored in each ϕ_j. The update can than be done using the following.

$$w_j^{\text{new}} = w_j^{\text{old}} + K_h(x_t, \phi_j) y_t, \tag{10}$$

$$\hat{y}_j^{\text{new}} = \frac{1}{w_j^{\text{new}}} \left[w_j^{\text{old}} \hat{y}_j^{\text{old}} + K_h(x_t, \phi_j) y_t \right], \tag{11}$$

The above can be further simplified to update only pivots close to x_t, which we define as pivots with similarity $K_h(x_t, \phi_j)$ greater than $K_h^{\min}$. This reduces the complexity of the update process, because only limited number of nearest pivots needs to be updated with each new event.

The update is outlined in Algorithm 1. First, a set of all pivots in $\varepsilon(x_i)$ in the $K_h(x_t, \phi_j)$ vicinity of a new event (y_t, x_t) is found and their estimates $\hat{y}$ are updated as defined in Equation (11). The γ parameter controls the distance (threshold on the similarity) upon which a new pivot is created with $\phi_{J+1} = x_t$, $\hat{y}_{J+1} = y_t$ and $w_{J+1} = 1$.

Data: Stream of events $(y_i, x_i), i = 1 \dots$
Result: Set of pivots $\Phi = \{\phi_j = (x_j, \hat{y}_j)\}_{j=1}^{J}$
Start with an empty set $\Phi = \varnothing$;
while *there is a new event* (y_i, x_i) **do**
  $\varepsilon(x_i) = \{\phi_k \in \Phi \mid K_h(x_i, \phi_k) > K_h^{\min}\}$;
  for $\phi_k \in \varepsilon(x_i)$ **do**
    Update $\hat{y}_k$ relevant to pivot ϕ_k using Equation (10) and (11);
  end
  Find the most similar pivot ϕ_N to the x_i:
  $\phi_N := \arg\max_{\phi \in \Phi} K_h(x_i, \phi)$;
  if $K_h(x_i, \phi_N) < \gamma$ **then**
    Create new pivot $\phi_{J+1} := x_i$;
    $w_{J+1} := 1$;
    $\hat{y}_{J+1} := y_i$;
    $\Phi = \Phi \cup \{\phi_{J+1}\}$;
  end
end

Algorithm 1: LAMS model update algorithm.

The algorithm is controlled by several parameters allowing to trade high locality (precision) for generalization and efficiency. The most important parameter is γ, which influences the total number of pivots in LAMS and hence the cost of the algorithm. When this parameter is set to $\max_{x,x' \in \mathcal{X}} K_h(x, x')$, the number of pivots is identical to the number of unique events. Departing from this extreme case the number of pivots decreases with decreasing its value with a possible impact (both positive or negative) on the accuracy of estimates if local variations are strong. An example of its effect is shown in Fig. 3. The minimal update weight K_h^{min} is not overly important and it is there more for computational speed-up, as sample's neighborhood are more controlled by the similarity function $K_h(x, x')$.

3.3. Querying the LAMS model

The query of the model follows the Equation (5) with the difference that the estimate is computed only over similar pivots, controlled by the same parameter γ as used in the update. Formally the estimate $\hat{y}_t$ is calculated as

$$\hat{y}_t = \frac{\sum_{\phi_j \in \varepsilon(x_t)} K_h(x_t, \phi_j) y_j}{\sum_{\phi_j \in \varepsilon(x_t)} K_h(x_t, \phi_j)}, \tag{12}$$

where $\varepsilon(x_t) = \{\phi_j \in \Phi \mid K_h(x_i, \phi_k) > K_h^{\min}\}$. Restricting the estimate to be calculated from the neighborhood $\varepsilon(x_t)$ is mainly to be consistent with the update procedure. In practice, $K_h^{\min}$ is so small that the effect of points outside $\varepsilon(x_t)$ on the points would be negligible.

When LAMS is deployed, it always performs an update before the query, which means that there is always at least one pivot from which the estimate can be calculated.

3.4. LAMS model feature selection

As in other machine learning algorithms the biggest influence on LAMS's accuracy has the selection of similarity measure and features defining it. The number of features that can be extracted from network traffic events can be high and defining the context with respect to all of them would lead to poor generalization — a phenomenon known as curse of dimensionality [15]. The usual approach is to reduce dimension by feature extraction algorithms [16] or by feature selection. In experiment described in the next section we have chosen the latter approach.

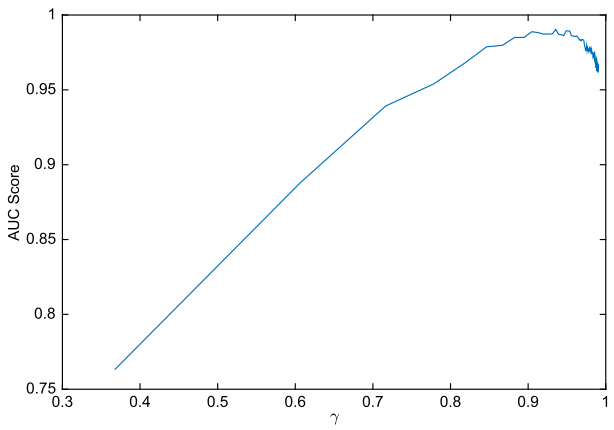


Fig. 3. The graph shows effects of γ parameter on the average AUC score (see Section 4 for definition). Too large γ reduces effect of the smoothing causing smaller efficacy. On the other hand, too small γ decreases model's efficacy, since the model cannot capture local variations in the events.

The features should be selected such that malicious network events are confined in a rather limited part of the space covered by only few pivots. This guarantees that such model will improve the accuracy, since if malicious events fall into a specific region of the feature space the long term anomaly estimate in this region will be consistently high. To select the features in practice, one either has to have examples of network attacks and use feature selection algorithm [17], or use a domain knowledge provided by an expert.

4. Experimental evaluation

We have evaluated the effectiveness of the proposed LAMS model in two different anomaly detection engines: the first, described in Subsection 4.1, uses NetFlow [18] records while the second, described in Section 4.2, uses HTTP proxy logs. Both anomaly detection engines are based on ensemble of simple yet diverse anomaly detectors allowing to efficiently process large data-streams while providing good detection accuracy.

The accuracy is measured by the area under Receiver Operating Characteristic (AUC) [19], which is frequently used when the operating point of the detector is not known.[3] AUC is equal to the probability that the anomaly score of a randomly chosen malicious sample is higher than a randomly chosen legitimate one. The AUC score one means that the detector is always correct, zero means it is always wrong, and 0.5 means that its accuracy is equal to random guessing.

4.1. NetFlow anomaly detection

The NetFlow anomaly detection engine [20,21] processes NetFlow [18] records exported by routers or other network traffic shaping devices. List of all features available in the NetFlow records can be seen in Table A.5 in Appendix A. The anomaly detection engine identifies anomalous traffic using ensemble of anomaly detection algorithms, some of them based on Principal component analysis [22–24], some detects abrupt changes detection [8], and some even uses fixed rules [25]. Furthermore, there are detectors designed to detect specific type of unwanted behavior like network scans [26] or malware with domain generating algorithm [27]. In total the NetFlow anomaly detection engine uses 16 anomaly detectors. The rationale behind the ensemble is that (a) it decreases the computational requirements, since individual detectors and their internal models are simpler, and (b) errors of individual detectors cancels out [28,29] decreasing the false positive rate in effect.

One of the problems to solve with ensemble systems is the combination of their outputs. Arithmetic mean or majority vote are preferred if individual classifiers have roughly the same accuracy [30]. This is not the case in our engine due to detectors specialized to some attacks. For such systems Evangelista [31] suggested to use average of mean and maximum scores of detectors within the ensemble, as it should be more robust to presence poor detectors without knowing which ones are poor. This combination function (further called Evangelista's) favors the highest anomaly scores and reduces the effect of poor detectors.

The combined output of 16 anomaly detectors is used as an input in five different LAMS models with different contexts. The rationale behind using more than one LAMS model is again the ensemble principle, as the hope is that errors of outputs of different LAMS models will be uncorrelated and they will cancel out. Contexts (features) of individual models are shown in Table 1 and they are named according to the publications which have used them in anomaly detection. Since all features

[3] By an operating point it is understood an upper bound on false positives/false negatives or their costs.

Table 1
Features defining similarity in NetFlow LAMS models.

Name	Context features
Xu-source [25]	entropy of source ports for the source IP entropy of destination ports for the source IP number of destination IP addresses for the source IP
Xu-destination [25]	entropy of source ports for the source IP entropy of destination ports for the source IP number of source IP addresses for the source IP
MINDS [8]	number of flows originating from the source IP number of flows originating from the destination IP number of different destination ports for the source IP
Lakhina [22]	number of flows originating from the source IP number of packets transferred from the source IP number of bytes transferred from the source IP
TAPS [26]	number of different destination IPs of the source IP number of unique ports of the source IP entropy of packets size of the source IP

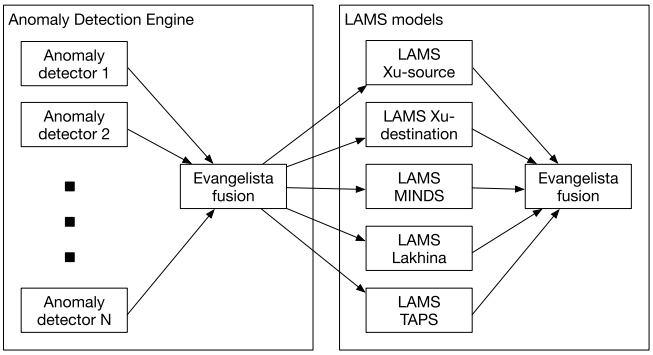


Fig. 4. NetFlow anomaly detection system architecture. In the experimental evaluation (Section 4.1.2) we compare the efficacy of the anomaly detection engine alone (depicted in the left box) with the efficacy when using anomaly detection followed by LAMS models (both the boxes).

are real numbers, the similarity is defined using the Gaussian kernel introduced in Section 3. The schema of the system is shown in Fig. 4. The system consists from two blocks, where the first (left) block represents the anomaly detection part, while the second (right) block represents the part with LAMS models.

4.1.1. Dataset of NetFlow

To evaluate the effects of LAMS model on the NetFlow Anomaly detection engine we have created several datasets from a traffic captured on the network of Czech Technical University (CTU) in Prague. We have used three different approaches to create labels: manual labeling, infecting virtual machines, and attacking our computers within the network by us.

Manual labeling. A week long capture from the university contains 41 517 828 flows between 19 261 different IP addresses. An experienced network operator has identified 10% as anomalous and 11% as legitimate. The most prevalent malicious traffic was ssh scan (55%) followed by p2p traffic (36%), horizontal scan (6.8%), and vertical scan (1.5%).

Malware infection. The second dataset was created by executing real malware inside controlled virtual machine (VM).[4] VMs were infected using malicious binaries captured during 24 hours of their traffic together with the traffic of the whole CTU network. The labels were created such that every NetFlow originating from the infected machines was labeled as malicious, divided into requests and responses according to the direction. NetFlows corresponding to normal traffic of Windows XP operating system running on the VMs, such as connection check, Windows Update checks, NTP checks, etc., were also labeled as normal.

In the first capture we have infected only one VM running Windows XP with Neeris [21] malware that has generated a lot of traffic (malware generated on average three NetFlows per second). Second, we have infected ten different VMs by

[4] The bandwidth of the connection was lowered to 150 kb/s to prevent generating huge amount of traffic.

Table 2
Comparison of AUC Scores without and with LAMS model.

Manual labeling	AUC	
	without LAMS	with LAMS
horizontal scan request	0.82	**0.83**
horizontal scan response	**0.77**	0.66
p2p request	0.87	**0.91**
p2p response	**0.89**	0.75
scan sql request	**1**	**1**
scan sql response	**0.92**	0.91
ssh cracking request	0.80	**0.83**
ssh cracking response	**0.84**	0.71
vertical scan request	**1**	**1**
vertical scan response	**0.98**	0.96
Malware infection		
Neris bot	0.69	**0.87**
FastFlux botnet	0.81	**0.88**
RBot ICMP flood	0.91	**0.94**
Artificial attack		
SSH scan request	0.93	**1**
SSH scan response	**0.84**	0.77
SSH cracking request	**0.83**	**0.83**
SSH cracking response	0.77	**0.78**
anomalous SSH download response	0.95	**0.97**
anomalous SSH download request	**0.95**	**0.95**

FastFlux botnet [21]. Finally, we have infected ten VMs by rbot we could control, and instruct it to perform ICMP flood attack against one of our servers.

Manual attacks. Third dataset was created by network specialist attacking one computer with an open SSH port inside the university network. She has started her attack by horizontal scan of university network searching for a computer with an open SSH port. Then, she performed brute force cracking of an ssh password with a help of dictionary of 1000 passwords with the last one being correct. She has tried all passwords within 5-minute long window. Finally, she has downloaded 0.5 GB from the attacked computer to simulate stealing the data. Corresponding NetFlows in the dataset were labeled on the basis of known attacker and attacked IP addresses and ports and time of the attack information.

4.1.2. Experimental results

Table 2 shows AUC scores of the anomaly detection engine after the first aggregation before using multiple LAMS models (denoted without LAMS), and after the second aggregation (denoted with LAMS) on individual datasets and attacks described previously. The results reveal that LAMS models improve the accuracy in detecting attack requests, but decrease the accuracy on responses. To investigate this, we have plot in Fig. 5 distribution of points corresponding to flows related to the horizontal scan in the context defined by Lakhina's entropy based features. The figure confirms the assumption that requests are located in a very small and distinct part of the space and therefore LAMS models can well estimate the anomaly score, whereas responses are scattered all over the space mixed with other mostly legitimate traffic. According to Table 1 Lakhina's entropy features are calculated from a set of flows with the same source IP address receiving the same entropy and also anomaly values. During horizontal scan, each attacked network host will probably have only one response flow to the attack, which will be well hidden among host's own legitimate traffic. This causes (a) the responses to have different entropies and locations in the space and (b) receiving low anomaly values. Contrary, requests of the attack stand out, since they have the same source IP address and the same values of entropies. Moreover, this entropy would be very different from that of other hosts in the network leading to high anomaly. This explanation is supported by the fact that LAMS models do not decrease anomaly values of response to attacks targeting a single host (e.g., vertical scan). Finally note that it is important to detect at least one part of the attack, either the attack, or the responds.

To demonstrate the decrease in false positive rate on structured false positives, we have used the dataset described in Section 4.1.1 and analyzed the traffic identified as the most anomalous traffic by the first part of the detection system (combined output of the anomaly detectors without LAMS models). Within we have identified a legitimate traffic meeting the criteria of the structured false positives (being explainable and corresponding to rare events). The complete list is in Table 3 and includes responses of NTP servers, software licenses servers, downloads from local data and database servers, etc. The decrease has been again measured by the AUC score, where the false positives were treated as negative samples and all attacks identified in the dataset 4.1.1 as positive samples. AUCs of identified structured false positives are shown in Table 3. According to it LAMS models almost always reduce the false positive rate and increase the accuracy of detection.

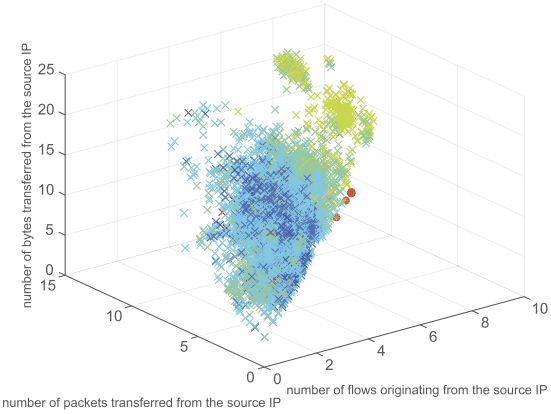


Fig. 5. Visualization of the anomaly scores assigned by the Lakhina LAMS model (see Table 1) to the horizontal scan and corresponding responses contained in the manually labeled dataset described in Section 4.1.1. Each point represents one scan request (cross) or scan response (dot). The color corresponds to obtained anomaly score with red being the most anomalous and blue being the least. Visualization of all the other LAMS models defined in Table 1 can be see in Appendix A in Fig. A.9. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
AUC scores of structured and unstructured false positives in the dataset introduced in Subsection 4.1.1 when LAMS models are used (right column) and not used (middle column). Higher AUC is better.

Structured false positive	AUC	
	without LAMS	with LAMS
NTP server response	0.98	**0.99**
ICQ servers	0.98	**0.99**
Google crawler	0.99	**1**
Local data server downloads	0.9	**0.93**
Local database server requests	**0.98**	**0.98**
Local database server responses	0.96	**0.98**
Local software license server	0.19	**0.89**
Subnetwork gateway requests	0.83	**0.90**
Subnetwork gateway responses	**0.30**	0.24
HTTP proxy requests	0.75	**0.77**
HTTP proxy responses	**0.76**	0.73

Unstructured false positive	AUC	
	without LAMS	with LAMS
DNS request	0.63	**0.70**
DNS response	0.45	**0.71**
Web browsing user 1	0.99	**1**
Web browsing user 2	**0.99**	**0.99**
Web browsing user 3	0.92	**0.94**
Gmail.com servers	0.98	**0.99**

The above investigation of false positives also revealed some unstructured false positives corresponding to excessive web and DNS traffic of some users, which anomaly detectors flagged as suspicious. LAMS models have decreased their anomaly values as is shown in the second part of the Table 3.

4.2. HTTP anomaly detection

Cisco Cognitive Threat Analytics (CTA) [32] is an anomaly-based IDS that uses HTTP proxy logs typically produced by the proxy servers located on the network perimeter and used by the network hosts to access web content. Contrary to NetFlow, these logs contain information extracted from HTTP header (time of the request, source IP address, destination IP address, url, MIME type, downloaded and uploaded bytes, User-Agent identifier, etc.). For the complete list of items see Table A.6 in Appendix A.

Anomaly detection part of the CTA consists from more than 30 various anomaly detectors detecting anomalies according to empirical probability estimates and conditional probability estimates (e.g., P(country), P(domain|host), P(User-Agent|sec-

Table 4
HTTP LAMS models.

Name	Context features
LAMS 1	number of unique referrers of the visited domain number of unique operating systems used to visit domain number of unique MIME types hosted on the domain
LAMS 2	number of unique referrers of the visited domain number of unique HTTP status responses of the visited domain amount of bytes downloaded using the User-Agent
LAMS 3	number of unique User-Agents used to visit the domain entropy of the MIME types hosted on the domain number of unique referrers of the visited domain
LAMS 4	URL length number of unique operating systems used to visit domain number of unique referrers of the visited domain

ond level domain), etc.), time series analysis (models of user activity over time, detection of sudden changes in activity, identification of periodical requests, etc.) and HTTP specific detectors (e.g., discrepancy in HTTP User-Agent field [33]).

The overall architecture of the system is the same as the NetFlow anomaly detection engine shown in Fig. 4. Again, there is a first block of 30 anomaly detectors, whose output is combined by Evangelista's fusion function to obtain one anomaly score value for each HTTP proxy log entry. This output is used as an input to the second block with several LAMS models with context features listed in Table 4 and Gaussian similarity function defined in Section 3. Outputs of four LAMS models are again combined by Evangelista's fusion function to the final output.

4.2.1. Dataset of HTTP proxy logs

The database of HTTP proxy logs was collected from networks of 30 different companies of various sizes and types with collection period ranging from six days to two weeks. There are more than seven billion HTTP logs in the database. A small team of security researchers has identified in them 2 666 users infected with a malware of 825 various families that have generated in total more than 129 millions HTTP logs. HTTP logs corresponding to malware requests represent less than 2% of the total traffic of the individual networks except for few cases, where the networks were infected by ZeroAccess malware [34]. ZeroAccess is very noisy, generating many HTTP logs which has reached up to 21% of the total number of logs. The other malware families worth to mention were: Cycbot, QBot, SpyEye, BitCoinMiner, Zbot and many others. All the malware was identified using several approaches starting with an analysis of the most anomalous HTTP logs as reported by the CTA, malware reported by the individual network administrators, using blacklists and other public feeds or third-party software. The rest of the unlabeled logs were considered to be legitimate traffic. We are aware that within them there might be another proxy logs corresponding to malware traffic, but we are not aware about any additional method providing the ground truth. Moreover since the most anomalous traffic was always investigated for a malware HTTP logs, the most important part corresponding to low false positive rate was always labeled. This labeling of the most anomalous part has been done for the detection engine using only anomaly detection part and using both parts including LAMS models.

4.3. Experimental results

To demonstrate the advantage of LAMS models we have calculated AUC score on sub-datasets with HTTP logs of each malware family considered as positives and all unlabeled HTTP logs as negatives. Since there are 825 different malware families in the dataset the table showing the AUC improvement would be huge. Therefore we have decided to show the effect of the LAMS model in a figure. The Fig. 6 shows a scatter plot where x-coordinate of each point is AUC score of output of the first part of the detection engine using only anomaly detectors and the y-coordinate is AUC score of the output of the second part with LAMS models. This means that if the point is above the diagonal, LAMS models improve the detection accuracy and vice versa. We can observe that LAMS models generally improve the accuracy of detection since majority (75%) of the points is above the diagonal. Particularly noticeable are points within the green rectangle that would not be detectable without LAMS models, but LAMS models significantly increase their anomaly values.

Figs. 7 and 8 show the distribution of anomaly scores of the anomaly detection engine without (left) and with (right) LAMS models. We can see that outputs of an anomaly detection engine without LAMS models on malware HTTP logs are more equally spread across the entire range of values. Contrary, when the anomaly detection engine uses LAMS models, the scores of malware's HTTP logs are located in the most anomalous part of the distribution.

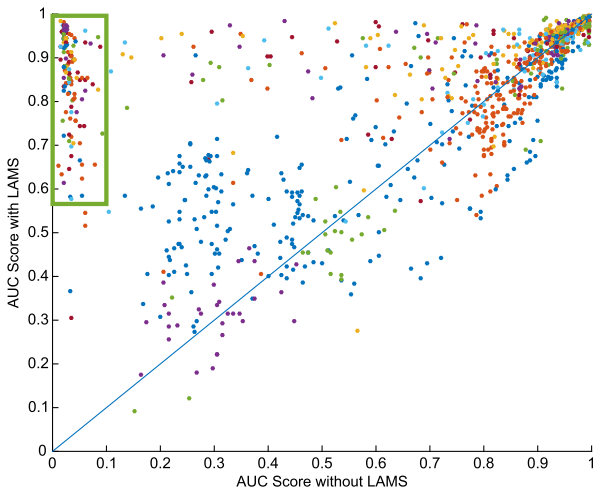


Fig. 6. Scatter plot of the AUC scores when running the anomaly detection engine without and with the LAMS model. Each point represents one malware sample and different colors denote different malware families to illustrate variety of the samples. Points that are above the blue line represent an AUC improvement when LAMS is used, whereas points below represent cases where LAMS decreases the efficacy. As can be seen the LAMS model significantly improves the AUC score of the samples with AUC smaller than 0.2, as depicted by the green bar in the figure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

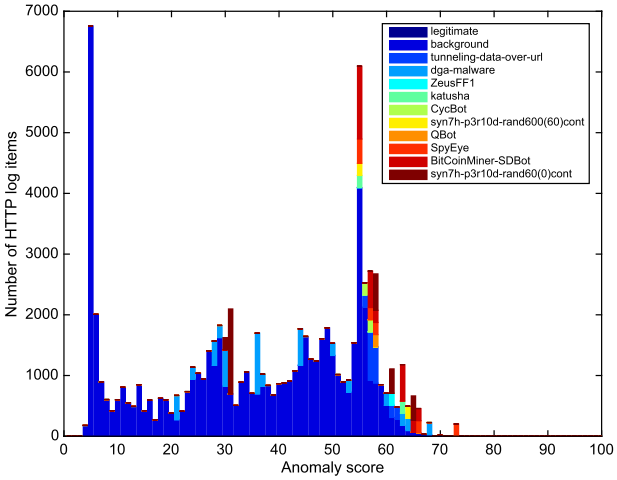


Fig. 7. Distributions of anomaly values of HTTP proxy logs without LAMS models.

5. Related work

The prior art in reducing the false positives used statistics [35], time series [36], and machine learning algorithms [37] to achieve this goal. Most approaches work with a notion of a network alert, which is a collection of networks flows, rather than with individual flows, which is one of the biggest differentiating factor of our work.

Bolzoni et al. [38] propose to correlate anomalies in incoming and outgoing traffic as successful attack should raise alarms in both directions.

Pietraszek [37] has created a classifier that is reducing the workload of the human analyst by classifying the alerts into true positives and false positives. The knowledge of how to classify alerts is learned adaptively by observing the analyst. Similarly, Tian et al. [39] use the analyst feedback to generate custom-made filtering rules, that automatically discard similar false alerts encountered in the future. The need of human interaction represent always a bottleneck since the effectiveness of the IDS depends on the human analyst.

Hooper [40] introduced intelligent network quarantine channels technique to get additional information about the suspected hosts. This information is further used to estimate the probability of a host being infected. Although, this technique

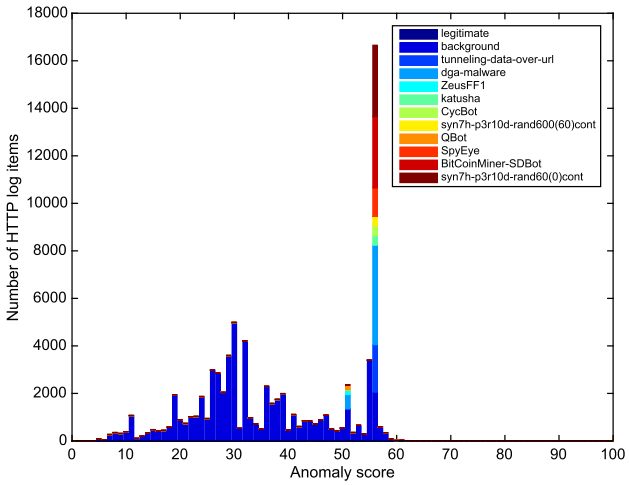


Fig. 8. Distributions of anomaly values of HTTP proxy logs with LAMS models.

provides additional information that can reduce the amount of false alarms, it is not always allowed by the organization's security policy to perform such a host checking.

Viinikka and Debar [36] proposed to use Exponentially Weighted Moving Average (EWMA) control charts of the flow of alerts and try to spot abnormalities. Their experimental results concern the production of less and more qualitative alerts.

The proposed model was inspired by trust models [41–44] which are specialized knowledge structures designed to maintain information about the trustworthiness of a communication partners, either acquired from interactions, observed from the interactions of others, or received from others by means of reputation mechanism [45]. The design of trust models emphasizes features such as fast learning, robustness in response to false reputation information [46], and robustness with respect to environmental noise.

Extended trust models (ETM) introduced by Rehak et al. [47] enhance the original trust models with context representations in order to reflect the context of trusting situation, putting it on par with the identity of trusting parties. The paper [48] took the work further and placed the ETMs in the context of network security problem and use them to aggregate the anomaly scores provided by several anomaly detectors. Each of the network anomaly detectors had its own ETM model with a feature space defined by identity and detector-specific context, serving mainly as a long term memory of past anomaly events. Our current work is an extension of the original ETM model, described in [48]. Unlike its predecessor, it has been simplified by omitting the fuzzy number representation of the anomaly value. Furthermore, we have discarded the identity, focusing only on the context of the observed network events. This simplifications have allowed us to re-formulate the problem more concisely as LAMS, and generalize its application in context of classification of a stream of events.

Pouzos et al. [14] introduced an adaptive kernel smoothing regression which to some extent similar to LAMS models. Contrary to the LAMS model, it updates only the nearest neighbor of the observation and allows the pivots to move towards regions with higher probability density. As discussed in Section 3, this behavior is undesirable, because it will cause LAMS models to forget rare events.

6. Conclusion

We have presented a method reducing false alarm rate of anomaly detection-based intrusion detection systems. The technique smooths detectors' output simultaneously over time and space, which improves the estimate of true anomaly score.

We have proved under mild assumptions that the method reduces unstructured false positives caused by stochasticity of the network traffic. The experiments also showed reduction of the structured FPs, while not having major negative effect in the remaining cases.

The method has been evaluated using large-number of samples from two domains with diverse sets of anomaly detectors. Furthermore the method is a critical component of Cognitive Threat Analytics [32] — an online malware detection security-as-a-service product delivered by Cisco, which analyzes more than 10 billions of requests per day. This illustrates one of the key advantages of our method: simplicity and flexibility. It also shows that the method is usable in real-life production IDS systems.

Acknowledgment

The work of T. Pevný has been partially supported by Czech Science Foundation project 15-08916S.

Appendix A. Examples of NetFlow record and HTTP logs

Table A.5
Example of one NetFlow record containing information about both communication participants (source and destination IP and port), time of the communication, protocol used, bitwise OR of all TCP flags, type of service (tos), number of packets and bytes transferred in both directions.

Feature	Example of values
start-time	1440870672
duration	5
protocol	TCP
source ip	192.168.1.2
destination ip	208.80.154.224
source port	1604
destination port	443
TCP flags	.AP.SF
type of service	0
number of packets	1201
number of bytes	1.8 M

Table A.6
Example of HTTP log. This is one of the HTTP logs created when downloading a wikipedia page. Each page download generates more HTTP logs since all the page resources have to be downloaded. To render the page from the example the browser generated additional 20 HTTP logs, containing page styles, scripts, pictures, etc. In the example the cssc prefixes denote the client to server and server to client communication respectively, so the sc-bytes represent the amount of bytes downloaded by the client and cs-bytes the amount of uploaded bytes to the server. The rest of the features is self-explanatory.

Feature	Value example
x-timestamp-unix	1440870672
sc-http-status	200
sc-bytes	16671
cs-bytes	0
cs-uri-scheme	https
cs-host	en.wikipedia.org
cs-uri-port	1604
cs-uri-path	/wiki/Anomaly_detection
cs-uri-query	
cs-username	Martin Grill
x-elapsed-time	5
s-ip	208.80.154.224
c-ip	192.168.1.2
Content-Type	text/html; charset=UTF-8
cs(Referer)	https://www.google.com/
cs-method	GET
cs(User-Agent)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36

References

[1] K. Scarfone, P. Mell, Guide to intrusion detection and prevention systems (idps), Tech. Rep. 800-94, NIST, US Dept. of Commerce, 2007.
[2] K. Julisch, M. Dacier, Mining intrusion detection alarms for actionable knowledge, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Minin, ACM, 2002, pp. 366–375.
[3] K. Julisch, Clustering intrusion detection alarms to support root cause analysis, ACM Trans. Inf. Syst. Secur. 6 (4) (2003) 443–471.
[4] M. Rehák, M. Grill, Categorisation of false positives: not all network anomalies are born equal, unpublished lecture, 2014.
[5] Cisco Systems, Cisco IOS NetFlow, http://www.cisco.com/go/netflow, 2007.
[6] D.E. Denning, An intrusion–detection model, IEEE Trans. Softw. Eng. 2 (1987) 222–232.
[7] W. Lee, S.J. Stolfo, P.K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, J. Zhang, Real time data mining-based intrusion detection, in: DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01, Proceedings, vol. 1, IEEE, 2001, pp. 89–100.
[8] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, P. Dokas, Minds-minnesota intrusion detection system, in: Next Generation Data Mining, 2004, pp. 199–218.
[9] E.A. Nadaraya, On estimating regression, Theory Probab. Appl. 9 (1) (1964) 141–142.
[10] G.S. Watson, Smooth regression analysis, Sankhya, Ser. A (1964) 359–372.

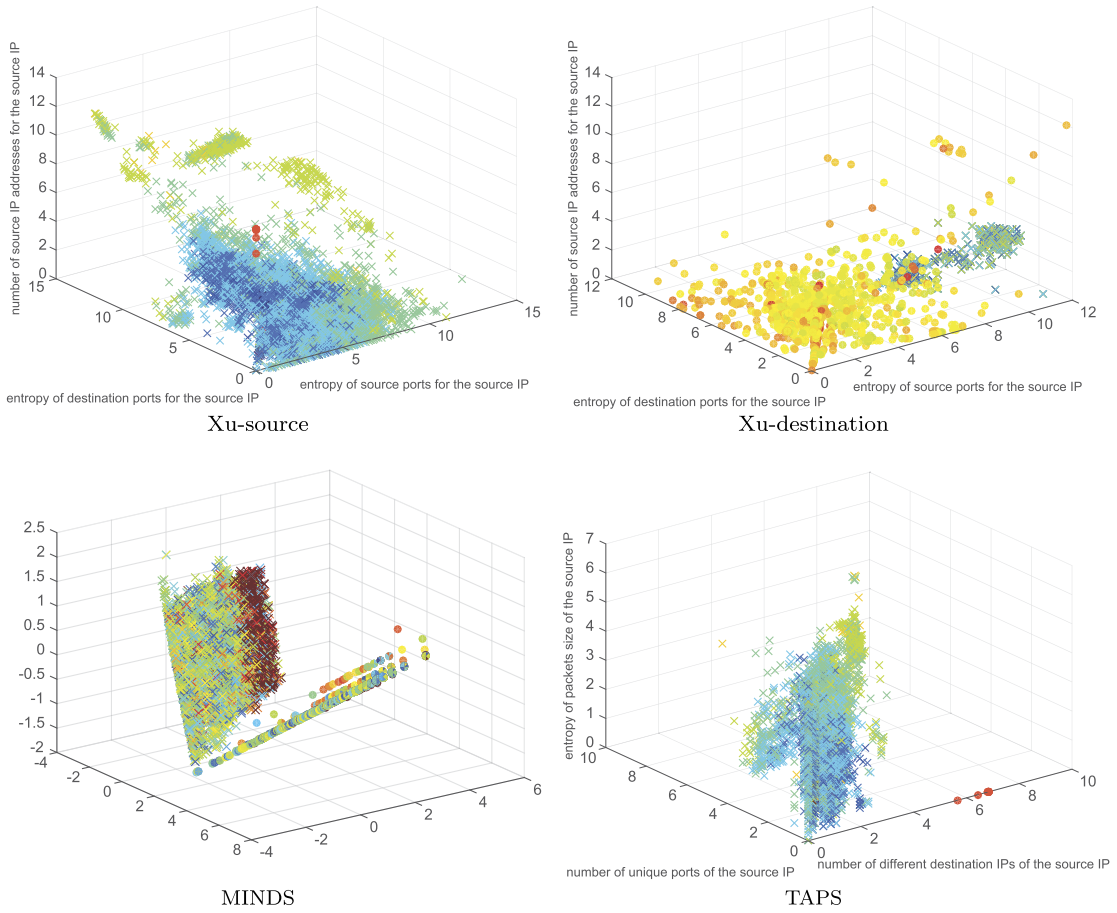


Fig. A.9. Additional visualization to the Fig. 5 of the horizontal scan and corresponding responses contained in the manually labeled dataset described in Section 4.1.1 in the context space of the rest of the LAMS models defined in Table 1. Each point on the individual scatter plots represents one scan request (cross) or scan response (dot). The color corresponds to obtained anomaly score with red being the most anomalous and blue being the least. Since the MINDS model uses four features to define its context we have used the Multidimensional scaling to be able to show the data in three dimensional plot (therefore there are no axis labels). The figures show that similarly to the Lakhina model (Fig. 5), Xu-source and TAPS models have the responses spread across a region of low anomaly score and the requests limited in a small region of high anomaly. For the Xu-destination the requests are spread in the context space but still maintaining high anomaly score. For this particular behavior the MINDS model as the only one increases anomaly score of a part of the responses and reduces the anomaly score of the requests. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

[11] L. Devroye, A. Krzyżak, An equivalence theorem for L1 convergence of the kernel regression estimate, J. Stat. Plan. Inference 23 (1) (1989) 71–82.
[12] M. Rehák, Multiagent trust modeling for open network environments, Doctoral Thesis, Czech Technical University in Prague, 2008.
[13] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, 2nd edition, John Wiley & Sons, New York, 2001.
[14] F. Pouzols, A. Lendasse, Adaptive kernel smoothing regression using vector quantization, in: 2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS), 2011, pp. 85–92.
[15] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning, Springer Ser. Stat., vol. 1, Springer, Berlin, 2001.
[16] I.K. Fodor, A survey of dimension reduction techniques, 2002.
[17] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (2003) 1157–1182.
[18] E.B. Claise, Cisco Systems NetFlow Services Export Version 9, 2004.
[19] T. Fawcett, An introduction to roc analysis, Pattern Recognit. Lett. 27 (8) (2006) 861–874.
[20] M. Rehak, M. Pechoucek, M. Grill, J. Stiborek, K. Bartoš, P. Celeda, Adaptive multiagent system for network traffic monitoring, IEEE Intell. Syst. 3 (2009) 16–25.
[21] S. Garcia, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, Comput. Secur. 45 (2014) 100–123.
[22] A. Lakhina, M. Crovella, C. Diot, Diagnosing network-wide traffic anomalies, in: ACM SIGCOMM Computer Communication Review, vol. 34, ACM, 2004, pp. 219–230.
[23] A. Lakhina, M. Crovella, C. Diot, Mining anomalies using traffic feature distributions, in: ACM SIGCOMM Computer Communication Review, vol. 35, ACM, 2005, pp. 217–228.
[24] T. Pevny, M. Rehak, M. Grill, Detecting anomalous network hosts by means of pca, in: 2012 IEEE International Workshop on Information Forensics and Security (WIFS), 2012, pp. 103–108.

Appendix 2 - Grill, M. and Pevný, T. and Rehák, M. : *Reducing false positives of network anomaly detection by local adaptive multivariate smoothing* [23]

[25] K. Xu, Z.-L. Zhang, S. Bhattacharyya, Profiling internet backbone traffic: behavior models and applications, in: ACM SIGCOMM Computer Communication Review, vol. 35, ACM, 2005, pp. 169–180.
[26] A. Sridharan, T. Ye, S. Bhattacharyya, Connectionless port scan detection on the backbone, in: Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International, IEEE, 2006, 10 pp.
[27] M. Grill, I. Nikolaev, V. Valeros, M. Rehak, Detecting dga malware using netflow, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 1304–1309.
[28] T.G. Dietterich, Ensemble methods in machine learning, in: Multiple Classifier Systems, in: LNCS, vol. 1857, Springer, 2000, pp. 1–15.
[29] R. Polikar, Ensemble based systems in decision making, IEEE Circuits Syst. Mag. 6 (3) (2006) 21–45.
[30] L. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley, 2004.
[31] P.F. Evangelista, M.J. Embrechts, B.K. Szymanski, Data fusion for outlier detection through pseudo-roc curves and rank distributions, in: International Joint Conference on Neural Networks, 2006. IJCNN'06, IEEE, 2006, pp. 2166–2173.
[32] Cisco Systems, CTA cisco cognitive threat analytics on cisco cloud web security, http://www.cisco.com/c/en/us/solutions/enterprise-networks/cognitive-threat-analytics, 2014–2015.
[33] M. Grill, M. Rehak, Malware detection using http user-agent discrepancy identification, in: 2014 IEEE International Workshop on Information Forensics and Security (WIFS), 2014, pp. 221–226.
[34] J. Wyke, The zeroaccess botnet–mining and fraud for massive financial gain, Sophos Technical Paper, https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/Sophos_ZeroAccess_Botnet.pdf, 2012.
[35] G.P. Spathoulas, S.K. Katsikas, Reducing false positives in intrusion detection systems, Comput. Secur. 29 (1) (2010) 35–44.
[36] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, M. Tarvainen, Processing intrusion detection alert aggregates with time series modeling, Inf. Fusion 10 (4) (2009) 312–324, special issue on Information Fusion in Computer Security.
[37] T. Pietraszek, Using adaptive alert classification to reduce false positives in intrusion detection, in: Recent Advances in Intrusion Detection, Springer, 2004, pp. 102–124.
[38] D. Bolzoni, S. Etalle, Aphrodite: an anomaly-based architecture for false positive reduction, preprint, arXiv:cs/0604026.
[39] Z. Tian, W. Zhang, J. Ye, X. Yu, H. Zhang, Reduction of false positives in intrusion detection via adaptive alert classifier, in: International Conference on Information and Automation, 2008. ICIA 2008, IEEE, 2008, pp. 1599–1602.
[40] E. Hooper, An intelligent detection and response strategy to false positives and network attacks, in: Fourth IEEE International Workshop on Information Assurance 2006. IWIA 2006, IEEE, 2006, pp. 20–31.
[41] J. Sabater, C. Sierra, Review on computational trust and reputation models, Artif. Intell. Rev. 24 (1) (2005) 33–60.
[42] J. Sabater, C. Sierra, Reputation and social network analysis in multi-agent systems, in: Proceedings of AAMAS '02, Bologna, Italy, 2002, pp. 475–482.
[43] S. Ramchurn, N. Jennings, C. Sierra, L. Godo, Devising a trust model for multi-agent interactions using confidence and reputation, Appl. Artif. Intell. 18 (9–10) (2004) 833–852.
[44] C. Castelfranchi, R. Falcone, Principles of trust for mas: cognitive anatomy, social importance, and quantification, in: Proceedings of the 3rd International Conference on Multi Agent Systems, IEEE Computer Society, 1998, p. 72.
[45] A. Josang, E. Gray, M. Kinateder, Simplification and analysis of transitive trust networks, Web Intelligence and Agent Systems 4 (2) (2006) 139–162.
[46] T.D. Huynh, N.R. Jennings, N.R. Shadbolt, An integrated trust and reputation model for open multi-agent systems, J. Autonomous Agents and Multi-Agent Systems 13 (2) (2006) 119–154.
[47] M. Rehak, M. Pechoucek, Trust modeling with context representation and generalized identities, in: Cooperative Information Agents XI, in: LNAI/LNCS, vol. 4676, Springer-Verlag, 2007.
[48] M. Rehák, M. Pěchouček, M. Grill, K. Bartos, Trust-based classifier combination for network anomaly detection, in: M. Klusch, M. Pěchouček, A. Polleres (Eds.), Cooperative Information Agents XII, in: Lect. Notes Comput. Sci., vol. 5180, Springer, Berlin, Heidelberg, 2008, pp. 116–130.

Appendix 3 - Ker, A. D. and Pevný, T. : *The Steganographer is the Outlier: Realistic Large-Scale Steganalysis* [30]

The Steganographer is the Outlier: Realistic Large-Scale Steganalysis

Andrew D. Ker, *Member, IEEE*, and Tomáš Pevný

***Abstract*— We present a method for a completely new kind of steganalysis to determine who, out of a large number of actors each transmitting a large number of objects, is hiding payload inside some of them. It has significant challenges, including unknown embedding parameters and natural deviation between innocent cover sources, which are usually avoided in steganalysis tested under laboratory conditions. Our method uses standard steganalysis features, the maximum mean discrepancy measure of distance, and ranks the actors by their degree of deviation from the rest: we show that it works reliably, completely unsupervised, when tested against some of the standard steganography methods available to nonexperts. We also determine good parameters for the detector and show that it creates a two-player game between the guilty actor and the steganalyst.**

***Index Terms*— Data security, information security.**

I. Introduction

STEGANALYSIS aims to detect the presence of hidden payload inside apparently-innocent covers. Although a refined discipline, particularly when the covers are still images, no research has yet considered how to detect payload when monitoring an entire network. In such a case the detector will see vast numbers of objects, transmitted by a variety of users each of whom uses slightly differing sources, the embedding methods used by "guilty" users may be unknown, and the amount of payload almost certainly is unknown. Such a situation is completely different to the "laboratory conditions" found in most steganalysis experiments, and the challenges are different from classifying an individual object as cover or stego. We address them in this paper.

After briefly surveying the state of art in steganalysis of individual objects (Subsection I-A), we explore the requirements of large-scale steganalysis (Section II). We then propose a new steganalysis paradigm (Section III), which differs from conventional steganalysis in two main ways: it takes as its unit the *actor* (network or social network user, or cover object source) rather than the single object, and it performs anomaly detection rather than classification, calibrating its expection of actors by the behaviour of the majority. This means that the method is entirely unsupervised, and robust to the challenges of large-scale steganalysis.

We then perform large-scale experiments, using a real-world social networking image set and steganography accessible to the non-expert (Section IV), to demonstrate that the method works robustly on a number of embedding algorithms (Section V), and to tune some of its parameters (Section VI). The experiments are performed using well-established embedding methods with available implementations, and use a well-established feature set: neither of these represents the academic state-of-art, but we stress that the contribution of this paper is the framework for large-scale steganalysis, rather than a detector using particular features. By testing in well-understood circumstances, we eliminate potential complications from the latest embedding methods and detection features. Finally, we conclude with discussion of many future directions for this line of research (Section VII).

This work extends our two previous conference papers on this subject [17], [18]: as well as new metrics for accuracy, we additionally evaluate a number of different parameters for the large-scale detector, uncover a game between the embedder's strategy and the detector's optimal behaviour, and compare to the (scarce) relevant prior art.

A. State of the Art in Binary Steganalysis

The contemporary approach to steganalysis involves three components. It extracts, from each object under examination, steganalytic *features* of high dimension; it supplies training sets of cover and stego objects; and it runs a machine learning algorithm on the training data. This creates a decision function for novel objects, classifying them as cover or stego, occasionally with some sort of associated level of confidence.

Recent feature sets for the domain of still images, where the literature is most advanced, comprise thousands of relatively weak features [7], [8], [19]. It seems that there may be a linear relationship between number (or rate) of embedding changes and the position of feature in the feature space, since linear classifiers [21], [24] are sufficient to devise very accurate detectors.

Some drawbacks of this approach are that detectors are targeted towards a given steganographic algorithm and payload size [26] (used to create the training stego set), and to a particular source of covers. If the objects under scrutiny

Manuscript received January 9, 2014; revised April 21, 2014; accepted June 6, 2014. Date of publication July 9, 2014; date of current version July 31, 2014. This work was supported by the European Office of Aerospace Research and Development under research grants FA8655-11-3035 and FA8655-13-1-3020. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation there on. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of EOARD or the U.S. Government. The work of T. Pevný was supported by the Grant Agency of Czech Republic under Project P103/12/P514. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Teddy Furon.

A. D. Ker is with the Department of Computer Science, Oxford University, Oxford OX3 0AU, U.K. (e-mail: adk@cs.ox.ac.uk).

T. Pevný is with the Agent Technology Center, Czech Technical University at Prague, Prague 166 36, Czech Republic, and also with Cisco Systems, Prague 117 21, Czech Republic (e-mail: pevnak@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2014.2336380

come from a different source (e.g. camera or settings) from the training data, the accuracy of detectors decreases, often dramatically [1]. This phenomenon, a cover-source mismatch, is unavoidable in reality unless the suspected steganographer is considerate enough to supply their enemy with their cover source.

Another drawback is that the design is for a decision function for single objects, which would only be applicable in the case when a steganalyst is presented with a small amount of data to classify individually. It does not address the challenges of large-scale steganalysis.

II. Large-Scale Steganalysis

Suppose that a steganalyst is monitoring a large network, with multiple users and many potentially suspect communications. For example, they might be scanning all the images on a social media site for hidden content, or acting as a corporate firewall to prevent data exfiltration. We identify four requirements for a steganalysis system:

Universality. The steganalyst may not know what steganography algorithm is being used on the network. As much as possible, their detector should be able to identify unknown or new embedding methods, with unknown sizes of payload. Most existing steganalysis methods do not have this property (see Subsection III-D for a survey of prior art).

Robustness. If the steganalyst has some training data, they cannot ensure that it comes from an identical source to that used by the actors they are monitoring. As much as possible, their detector should not suffer unpredictably from cover-source mismatch. Again, existing steganalysis methods generally fail this condition [27].

Multiple actor. The network has multiple users, some (most) of them innocent of steganographic embedding, but each with a slightly different cover source. The detector needs to determine *who* is guilty, not necessarily which of their objects specifically contain payload. No previous steganalysis methods have considered this case. Exactly what output is required depends on the situation: it might be known that at least one guilty actor exists, or not, and it might be required to obtain a probability of guilt for each or simply a ranking. In this paper we assume that it is sufficient to rank the actors in order of likeliness of guilt.

Multiple object. Each actor emits many objects. For innocent actors, all of their objects are plain covers. For guilty actors, some (not necessarily all) of them contain payload. The detector must aggregate the evidence in the objects, and this is the *pooled steganalysis* problem from [14], which has not yet been addressed successfully.

We also require a certain computational efficiency, ideally linear in the number of objects captured from the network. Large-scale monitoring, in real-world scenarios, may have to cope with vast amounts of intercepted data.

Naturally, we expect some penalty for universality and robustness: a hypothetical detector for single images with these properties would likely be inferior to existing binary classification steganalysis when tested under laboratory conditions (known algorithm and payload size, no mismatch). However, we are able to turn the large-scale situation, with multiple actors and objects, to our advantage. More evidence is available: of individual actor's guilt, of the behaviour of innocent actors, and (crucially) of how much innocent actors sources tend to differ from each other. As with the original pooled steganalysis problem, the difficulty is how to aggregate the evidence.

III. Detecting Anomalous Actors

Our proposed detector identifies actors that significantly deviate from the majority. We assume the scenario of multiple actors each emitting multiple objects, all of which are seen by the detector, who also knows which actor sent what. In the discussion below, we assume that the objects are digital images, but the same system could be used for any domain with good steganalytic features.

The detector works in three steps: first, extracting standard steganalytic features from all objects; second, calculating distances between each pair of *actors* based on the cloud of feature points that they have emitted; third, identifying actors deviating from the majority using an anomaly measure computed from the distances. If steganalytic features are sensitive to hidden payload, and relatively insensitive to other characteristics of the objects, then an actor's deviation is evidence of their guilt: the steganographer is the outlier. These three steps are now described in detail, with a discussion of the design choices to be made by the steganalyst.

A. Features

The detector extracts features from every image transmitted by every actor. The steganalyst's first design decision is to select a suitable feature set: in theory, a detector should work with any steganalytic features sensitive to embedding changes and relatively insensitive to image content.

In experiments performed in this paper, which use JPEG images, we have chosen so-called PF274 features [28], because they reliably detect the steganographic algorithms used (described in Subsection IV-B), their extraction is fast, and they have good signal to noise ratio. In other work we have shown that the detector works with high dimensional features as well [20], but due to their sensitivity to image content, they have to be made robust with respect to it [31].

Once features are extracted from all images, the steganalyst must pre-process them, to make the contribution of each feature equal and hence the distance (below) meaningful. We determined (see Section VI-A) that a global whitening works best. The whitening projects features into a new space, of slightly lower dimension, where features are uncorrelated and they have unit variance in each direction. The base of the projection space is found by eigenvalue decomposition of the features' covariance matrix (the same operation is used in principal component analysis) calculated from all images. For numerical stability, projections with corresponding eigenvalues smaller than 0.01 (see Section VI-A) are discarded.

B. Distance Between Actors

We propose to measure distance between actors using an empirical Maximum Mean Discrepancy (MMD) [9], which

is a measure of similarity between probability distributions. It has the useful property that it can be estimated robustly, even for high dimensional probability distributions, from relatively little data. MMD corresponds to an L_2 distance in some Hilbert space implicitly defined through a positive definite kernel function $\kappa(x, y) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ (if pre-processed features are real and of dimension d). Popular kernels include the linear kernel $\kappa(x, y) = x^{\mathrm{T}} y$ and the Gaussian kernel $\kappa(x, y) = \exp(-\gamma \|x - y\|^2)$, where γ is the inverse kernel width. Assuming n samples $\{x_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$, pre-processed feature vectors from actors X and Y, a sample estimate of the MMD distance has the following simple form

$$MMD(X, Y) = \frac{1}{\binom{n}{2}} \sum_{1 \le i < j \le n} \kappa(x_i, x_j) - \kappa(x_j, y_i) - \kappa(x_i, y_j) + \kappa(y_i, y_j). \quad (1)$$

An adjustment can be made when the number of samples is different between X and Y, for which we refer to the original publication [9].

The above formula requires $O(n^2)$ computations, where n is the number of images emitted by each actor. This is undesirable for large-scale application, but a simple approximation is available: in case of a the linear kernel, MMD converges with $n \to \infty$ to $\|\bar{x} - \bar{y}\|_2^2$, the L_2 norm between the actors' centroids $\bar{x}$ and $\bar{y}$ in the feature space (see Appendix B). We call this the centroid 'kernel' (it is not really a kernel, but an asymptotic approximation) and use it extensively in our experiments because it can be computed in $O(n)$ time. The influence of the MMD kernel on the quality of detection is studied in greater detail in Section VI-B.

C. Anomaly Detector

Once distances between actors are calculated, we identify outlying actors. From the plethora of outlier detection methods [3], we have chosen the local outlier factor (LOF) method [2], as it has several desirable features: (a) it detects outliers in probability distributions with clusters of different densities; (b) the provided anomaly score is interpretable, as values around one corresponds to true 'inliers' and values greater than two correspond to outliers. Given a set P of points (actors) with a metric $d : P \times P \to [0, \infty)$ and an integer parameter $1 < k < |P|$, the LOF is calculated as follows.[1]

The **reachability distance of point p from q**, $r_k(p, q)$, is the greater of $d(p, q)$ and $d(q, q')$, where q' is q's k-nearest neighbour. Compared with the metric d, the reachability distance reduces statistical fluctuations for close objects, with smoothing controlled by the parameter k.

Fix a point p, and write P_k for the k-nearest neighbourhood of p in P. The **local reachability density of** p is defined as an inverse of the average reachability distance of point p from all points $q \in P_k$,

$$lrd_k(p) = \left(\frac{1}{k} \sum_{q \in P_k} r_k(p, q) \right)^{-1},$$

and the **local outlier factor (LOF) of** p is

$$lof_k(p) = \frac{1}{k} \sum_{q \in P_k} \frac{lrd_k(q)}{lrd_k(p)}.$$

Thus $lof_k(p)$ captures the degree to which p is further from its k-nearest neighbours than they are from theirs. Defining it as a relative number means that it does not depend on absolute values of distances $d(p, q)$.

The original publication recommends $k = 10$, and we have used this value throughout all experiments except in Section VI-C. The results will show that the optimal value of k depends on the number of guilty actors and their embedding strategies. The LOF calculation is quadratic in the number of actors (as it must compute and rank all pairwise distances) which is slightly undesirable, but the number of actors is likely to be orders of magntitude smaller than the number of images.

By design, this detector works (only) for a multi-actor, multi-image scenario. And because it is completely unsupervised, it cannot suffer from mismatch between training and testing data. Thus our requirement of robustness is automatically met. It remains to demonstrate that it works and has good universality.

D. Relation to Prior Art

The vast majority of published work on steganalysis attacks a different problem: analysis of one image at a time. The first work proposing to investigate multiple images from a one actor was [14]: it describes different strategies of aggregating detection results from individual images to find whether one actor is guilty or not. Although the work assumes a targeted detector for a single image, it could be used with universal steganalyzer as well. It does not consider the scenario of multiple actors.

Universal steganalysis, where the steganalyst does not know the embedding algorithm, is a largely neglected field of research. Probably the first work in this field was [25], which modelled distribution of cover images by one-class SVM and classified deviations as stego images. The problem was further studied in [30], where it is shown that universal steganalyzers are sensitive to what is now called cover-source mismatch.

To the best of our knowledge, the large-scale steganalysis scenario described here has not been attacked at all, except in our prior work [16]–[18]. Combining the universal single-image steganalyzer [30] with aggregation methods published in [14] is the only prior art we can find. We compare our detector to it in Section V-B.

IV. Simulating the Real World

We wish to validate this new detection paradigm, in a situation which mimics as much as possible a real-world network scanning problem. We therefore selected covers, steganographic embedding methods, and strategies for guilty actors to allocate payload between covers, to mimic a hypothetical steganographer inserting payload into social media images.

[1] For this exposition we assume no exact duplicates in P or exactly tied distances between members of P, which simplifies the description considerably. For full details, see the original publication [2].

A. Cover Images

The images were obtained from a leading social network site, which is popular for sharing pictures. Such a service could provide an ideal steganographic channel, because uploading and downloading many images should not raise suspicion. We used a web crawler to download all *public* images from users who identified themselves as members of Oxford University. We stopped after downloading more than 4 million images from more than 70 000 users. All personally identifiable information was removed, and the files anonymized except for grouping images uploaded by the same user. The *actors* in our experiments are the uploaders, which mimic well the behaviour of real-world actors: sometimes a single actor uses two or three cameras. In the experiments described here, we used a randomly selected subset of 4000 actors and 200 images for each actor, for a total of 800 000 images.

At the time of crawling, the social networking site automatically resized large images, to approximately 1Mpix, and then JPEG compressed them with quality factor 85. This simplifies steganalysis, since it is known that steganalytic features are very sensitive to different quantisation matrices [29], but does introduce a second compression (if the files were originally uploaded as JPEGs); double compression is usually considered a difficult nuisance parameter in steganalysis [29].

Apart from a fairly uniform size and completely uniform quality factor, the images in the database are very diverse, as they (a) come from different sources (cameras, flatbed scanners), (b) are of different types (indoor party pictures, cities, outdoor nature scenes, etc.), and (c) underwent different image processing from acquisition to download. Some of them are not natural images at all, but synthetic images or mosaics. Most steganalysis literature would perform experiments on images with such "impurities" removed, but we did not remove them: the impurities are there in practice and these images are a good prototype for what might be expected when monitoring a real network. Their proportion in the database should reflect the proportion we can expect in the wild, since it was crawled from a real-world source.

B. Embedding Algorithms

In our experiments, we have used the following five steganographic algorithms: F5 [38], [39], F5 with shrinkage removed by wet paper codes and matrix embedding turned off (nsF5), JPHide&Seek [23], OutGuess [32], [33], and Steghide [10], [11]. These algorithms have diverse embedding mechanisms, software implementations are all publicly available (except for nsF5 where only a simulator exists), and they do not utilise side information in the form of the raw image. Thus they could be applied by a non-expert. Furthermore, there is copious evidence that these embedding methods can be detected by the chosen steganalytic feature set. Below, the ideas behind each algorithm are briefly described. For further details we refer to the original publications.

OutGuess [32] is an improved version of **JSteg** [37]. OutGuess inserts the message by using standard LSB replacement, while it avoids changing zeros and ones. Since this embedding operation changes the first-order histogram of DCT coefficients, OutGuess reserves some DCT coefficients to restore, approximately, the first order histogram. By doing so, OutGuess performs approximately twice as many changes as JSteg, which makes the algorithm more detectable by methods (features) modelling higher order dependencies.

Unlike OutGuess, **F5** [38] does not try to preserve the first-order histogram of DCT coefficients. Instead, it preserves the shape of the histogram, making it similar to that of the cover image. The message is embedded by changing the absolute values of DCT coefficients toward zero. DCT coefficients equal to zero are skipped, and if the coefficient is changed to zero during embedding, it is skipped as well and a new one is utilised for re-embedding. The F5 algorithm was also the first algorithm to use matrix embedding, a coding scheme that increases embedding efficiency, here measured as the number of bits embedded per embedding change.

Steghide [11] tries to preserve first-order statistics, but without making additional embedding changes like OutGuess. The algorithm starts by constructing a graph, where each vertex corresponds to a group of pixels that need to be changed. The weight of an edge between two vertices is proportional to the distortion caused by modification of both vertices such that they code the message. During the embedding, the algorithm finds the partition of the graph minimising the cost, subject to the chosen message being coded.

Despite the C source code for **JPHide&Seek** being available, its method of operation has not been described. To our knowledge, the algorithm has not been published in any scientific or other paper.

The **nsF5** algorithm uses the same type of embedding changes as the F5 algorithm. To avoid introducing more zeros (the shrinkage effect), nsF5 uses wet paper codes with improved efficiency [6]. The experiments in this paper used the version of the algorithm from 2008, which simulates the embedding efficiency of particular wet paper codes; this differs from the version currently published by the author, which simulates the theoretically-optimal efficiency.

C. Embedding Strategies

Embedding in multiple images poses new problems, originally described in [14]. The steganographer must choose how to spread a message of total length M bits into n covers $(X_1, \ldots, X_n)$ with capacities $(c_1, \ldots, c_n)$ by using the chosen steganographic algorithm. We distinguish the *embedding strategy*, which allocates payload amongst objects, from the *embedding algorithm* which inserts the payload steganographically.

In [17], we have identified five simple strategies to break the message into fragments of lengths $(m_1, \ldots, m_n)$ such that $M = \sum_{i=1}^{n} m_i$. Since one of the strategies had little practical value, we omit it here. None of the strategies is theoretically optimal, and indeed the batch steganography problem has not been solved. We have chosen strategies that could be applied by a non-expert, similarly to our choice of embedding algorithms.

The **greedy** strategy tries to use as few images as possible. The steganographer chooses the cover with highest capacity, and embeds part of his message up to maximum capacity. If more message remains, he repeats with the cover of next highest capacity, until the whole message is embedded.

If the images are ordered by capacity so that $c_1 \geq c_2 \geq \ldots \geq c_n$, this leads to the following message lengths:

$$\begin{aligned} m_i &= c_i, \quad \forall i \in \{1, \ldots, I-1\}, \\ m_I &= M - \sum_{i=1}^{I-1} m_i, \\ m_i &= 0, \quad \forall i \in \{I+1, \ldots, n\}, \end{aligned}$$

where I denotes the smallest possible number of images with sufficient capacity, i.e.

$$I = \arg\min_i M \leq \sum_{j=1}^{i} c_j.$$

The **maximum** strategy is a variation of the greedy strategy, where the images for embedding are selected in random order and used to full capacity. This simulates a case of a steganographer who is not able to estimate capacity until they embed.

The **linear** strategy distributes the message into all available covers proportionately to their capacity. This means that

$$m_i = \frac{c_i M}{\sum_{j=1}^{n} c_j}.$$

(Fractional bits are ignored in this study.)

In the **even** strategy, the message is distributed evenly into all available covers regardless of their capacity. Thus

$$m_i = \frac{M}{n}.$$

For relatively large payloads and covers of uneven capacity, sometimes m_i exceeds c_i. In this cases, we set $m_i = c_i$ and recalculate an even message length for the remaining images.

We do not consider, in this work, how the receiver is to reconstruct the original message. The allocation of payload might be part of a shared secret key, or stored in the first few bits of payload in a fixed position.

V. Main Experimental Results

We simulated large-scale steganalysis using tens of thousands of experiments. In each experiment, we randomly selected N_A actors out of the 4000 in our data set, and N_I images from each actor. Exactly one guilty actor is simulated, by using the chosen embedding algorithm (from Subsection IV-B) and embedding strategy (Subsection IV-C) to insert of payload size np, where $0 \leq p < 0.25$ and n is the total number of nonzero coefficients in their images. Thus p is the number of bits per nonzero coefficient (bpnc). It is important to measure the payload size relative to a fixed quantity, not to the capacity of an individual embedding algorithm, otherwise the results are incomparable. We then calculate features from each of the $N_A N_I$ images, MMD between each pair of actors, and LOF scores for each actor.

For each combination of parameters, each experiment is repeated 500 times with a different selection of actors and guilty actor. We need a benchmark to reflect how well the guilty actor is identified, and we have chosen the *average rank of the guilty actor*. An average rank of one corresponds to perfect detection — the guilty actor is always ranked most suspicious — and an average rank of $\frac{N_A+1}{2}$ corresponds to random guessing. We would not expect a universal, unsupervised detector to achieve perfect accuracy, but instead hope that it provides intelligence by ranking a truly guilty actor amongst the top 5-10%, say, of all actors.

A. Detecting Different Algorithms

We first demonstrate that the proposed detector is capable of detecting a wide range of algorithms (it has good universality). We tested $N_A \in \{100, 400, 1600\}$ actors with $N_I = 100$ images each, and every combination of embedding algorithm and strategy from subsects. IV-B and IV-C. The steganalyser used the centroid 'kernel' for the MMD distance measure, whitening of raw features (more on this in Subsection VI-A), and the LOF parameter $k = 10$.

The average ranks of the true guilty actor, when hiding payloads $p \in \{0.025, 0.05, \ldots, 0.25\}$, are shown in Figure 1. Our first observation is that the method works, when the total payload is large enough. There are differences between embedding algorithms (it confirms the known relative security of the embedding algorithms [28], that nsF5 is most secure and OutGuess/Steghide least secure) and between embedding strategies, but the overall pattern is consistent. With weaker embedding algorithms, perfect identification of the guilty actor is achievable around 0.1–0.2 bpnc payload sizes; perfect detection is not observed with nsF5 and F5, but the guilty actor is consistantly ranked as one of the 2–6 most suspicious out of 100. Monitoring larger number of actors does not substantially change the results, except that the average rank of the guilty actor apparently scales slightly sublinearly with the total number of actors N_A. A similar phenomenon was observed in [17], using a different metric for the anomaly detector, and this finding would seem favourable for large-scale steganalysis, but it has not yet been explained.

Second, we observe that the greedy strategy is consistently the most secure for the embedder: the average rank of the guilty actor is higher, in all algorithms and payloads. We will later show that this is only true when the steganalyst uses the centroid 'kernel'. The second most secure strategy is maximum, except in the case of F5 where the matrix embedding induces a nonlinear relationship between payload size and steganographic distortion. The linear strategy is next, and even is most insecure. The reason that the greedy/maximum strategy is more secure than linear/even has been explained in detail in our prior work [17].[2] In brief, this effect is caused by whitening the features in the pre-processing stage.

[2]In [17] the greedy and maximum strategies are called max-greedy and max-random, respectively.

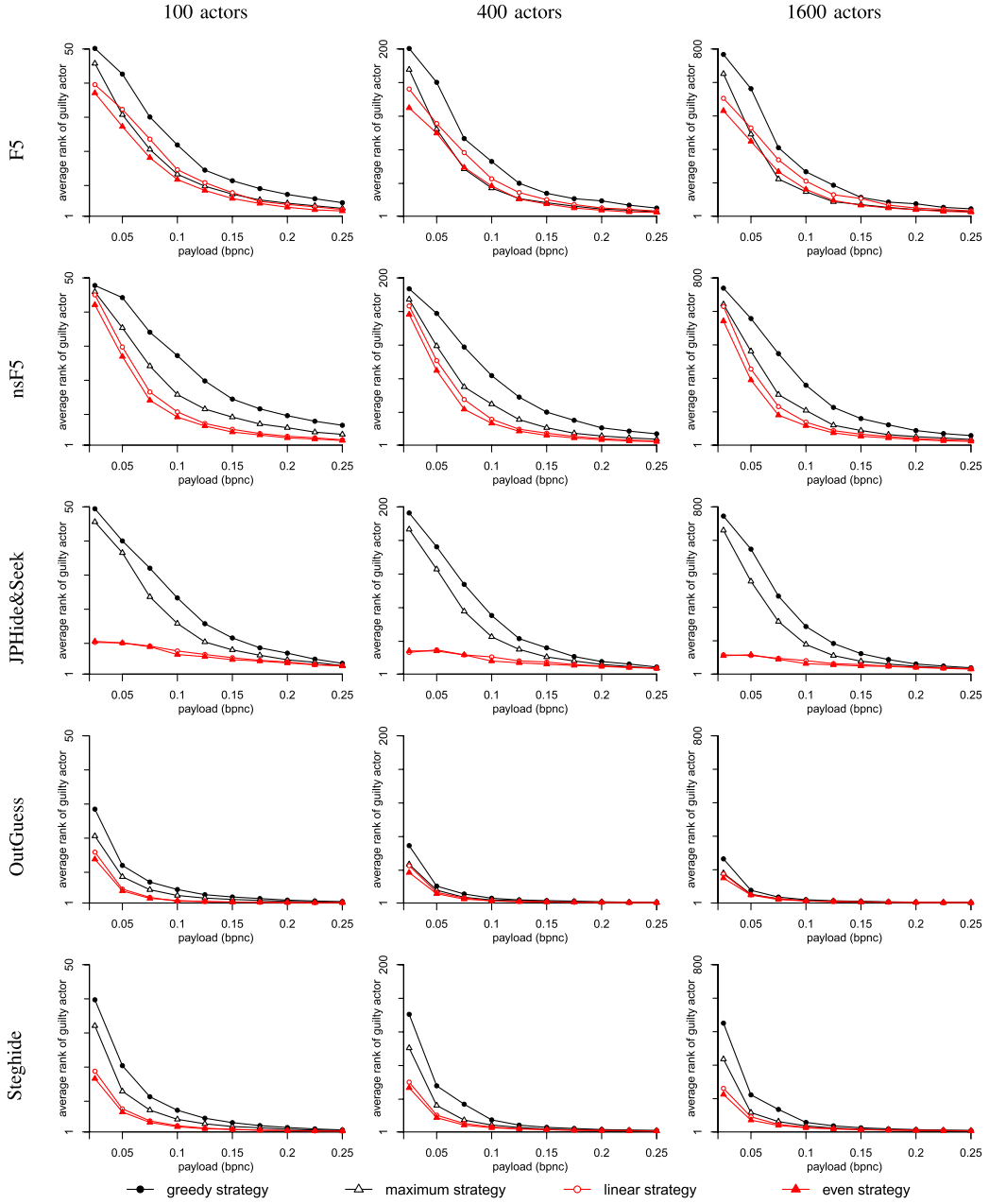


Fig. 1. Performance of the proposed detector: from top to bottom, five different embedding algorithms; from left to right, different numbers of actors N_A ($N_I = 100$ in each case); lines in each chart denote different embedding strategies; each x-axis represents total payload (bpnc) and each y-axis represents the average rank of the truly guilty actor. The detector parameters are: centroid 'kernel', whitened features, $k = 10$.

But, as will be shown in Subsection VI-A, such preprocessing is needed to achieve good accuracy. We discuss this further in Section VII.

Because the greedy strategy dominates maximum, and linear dominates even, in subsequent experiments we will discard the maximum and even strategies.

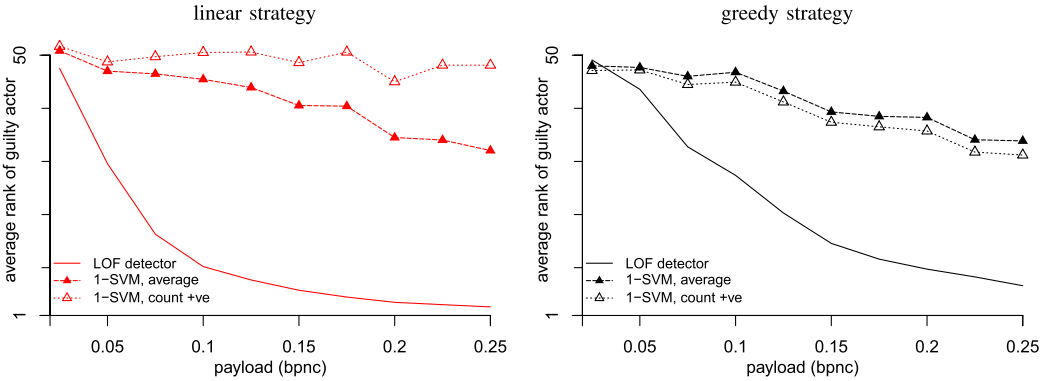


Fig. 2. Comparison of proposed detector with prior art. Each x-axis represents total payload (bpnc) and each y-axis represents the average rank of the truly guilty actor; left, using the linear strategy; right, using the greedy strategy. In both cases $N_A = N_I = 100$, and the detector parameters are: centroid 'kernel', whitened features, $k = 10$.

B. Comparison With Prior Art

As mentioned above, no other literature addresses the large-scale steganalysis problem, so there is no direct prior art that we can compare to. The only method we can identify, for ranking guilty actors without knowledge of the embedding algorithm, is a combination of the universal detector proposed in [30] and *pooling strategies* described in [14] (which aggregate the scores for each actor).

The universal steganalyzer, here implemented as a one-class support machine (1-SVM) [34], assigns to each image a score $f(x) = \langle w, x\rangle_{\mathcal{H}} - b$, where x is a feature vector of a given image and (w, b) defines a hyperplane in a Hilbert space $\mathcal{H}$, which is determined from the training data. Following [14], we implemented two pooling strategies to rank the guiltiness of each actor from the scores of their images: (i) we calculated the average score

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i),$$

or (ii) we calculated the number of positive scores (number of images classified as outliers)

$$\#\{x_i \mid f(x_i) > 0\}.$$

There may well be better methods for aggregation, but the literature does not yet contain them.

The 1-SVMs were trained on 6000 cover images from 60 actors; the pool of actors used for this training was disjoint from, but from the same social media source as, all the other experiments in the paper. To avoid bias from picking a particularly good or poor set of training images, we trained 20 different 1-SVMs, on different cover examples, and picked one of the machines at random for each experiment. The 1-SVM hyperparameters were $\nu = 0.01$ (proportion of outliers) and γ (width of Gaussian kernel) using the following common heuristic: inverse median of squared distances between cover images (features) in the training set. The features were normalised to have zero mean and unit variance. Note the distinction between the method presented in this paper, which is completely untrained, and the use of 1-SVMs, which do require cover training data.

The average rank of the guilty actor, hiding payloads $p \in \{0.025, 0.05, \ldots, 0.25\}$ using only the nsF5 algorithm and greedy/linear strategies, is shown in Figure 2. The steganalyst used the two aggregations of 1-SVM scores, or the proposed detector with the same settings as in the previous subsection. Here $N_A = 100$ and $N_I = 100$ (similar results are observed, but not included here, with other combinations of parameter). The graphs clearly show that the proposed solution is substantially more accurate than prior art (except for tiny payloads when both are guessing randomly), which is not aggregating effectively the evidence from the multiple images.

VI. Supplemental Experiments

The detector described in Section III has several hyperparameters, which influence its performance. In the previous section, we used parameters based on our previous experiments published in [16] and [18]. Here, we re-examine the choices one-by-one.

Unless otherwise indicated, all experiments in this section share the same setting of one guilty actor emitting a payload of 0.1 bpnc, using nsF5 algorithm and the greedy or linear strategy. The steganalyst uses the proposed detector with whitened features, the centroid 'kernel', and LOF parameter $k = 10$. $N_I = 100$ and $N_A \in \{100, 400, 1600\}$.

A. Pre-Processing

The pre-processing of features has a significant impact on the accuracy of detection. In steganalysis literature a common pre-processing is normalisation, where each feature is individually scaled to have zero mean and unit variance (usually across the cover training set). The goal is to prevent features with high variance from dominating other, perhaps more informative, features of low variance. In this application it is essential, so that the MMD distances are meaningful and not dominated by noisy components.

For additional stabilization, one can also apply whitening (principal component transform), to decorrelate the features

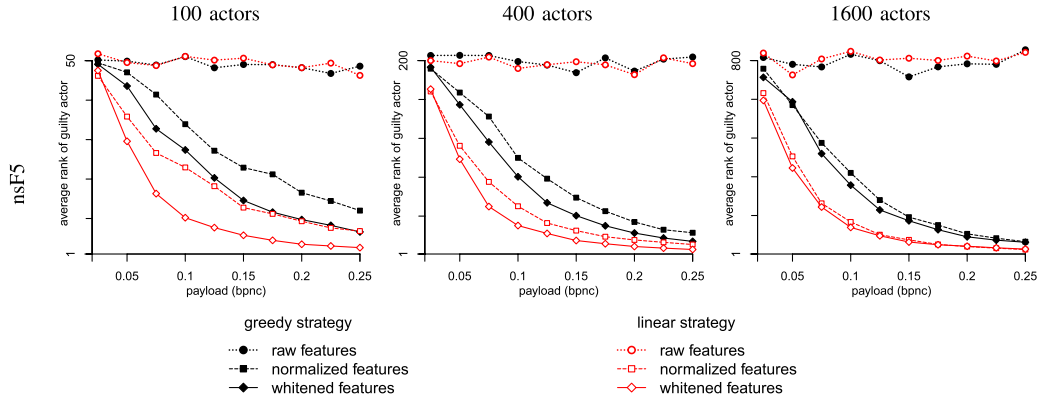


Fig. 3. The effect of different feature pre-processing. From left to right, different numbers of actors N_A ($N_I = 100$ in each case); lines in each chart denote different pre-processing options and embedding strategy; each x-axis represents total payload (bpnc) and each y-axis represents the average rank of the truly guilty actor. The embedder uses nsF5 embedding. The other detector parameters are: centroid 'kernel', $k = 10$.

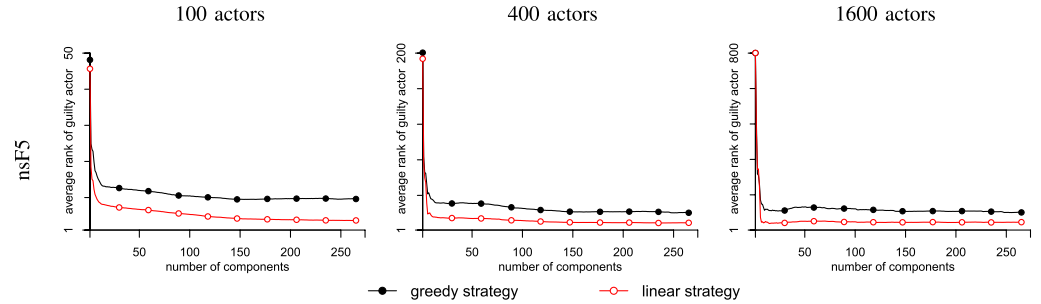


Fig. 4. The effect of the number of components retained after whitening. From left to right, different numbers of actors N_A ($N_I = 100$ in each case); lines in each chart denote different embedding strategy; each x-axis denotes the number of components kept for the LOF analysis, and each y-axis represents the average rank of the truly guilty actor. The embedder uses nsF5 embedding with total payload 0.2 bpnc. The other detector parameters are: centroid 'kernel', $k = 10$.

(again, usually on the cover training set); in this application, because equally-scaled components are essential, we further apply normalization after whitening.

Figure 3 shows the performance of the detector, when the steganalyst uses unprocessed (raw), normalised, and whitened features. In all situations (different embedding strategies of the guilty actor, different number of actors) the raw features are near-useless, confirming the importance of equal scaling of features in the anomaly detector: a different situation may hold in supervised classifiers, where the training phase can learn to ignore noisy features. Whitened features are consistently the best option. Surprisingly, with an increasing number of actors, the advantage of whitening over normalization decreases.

In image recognition applications, principal component analysis (PCA) is frequently used as a denoising filter, discarding components corresponding to small eigenvalues in the correlation matrix. We use the same in our application, applying it to all feature vectors pooled across all actors in each experiment, and discarding the components with low eigenvalue after whitening the features. In the previous experiments we discarded such components with corresponding eigenvalues lower than 0.01, as we expected them to carry noise. But is this really a sensible choice? Figure 4 shows the average rank of the guilty user, as we vary the number of components retained after whitening: the components were sorted from those highest eigenvalue (variance) to lowest, and we kept only the highest. The results show that better accuracy is acheived when most components are used; for large number of actors the improvement is very negligible and it looks like that the optimum is near, but not quite at, the maximum of retaining all 274 components. Due to the difficulty of finding this maximum in advance, and for good numerical stability, we suggest continuing to use the eigenvalue threshold 0.01.

B. Kernel

The kernel function used in the calculation of MMD distance underpins the entire system. Thus far we have used the centroid 'kernel', which approximates the linear kernel $\kappa(x, y) = x^{\mathrm{T}}y$, because it has linear time complexity. Experiments in [18] report its superiority with respect to other kernels, but only one embedding strategy was examined.

There is reason to believe that alternative kernels should have advantages against certain embedding strategies.

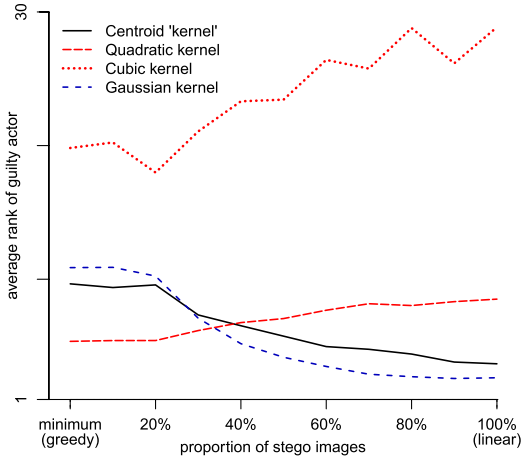


Fig. 5. The effect of the MMD kernel. The x-axis denotes strategies interpolating smoothly between greedy (left) and linear (right), the y-axis the average rank of the truly guilty actor. The different lines indicate diffeerent kernels. Here $N_A = N_I = 100$, the payload is 0.2 bpnc and the embedding is by nsF5, the features were whitened, and the LOF parameter $k = 10$.

TABLE I

THE AVERAGE RANK OF 1, 2, 4, OR 8 GUILTY ACTORS USING EITHER LINEAR OR GREEDY STRATEGY. THE DETECTOR VARIES THE NEAREST NEIGHBOUR PARAMETER k IN THE LOF METHOD. HERE $N_A = N_I = 100$, THE PAYLOAD IS 0.2 BPNC, THE EMBEDDING IS BY NSF5, AND THE FEATURES WERE WHITENED. THE LAST ROW CAPTIONED "PERFECT" SHOWS THE AVERAGE RANK OF A PERFECT DETECTOR

	linear strategy #guilty actors				greedy strategy #guilty actors			
k	1	2	4	8	1	2	4	8
2	2.7	7.0	63.1	62.2	6.9	14.5	29.3	42.7
4	2.9	5.0	16.3	62.5	7.8	11.3	19.1	31.6
6	3.1	4.9	10.0	43.0	8.4	11.8	17.9	28.2
8	3.4	4.8	9.0	24.1	9.2	12.4	17.6	26.5
10	3.5	5.1	8.7	17.8	9.7	12.1	17.7	26.5
12	3.7	5.1	8.5	15.4	9.8	12.5	17.8	26.1
14	3.8	5.1	8.2	15.1	10.3	13.0	18.3	27.4
16	3.9	5.2	8.2	14.3	10.7	13.0	18.9	27.3
18	3.9	5.5	8.2	13.8	10.7	13.4	19.6	28.2
20	4.1	5.4	8.0	13.6	10.9	14.0	19.7	28.2
Perfect	1	1.5	2.5	4.5	1	1.5	2.5	4.5

To summarise the argument in [17]: because of noisy components, steganalysis features per object are distorted by an amount *sublinear* in the payload, and the linear kernel only agglomerates this distortion linearly. Hence the apparent superiority of the greedy strategy. But consider some of the theoretical steganography work on small payloads: in principle, distortion relating to statistical detectability (KL divergence) is locally quadratic in the payload size [5]. Thus a kernel which captures such distortion would be more powerful against the greedy strategy. This motivates us to examine polynomial kernels $k(x, y) = (x^{\mathrm{T}}y)^2$ (quadratic, which should be powerful against the greedy strategy) and $k(x, y) = (x^{\mathrm{T}}y)^3$ (cubic), as well as Gaussian $k(x, y) = \exp(-\gamma \|x - y\|^2)$. The problem of setting γ is treated in more detail in Appendix A. We also investigated higher-order polynomials, but the results were weak and are not included here.

For a more fine-grained analysis of embedding strategy, we used methods which apply linear embedding to the greatest-capacity proportion P of the guilty actor's images. For small P, this is equivalent to the greedy strategy (maximum payload in fewest covers); for $P = 100\%$ it is the linear strategy, and in between it exchanges size of payload-per-image for number of images used. The average rank of the guilty actor, when testing each kernel against strategies for various P, is shown in Figure 5. The results validate our theory that the embedding strategies have a detection counter-strategy: for low P (greedy) the quadratic kernel is indeed most accurate, while for high P it is the Gaussian kernel with best performance. Our use of the centroid 'kernel' was slightly sub-optimal, in that it is dominated by the Gaussian or quadratic kernels, but the difference between centroid and Gaussian is not so great as to outweigh the benefits of its linear time complexity.

As predicted in [14] and [15], we find a two-player game between the embedder and detector. We could define the zero-sum payoff to be the average rank of the guilty actor. In that case we can even, purely for illustration, use standard linear-programming techniques to compute the equilibrium strategies from our empirical data. It turns out that both players should use *mixed* (randomized) strategies to avoid being exploited by their opponent: the embedder should pick the greedy or linear strategy at random, with probability approximately 0.52 and 0.48, respectively (intermediate options are dominated), while the detector should pick the Gaussian kernel with probability 0.27, and quadratic with probability 0.73. The average rank is then approximately 6.9. Of course, these strategies are only optimal within the narrow confines of this game, in particular for this embedding algorithm, payload size, number of images, and number of actors, and other possible kernels or embedding strategies may change the results. The game theory of steganography is still in its infancy [36], and has not reached the batch steganography case.

C. Multiple Guilty Actors, Effect of k

Thus far, our experiments have simulated only one guilty actor, and the number of nearest neighbours in LOF method was set to $k = 10$, as recommended in [2]. If there are multiple guilty actors in a tight cluster then the optimal choice of k depends on the size of that cluster: too small a value k causes the cluster to be deemed 'normal', too large will smooth out its anomaly level. To briefly investigate this, we vary the number of guilty actors from $\{1, 2, 4, 8\}$ out of a total of $N_A = 100$ actors, and tested $k \in \{2, 4, 6, \ldots, 20\}$. We compute the same metric: average rank of the guilty actor.

The results in Table I reveal that smaller values of k are slightly more accurate for detecting a single guilty actor, and larger values better for detecting more actors. This suggests that the guilty actors indeed form some sort of cluster, which would probably not happen if the guilty actors used different

embedding methods or strategies (we postpone such further experiments to future work). The default value of $k = 10$, which we have used in this paper, seems to be a good compromise, but the optimal k depends also on the embedding strategy used by the guilty actors, introducing another potential game between embedder and detector.

VII. Conclusion

We have presented a first method for addressing a new and realistic problem in steganalysis: detecting the guilty actor, rather than individual image, in the setting of network monitoring when there are many actors and images to consider. To our knowledge this is the first work to address this problem or the pooled steganalysis problem [14], and the first to use MMD in steganalysis detection. A key element is to turn some of the difficulties in the problem – the large number of users and images – to our advantage by calibrating the behaviour of outliers against that of the majority, allowing completely unsupervised detection.

Our experiments, which simulated embedding and detection in $100 - 1600$ actors, each transmitting 100 images,[3] repeated thousands of times with different combinations of embedding parameters, payload size, and detection parameters.

Essential to every modern steganalysis method are the features that drive it. We used a well-established feature set, which is not one of the recent "rich model" [7], [13], [20] feature sets that have been published recently, for two reasons. First, for experiments on such a scale it is essential that the features be calculated quickly and the many-thousand dimensional rich model features are not quick to extract. Second, and a subject of our current work, is that larger feature sets perform *worse* when plugged into the same anomaly detection framework despite being considerably more effective in traditional binary steganalysis. We have examined this phenomenon and it is an unavoidable consequence of unsupervised learning: there is no way to discard or weight down the weaker features in a training phase, so the anomaly detector is overwhelmed by noise. It illustrates that the design of good features for the standard binary-classification steganalysis, where extra features are essentially free of cost, does not translate well into an unsupervised case. Our current work involves partially-supervised dimensionality reduction to focus the power of large feature sets into a smaller set which is suitable for unsupervised anomaly detection [31].

We also tested only well-understood embedding algorithms which have long been known to be detectable by statistical analysis, and naive embedding strategies with little or no adaptive allocation of payload between images. Again this is required for experimental efficiency, but we were also motivated by simulating a "real-world" steganographer who uses tools currently easily available on the internet. For the same reason we used a large image set downloaded from social media. Our method, however, is applicable to any embedding method where the features are more sensitive to stego content than payload content, and it would be very surprising if even the most recent adaptive embedding methods [4], [12] were not detectable, albeit at higher payloads because of their lower distortion, than their simpler ancestors tested here.

[3]Further experiments with as few as 20 and as many as 200 images were performed, but the results were so similar to those presented here that we do not include them.

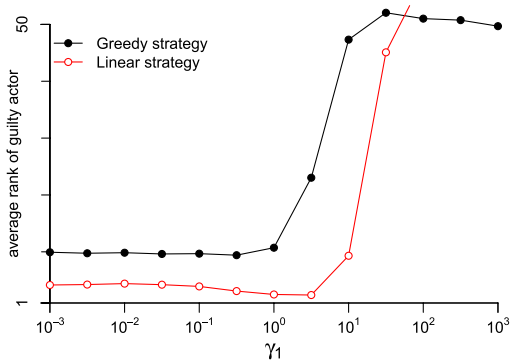


Fig. 6. Effect of the Gaussian kernel width. The embedder uses nsF5 embedding with total payload 0.2 bpnc, the features have been whitened, and the LOF parameter is $k = 10$.

Although we have investigated good parameters (feature preprocessing, kernel, LOF parameter) for the detector, this demonstrated the game at the heart of steganography and steganalysis: fixed (known) choices by the embedder allows the detector to tune parameters to enhance accuracy, whereas fixed (known) choices by the detector allow the embedding to tune their parameters to reduce detection accuracy. We cannot solve such games until all the options, for embedder and detector, are properly understood, but in the future we might hope to find both equilibria and, more practically usefully, conservative minimax strategies for each player.

Other future work should be to examine more closely the cases of multiple guilty actors, as well as the case of zero guilty actors: throughout this paper we assumed that at least one actor was guilty, which may not be true in application. Ideally we would like to estimates the probability of guilt for each actor, which is a known and difficult problem in anomaly detection. Another direction might be to use multiple anomaly detectors instead of a single LOF, hoping that diversity amongst outlier-detection methods will lead to better results, but this runs into the known, nontrivial, problem of aggregating scores [22].

Appendix

A. Gaussian Kernel Width

The quality of detection with the Gaussian MMD kernel $\kappa(x, y) = \exp(-\gamma \|x - y\|^2)$ depends on the choice of γ, and a wrong choice can decrease the performance. With the recommendation of [35], we investigated γ in a range around γ_0, the latter defined as the inverse of the median squared distances between (whitened) image features. We set $\gamma = \gamma_0 \cdot \gamma_1$ where $\gamma_1 \in \{10^n | n \in \{-3, -2, \ldots, 3\}\}$. Fixing on a payload of 0.2 bpnc, the nsF5 embedding algorithm, and the greedy and linear strategies, we tested all such γ and display the results in Fig. 6.

Because of the connections between linear and Gaussian kernel with small γ (see Appendix B), it is not surprising that small kernel γ does not penalise the performance very much; large γ causes performance to decrease to random guessing because the MMD calculations are dominated by outliers in the feature point clouds of each actor. The optimum is indeed around the default $\gamma = \gamma_0$ in each case.

B. Connections Between Kernels

We have used primarily the centroid 'kernel'; here we demonstrate its connection with the true linear kernel, and also the Gaussian kernel for large kernel width.

First, use symmetry of the kernel to write

$$MMD_{linear}(X,Y) = \frac{1}{n(n-1)} \times \sum_{1\le i\ne j\le n} x_i^{\mathrm{T}} x_j - 2x_i^{\mathrm{T}} y_j + y_i^{\mathrm{T}} y_j.$$

Then we expand

$$\begin{aligned} &MMD_{centroid}(X,Y) \\ &= \Big(\frac{1}{n}\sum_i x_i - \frac{1}{n}\sum_i y_i\Big)^{\mathrm{T}}\Big(\frac{1}{n}\sum_i x_i - \frac{1}{n}\sum_i y_i\Big) \\ &= \frac{1}{n^2}\sum_{i,j} x_i^{\mathrm{T}} x_j - 2x_i^{\mathrm{T}} y_j + y_i^{\mathrm{T}} y_j \\ &= \frac{n}{n-1} MMD_{linear}(X,Y) + \frac{1}{n^2}\sum_i (x_i^{\mathrm{T}} - y_i)(x_i - y_i). \end{aligned}$$

This demonstrates that the centroid 'kernel' approximates the true linear MMD for large n.

Now let k be the Gaussian kernel with inverse width γ, then

$$k(x,y) = 1 - \gamma\|x-y\|^2 + O(\gamma^2),$$

so that for small γ we have

$$MMD_{Gaussian}(X,Y) \approx \frac{\gamma}{n(n-1)} \sum_{1\le i\ne j\le n} 2\|x_i - y_j\|^2 - \|x_i - x_j\|^2 - \|y_i - y_j\|^2.$$

The first term measures average distances *between* the distributions X and Y, and the other terms measure average distances *within* them. Indeed, if the x_i (respectively y_i) are drawn from any multivariate distribution with mean μ_x (μ_y) and finite covariance matrix Σ_x (Σ_y) then elementary calculations give

$$\begin{aligned} E\big[\|x_i - y_j\|^2\big] &= \mathrm{Tr}(\Sigma_x + \Sigma_y) + \|\mu_x - \mu_y\|^2, \\ E\big[\|x_i - x_j\|^2\big] &= 2\,\mathrm{Tr}(\Sigma_x), \\ E\big[\|y_i - y_j\|^2\big] &= 2\,\mathrm{Tr}(\Sigma_y), \end{aligned}$$

and hence by the law of large numbers $MMD_{Gaussian}(X,Y) \to \gamma \cdot MMD_{centroid}(X,Y) + O(\gamma^2)$ as $n \to \infty$. The LOF method is scale insensitive, so the factor γ has no effect on it. This explains the behaviour seen in Fig. 6.

REFERENCES

[1] P. Bas, T. Filler, and T. Pevný, "'Break our steganographic system': The ins and outs of organizing BOSS," in *Proc. 13th Inf. Hiding Conf.*, 2011, pp. 59–70.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 93–104.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[4] T. Filler and J. Fridrich, "Design of adaptive steganographic schemes for digital images," *Proc. SPIE, Media Watermark., Security, Forensics XIV*, vol. 7880, p. 78800F, Feb. 2011.

[5] T. Filler, A. D. Ker, and J. Fridrich, "The square root law of steganographic capacity for Markov covers," *Proc. SPIE, Media Forensics Security XI*, vol. 7254, pp. 801–811, Feb. 2009.

[6] J. Fridrich, M. Goljan, and D. Soukal, "Wet paper codes with improved embedding efficiency," *IEEE Trans. Inf. Forensics Security*, vol. 1, no. 1, pp. 102–110, Mar. 2006.

[7] J. Fridrich and J. Kodovský, "Rich models for steganalysis of digital images," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 3, pp. 868–882, Jun. 2012.

[8] J. Fridrich, J. Kodovský, V. Holub, and M. Goljan, "Steganalysis of content-adaptive steganography in spatial domain," in *Proc. 13th Inf. Hiding Conf.*, 2011, pp. 102–117.

[9] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample problem," in *Advances in Neural Information Processing Systems 19*. Cambridge, MA, USA: MIT Press, 2007, pp. 513–520.

[10] S. Hetzl. (2003, Oct.). *Implementation of the Steghide Algorithm ver. 0.5.1* [Online]. Available: http://steghide.sourceforge.net/

[11] S. Hetzl and P. Mutzel, "A graph–theoretic approach to steganography," in *Proc. 9th Int. Conf. Commun. Multimedia Security*, 2005, pp. 119–128.

[12] V. Holub and J. Fridrich, "Digital image steganography using universal distortion," in *Proc. 1st ACM Workshop Inf. Hiding Multimedia Security (IH&MMSec)*, New York, NY, USA, 2013, pp. 59–68.

[13] V. Holub and J. Fridrich, "Random projections of residuals for digital image steganalysis," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 1996–2006, Dec. 2013.

[14] A. D. Ker, "Batch steganography and pooled steganalysis," in *Proc. 8th Inf. Hiding Workshop*, vol. 4437. 2006, pp. 265–281.

[15] A. D. Ker, "Batch steganography and the threshold game," *Proc. SPIE, Security, Steganography, Watermark. Multimedia Contents IX*, vol. 6505, pp. 401–413, Mar. 2007.

[16] A. D. Ker and T. Pevný, "A new paradigm for steganalysis via clustering," *Proc. SPIE, Media Watermark., Security, Forensics III*, vol. 7880, pp. U01–U13, Feb. 2011.

[17] A. D. Ker and T. Pevný, "Batch steganography in the real world," in *Proc. 14th ACM Workshop Multimedia Security (MM&Sec)*, 2012, pp. 1–10.

[18] A. D. Ker and T. Pevný, "Identifying a steganographer in realistic and heterogeneous data sets," *Proc. SPIE, Media Watermark., Security, Forensics XIV*, vol. 8303, pp. N01–N13, May 2012.

[19] J. Kodovský and J. Fridrich, "Steganalysis in high dimensions: Fusing classifiers built on random subspaces," *Proc. IS&T/SPIE Electron. Imag.*, vol. 7880, p. 78800L, 2011.

[20] J. Kodovský and J. Fridrich, "Steganalysis of JPEG images using rich models," *Proc. SPIE, Media Watermark., Security, Forensics XIV*, vol. 8303, p. 83030A, Feb. 2012.

[21] J. Kodovský, J. Fridrich, and V. Holub, "Ensemble classifiers for steganalysis of digital media," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 432–444, Apr. 2012.

[22] H. P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Interpreting and unifying outlier scores," in *Proc. 11th SIAM Int. Conf. Data Mining*, 2011, pp. 13–24.

[23] A. Latham. (1999, Aug.). *Implementation of the JPHide and JPSeek Algorithms ver 0.3* [Online]. Available: http://linux01.gwdg.de/~alatham/stego.html

[24] I. Lubenko and A. D. Ker, "Going from small to large data in steganalysis," *Proc. IS&T/SPIE Electron. Imag.*, vol. 8303, p. 83030M, Feb. 2012.

[25] S. Lyu and H. Farid, "Steganalysis using higher-order image statistics," *IEEE Trans. Inf. Forensics Security*, vol. 1, no. 1, pp. 111–119, Mar. 2006.

[26] T. Pevný, "Detecting messages of unknown length," *Proc. SPIE, Electron. Imag., Media Watermark., Security Forensics Multimedia XIII*, vol. 7880, pp. 23–26, Jan. 2011.

[27] T. Pevný, P. Bas, and J. Fridrich, "Steganalysis by subtractive pixel adjacency matrix," *IEEE Trans. Inf. Forensics Security*, vol. 5, no. 2, pp. 215–224, Jun. 2010.
[28] T. Pevný and J. Fridrich, "Merging Markov and DCT features for multi-class JPEG steganalysis," *Proc. SPIE, Media Watermark., Security, Forensics IX*, vol. 6505, pp. 3–14, Feb. 2007.
[29] T. Pevný and J. Fridrich, "Multiclass detector of current steganographic methods for JPEG format," *IEEE Trans. Inf. Forensics Security*, vol. 3, no. 4, pp. 635–650, Dec. 2008.
[30] T. Pevný and J. Fridrich, "Novelty detection in blind steganalysis," in *Proc. 10th Workshop Multimedia Security*, 2008.
[31] T. Pevný and A. D. Ker, "The challenges of rich features in universal steganalysis," *Proc. SPIE, Media Watermark., Security, Forensics*, vol. 8665, pp. M01–M15, Mar. 2013.
[32] N. Provos, "Defending against statistical steganalysis," in *Proc. 10th Conf. USENIX Security Symp.*, vol. 10. 2001, pp. 323–335.
[33] N. Provos. (2001, Oct.). *Implementation of the OutGuess Algorithm ver. 2.0* [Online]. Available: http://www.outguess.org/
[34] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, 2001.
[35] B. Schölkopf and A. J. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.
[36] P. Schöttle, S. Korff, and R. Böhme, "Weighted stego-image steganalysis for naive content-adaptive embedding," in *Proc. 4th IEEE Int. Workshop Inf. Forensics Security (WIFS)*, Dec. 2012, pp. 193–198.
[37] D. Upham. (2012, Apr.). *Implementation of the JSteg Steganographic Algorithm* [Online]. Available: http://zooid.org/~paul/crypto/jsteg/
[38] A. Westfeld, "F5—A steganographic algorithm," in *Proc. 4th Inf. Hiding Workshop*, vol. 2137. 2001, pp. 289–302.
[39] A. Westfeld. (2012, Apr.). *Implementation of the F5 Steganographic Algorithm* [Online]. Available: http://code.google.com/p/f5-steganography/

Andrew D. Ker (M'06) was born in Birmingham, U.K., in 1976. He received the B.A. degree in mathematics and computer science and the D.Phil. degree in computer science from Oxford University, Oxford, U.K., in 1997 and 2001, respectively.

He is a fellow and Praelector in Computer Science with University College, Oxford, and a University Lecturer with the Department of Computer Science, Oxford University. He has published widely on steganography and steganalysis, in practice and theory.

Dr. Ker is also a member of the Association for Computing Machinery.

Tomáš Pevný is a Researcher with the Czech Technical University of Prague, Prague, Czech Republic. He received the Ph.D. degree in computer science from the State University of New York, Binghamton, NY, USA, in 2008, and the M.S. degree in computer science from the Czech Technical University of Prague in 2003. From 2008 to 2009, he was with Gipsa-Lab, Grenoble, France, as a Post-Doctoral Researcher. His main research interests are in nonparametric statistics with a focus on steganography, steganalysis, network security, and intrusion detection.

Appendix 4 - Kohout, J. and Pevný, T. : *Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test* [33]

Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test

Jan Kohout and Tomáš Pevný

***Abstract*— Many applications and communication protocols exhibit unique communication patterns that can be exploited to identify them in network traffic. This paper proposes a method to represent these patterns compactly, such that they can be used in different analytical tasks. The method treats each communication as a set of observations of a random variable with unknown probability distribution. This view allows us to derive the representation from a distance between two probability distributions used in maximum mean discrepancy—a non-parametric kernel test. The representation (and distance) can be then easily used in various algorithms for identification of communicating application and data analysis, independently of the specific type of input data.**

***Index Terms*— Communication fingerprinting, maximum mean discrepancy, application identification.**

I. Introduction

IDENTIFICATION of applications based on information obtained by monitoring their network traffic is a well established problem in the network security domain. Knowing which application is communicating with which server improves the understanding of a monitored network, resulting into more precise identification of infected computers and better enforcement of policies, because in some enterprises, the use of specific types of programs is against policy. Even though approaches relying on content inspection (also known as deep packet inspection, DPI) are still widely used ([1]–[8]), the need of lightweight methods not inspecting the content has been recognized by both the industry and the research community. This need is currently driven by the increased adoption of encryption, which is on the rise and which complicates content inspection. Furthermore, the constant growth of network traffic volume makes content inspection prohibitively expensive, because detection systems should work in real-time.

It has turned out that the identification of applications from their communication does not always require to inspect the content ([9]–[15]). As these works show, it is often sufficient to observe just basic features of flows or packets based on their sizes and timing to distinguish individual protocols or applications with acceptable accuracy. This is because most applications and protocols have unique statistical distributions of these variables, which allows to use machine learning algorithms to solve problems of interest, e.g. distinguishing applications based on their network footprint. This brings the need to represent the distributions in a compact way so that they can be effectively handled by the algorithms. Many works solve this by using either aggregate statistics (e.g., mean, variance or entropy) to represent the distributions or by capturing marginal distributions of observed features ([15]–[19]). This can reduce the amount of information about the distribution that the representation is able to capture. Moreover, these representations are often suited for specific scenarios and adapting them to different setups might demand considerable effort.

This work presents a framework for fingerprinting repeated communication over the network. The proposed fingerprint effectively captures joint distribution of features and represents them in a compact way. It consists of a single vector of fixed dimension designed such that the Euclidean distance between fingerprints approximates Maximum Mean Discrepancy [20], a well established measure of similarity of two probability distributions observed through a set of finite number of samples. The advantage of the proposed fingerprint is (i) strong theoretical justification, (ii) finite dimension of the fingerprint, which allows it to be used with most contemporary machine learning algorithms, and (iii) easy adaption of the framework to new data with different features. The experimental evaluation reveals that the proposed representation outperforms methods adopted from previous works, it can successfully handle different types of data (packets and flows), and is generally usable in a detection system relying on machine learning algorithms.

The main contribution of this work is the efficient approximation of the Maximum Mean Discrepancy measure described in Section II and the representation of a set of observations as a single real vector of finite dimension. This makes the proposed representation scalable and usable beyond the field of network traffic analysis, whenever the sample consists of a set of observations.

This paper is organized as follows: In Section II, statistical fingerprints of communication are defined and their effective representation is proposed. In Section III, the related work is reviewed, emphasizing the differences from this work.

Manuscript received March 9, 2017; revised August 10, 2017; accepted October 18, 2017. Date of publication October 30, 2017; date of current version December 19, 2017. This work was supported by Research Grant GAČR 15-08916S. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mauro Barni. *(Corresponding author: Jan Kohout.)*

The authors are with the Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, 16636 Prague, Czech Republic and also with Cisco Systems, Inc., San Jose, CA 95134 USA (e-mail: jkohout@cisco.com; pevnak@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.2768018

In Sections IV and V, methods selected from previous works for comparison are characterized in more detail and datasets used in evaluation are described. Sections VI and VII are then devoted to the evaluation, showing applications of the proposed technique in different scenarios. Finally, the work is concluded in Section VIII.

II. Communication Fingerprinting

Before diving into details of the proposed characterization of a sample of communication as a vector in Euclidean space of fixed dimension, key terms and notations are introduced.

Definition 1: Message is a basic unit of communication exchanged between two communicating peers. The exact definition of message is dependent on the level at which the communication is observed. It can be a single IP-layer or transport layer packet, a transport layer flow or even a request-response pair in a client-server communication. It is assumed that each message can be represented as a point $m \in \mathbb{R}^d$ in d-dimensional Euclidean space and is characterized by a set of d features. Attributes of a single message can be its size, duration (if measurable), or time from the previous message. However, in general, they can be any features observable and relevant for messages in the given scenario.

Definition 2: Message set is a set of messages $\mathcal{R} = \{m_1, \ldots, m_n\}$ sharing the same identifier of the communication. Again, the specific identifier of the message set is dependent on the level of observation and it can vary from a single flow specified by the both communicating endpoints to entire server specified by its IP address or hostname. For example, if messages are defined as individual TCP packets, then a message set can be defined as one TCP flow of packets that share the same source and destination IPs and ports.

Definitions 1 and 2 are general and inference of the representation presented below was made independently of any specific features observed on messages and or type of communication identifier. Thanks to this the entire fingerprinting framework can be easily adapted to different scenarios and sources of data.

For purposes of a message set representation, each message $m \in \mathbb{R}^d$ in a message set $\mathcal{R}$ is treated as a realization of a d-dimensional random variable with probability distribution $P_{\mathcal{R}} \in \mathcal{P}$, where $\mathcal{P}$ is the set of all probability distribution on the space of messages $\mathbb{R}^d$. The underlying assumption is that the message set $\mathcal{R}$ is fully determined by its distribution $P_{\mathcal{R}}$ followed by its messages. In practice, $P_{\mathcal{R}}$ is never precisely known, but it is observed through that finite set of messages $\mathcal{R} = \{m_i \in \mathbb{R}^d \mid i = 1, ..., n\}$. Naturally, different message sets might have different cardinalities due to the different number of observed messages in them.

Assuming the message set $\mathcal{R}$ to be fully characterized by its probability distribution $P_{\mathcal{R}} \in \mathcal{P}$, the fingerprint of the message set and a metric between two message sets $\mathcal{R}_1, \mathcal{R}_2$ should reflect distance between two probability distributions $P_{\mathcal{R}_1}$, $P_{\mathcal{R}_2}$. This would enable to easily compare two message sets and also paves the road to use algorithms from the rich state of the art in machine learning to achieve desired goals.

There exists plethora of distances and pseudo-distances on the space of probability distributions [21]: Kullback-Leibler divergence, Total Variation, Bhattacharyya distance, Hellinger distance, and Rényi entropy to name a few. However, the proposed fingerprint is derived from Maximum Mean Discrepancy (MMD) [20], because it can be efficiently estimated from a small number of observations. Moreover, it enables to compare joint distributions of features without explicitly estimating their probability density functions. This is important, because joint distributions can provide rich information about the features but direct estimation of the probability density would be often impossible due to demand for extremely high number of observations. MMD is defined as

$$\mathrm{MMD}(\mathcal{F}, P_{\mathcal{R}_1}, P_{\mathcal{R}_2}) = \sup_{f \in \mathcal{F}} \left\{ \mathrm{E}_{x \sim P_{\mathcal{R}_1}}[f(x)] - \mathrm{E}_{y \sim P_{\mathcal{R}_2}}[f(y)] \right\},$$

where $\mathcal{F}$ is a unit ball in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ ([22], [23]) associated with a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (here $\mathcal{X} = \mathbb{R}^d$). If the kernel is characteristic ([20], [24]), then MMD defines a metric on the space $\mathcal{P}$ and it holds that $\mathrm{MMD}(\mathcal{F}, P_{\mathcal{R}_1}, P_{\mathcal{R}_2}) = 0$ if and only if $P_{\mathcal{R}_1} = P_{\mathcal{R}_2}$. An example of a characteristic kernel is the Gaussian kernel used in [20], defined as $k(x, y) = \exp\left(-\gamma \|x - y\|_2^2\right)$.

As already mentioned, an important advantage of MMD is that it can be well estimated from a limited set of observations. Given two message sets $\mathcal{R}_1 = \{m_i^1\}_{i=1}^{n_1}$, $\mathcal{R}_2 = \{m_i^2\}_{i=1}^{n_2}$ (with distributions $P_{\mathcal{R}_1}$ and $P_{\mathcal{R}_2}$), estimate of $\mathrm{MMD}^2(\mathcal{F}, P_{\mathcal{R}_1}, P_{\mathcal{R}_2})$ can be calculated ([20]) as

$$\begin{aligned} &\mathrm{MMD}^2(\mathcal{F}, \mathcal{R}_1, \mathcal{R}_2) \\ &\quad = \frac{1}{n_1^2} \sum_{i,j=1}^{n_1,n_1} k(m_i^1, m_j^1) \\ &\qquad + \frac{1}{n_2^2} \sum_{i,j=1}^{n_2,n_2} k(m_i^2, m_j^2) - \frac{2}{n_2 n_1} \sum_{i,j=1}^{n_1,n_2} k(m_i^1, m_j^2) \end{aligned} \quad (1)$$

The above estimate can be compactly written as the norm of estimates of means in the Hilbert space $\mathcal{H}$ as

$$\mathrm{MMD}^2(\mathcal{F}, \mathcal{R}_1, \mathcal{R}_2) = ||\mu_1 - \mu_2||_{\mathcal{H}}^2, \quad (2)$$

where

$$\mu_1 = \frac{1}{n_1} \sum_{i=1}^{n_1} k(m_i^1, \cdot), \quad \mu_2 = \frac{1}{n_2} \sum_{i=1}^{n_2} k(m_i^2, \cdot) \quad (3)$$

are so called *mean maps* of the distributions $P_{\mathcal{R}_1}$ and $P_{\mathcal{R}_2}$. Note that the mean map is a function of one real variable, which is marked by the $\cdot$ symbol in the definitions above. Therefore, $\mu(t) = \frac{1}{n} \sum_{i=1}^{n} k(m_i, t), t \in \mathbb{R}$. Reader interested in more details about MMD and its calculation is referred to [20].

Although the MMD is a theoretically well justified distance, its practical use is limited due to the excessive computational and memory requirements. Memory requirements grow linearly with the size of $\mathcal{R}$ (number of observed messages) and system's memory would be quickly exhausted (discarding old messages should be avoided to prevent loss of information). Moreover, calculation of the distance between two message sets $\mathcal{R}_1$ and $\mathcal{R}_2$ is of order $O(\max\{|\mathcal{R}_1|, |\mathcal{R}_2|\}^2)$, which is prohibitive in practice. These constrains motivated the search for an alternative representation and calculation that would

approximate the MMD (1) with computational complexity independent of sizes of individual message sets.

The proposed method approximates the mean maps (3) as a linear combination of a fixed base set of basis functions $\mathcal{K} = \{k(l_i, \cdot)|i \in \{1, \ldots, |\mathcal{L}|\}\}$, where $\mathcal{L} \subset \mathbb{R}^d$ is a set of points chosen from the same space in which the messages exist. For now, it is assumed that the set $\mathcal{L}$ is already given and its construction is deferred to the end of this section. To express given mean map μ in $\mathcal{K}$, i.e. to find coefficients of basis function amounts α, the following optimization problem is solved:

$$\alpha = \text{argmin}_\alpha \left\| \sum_{i=1}^{|\mathcal{L}|} \alpha_i k(l_i, \cdot) - \mu \right\|_{\mathcal{H}}^2 \quad (4)$$

$$= \text{argmin}_\alpha \left\| \sum_{i=1}^{|\mathcal{L}|} \alpha_i k(l_i, \cdot) - \frac{1}{n} \sum_{j=1}^{n} k(m_j, \cdot) \right\|_{\mathcal{H}}^2, \quad (5)$$

where $n = |\mathcal{R}|$. Denoting kernel matrix of points from $\mathcal{L}$ as $\mathbf{K}_{ij} = k(l_i, l_j)$, $\mathbf{K} \in \mathbb{R}^{|\mathcal{L}|,|\mathcal{L}|}$ and kernel matrix of points from $\mathcal{L}$ and messages from $\mathcal{R}$ as $\mathbf{Q}_{ij} = k(l_i, m_j)$, $\mathbf{Q} \in \mathbb{R}^{|\mathcal{L}|,n}$, the solution of (5) is

$$\alpha = \frac{1}{n} \mathbf{K}^{-1} \mathbf{Q} \mathbf{1}_n, \quad (6)$$

where $\mathbf{1}_n$ is a vector of ones of size n. The solution vector α is therefore of fixed dimension $\alpha \in \mathbb{R}^{|\mathcal{L}|}$, which depends on the cardinality of the set $\mathcal{L}$.

Let's now assume two message sets $\mathcal{R}_1$ and $\mathcal{R}_2$ with mean maps expressed in the base $\mathcal{K}$ as α^1 and α^2. Their MMD distance can be then approximated as:

$$\text{MMD}^2(\mathcal{F}, \mathcal{R}_1, \mathcal{R}_2) \approx \left\| \sum_i \alpha_i^1 k(l_i, \cdot) - \sum_i \alpha_i^2 k(l_i, \cdot) \right\|^2 \quad (7)$$

$$= {\alpha^1}^{\mathrm{T}} \mathbf{K} \alpha^1 - 2{\alpha^1}^{\mathrm{T}} \mathbf{K} \alpha^2 + {\alpha^2}^{\mathrm{T}} \mathbf{K} \alpha^2. \quad (8)$$

Since $\mathbf{K}$ is positive semi-definite (if k is Mercer's kernel), there exists a Cholesky decomposition such that $\mathbf{K} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ and $\mathbf{L}$ is a triangular matrix. Using $\mathbf{L}$, the approximated MMD distance (8) can be compactly expressed as

$$\text{MMD}^2(\mathcal{F}, \mathcal{R}_1, \mathcal{R}_2) \approx \left\| \mathbf{L}\alpha^1 - \mathbf{L}\alpha^2 \right\|^2. \quad (9)$$

Substituting the calculation of α from (6) and using the property of Cholesky decomposition that $\mathbf{K}^{-1} = \mathbf{L}^{-1}{\mathbf{L}^{-1}}^{\mathrm{T}}$, the approximate MMD becomes

$$\begin{aligned} \text{MMD}^2(\mathcal{F}, \mathcal{R}_1, \mathcal{R}_2) &\approx \left\| \mathbf{L}(\alpha^1 - \alpha^2) \right\|^2 \\ &= \left\| \mathbf{L}\mathbf{K}^{-1} \left(\frac{1}{n_1} \mathbf{Q}^1 \mathbf{1}_{n_1} - \frac{1}{n_2} \mathbf{Q}^2 \mathbf{1}_{n_2} \right) \right\|^2 \\ &= \left\| \mathbf{L}\mathbf{L}^{-1}{\mathbf{L}^{-1}}^{\mathrm{T}} \left(\varphi(\mu_1) - \varphi(\mu_2) \right) \right\|^2 \\ &= \left\| {\mathbf{L}^{-1}}^{\mathrm{T}} \left(\varphi(\mu_1) - \varphi(\mu_2) \right) \right\|^2. \end{aligned}$$

where $\varphi : \mathcal{H} \to \mathbb{R}^{|\mathcal{L}|}$ is a functional evaluating the mean map $\mu = \frac{1}{n} \sum_{i=1}^{n} k(m_i, \cdot)$ (i.e., Parzen window estimates) at points from $\mathcal{L}$

$$\varphi(\mu) = (\mu(l_1), ..., \mu(l_{|\mathcal{L}|})).$$

Note that it holds that

$$\begin{aligned} \varphi(\mu) &= \frac{1}{n} \mathbf{Q} \mathbf{1}_n \\ &= \frac{1}{n} \left(\sum_{i=1}^{n} k(m_i, l_1), \ldots, \sum_{i=1}^{n} k(m_i, l_{|\mathcal{L}|}) \right). \end{aligned}$$

Therefore, after pre-computing the inverse $\mathbf{L}^{-1}$ which is relatively easy as the matrix $\mathbf{L}$ is triangular, MMD distance $\|\mu_1 - \mu_2\|_{\mathcal{H}}^2$ can be approxiamted by Euclidean distance $\|{\mathbf{L}^{-1}}^{\mathrm{T}}(\varphi(\mu_1) - \varphi(\mu_2))\|_2^2$. In other words, the MMD distance between μ_1 and μ_2 in $\mathcal{H}$ is approximated by generalized Mahalanobis distance (as introduced in [25]) between $\varphi(\mu_1)$ and $\varphi(\mu_2)$ in $\mathbb{R}^{|\mathcal{L}|}$, defined by the matrix $(\mathbf{L}^{-1})\mathbf{L}^{-1^{\mathrm{T}}} = \mathbf{K}^{-1}$.

A fingerprint of a message set is therefore computed and stored using the function φ in a form of an $|\mathcal{L}|$-dimensional real vector, which can be updated on-line as new messages are observed, which removes the need to keep all messages in memory. Storing fingerprint as $\varphi(\mu)$ is preferred to $\mathbf{L}^{-1^{\mathrm{T}}}\varphi(\mu)$ (which would save computational resources) as most of its components are equal to zero (or very small such they can be replaced by zero without loss of accuracy), enabling efficient storage in sparse structures.

In further text, this proposed representation is called Approximated MMD representation — AMRep. Computing the AMRep representation of message sets from an input set of messages thus include these steps:

1) Forming the set $\mathcal{L}$ with respect to the given input set of messages.
2) Computing the matrix $\mathbf{L}^{-1}$.
3) Computing and maintaining the function φ for each message set.

To this end it has been assumed that the set $\mathcal{L}$ is given. The method how $\mathcal{L}$ is constructed should reflect the intention of its use. For determining the set $\mathcal{L}$, it is assumed that there is an input training set of messages $\mathcal{M}$ available, that can be used to compute $\mathcal{L}$. In a typical case, $\mathcal{M}$ will be a set of all messages from message sets obtained as training data for a classifier, or all messages contained in some unknown data that are about to be analyzed. A common requirement would be to select $\mathcal{L}$ such that $|\mathcal{L}| \ll |\mathcal{M}|$. Otherwise, storing of fingerprints and all computations would be expensive. If the fingerprint would be used to model *majority* of traffic, then items of $\mathcal{L}$ should be located in parts of the space where most messages are located (i.e., in the most "interesting" areas of the input space) and algorithms like self-organising maps [26] are appropriate. Contrary, if fingerprints will be used to distinguish many different communictions or to detect rare events, elements of $\mathcal{L}$ should *cover* the entire space of all messages, including regions with low density of messages. The latter approach is followed in this work. Formally, the cover should in this case satisfy the following condition: $\forall m \in \mathcal{M}$ $\exists m' \in \mathcal{L}$ such that $\|m' - m\|_2 \leq \epsilon$, where ϵ is a preset constant. This parameter determines granularity and size of the cover $\mathcal{L}$. Below experiments used this condition to build

Algorithm 1 Greedy ϵ-Cover of a Set $\mathcal{M}$

```
procedure COVER(M, ε)
    L ← {m}, m randomly selected from M
    for m ∈ M do
        if ∀m' ∈ L : ‖m − m'‖_2 > ε then
            L ← L ∪ {m}
        end if
    end for
    return L
end procedure
```

$\mathcal{L}$, when a sub-optimal solution to the class cover problem ([27]–[29]) was obtained by a greedy algorithm shown in the listing of Algorithm 1. However, the AMRep representation is not limited to this specific algorithm and any method which is able to construct $\mathcal{L}$ with desired properties can be considered.

III. Related Work

Modelling applications based on statistical properties of their network communication has been already investigated in the related works differing mainly in the goal of the model. Large part of published methods models communication to detect command and control (C&C) channels of botnets that are used by a botmaster to communicate with infected computers. The main reason is that the presence of a C&C channel is an intrinsic property of botnets and they frequently have a distinct statistical properties.

BotMiner [30] is a complex system detecting C&C channels by correlating communication patterns of network hosts. From a sequence of TCP flows (called C-flows in the reference), it extracts the number of flows per hour, number of packets per flow, average number of bytes per one packet, and average number of bytes per second in a flow, and uses marginal distributions of these quantities to identify groups of possible C&C channels. The main difference to this work is in features and their scope, since the cited work is targeted only to detect botnets and therefore the algorithms rely on additional assumptions about the traffic of C&C channels, such as periodicity or, in general, low entropy (i.e., well predictable behaviour) of the communication.

BotFinder [15], Disclosure [17], and [18], [19], [31] all represent communication by aggregate statistics (mean and standard deviations, Aiello *et al.* in [31] also uses kurtosis and skewness) of marginal distribution of different features such as transferred bytes, inter-arrival times, and durations of either flows or packets. Individual works differ by features they use, by the definition of a message (flow or packet), by the statistics (only [31] uses kurtosis and skewness), and finally by their aim, as the works [15], [17] aim to detect botnets while the work [31] detects DNS tunnelling. Nevertheless, their common property is the use of simple statistics over marginals, not joint distributions.

Wright *et al.* [9] aims at identification of application protocols, namely HTTP(S), AIM, SSH, SMTP and Telnet, from a stream of TCP packets. The work utilizes only sizes of packets and their directions (i.e., from client to server or from server to client) to model the protocols' behaviour. The representation is a histogram of four types of packet according to their size and direction ($\{\text{incoming, outgoing}\} \times \{\text{small,large}\}$) seen during the observation window. To make the representation richer, histograms from several consecutive widows are concatenated together. Contrary to this reference, the proposed work uses finer quantization (specified by the cover $\mathcal{L}$) of the data and uses a joint distribution.

Packets' sizes and inter-arrival times (the time between two consecutive packets in the same direction) extracted from TCP flows of packets are used in works [13] and [32] to build statistical representation of application layer protocols. The representation is a collection of N joint histograms, where each histogram models size and inter-arrival time of i-th packet in the observed set of flows. This makes this representation dependent on information about messages' order in a message set which might not be always available and it is subject to noise caused by reordering during transit. Contrary to these references, the proposed work treats messages as i.i.d. realizations of a random variable.

Hermann *et al.* in [33] aims at fingerprinting of websites from streams of TCP packets captured during loading of web pages to identify which web page was accessed by encrypted communication. The classifier is implemented as naive Bayes with histograms of packets' sizes as the feature vector. SPID [16] is a modular framework for identification of application layer protocols (BitTorrent, HTTP, SSH,...). Depending on the specific source of data, it can use different number of features. Independently on the particular feature set which is used, observed values of each feature are modelled separately by a histogram. The classifier is implemented as a nearest prototype classifier with distance defined by Kullback-Leibler divergence between the known protocol models and a representation created from the classified sample of communication. AMRep differs from these works by relying on joint distribution and by using a distance derived from MMD measure.

The work [10] proposes a representation of TCP flows based on aggregate statistics of transferred packets and bytes in each flow. This representation is then used by an SVM classifier to identify communicating application, browser or operating system. While the domain in which that representation is applied overlaps with aim of this work, the difference is in the way how the representation is designed (AMRep treats messages individually while [10] uses aggregate statistics). Furthermore, this representation is not easily extendable to cases when it is needed to represent unordered sets of messages (e.g., for representing servers instead of individual flows as shown in Section VII).

In [34], a mechanism based on Hidden Markov Models (HMM) is introduced to build profiles of users behind a NAT. The representation uses only elementary features like sizes and inter-arrival times of NetFlows to represent users' behaviour. However, the training of the underlying HMMs is supervised (needs a training set with labels determining which NetFlow belongs to which user), which makes the representation unusable for cases with little or no labelled data.

Appendix 4 - Kohout, J. and Pevný, T. : *Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test* [33]

The experimental section of this article compares the presented method (AMRep) to [32], [33] and [16], because representations used in these works are the most similar to that proposed in this work. They all build probability distributions of observed features to represent network communication. Detailed description of the compared methods can be found in Section IV.

IV. Compared Methods

The representation proposed in this work is compared to the related methods [33], [16] and [32] selected because (i) they can be extended to different definitions of messages and message sets and (ii) they cover principles of most approaches proposed by the state of the art algorithms that rely on modelling of probability distributions.

A. Multinomial Naive Bayes (MNB)

Hermann *et al.* [33] models encrypted communication by one-dimensional histogram of datagram sizes with datagrams' direction encoded by sign. In the definition used in this paper, one message in [33] corresponds to one packet and a message set corresponds to a set of packets interchanged during loading of one specific web page. As individual messages are represented by scalar values (their sizes), MNB uses a one-dimensional histogram $h^{\mathcal{R}}$ to represent a message set $\mathcal{R}$. The histogram has m bins $h^{\mathcal{R}}_1, ..., h^{\mathcal{R}}_m$ and the i-th bin caputres number of messages of size i observed in $\mathcal{R}$. Each target class ω is represented by a set of empirical probabilities $\{P(i|\omega)|i = 1, ..., m\}$, where $P(i|\omega)$ is an estimation of probability that a message of size i will be observed in a message set belonging to class ω. An unknown message set $\mathcal{R}$ is then assigned to the most probable class ω^*, which satisfies:

$$\omega^* = \arg\min_{\omega} - \sum_{i=1}^{m} h^{\mathcal{R}}_i \log P(i|\omega).$$

Since the motivation behind this representation was to treat histogram bins as words, the histogram bins are scaled using term-frequency inverse document frequency (TF-IDF) paradigm. The best results were reported for the combination of term-frequency normalization (which is implemented by using logarithmic scale for bins' sizes in the histograms) combined with L_2 normalizations of the histograms. These transformations were implemented in the below comparison.

Despite the above representation was originally proposed for fingerprinting web sites, it can obviously be used with other definitions of messages and message sets.

B. Statistical Protocol Identification (SPID)

SPID [16] is designed to identify application protocols on basis of variable number of messages' attributes. Looking at this framework by the formalism used in this work, SPID represents a single message set as well as a model of a single protocol by a set of d empirical estimates of marginal probability densities of d features observed on the messages. A classified message set $\mathcal{R}$ is then assigned to protocol ω^* which minimizes the following Kullback-Leibler divergence averaged over the observed features:

$$\omega^* = \arg\min_{\omega} \frac{1}{d} \sum_{i=1}^{d} D_{\mathrm{KL}}(P^{\omega}_i \| P^{\mathcal{R}}_i),$$

where P^{ω}_i and $P^{\mathcal{R}}_i$ are estimates of probability density functions of the i-th feature for the protocol model ω and for the classified message set $\mathcal{R}$, respectively. Individual features are therefore considered to be independent of each other and only their marginal distributions are modelled, which is one of the biggest differences to the approach proposed in this work which models joint distributions.

C. Tunnel Hunter (TunHunter)

Tunnel Hunter [32] identifies application-layer protocols tunnelled through other protocols. Its main difference to all related methods is that it captures packets' order by modelling separately the first, the second, …, up to the N-th packet in a TCP flow. Models of protocols are implemented as joint empirical histograms of packets' sizes and inter-arrival times and they are estimated from a set of TCP flows with the same protocol label. Unlike the fingerprinting framework proposed in this work, TunHunter has stricter requirements for deployment, as the order of packets matters. The histograms are then smoothed by a Parzen window estimator (in implementation used for comparison in this article, Gaussian kernel was used for this smoothing as suggested in the original work). The classification is a variation of Naive-Bayes, where a message set $\mathcal{R} = \{m_i\}_{i=1}^{N'}$ (a TCP flow) is assigned to a protocol ω^* which minimizes the following log-likelihood:

$$\omega^* = \arg\min_{\omega} - \frac{1}{\min\{N', N\}} \sum_{i=1}^{\min\{N', N\}} \log P(m_i|\omega),$$

where N is the number of messages (packets) used to model each protocol, N' is the number of messages observed in the classified message set $\mathcal{R}$ and $P(m_i|\omega)$ is a probability of observing message $m_i \in \mathcal{R}$ in a TCP flow belonging to the protocol ω.

V. Datasets

To demonstrate the generality of the fingerprint with respect to the definition of a single message (one of the main design goals), the experiments used two sources of data described below.

A. AnyConnect Data

This dataset contains TCP flows collected during one work day in February 2015 in a corporate network of a company with 550 active users with computers running mostly Windows or OSX operating systems. To access internal network and the Internet, these users used AnyConnect VPN client [35] modified such that it exported names of the processes running on users' machines that generated the network communication (this feature is not present in the standard AnyConnect VPN client and the users were informed about the extension).

```
{"flow" :
{"an" : "APSDaemon.exe",
"t_start" : 1424178205, "t_end" : 1424178207,
"sa" : "192.168.2.105" , "sp" : 59453,
"da" : "80.239.137.24", "dp" : 80,
"user" : "john.smith",
"packets" : [
{"dir" : ">", "b" : 138, "ipt" : 0}
{"dir" : ">", "b" : 115, "ipt" : 50}
,{"dir" : "<", "b" : 268, "ipt" : 0}
,{"dir" : "<", "b" : 1336, "ipt" : 248}
] }}
```

Fig. 1. Data of a single TCP flow collected by the modified AnyConnect client (in JSON format), including the application label (the "an" field).

The modified client collected following information about each TCP flow: name of the process (and its hash), local user IP address, remote server IP address, remote server port, number of bytes and packets sent to and received from the server, and duration of the particular flow. Additionally, since the AnyConnect client sees also packets at the transport layer within each flow, it was to able log the sequence of their sizes and inter-arrival times for each flow. An example of collected data of single TCP flow is shown in Figure 1.

The data collected by the AnyConnect client thus allowed to define the message either:

- as a TCP packet at the transport layer in which case single TCP flow corresponds to one message set (this dataset is further called *Any-P*);
- or as an entire TCP flow in which case one message set corresponds to all TCP flows interchanged between the client and the server endpoint, with identifier of a message set defined as the triplet client IP, server IP, server port (this dataset is further called *Any-F*).

Difference in both definitions is outlined in Figure 2. Each message at the transport layer (in the Any-P dataset) is described by its size (in number of bytes), direction and time elapsed from its predecessor (the inter-arrival time). Similarly as in ([33]), the direction of each message is encoded by the sign of its size and inter-arrival time. In the Any-F dataset, each message is described by the number of bytes and packets sent to and received from the server, and by the duration of the flow.

Since the modified AnyConnect Client included names of clients' processes initiating the connection, this information was used as ground-truth labels for the applications' classification experiments (Section VI). After dropping flows with less than 10 packets, both datasets contain message sets belonging to 69 applications, with 198 786 flows in total. The number of flows per application varied from 550 to 9800. A complete list of application names is provided in the appendix.

B. Proxy Logs Data

The dataset contains records collected by a cloud web proxy, where each entry corresponds to one HTTP request-response pair [36]. Web proxies like,

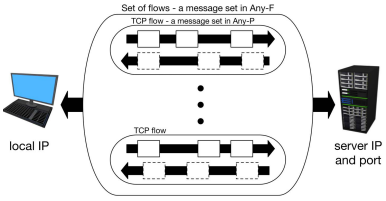


Fig. 2. Two definitions of messages in the AnyConnect data. Any-P dataset defines a message as one packet at the transport layer (solid and dashed rectangles), while a message set is defined as one TCP flow of packets. Any-F dataset defines a message as one TCP flow and a message set as a set of all TCP flows interchanged between a client and a server.

for example, Squid [1] are commonly used in corporate environments to access the World Wide Web as they allow easy deployment of security policies and they are also able to produce logs of users' communication for forensic analysis. The data were collected during one week in July, 2016 from networks of 30 international companies that use cloud web proxies to manage and secure Internet communication of their employees. This data source, though limited only to HTTP traffic, is interesting as more and more applications and malware tunnel their communication through the HTTP protocol to prevent blocking on firewalls and therefore it is increasingly more popular among security analysts. Although the visibility of communication on the application layer brings the possibility to use number of attributes that are specific for the HTTP protocol, such as URLs, referrers or user agents strings, due to the increasing popularity of HTTPs these attributes are frequently hidden. In case of encrypted communication, available attributes are typically limited to only sizes and timings of these request-response pairs. Therefore these features were used in below experiments. An example of a single datum provided by the cloud web proxy is shown in Table I.

The proposed fingerprint is used to define message set comprising of all HTTP requests exchanged between users and a particular domain (or destination IP address if domain is invisible). Message therefore corresponds to an HTTP request-response pair described by the size of the request and response (in bytes) and duration of the request-response loop. This dataset is further called *Proxy* dataset. This dataset was used in experiments presented in Section VII. These experiments used 5903 request-response pairs to 146 different servers (identified by their hostnames) of the Dropbox service, and 32000 requests-response pairs to totally 880 servers in the entire autonomous system owned by Facebook, Inc.

VI. Experimental Evaluation — Classification

A. Application Identification

In this section, the AMRep representation of message sets is compared to the methods described in Section IV on a problem of identification of application to which the observed network traffic belongs. The comparison is done at the level

[1] http://www.squid-cache.org/

Appendix 4 - Kohout, J. and Pevný, T. : *Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test* [33]

TABLE I
EXAMPLE OF FIELDS LOGGED BY A WEB PROXY FOR ONE HTTPS REQUEST-RESPONSE PAIR

timestamp	user	URL	duration	requestBytes	responseBytes	port
1467816731	j.smith	https://www.google.com/	253	120	560	443

TABLE II
AVERAGE ACCURACY OF APPLICATION IDENTIFICATION ON ANY-P DATASET (TOP HALF OF THE TABLE) AND ON ANY-F DATASET (LOWER)

Dataset	Features	MNB	SPID	TunHunter	AMRep + k-NN
Any-P	Sizes	33.43%	41.23%	—	**66.83%**
	Sizes, Inter-arrival times	—	43.59%	58.31%	**67.19%**
Any-F	Sizes	21.80%	39.67%	—	**51.94%**
	Sizes, durations	—	43.21%	—	**53.29%**
	Sizes, durations, packets	—	44.17%	—	**53.26%**

of individual TCP flows, where one message corresponds to a single TCP packet (Any-P dataset) and at the level where a message set is identified by unique tuple of client IP, server IP and server's destination port, and one message corresponds to one TCP flow (Any-F dataset). Since AMRep representation is designed to approximate MMD distance between two probability distributions, it is primarily used with a k-nearest neighbours (k-NN) classifier [37] for which a meaningful definition of distance is crucial. Also, the k-NN classifier naturally well extends to multi-class problems (recall that the dataset used in this section contains 69 different classes — the individual applications).

The k-NN classifier was taken from the PRTools toolbox for Matlab™ [38], which has advantage in having its own routine for determining the proper value of k from the training data. Other parameters of the AMRep representation (ϵ for the cover computation and γ for the width of the Gaussian kernel used in the mean map) were determined by exhaustive search over all combinations of values $\epsilon \in \{0.5, 0.6, ..., 2\}$ (Any-P dataset) and $\epsilon \in \{0.5, 0.6, ..., 5\}$ (Any-F dataset) and $\gamma \in \{2^n | n \in \{-2, -1, ..., 4\}\}$ for the kernel width values. Each combination of the parameters' values was evaluated by 5-fold cross-validation and the combination with the highest classification accuracy was used to train the final classifier which was then evaluated on a testing set. Note that the classification accuracy is defined as the ratio of the number of correctly classified messages sets to the total number of classified messages sets.

Presented experimental results are averages of ten repetitions, where each iteration used 10 000 randomly selected samples (message sets) from the entire dataset (Any-P or Any-F). This sample was used to create the training and the testing sets in the given iteration. Each iteration used 10-fold cross-validation to estimate the classification accuracy. However, since ground-truth (true labels) are notoriously difficult to obtain in network security, this scenario was simulated by using only 10% of samples available in each iteration for training in one fold of the cross-validation. Remaining 90% of samples were used to estimate the accuracy of identifying the application corresponding to testing samples.

TABLE III
AVERAGE EXECUTION TIMES OF INDIVIDUAL METHODS NEEDED FOR CLASSIFICATION OF THE TEST DATA

Method	Any-P dataset	Any-F dataset
AMRep + k-NN	0.13s	0.03s
SPID	0.46s	0.26s
MNB	0.07s	0.03s
TunHunter	4.36s	—

Average accuracy computed over all 69 applications (and all 10 iterations) is shown in Table II. According to the results, AMRep with k-NN outperforms representations and methods proposed in the related works. Moreover, the results show that statistics of packets are more informative for application identification than statistics of flows, which is expected. Interestingly, according to these experiments, inter-arrival times of packets improved the accuracy only by 0.3% on the Any-P dataset. However, paired t-test at 5% significance level does not reject the hypothesis that the accuracies achieved with and without the inter-arrival times are the same. Hence, this improvement can be considered statistically significant.

Runtimes (in seconds) of individual methods needed for computing classifications of the testing data in one fold (9000 message sets) are compared in Table III. The table presents runtimes averaged over all folds of the cross-validation on a personal computer with Intel Core-i7 CPU (4 cores, 2.2GHz) and 16GB RAM. The evaluation environment was implemented in Matlab R2016a software. The AMRep representation used with the k-NN classifier is significantly faster than SPID and TunHunter methods. The only faster method is the Multinomial Naive Bayes classification (MNB), but at the cost of notably lower accuracy (see results in Table II). The fast computation of MNB is thanks to its simplicity, as it uses only one feature (message sizes) and a Naive Bayes classifier.

Since results in Table II used only 10% of available samples for training it is interesting to study, how the accuracy improves as the number of training samples increases. This has been investigated by holding 20% of samples in each

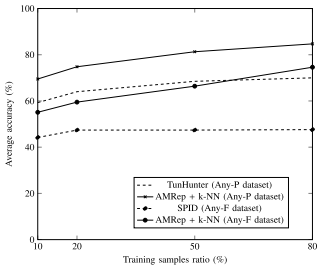


Fig. 3. Comparison of average accuracy for different methods depending on the ratio of samples used for training.

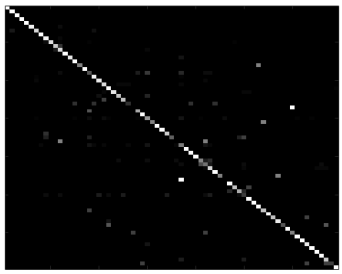


Fig. 4. Confusion matrix of AMRep on Any-P dataset for all 69 applications. Each row and each column in the figure corresponds to one application and the colour of each cell (i, j) represents the ratio of message sets of application i classified as application j. The lighter the colour of a cell, the higher the ratio which the cell represents.

iteration for testing and varying the size of the training set from 10% to 80% randomly selected from remaining 80% of samples. Resulting graphs for AMRep and the best state of the art method for each version of the dataset are shown in Figure 3. As expected, the accuracy improves as the size of the training set increases, especially for models based on AMRep. Contrary, the accuracy of SPID quickly reaches flat region when the size of the training set is 20% of all data. SPID's behaviour can be explained by histograms in its model being estimated with sufficient accuracy, therefore the approach hits its boundary and only adding new features could probably further improve the accuracy. As seen in Figure 3, the accuracy of AMRep with k-NN classifier improves as the amount of training data increases, reaching accuracy of 84.7% on Any-P dataset and 74.6% on Any-F dataset when 80% of messages sets are used for training. This makes it always the best performing classifier on given dataset.

To further discuss AMRep's performance, Figure 4 shows confusion matrix in form of a heatmap for all 69 applications on the Any-P dataset. Each row and each column in the figure corresponds to one application and the colour of each cell (i, j) represents the ratio of message sets of application i classified as application j. The lighter the colour of a cell, the higher the ratio which the cell represents. Cells on the main diagonal therefore represent accuracies achieved on individual applications while cells outside the diagonal represent mis-classifications. Due to the large number of classes, the names of individual applications are omitted in the figure. The confusion matrix shows that for the vast majority of applications the classification is very accurate, which also means that statistics of their packets are very distinct. However, there are few pairs which are frequently confused, some of them are discussed in detail below:

- `GoogleSoftwareUpdateAgent` and `ksfetch` — `ksfetch` is an OSX process involved in downloading updates for Google products, hence these two misclassified applications are related and might share the code base.
- `SoftwareUpdateCheck` and `softwareupdate` — Both of these are names of processes running on OSX systems and involved in checking for updates of software installed on the system. Hence, there is clear relation between these processes.
- `WmiPrvSE.exe` and `lsass.exe` are essential Windows processes and therefore they probably use the same libraries for network communication.
- `cma` and `DropboxOriginal` — `cma` is a part of McAfee software, while `DropboxOriginal` is part of the Dropbox file storage service. These two are likely unrelated and this is a true mis-classification.
- `helpd` and `com.apple.WebKit.Networking` — `helpd` is a process running on Apple devices which connects to on-line support of Apple and downloads the help pages. WebKit is an engine for rendering web pages used, for example, by Apple Safari browser. It is likely that the `helpd` process could use this engine too but authors were not able to verify this.
- `googledrivesync` and `netsession_win` — while the `googledrivesync` process is responsible for synchronizing Google Drive content between a user's computer and the on-line storage, the `netsession_win` process is part of Akamai NetSession Client which is, as stated by Akamai, "a tool to improve the speed, reliability, and efficiency for downloads and streams". While these two applications are not closely related, their basic behavioural patterns might be similar — both of them are likely transferring batches of larger data which explains why the classifier mismatched them.

B. Accuracy With Limited Number of Observations

The quality of representation of probability distributions depends on the number of observations. To assess how the compared classification methods are sensitive to limited number of observations, the main classification experiment described in Section VI-A was repeated while limiting the number of observed messages for each repeated communication to 5, 10, 15 and 20. The rest of the experimental settings remained the same as in the previous subsection and the classifiers used all features available for a given dataset. Classification accuracies of different methods are summarized in Figures 5a 5b for Any-P and Any-F datasets. For easier comparison of the results achieved by the version of the classifier without upper bound on the number of messages per message set (individual methods from Table II) are shown

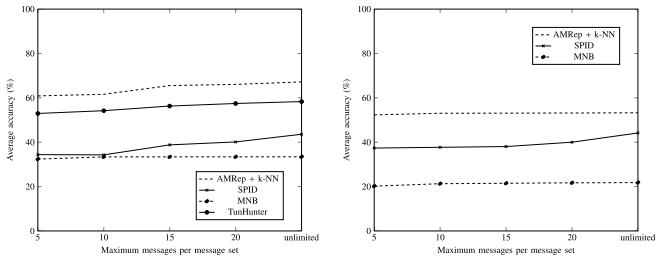


Fig. 5. Comparison of average accuracy depending on the maximal number of messages used from each message set. (a) Any-P dataset. (b) Any-F dataset.

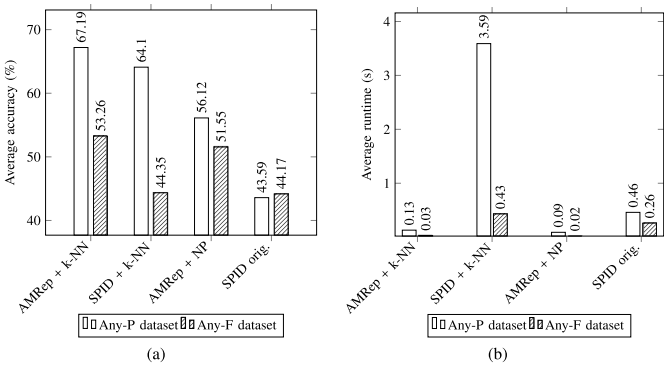


Fig. 6. Average accuracy and running times for AMRep and SPID used either with the k-NN classifier or with the nearest prototype (NP) classifier. (a) Average accuracy (higher is better). (b) Average runtime (lower is better).

in Figures 5a 5b as well under the tick "unlimited" on the x-axis. As expected with decreasing the limit on maximum messages used the accuracy of methods decreases. The only exception seems to the MNB classifier which only the messages' sizes and is therefore less prone to errors caused but insufficient number of observations (messages). However, the AMRep representation with the k-NN classifier still outperforms the other methods, despite that it uses joint distributions of all the features. This is probably because MMD metric does not need to explicitly estimate the shape of probability distributions, it can effectively operate even with very limited number of observations from each distribution. This efficiency in terms of number of observations is in fact one of the main feature of MMD.

C. Dependency on the Classifier

The k-NN classifier is for AMRep a natural choice, since it leverages the well-defined pairwise distances between classified objects. Because the classifier uses local neighborhoods of the queried objects, it can well capture multi-modal behavior of the applications (k-NN classifier is asymptotically consistent [39]). Flows belonging to the same application (e.g., Dropbox) can belong to different modes of the application's behaviour (e.g., regular polling for change notifications, files upload, filed download etc.). While flows from different modes are dissimilar to each other, behaviour within one mode is homogenous which allows the k-NN classifier to correctly classify most of the unknown flows (this is illustrated in the visualisation of the clustering of flows in the next section). However, the AMRep representation is not tightly connected with one specific classifier, as it is a general algorithm for building representations of message sets. Similarly, the SPID method can be viewed as an implementation of a nearest prototype (NP) classifier paradigm [40] with the Kullback-Leibler divergence as distance metric. The SPID method can be therefore extended to use the k-NN classifier with Kullback-Leibler divergence instead of the originally proposed NP classifier while AMRep can be used with the NP classifier by creating the applications' prototypes by merging all training messages belonging to the same application into one message set and then using the AMRep algorithm to create its representation.

This motivates the experimental comparison of these four different configurations of these two methods, namely AMRep with the k-NN classifier, SPID with the k-NN classifier, AMRep with the NP classifier and the original version of the SPID method (with the NP classifier). The method are compared in the same evaluation setup as in Section VI-A and the average classification accuracy and running times are measured. According to classification accuracies presented in Figure 6, the k-NN classifier always significantly

outperforms the NP classifier, which is likely due to the multimodality of individual classes discussed above. The unimodal behaviour implicitly assumed by the NP classifier seems to be oversimplified. Furthermore, AMRep outperforms the SPID representation regardless the used classifiers. Finally, running times of classifiers that use AMRep are considerably lower, which makes them more suitable solution, especially for real-time deployment.

VII. Experimental Evaluation — Clustering

Clustering algorithms are essential in unsupervised analysis of unlabelled data, since they reveal groups of similar objects. This section demonstrates how the AMRep representation can be used in analysis of unlabelled traffic in two different scenarios. The first scenario clusters individual flows (Any-P dataset), which can be used to identify similar applications active in a network. Results of this clustering can be used, for example, to uncover new and possibly unwanted applications in the network or as a basis of anomaly detectors that discover outlying flows that do not belong to any widely used application. The second scenario identifies groups of web servers running the same service. Knowledge of such structure can significantly improve, besides other applications, the behavioural models used in user behaviour analysis (UBA) systems ([41], [42]). The demonstrated clustering of servers can be used in tools that support work of security analysts and architects to design the UBA monitoring systems.

Since graphical representation of clustering results enables an analyst to quickly identify groups of similar objects, this work uses similarity graphs to visualise the results. In a similarity graph vertices correspond to individual objects that are clustered, while weights of edges correspond to their similarities. Since AMRep is designed such that Euclidean distance between two fingerprints approximates the MMD distance between the underlying distributions, calculating similarity with Gaussian kernel as

$$\mathrm{sim}(s_1, s_2) = \exp(-\gamma' \|s_1 - s_2\|^2), \tag{10}$$

corresponds to kernel over the space of probability distributions as introduced in [43]. $s_i = \mathbf{L}^{-1^{\mathrm{T}}}\varphi(\mu_i)$, $i \in \{1, 2\}$, in Equation 10 are AMRep fingerprints of the two compared message sets as defined in Section II and γ' is the width parameter of the Gaussian kernel on the space of fingerprints which is different from γ used to calculate s_i. The width parameter γ' can be used to change sensitivity of the similarity function, the experiments presented here used inverse of the median of squared distances between all pairs of fingerprints, which is a "rule of thumb" recommended in [44]. The similarity graph is then visualized in two-dimensional plane by means of ForceAtlas2 algorithm [45], which causes highly similar vertices to be attracted to each other, while those with low similarity are repulsed. To further improve visualization qualities of the graph, edges with weights close to zero can be omitted to make groups of similar vertices more distinct.

Finally note that the visualization algorithm and clustering were chosen to enable easy assessment by a human analyst. However, any other clustering algorithm requiring only pairwise distances, similarities, or even existence of the clustered objects in a metric space could be used. Examples include k-means, spectral clustering ([46], [47]) or the Louvain method [48], which can be even applied directly on the similarity graph and is able to optimise also the number of clusters.

TABLE IV
Example of 8 Clusters With the Highest Silhouette Score Produced by k-Means Algorithm Applied on Data Used in Section VII-A. For Each Cluster, Applications With at Least 1% Representation in the Cluster are Presented

Cluster	Application	Percentage
1	`Dropbox.exe`	50.00%
	`DropboxOriginal`	50.00%
2	`Box Sync`	100.00%
	`thunderbird.exe`	89.33%
	`vmnat.exe`	3.37%
	`Meeting Center`	3.37%
	`vmware-vmrc.exe`	2.25%
3	`ssh`	100.00%
4	`APSDaemon.exe`	99.58%
5	`Meeting Center`	55.56%
	`atmgr.exe`	44.45%
6	`com.apple.WebKit.Networking`	89.47%
	`Google Chrome`	5.26%
	`firefox.exe`	5.26%
7	`Dropbox.exe`	50.00%
	`Dropbox109`	12.50%
	`DropboxOriginal`	12.50%
	`Cisco Jabber`	12.50%
	`Google Chrome`	12.50%
8	`Cisco Jabber`	45.45%
	`Meeting Center`	18.18%
	`Cisco Webex Start`	15.15%
	`CiscoJabber.exe`	9.09%
	`com.apple.WebKit.Networking`	6.06%

A. TCP Flows Clustering

The demonstration of clustering of application flows (Any-P dataset) uses a subset of 5000 TCP flows of 69 different applications used in the previous section. The subset was selected randomly, but with the restriction that there were at least 10 flows from each application. Restricting the experiment to only 5000 samples was to enable comprehensive visualisation of results, as more samples would make the graph cluttered. The similarity graph drawn as described above is shown in Figure 7. Vertices are colored according to different applications (recall that ground truth is known in this case) to which the respective flows belong. Flows (vertices) belonging to same applications mostly form well separated clusters which asserts that AMRep representation is able to distinguish the applications, using only information about packets' sizes and their inter-arrival times. This visualisation also helps to understand why the k-NN classifier outperforms the NP classifier, as discussed in Section VI-C. For example, the application "Cisco Jabber" is spread over multiple clusters which indicates that it has multiple types of behaviour. However, each cluster is homogenous which allows the k-NN classifier to correctly classify flows from different clusters (it uses the k nearest

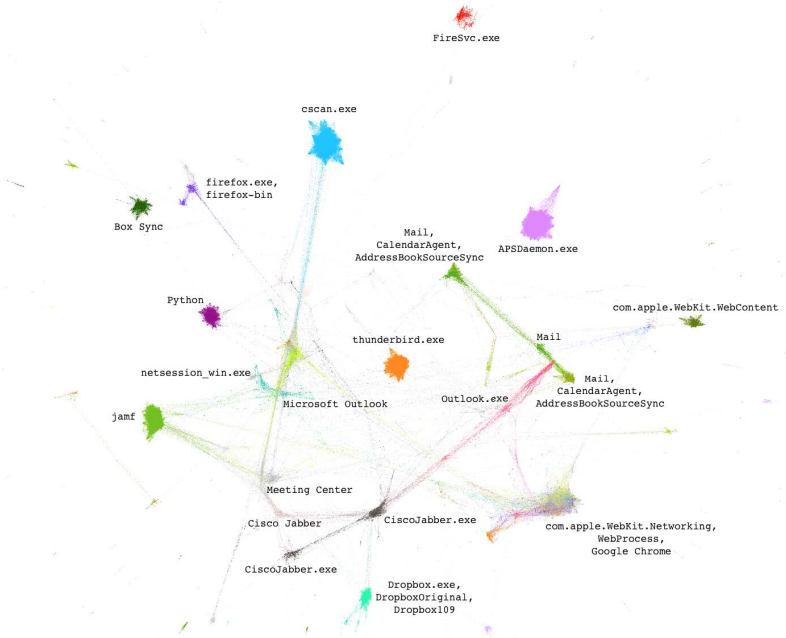


Fig. 7. Visualization of similarities of TCP flows (each treated as a message set of packets) belonging to different applications. Each vertex represents one TCP flow. The vertices are coloured according to the ground-truth application labels as logged by the AnyConnect client. Clusters of vertices containing the most prevalent applications are labelled with those applications' names.

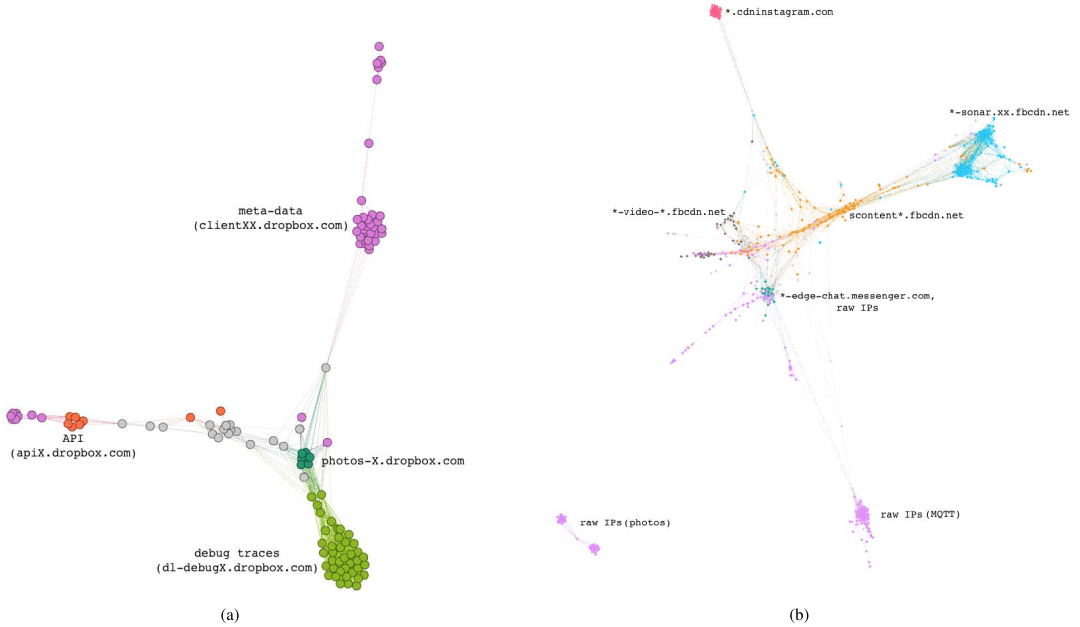
neighbours that will be likely from the same cluster). As in previous section, some clusters contain multiple applications mixed together. A closer inspection of flows of these clusters shows that they have something in common. Few larger clusters with mixed application contain:

- `com.apple.WebKit.Networking`, `WebProcess` and `Google Chrome`, which can be explained by WebKit (as already mentioned in the previous section) being a widely used rendering engine for web pages and being used as a component of many other applications including the Chrome browser.
- `Dropbox.exe`, `DropboxOriginal` and `Dropbox109` are different names of the Dropbox application running on different operating systems that appeared in the network from which the data were collected.
- `Mail`, `CalendarAgent` and `AddressBookSourceSync` are all processes running on Apple's devices responsible (as their names suggest) for a mail client operation and the Calendar application. These two applications frequently work in tandem, e.g., when a user connects to Microsoft Exchange server which manages e-mails, meeting invitations etc.

To illustrate flexibility of AMRep with respect to clustering method used, the k-means clustering algorithm was applied to the same 5000 TCP flows used in visualization in Figure 7. Implementation of the k-means clustering was provided by Matlab™'s statistics and machine learning toolbox, which determines the optimal value of k by maximizing the *silhoutte* score [49]. Since the k-means algorithm uses Euclidean distances, it directly used fingerprints as described in Section II. The algorithm returned 73 clusters, which is surprisingly close to the true number of different applications in the data, which was 69. *Homogeneity* [50] measuring purity of clusters achieved 0.67 (higher is better, with 0 being minimum and 1 being maximum which is achieved if and only if the individual clusters are identical to individual ground-truth classes, i.e., application labels). Although 0.67 might seem to be low, an inspection of applications in 8 clusters with the highest silhouette score (shown in Table IV) reveals that the real number will be higher, because of multiple applications with different labels are in fact very similar or the same. For example, Dropbox clients running on different platforms, as can be seen in Figure 7, or the cluster number 5 in Table IV containing only flows belonging to applications `Meeting Center` and `atmgr.exe` that are both related to Cisco Webex service, or the cluster number 8 which contains mixture of Cisco Jabber and Cisco Webex/Meeting Center applications that can all work integrated together to allow communication and organizing calls or teleconferences.

B. Web Servers Clustering

The second clustering scenario demonstrates how AMRep can be used to identify functional parts of a single web service accessed via HTTP(s) protocol. This scenario is interesting in



(a) (b)

Fig. 8. Visualization of similarities between fingerprints of Dropbox servers and between servers in the autonomous system belonging to Facebook, Inc.. Each vertex represents one server, the closer two vertices are, the more similar are their respective fingerprints. The vertices are coloured according to the servers' specializations. (a) Servers belonging to Dropbox. (b) Servers in the autonomous system belonging to Facebook, Inc.

cases of web services being spread over multiple servers with several groups of servers having the same purpose. An example is Dropbox — a widely used on-line file store, which has servers specialized for API calls, notifying clients about changes, serving the content, etc. For network analysts aiming at modelling users' behaviour, it is important to recognize groups of similar servers, as this knowledge can improve their models. Servers' roles can sometimes be inferred from patterns of their hostnames. However, the hostnames might not be always available in the logs as, depending on the settings of the logging proxy, the hostnames can be hidden due to encryption. This motivates the modelling of servers' behaviour based only on statistics of their communication.

In this scenario, one message set corresponds to all web communication of all clients to a single server (identifier of a message set is the server's hostname or IP, if the hostname is not available), and one message corresponds to a single HTTP(s) request-response pair described by three features: sent and received bytes and duration. AMRep framework is easily extended to model entire communication to a server, because it treats individual messages in a message set independently.

Figure 8a shows the similarity graph for Dropbox service (146 servers in total), while Figure 8b shows the similarity graph of 880 servers hosted in the entire autonomous system belonging to Facebook, Inc.. Although the exact ground-truth labelling was not available, for Dropbox service partial ground-truth exists thanks to [51] and for Facebook's autonomous system (Figure 8b) servers implementing the same sub-service were estimated from patterns of hostnames. If hostname's service was unknown or could not be clearly estimated, it is marked by grey color. The similarity graph of Facebook servers also contains multiple vertices labelled as "raw IPs" (marked by violet color). These represent servers for which the hostname was not available and only their IP addresses were logged by the proxy. As discussed earlier, this can be caused by using HTTPs or the clients contacted directly the server's IP address, without specifying the hostname. To verify that even these groups of servers represented just by IPs gather servers with similar specializations, additional analysis was performed using publicly available services including SenderBase[2] and VirusTotal.[3] These services provide information about hostnames observed on IP addresses in the Internet. This analysis revealed that the servers in separate groups indeed serve different purposes. For example, the group of raw IPs in the bottom left corner of Figure 8b gathers servers that are operated by Edge Networks Limited, a subsidiary company of Facebook, Inc. and that are used for handling photos (hostnames with patterns like `sphotos-a.xx.fbcdn.net`). Similarly, the second isolated group of raw IPs in the bottom of Figure 8b contains servers that are running MQTT protocol, which was used by Facebook as part of the Messenger service [52].

[2]http://www.senderbase.org/
[3]https://www.virustotal.com/

Appendix 4 - Kohout, J. and Pevný, T. : *Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test* [33]

Both figures nicely show how servers with different specializations are separated and those serving the similar purpose are close to each other. This demonstrates that AMRep fingerprints of two servers with the same specialization are indeed similar.

VIII. Conclusion

This work has addressed the problem of modeling and representing a set of messages of variable size exchanged between network entities as a vector in Euclidean space. The proposed representation, based on a well defined distance on spaces of probability distributions used in kernel two sample hypothesis test (known as Maximum Mean Discrepancy) is general and not restricted to security domains.

The representation was evaluated and compared to the state of the art in the supervised setting on the problem of identification of application from the observed traffic at two different levels: (i) identification of application from a set of measurements on TCP packets and (ii) identification of application from a set of flows that a client exchanged with a server. The representation was also evaluated in unsupervised problems of (i) identifying groups of flows initiated by the same applications and (ii) identifying groups of servers providing the same service (e.g., a set of servers providing content in Dropbox). The experimental results demonstrated the generality of the proposed representation and superiority to the state of the art. Moreover, the running times and computationally complexity of the proposed method is significantly lower than the prior art.

Appendix

Here follows a complete list of 69 process names logged by the AnyConnect VPN client that were used as application labels for Any-P and Any-F datasets in the classification experiments (Section VI):

```
'APSDaemon.exe', 'AddressBookSourceSync',
'AgentService.exe', 'App Store', 'AppleIEDAV.exe',
'Arellia.Agent.Service.exe', 'Box Sync',
'CalendarAgent', 'CcmExec.exe', 'CiscoJabber.exe',
'Cisco Jabber', 'Cisco.WebEx.Start', 'Dropbox109',
'DropboxOriginal', 'Dropbox.exe', 'Evernote',
'FireSvc.exe', 'GoogleSoftwareUpdateAgent',
'GoogleUpdate.exe', 'Google Chrome', 'Mail',
'Meeting Center', 'Microsoft Outlook',
'OUTLOOK.EXE', 'Python', 'Safari', 'SkyDrive.exe',
'Skype', 'SoftwareUpdateCheck', 'Spotify',
'SubmitDiagInfo', 'VpxClient.exe', 'WebProcess',
'WmiPrvSe.exe', 'apsd', 'atmgr.exe', 'chrome.exe',
'cma', 'com.apple.WebKit.Networking',
'com.apple.WebKit.WebContent',
'com.apple.iCloudHelper', 'cscan.exe',
'firefox-bin', 'firefox.exe', 'fpsaud',
'gconsync', 'googledrivesync.exe',
'helpd', 'iCloudServices.exe', 'iTunes',
'iTunes.exe', 'iexplore.exe', 'jamf',
'java', 'ksfetch', 'lsass.exe', 'mdmclient',
'mutt', 'netsession_win.exe', 'pcdrcui.exe',
'sfc.exe', 'softwareupdate', 'softwareupdated',
'splwow64.exe', 'ssh', 'storeagent', 'taskhost.exe',
'thunderbird.exe', 'vmnat.exe', 'vmware-vmrc.exe'
```

References

[1] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 512–521.

[2] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. PAM*, 2005, pp. 41–54.

[3] N. Cascarano, L. Ciminiera, and F. Risso, "Optimizing deep packet inspection for high-speed traffic analysis," *J. Netw. Syst. Manage.*, vol. 19, no. 1, pp. 7–31, 2011.

[4] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 781–789.

[5] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 229–240, Oct. 2005.

[6] W. De Donato, A. Pescapé, and A. Dainotti, "Traffic identification engine: An open platform for traffic classification," *IEEE Netw.*, vol. 1, no. 2, pp. 56–64, Mar./Apr. 2014.

[7] N. Khamphakdee, N. Benjamas, and S. Saiyod, "Improving intrusion detection system based on snort rules for network probe attack detection," in *Proc. 2nd Int. Conf. Inf. Commun. Technol. (ICoICT)*, May 2014, pp. 69–74.

[8] E. Albin and N. C. Rowe, "A realistic experimental comparison of the suricata and snort intrusion-detection systems," in *Proc. 26th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, 2012, pp. 122–127.

[9] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *J. Mach. Learn. Res.*, vol. 7, pp. 2745–2769, Dec. 2006.

[10] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, and O. Pele. (2016). "Analyzing HTTPS traffic for a robust identification of operating system, browser and application." [Online]. Available: https://arxiv.org/abs/1603.04865

[11] T. Mori, T. Inoue, A. Shimoda, K. Sato, K. Ishibashi, and S. Goto, "SFMap: Inferring services over encrypted Web flows using dynamical domain name graphs," in *Proc. Int. Workshop Traffic Monitor. Anal.*, 2015, pp. 126–139.

[12] V. Perelman, N. Melnikov, and J. Schönwälder, "Flow signatures of popular applications," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, 2011, pp. 9–16.

[13] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 5–16, 2007.

[14] H. Asai, K. Fukuda, P. Abry, P. Borgnat, and H. Esaki, "Network application profiling with traffic causality graphs," *Int. J. Netw. Manage.*, vol. 24, no. 4, pp. 289–303, 2014.

[15] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding bots in network traffic without deep packet inspection," in *Proc. 8th Int. Conf. Emerg. Netw. Experim. Technol.*, 2012, pp. 349–360.

[16] E. Hjelmvik and W. John, "Statistical protocol identification with SPID: Preliminary results," in *Proc. Swedish Nat. Comput. Netwo. Workshop*, 2009, pp. 399–410.

[17] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, 2012, pp. 129–138.

[18] L. Boero, M. Cello, M. Marchese, E. Mariconti, T. Naqash, and S. Zappatore, "Statistical fingerprint-based intrusion detection system (SF-IDS)," *Int. J. Commun. Syst.*, vol. 30, no. 10, p. e3225, 2017.

[19] G. Fedynyshyn, M. C. Chuah, and G. Tan, "Detection and classification of different botnet C&C channels," in *Proc. 8th Int. Conf. Auto. Trusted Comput.*, 2011, pp. 228–242.

[20] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 723–773, 2012.

[21] S.-H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *Int. J. Math. Models Methods Appl. Sci.*, vol. 1, no. 4, pp. 300–307, 2007.

[22] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, no. 3, pp. 337–404, 1950.

[23] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Schölkopf, and G. R. G. Lanckriet, "Hilbert space embeddings and metrics on probability measures," *J. Mach. Learn. Res.*, vol. 11, pp. 1517–1561, Jan. 2010.

[24] K. Fukumizu, A. Gretton, X. Sun, and B. Schölkopf, "Kernel measures of conditional dependence," in *Proc. NIPS*, 2008, pp. 489–496.
[25] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *J. Mach. Learn. Res.*, vol. 10, pp. 207–244, 2009.
[26] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.
[27] A. H. Cannon and L. J. Cowen, "Approximation algorithms for the class cover problem," *Ann. Math. Artif. Intell.*, vol. 40, nos. 3–4, pp. 215–223, 2004.
[28] A. Manukyan and E. Ceyhan, "Classification of imbalanced data with a geometric digraph family," *J. Mach. Learn. Res.*, vol. 17, no. 189, pp. 1–40, 2016.
[29] C. E. Priebe, D. J. Marchette, J. G. DeVinney, and D. A. Socolinsky, "Classification using class cover catch digraphs," *J. Classification*, vol. 20, no. 1, pp. 003–023, 2003.
[30] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent Botnet detection," in *Proc. 17th Conf. Secur. Symp.*, vol. 5, no. 2, pp. 139–154, 2008.
[31] M. Aiello, M. Mongelli, and G. Papaleo, "DNS tunneling detection through statistical fingerprints of protocol messages and machine learning," *Int. J. Commun. Syst.*, vol. 28, no. 14, pp. 1987–2002, 2015.
[32] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Comput. Netw.*, vol. 53, no. 1, pp. 81–97, 2009.
[33] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier," in *Proc. Workshop Cloud Comput. Secur.*, 2009, pp. 31–42.
[34] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, "No NAT'd user left behind: Fingerprinting users behind NAT from NetFlow records alone," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2014, pp. 218–227.
[35] Cisco AnyConnect Secure Mobility Client. Accessed: Feb. 2017. [Online]. Available: https://www.cisco.com/c/en/us/support/security/anyconnect-secure-mobility-client/tsd-products-support-series-home.html
[36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (1999). *Hypertext Transfer Protocol—HTTP/1.1*. [Online]. Available: https://www.ietf.org/rfc/rfc2616.txt
[37] J. Friedman, T. Hastie, and R. Tibshirani, "The elements of statistical learning," in *Springer Series in Statistics*. Berlin, Germany: Springer, 2001.
[38] R. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. de Ridder, D. M. J. Tax, and S. Verzakov. (2015). *PRTools4 A Matlab Toolbox for Pattern Recognition*. [Online]. Available: http://prtools.org
[39] L. Devroye, L. Györfi, and G. Lugosi, *Consistency of the k-Nearest Neighbor Rule*. New York, NY, USA: Springer, 1996.
[40] J. C. Bezdek and L. I. Kuncheva, "Nearest prototype classifier designs: An experimental study," *Int. J. Intell. Syst.*, vol. 16, no. 12, pp. 1445–1473, 2001.
[41] Gartner. (2015). *Market Guide for User and Entity Behavior Analytics*. ahttps://www.gartner.com/doc/3134524/market-guide-user-entity-behavior
[42] J. T. Johnson. (2015). *User Behavioral Analytics Tools Can Thwart Security Attacks*. [Online]. Available: http://searchsecurity.techtarget.com/feature/User-behavioral-analytics-tools-can-thwart-security-attacks
[43] A. Christmann and I. Steinwart, "Universal kernels on non-standard input spaces," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 406–414.
[44] B. Schölkopf and A. J. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. Cambridge, MA, USA: MIT Press, 2001.
[45] M. Jacomy, S. Heymann, T. Venturini, and M. Bastian, "ForceAtlas2, a continuous graph layout algorithm for handy network visualization," *Medialab Center Res.*, vol. 9, no. 6, p. e98679, 2011.
[46] J. A. Hartigan, *Clustering Algorithms*. New York, NY, USA: Wiley, 1975.
[47] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2001.
[48] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.*, vol. 2008, no. 10, p. P10008, 2008.
[49] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, Nov. 1987.
[50] A. Rosenberg and J. Hirschberg, "V-Measure: A conditional entropy-based external cluster evaluation measure," in *Proc. EMNLP-CoNLL*, 2007, vol. 7, pp. 410–420.
[51] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," in *Proc. Conf. Internet Meas. Conf.*, 2012, pp. 481–494.
[52] L. Zhang. *Building Facebook Messenger*. Accessed: Nov. 4, 2017. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920

Jan Kohout received the master's degree in theoretical computer science from the Faculty of Mathematics and Physics, Charles University, Prague, in 2012. He is currently pursuing the Ph.D. degree with the Czech Technical University, Prague. He is also a Researcher with Cisco Systems focusing on applications of machine learning in computer networks security.

Tomáš Pevný received the master's degree in computer science from the School of Nuclear Sciences and Physical Engineering, Czech Technical University, Prague, in 2003, and the Ph.D. degree in computer science from the State University of New York, Binghamton, in 2008. He holds a position of Researcher with the Czech Technical University, Prague. From 2008 to 2009, he held a Post-Doctoral position at Gipsa-lab, Grenoble, France. His research interests are applications of non-parametric statistics (machine learning, data modeling) with a focus on steganography, steganalysis, and intrusion detection.

Using Neural Network Formalism to Solve Multiple-Instance Problems

Tomáš Pevný[1,2] and Petr Somol[1,3]

[1]Cisco Systems, Charles Square 10, Prague 2, Czech Republic
[2]Faculty of Electrical Engineering, Czech Technical University
[3]UTIA, Czech Academy of Sciences, Prague, Czech Republic

Abstract. Many objects in the real world are difficult to describe by means of a single numerical vector of a fixed length, whereas describing them by means of a set of vectors is more natural. Therefore, *Multiple instance learning* (MIL) techniques have been constantly gaining in importance throughout the last years. MIL formalism assumes that each object (sample) is represented by a set (bag) of feature vectors (instances) of fixed length, where knowledge about objects (e.g., class label) is available on bag level but not necessarily on instance level. Many standard tools including supervised classifiers have been already adapted to MIL setting since the problem got formalized in the late nineties. In this work we propose a neural network (NN) based formalism that intuitively bridges the gap between MIL problem definition and the vast existing knowledge-base of standard models and classifiers. We show that the proposed NN formalism is effectively optimizable by a back-propagation algorithm and can reveal unknown patterns inside bags. Comparison to 14 types of classifiers from the prior art on a set of 20 publicly available benchmark datasets confirms the advantages and accuracy of the proposed solution.

1 Motivation

The constant growth of data sizes and data complexity in real world problems has increasingly put strain on traditional modeling and classification techniques. Many assumptions cease to hold; it can no longer be expected that a complete set of training data is available for training at once, models fail to reflect information in complex data unless a prohibitively high number of parameters is employed, availability of class labels for all samples can not be realistically expected, and particularly the common assumption about each sample to be represented by a fixed-size vector seems to no longer hold in many real world problems.

Multiple instance learning (MIL) techniques address some of these concerns by allowing samples to be represented by an arbitrarily large set of fixed-sized vectors instead of a single fixed-size vector. Any explicit ground truth information (e.g., class label) is assumed to be available on the (higher) level of samples but not on the (lower) level of instances. The aim is to utilize unknown patterns on instance-level to enable sample-level modeling and decision making. Note that

MIL does not address the Representation Learning problem [?]. Instead it aims at better utilization of information in cases when ground truth knowledge about a dataset may be granular and available on various levels of abstraction only.

From a practical point of view MIL promises to i) save ground truth acquisition cost – labels are needed on sample-level, i.e., on higher-level(s) of abstraction only, ii) reveal patterns on instance level based on the available sample-level ground truth information, and eventually iii) achieve high accuracy of models through better use of information present in data.

Despite significant progress in recent years, the current battery of MIL tools is still burdened with compromises. The existing models (see next Section **??** for a brief discussion) clearly leave open space for more efficient utilization of information in samples and for a clearer formalism to provide easily interpretable models with higher accuracy. The goal of this paper is to provide a clean formalism bridging the gap between the MIL problem formulation and classification techniques of neural networks (NNs). This opens the door to applying latest results in NNs to MIL problems.

2 Prior art on multi-instance problem

The pioneering work [?] coined *multiple-instance* or *multi-instance* learning as a problem where each sample b (called *bag* in the following) consists of a set of instances x, i.e., $b = \{x_i \in \mathcal{X} | i \in \{1, \ldots, |b|\}\}$, equivalently $b \in \mathcal{B} = \cup_{k>1}\{x_i \in \mathcal{X} | i \in \{1, \ldots, k\}\}$ and each instance x can be attributed a label $y_x \in \{-1, +1\}$, but these instance-level labels are not known even in the training set. The sample b is deemed positive if at least one of its instances had a positive label, i.e., label of a sample b is $y = \max_{x \in b} y_x$. Most approaches solving this definition of MIL problem belong to *instance-space paradigm*, in which the classifier is trained on the level of individual instances $f : \mathcal{X} \mapsto \{-1, +1\}$ and the label of the bag b is inferred as $\max_{x \in b} f(x)$. Examples of such methods include: Diverse-density [?], EM-DD [?], MILBoost [?], and MI-SVM [?].

Later works (see reviews [?,?]) have introduced different assumptions on relationships between labels on the instance level and labels of bags or even dropped the notion of instance-level labels and considered only labels on the level of bags, i.e., it is assumed that each bag b has a corresponding label $y \in \mathcal{Y}$, which is for simplicity assumed to be binary, i.e., $\mathcal{Y} = \{-1, +1\}$ in the following. Most approaches solving this general definition of the problem follow either the *bag-space paradigm* and define a measure of distance (or kernel) between bags [?,?,?]or the *embedded-space paradigm* and define a transformation of the bag to a fixed-size vector [?,?,?].

Prior art on neural networks for MIL problems is scarce and aimed for *instance-space paradigm*. Ref. [?] proposes a smooth approximation of the maximum pooling in the last neuron as $\frac{1}{|b|} \ln \left(\sum_{x \in b} \exp(f(x)) \right)$, where $f(x) : \mathcal{X} \mapsto \mathbb{R}$ is the output of the network before the pooling. Ref. [?] drops the requirement on smooth pooling and uses the maximum pooling function in the last neuron. Both approaches optimize the L_2 error function.

Due to space limits, the above review of the prior art was brief. The Interested reader is referred to [**?**,**?**,**?**] for a more thorough discussion of a problem and algorithms.

3 Neural network formalism

The proposed neural network formalism is intended for a general formulation of MIL problems introduced in [**?**]. It assumes a non-empty space $\mathcal{X}$ where instances live with a set of all probability distributions $\mathcal{P}^{\mathcal{X}}$ on $\mathcal{X}$. Each bag corresponds to some probability distribution $p_b \in \mathcal{P}^{\mathcal{X}}$ with its instances being realizations of random a variable with distribution p_b. Each bag b is therefore assumed to be a realization of a random variable distributed according to $P(p_b, y)$, where $y \in \mathcal{Y}$ is the bag label. During the learning process each concrete bag b is thus viewed as a realization of a random variable with probability distribution p_b that can only be inferred from a set of instances $\{x \in b | x \sim p_b\}$ observed in data. The goal is to learn a discrimination function $f : \mathcal{B} \mapsto \mathcal{Y}$, where $\mathcal{B}$ is the set of all possible realizations of distributions $p \in \mathcal{P}^{\mathcal{X}}$, i.e., $\mathcal{B} = \left\{x_i | p \in \mathcal{P}^{\mathcal{X}}, x_i \sim p, i \in \{1, \dots l\}, l \in \mathbb{N}\right\}$. This definition includes the original used in [**?**], but it also includes the general case where every instance can occur in positive and negative bags, but some instances are more frequent in one class.

The proposed formalism is based on the *embedded-space* paradigm representing bag b in an m-dimensional Euclidean space $\mathbb{R}^m$ through a set of mappings

$$(\phi_1(b), \phi_2(b), \dots, \phi_m(b)) \in \mathbb{R}^m \tag{1}$$

with $\phi : \mathcal{B} \mapsto \mathbb{R}$. Many existing methods implement embedding function as

$$\phi_i = g\left(\{k(x, \theta_i)\}_{x \in b}\right), \tag{2}$$

where $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_0^+$ is a suitably chosen distance function, $g : \cup_{k=1}^{\infty} \mathbb{R}^k \mapsto \mathbb{R}$ is the pooling function (e.g. minimum, mean or maximum), and finally $\Theta = \{\theta_i \in \mathcal{X} | i \in \{1, \dots, m\}\}$ is the dictionary with instances as items. Prior art methods differ in the choice of aggregation function g, distance function k, and finally in the selection of dictionary items, Θ. A generalization was recently proposed in [**?**] defining ϕ using a distance function (or kernel) over the bags $k : \mathcal{B} \times \mathcal{B} \mapsto \mathbb{R}$ and dictionary Θ containing bags rather instances. This generalization can be seen as a crude approximation of kernels over probability measures used in [**?**].

The computational model defined by (**??**) and (**??**) can be viewed as a neural network sketched in Figure **??**. One (or more) lower layers implement a set of distance functions $\{k(x, \theta_i)\}_{i=1}^m$ (denoted in Fig. **??** in vector form as $k(x, \theta)$) projecting each instance x_i from the bag $\{x_i\}_{i=1}^m$ from the input space $\mathbb{R}^d$ for $\mathbb{R}^m$. The pooling layer implementing the pooling function g produces a single vector $\bar{x}$ of the same dimension $\mathbb{R}^m$. Finally subsequent layers denoted in the figure as $f(\bar{x})$ implement the classifier that already uses a representation of the bag as a feature vector of fixed length m. The biggest advantage of this

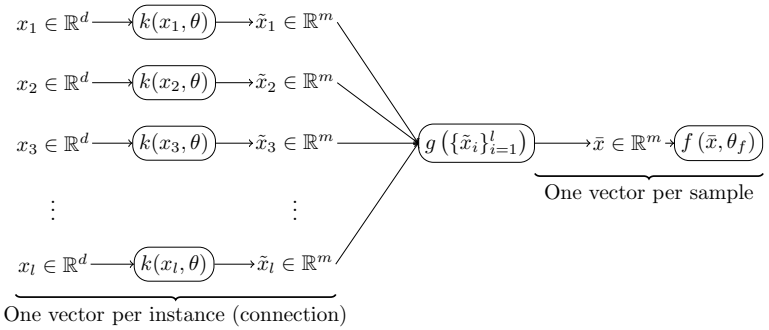


Fig. 1. Sketch of the neural network optimizing the embedding in embedding-space paradigm.

formalism is that with a right choice of pooling function $g(\cdot)$ (e.g. mean or maximum) all parameters of the embedding functions $k(x, \theta)$ can be optimized by the standard back-propagation algorithm. Therefore embedding at the instance-level (layers before pooling) is effectively optimized while requiring labels only on the bag-level. This mechanism identifies parts of the instance-space $\mathcal{X}$ with the largest differences between probability distributions generating instances in positive and negative bags with respect to the chosen pooling function. This is also the most differentiating feature of the proposed formalism to most prior art, which typically optimizes embedding parameters θ_i regardless of the labels.

The choice of a pooling function depends on the type of the MIL problem. If the bag's label depends on a single instance, as it is the case for the instance-level paradigm, then the maximum pooling function is appropriate, since its output also depends on a single instance. On the other hand if a bag's label depends on properties of all instances, then the mean pooling function is appropriate, since its output depends on all instances and therefore it characterizes the overall distribution.

Remark: the key difference of the above approach to the prior art [**?**] is in performing pooling *inside the network* as opposed to after the last neuron or layer as in the cited reference. This difference is key to the shift from instance-centric modeling in prior art to bag-centric advocated here. However the proposed formalism is general and includes [**?**] as a special case, where instances are projected into the space of dimension one ($m = 1$), pooling function g is set to maximum, and layers after the pooling functions are not present (f is equal to identity).

4 Experimental evaluation

The evaluation of the proposed formalism uses publicly available datasets from a recent study of properties of MIL problems [**?**], namely *BrownCreeper, CorelAfrican, CorelBeach, Elephant, Fox, Musk1, Musk2, Mutagenesis1, Mutagenesis2, Newsgroups1, Newsgroups2, Newsgroups3, Protein, Tiger, UCSBBreast-*

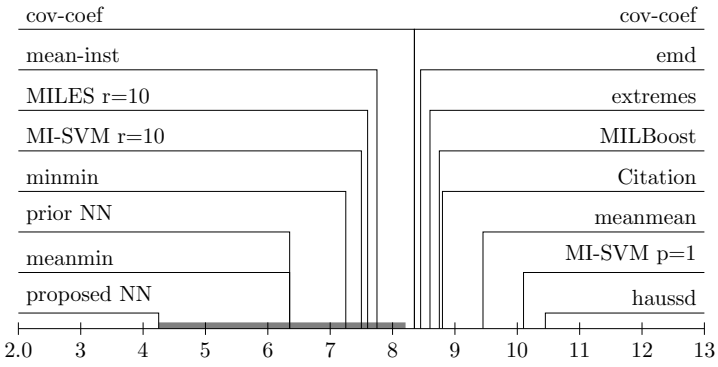


Fig. 2. Critical difference diagram shows average rank of each method over 20 problems. The thick black line shows the confidence interval of corrected Bonferroni-Dunn test with significance 0.05 testing whether two classifiers have equal performance.

Cancer, Web1, Web2, Web3, Web4, and *WinterWren*. The supplemental material [**?**] contains equal error rate (EER) of 28 MIL classifiers (and their variants) from prior art implemented in the MIL matlab toolbox [**?**] together with the exact experimental protocol and indexes of all splits in 5-times repeated 10-fold cross-validation. Therefore the experimental protocol has been exactly reproduced and results from [**?**] are used in the comparison to prior art.

The proposed formalism has been compared to those algorithms from prior art that has achieved the lowest error on at least one dataset. This selection yielded 14 classifiers for 20 test problems, which demonstrates diversity of MIL problems and difficulty to choose suitable method. Selected algorithms include representatives of instance-space paradigm: *MIL Boost* [**?**], *SimpleMIL*, *MI-SVM* [**?**] with Gaussian and polynomial kernel, and prior art in Neural Networks (denoted *prior NN*) [**?**]; bag-level paradigm: k-nearest neighbor with *citation* distance [**?**] using 5 nearest neighbors; and finally embedded-space paradigm: *Miles* [**?**] with Gaussian kernel, Bag dissimilarity [**?**] with *minmin*, *meanmin*, *meanmean*, *Hausdorff*, and Earth-moving distance (EMD), *cov-coef* [**?**] embedding bags by calculating covariances of all pairs of features over the bag, and finally *extremes* and *mean* embedding bags by using extreme and mean values of each feature over instances of the bag. All embedded space paradigm methods except Miles used a logistic regression classifier.

The proposed MIL neural network consists of a single layer of rectified linear units (ReLu) [**?**] with transfer function $\max\{0, x\}$, followed by a mean-pooling layer and a single linear output unit. The training minimized a hinge loss function using the Adam [**?**] variant of stochastic gradient descend algorithm with mini-batch of size 100, maximum of 10 000 iterations, and default settings. L1 regularization on weights of the network was used to decrease overfitting. The topology had two parameters — the number of neurons in the first layer defining the dimension of bag representation, m, and the strength of the L1 regularization, λ.

	Error of NN on		prior art	
	training set	testing set	error	algorithm
BrownCreeper	0	**5.0**	11.2	MILBoost
CorelAfrican	2.6	**5.5**	11.2	minmin
CorelBeach	0.2	**1.2**	17	extremes
Elephant	0	**13.8**	16.2	minmin
Fox	0.4	**33.7**	36.1	meanmin
Musk1	0	17.5	**12.8**	Citation
Musk2	0	**11.4**	11.8	Hausdorff
Mutagenesis1	7.5	**11.8**	16.9	cov-coef
Mutagenesis2	14.9	**10.0**	17.2	emd
Newsgroups1	0	42.5	**18.4**	meanmean
Newsgroups2	0	35	**27.5**	prior NN
Newsgroups3	0	37.5	**31.2**	meanmean
Protein	2.5	**7.5**	15.5	minmin
Tiger	0	20.0	**19**	MILES
UCSBBreastCancer	0	25	**13.6**	MI-SVM g
Web1	0	40.6	**20.9**	MILES
Web2	0	28.1	**7.1**	MI-SVM p
Web3	0	25	**13.6**	MI-SVM g
Web4	0	18.8	**1.5**	mean-inst
WinterWren	0	5.9	**2.1**	emd

Table 1. Average equal error rate of the proposed NN formalism on training and testing set and average equal error rate on the testing set of the best prior art for the given problem. Abbreviations of the prior art are as introduced in Section **??**.

Suitable parameters were found by estimating equal error rates by five-fold cross-validation (on training samples) on all combinations of $k \in \{2, 4, 8, 12, 16, 20\}$ and $\lambda \in \{10^{-7}, 10^{-6}, \ldots, 10^{-3}\}$ and using the combination achieving the lowest error. The prior art of [**?**] was implemented and optimized exactly as the proposed approach with the difference that the max pooling layer was *after* the last linear output unit.

Figure **??** summarizes results in critical difference diagram [**?**] showing the average rank of each classifier over the problems together with the confidence interval of corrected Bonferroni-Dunn test with significance 0.05 testing whether two classifiers have equal performance. The critical diagram reveals that the classifier implemented using the proposed neural net formalism (caption *proposed NN*) achieved overall the best performance, having the average rank 4.3. In fact, Table **??** shows that it provides the lowest error on nine out of 20 problems. Note that the second best, Bag dissimilarity [**?**] with minmin distance and prior art in NN [**?**], achieved the average rank 6.4 and was the best only on three and one problems respectively.

Exact values of EER of the best algorithm from the prior art and that of the proposed NN formalism is summarized in Table **??**. From the results it is obvious that the proposed neural network formalism have scored poorly on problems with a large dimension and a small number of samples, namely Newsgroups and Web

(see Table 1 of [**?**] for details on the data). The neural network formalism has easily overfit to the training data, which is supported by zero errors on the training sets.

5 Conclusion

This work has presented a generalization of neural networks to multi-instance problems. Unlike the prior art, the proposed formalism embeds samples consisting of multiple instances into vector space, enabling subsequent use with standard decision-making techniques. The key advantage of the proposed solution is that it simultaneously optimizes the classifier and the embedding. This advantage was illustrated on a set of real-world examples, comparing results to a large number of algorithms from the prior art. The proposed formalism seems to outperform the majority of standard MIL methods in terms of accuracy. It should be stressed though that results were compared to those published by authors of survey benchmarks; not all methods in referred tests may have been set in the best possible way. However, as many such cases would be very computationally expensive, the proposed formalism becomes competitive also due to its relatively modest computational complexity that does not exceed that of a standard 3-layer neural network. The proposed formalism opens up a variety of options for further development. A better and possibly more automated choice of pooling functions is one of the promising ways to improve performance on some types of data.

Acknowledgements

This work has been partially supported by Czech Science Foundation project 15-08916S.

References

1. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. arXiv preprint arXiv:1206.5538v2 (2012)
2. Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial intelligence **89**(1) (1997) 31–71
3. Maron, O., Lozano-Pérez, T.: A framework for multiple-instance learning. In Jordan, M.I., Kearns, M.J., Solla, S.A., eds.: Advances in Neural Information Processing Systems 10. MIT Press (1998) 570–576
4. Zhang, Q., Goldman, S.A.: Em-dd: An improved multiple-instance learning technique. In Dietterich, T.G., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems 14. MIT Press (2002) 1073–1080
5. Zhang, C., Platt, J.C., Viola, P.A.: Multiple instance boosting for object detection. In Weiss, Y., Schölkopf, B., Platt, J.C., eds.: Advances in Neural Information Processing Systems 18. MIT Press (2006) 1417–1424
6. Andrews, S., Tsochantaridis, I., Hofmann, T.: Support vector machines for multiple-instance learning. In Becker, S., Thrun, S., Obermayer, K., eds.: Advances in Neural Information Processing Systems 15. MIT Press (2003) 577–584

7. Amores, J.: Multiple instance classification: Review, taxonomy and comparative study. Artificial Intelligence **201** (2013) 81–105
8. Foulds, J., Frank, E.: A review of multi-instance learning assumptions. The Knowledge Engineering Review **25**(01) (2010) 1–25
9. Haussler, D.: Convolution kernels on discrete structures. Technical report, Universityof California at Santa Cruz (1999)
10. Muandet, K., Fukumizu, K., Dinuzzo, F., Schölkopf, B.: Learning from distributions via support measure machines. In: Advances in neural information processing systems. (2012) 10–18
11. Gärtner, T., Flach, P.A., Kowalczyk, A., Smola, A.J.: Multi-instance kernels. In: ICML. Volume 2. (2002) 179–186
12. Wang, J., Zucker, J.D.: Solving the multiple-instance problem: A lazy learning approach. In: Proceedings of the Seventeenth International Conference on Machine Learning. ICML '00, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2000) 1119–1126
13. Cheplygina, V., Tax, D.M., Loog, M.: Multiple instance learning with bag dissimilarities. Pattern Recognition **48**(1) (2015) 264 – 275
14. Chen, Y., Bi, J., Wang, J.Z.: Miles: Multiple-instance learning via embedded instance selection. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(12) (Dec 2006) 1931–1947
15. Ramon, J., De Raedt, L.: Multi instance neural networks. (2000)
16. Zhou, Z.h., Zhang, M.l.: Neural networks for multi-instance learning. In: Proceedings of the international conference on intelligent information technology. Volume 182., Citeseer (2002)
17. Carbonneau, M.A., Cheplygina, V., Granger, E., Gagnon, G.: Multiple instance learning: A survey of problem characteristics and applications. arXiv preprint arXiv:1612.03365 (2016)
18. Cheplygina, V., Tax, D.M.J.: Characterizing Multiple Instance Datasets. In: Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings. Springer International Publishing, Cham (2015) 15–27
19. Cheplygina, V., Tax, D.M.J., Loog, M.: Supplemental documents to characterizing multiple instance datasets
20. Tax, D.M.J., C.V.: MIL, a Matlab toolbox for multiple instance learning (Jun 2016) version 1.2.1.
21. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition? In: Computer Vision, 2009 IEEE 12th International Conference on. (Sept 2009) 2146–2153
22. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research **7** (2006) 1–30
24. Cheplygina, V., Tax, D.M.J. In: Characterizing Multiple Instance Datasets. Springer International Publishing, Cham (2015) 15–27

Discriminative Models for Multi-instance Problems with Tree Structure

Tomáš Pevný
CTU in Prague
Cisco R&D Center in Prague
Prague, Czech Republic
pevnak@gmail.com

Petr Somol
Cisco R&D Center in Prague
UTIA, Czech Academy of Sciences
Prague, Czech Republic
psomol@cisco.com

ABSTRACT

Modelling network traffic is gaining importance to counter modern security threats of ever increasing sophistication. It is though surprisingly difficult and costly to construct reliable classifiers on top of telemetry data due to the variety and complexity of signals that no human can manage to interpret in full. Obtaining training data with sufficiently large and variable body of labels can thus be seen as a prohibitive problem. The goal of this work is to detect infected computers by observing their HTTP(S) traffic collected from network sensors, which are typically proxy servers or network firewalls, while relying on only minimal human input in the model training phase. We propose a discriminative model that makes decisions based on a computer's all traffic observed during a predefined time window (5 minutes in our case). The model is trained on traffic samples collected over equally-sized time windows for a large number of computers, where the only labels needed are (human) verdicts about the computer as a whole (presumed infected vs. presumed clean). As part of training, the model itself learns discriminative patterns in traffic targeted to individual servers and constructs the final high-level classifier on top of them. We show the classifier to perform with very high precision, and demonstrate that the learned traffic patterns can be interpreted as Indicators of Compromise. We implement the discriminative model as a neural network with special structure reflecting two stacked multi-instance problems. The main advantages of the proposed configuration include not only improved accuracy and ability to learn from gross labels, but also automatic learning of server types (together with their detectors) that are typically visited by infected computers.

Keywords

Neural network; user modeling; malware detection; big data; learning indicators of compromise

AISec'16, October 28 2016, Vienna, Austria

ISBN 978-1-4503-4573-6/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2996758.2996761

1. MOTIVATION

In network security it is increasingly more difficult to react to the influx of new malicious programs such as trojans, viruses and others (further called malware). Traditional defense solutions rely on identifying pre-specified patterns (called signatures) known to distinguish malware in incoming network connections, e-mails, locally stored programs, etc. But signature-matching now runs out of breath with the rapid increase in malware sophistication. Contemporary malware deploys many evasion techniques such as polymorphism, encryption, obfuscation, randomization, etc., which critically decrease recall of signature-based methods. One of possible perpendicular approaches is identifying infected computers on the basis of their behavior, i.e., usually by monitoring and evaluating network activity or system calls. The advantage of such an approach is higher recall, because it is much harder to evade behavior-based detection. For example, computers infected by spamming malware almost inevitably display an increase in the number of sent e-mails. Click-fraud, where infected computers earn money to the originator of the infection by showing or accessing advertisements, is another example where the increased volume of certain traffic is a good indicator of compromise. On the other hand, behavior-based malware detection frequently suffers from higher false positive rates compared to signature based solutions.

Machine learning methods have recently attracted attention due to their promise to improve false-positive rates of behavioral malware detection[2]. However, the use of off-the-shelf machine learning methods to detect malware is typically hindered by the difficulty of obtaining accurate labels, especially if classification is to be done at the level of individual network connections (TCP flow, HTTP request, etc.)[11, 13]. Even for an experienced security analyst it is almost impossible to determine which network connections are initiated by malware and which by a benign user or application,[1] since malware often mimics the behavior of benign connections. We have observed malware connecting to `google.com` for seemingly benign connection checks, displaying advertisements, or sending e-mail as mentioned above. Labeling individual network connections is thus pro-

[1]Even though one has access to the machine infected by malware and can obtain hashes of processes issuing connections, malicious browser plugins will have the hash of the browser, which is a legitimate application, which renders this technique useless. Also, the database of hashes used to identify malware processes might not be complete, resulting in incomplete labeling.

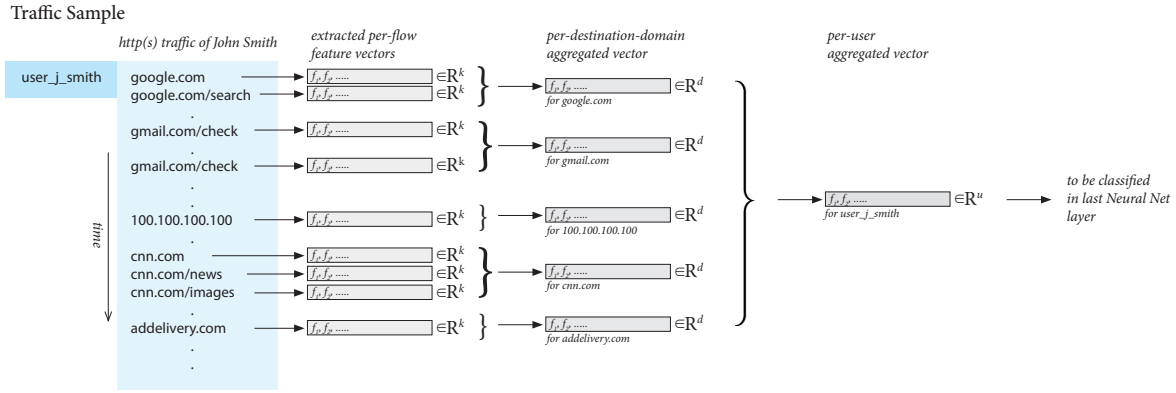


Figure 1: Sketch of the traffic of a single computer.

hibitive not only due to their huge numbers but also due to ambiguity in individual connections' classification. Automatic and large-scale training of accurate classifiers is thus very difficult.

In this work we sidestep this problem by moving the object of classification one level up, i.e., instead of classifying individual connections we classify the computer (represented by a collection of all its traffic) as a whole. The immediate benefit is twofold. First, the labeling is much simpler, as it is sufficient to say "this computer is infected / clean" rather than "this connection has been caused by malware". Second, a grouping of connections provides less ambiguous evidence than a single connection (see cases described above where a single access to an ad server does not tell much, but a multitude of such accesses does). This latter property is in fact the main motivation behind our present work.

The biggest obstacle in implementing a classifier on basis of all observed traffic is the variability in the number of network connections (hereafter called *flows*). This property effectively rules out the majority of machine learning algorithms requiring each sample to be described by a fixed dimensional vector, because the number of observed flows supposed to characterize one computer can range from dozens to millions while information content of the flows may vary significantly. Our problem thus belongs to the family of multi-instance learning (MIL) problems [3, 7] where one sample is commonly called a *bag* (in our case representing a computer) and consists of a variable number of *instances* (in our case one instance is one flow), each described by a fixed dimensional vector.

The solution proposed below differs from the current MIL paradigm by taking a step further and representing data not as a collection of bags, but as a hierarchy of bags. We show that such approach is highly advantageous as it effectively utilizes the natural hierarchy inherent in our data. Flows emitted or observed by one computer can be easily grouped according to servers they connect to (these groups are called *sub-bags)*, so that the *bag* representing the particular computer becomes a collection of *sub-bags*. This hierarchy can be viewed as a tree with leafs representing flows (instances), inner nodes representing servers (sub-bags), and finally the root representing the computer (bag). The structure of the problem is shown in Figure 1. Note that trees representing different computers will have different number of inner nodes and leafs. The proposed classifier exploits this structure by first modeling servers (sub-bags) on the basis of flows targeted to them and then modeling the computer on top of the server models. This approach can be viewed as two MIL problems stacked one on top of the other. In Section 3 we show how the hierarchical MIL problem can be mapped into a neural-network architecture, enabling direct use of standard back-propagation as well as many recent developments in the field of deep learning. Once trained, the architecture can be used for classification but it can also be decomposed to identify types of traffic significant for distinguishing benign from infected computers, i.e., it allows to extract learned indicators of compromise (IOCs). Finally, using an approach similar to URCA [17], it is possible to identify particular connections which made the neural network decide that the computer is infected; hence effectively providing an explanation of the learned IOC.

Section 4 demonstrates the proposed approach on a large scale real-world problem of detecting infected computers from proxy logs. It is shown that the neural network can learn to identify infected computers in computer networks, as well as provide sound explanations of its verdicts to the consumer. Neurons in lower layers are shown to have learned weak indicators of compromise typical for malware.

The proposed neural network architecture is shown to have multiple advantageous properties. Its hierarchal MIL nature dramatically reduces the cost of label acquisition. By using labels on high-level entities such as computers or other network devices the creation of training data is much simpler. The ability to decompose the encoded structure is no less important as it provides a definition of learned indicators of compromise. Finally, it allows for human-intelligible explanations of classifier verdicts as security incidents, which simplifies the job of the network administrator.

This paper is organized as follows. The next section formulates the problem of multiple instance learning and reviews important work we build upon. The proposed approach is presented in Section 3. Experimental evaluation is provided in Section 4.

2. RELATED WORK

Here we review the evolution of paradigms leading to the solution proposed in the next section.

2.1 Multi instance learning problem

The pioneering work [6] coined *multiple-instance* or *multi-instance* learning as a problem, where each sample b (to be referred to as *bag* in the following) consists of a set of instances x, i.e., $b = \{x_i \in \mathcal{X} | i \in \{1, \ldots, |b|\}\}$. Each instance x can be attributed a label $y_x \in \{-1, +1\}$, but these instance-level labels are not assumed to be known even in the training set. The sample b is deemed positive if at least one of its instances has a positive label, i.e., label of a sample b is $y = \max_{x \in b} y_x$. For this scenario the prevalent approach is the so-called *instance-space paradigm*, i.e., to train a classifier on the level of individual instances $f : \mathcal{X} \mapsto \{-1, +1\}$ and then infer the label of the bag b as $\max_{x \in b} f(x)$.

2.1.1 Embedded-Space Paradigm

Later works (see reviews [3, 7]) have introduced different assumptions on relationships between the labels on the instance level and labels of bags or even dropped the notion of instance-level labels and considered only labels at the level of bags, i.e., it is assumed that each bag b has a corresponding label $y \in \mathcal{Y}$, which for simplicity we will assume to be binary, i.e., $\mathcal{Y} = \{-1, +1\}$ in the following. The common approach of the latter type is either to follow a *bag-space paradigm* and define a measure of distance (or kernel) between bags, or to follow an *embedded-space paradigm* and define a transformation of the bag to a fixed-size vector.

Since the solution presented in Section 3 belongs to the embedded-space paradigm, we describe this class of methods in necessary detail and adopt the formalism of [16], which is suitable for presenting our solution. The formalism of [16] is intended for a general formulation of MIL problems, where labels are assumed only at the level of bags without any labels at the level of instances. Each bag b consists of a set of instances, which are viewed as a realization of some probability distribution p_b defined over the instance space $\mathcal{X}$. To allow more flexibility between bags even within the same class, the formalism assumes that probability distributions p_b of different bags are different, which is captured as p_b being realization of a probability $P(p_b, y)$, where $y \in \mathcal{Y}$ is the bag label.

During the learning process each concrete bag b is thus viewed as a realization of an unknown probability distribution p_b that can be inferred only from groups of instances $\{x \in b | x \sim p_b\}$ observed in data. The goal is to learn a discrimination function $f : \mathcal{B} \mapsto \mathcal{Y}$, where $\mathcal{B}$ is the set of all possible realizations of all distributions $p \in \mathcal{P}^{\mathcal{X}}$, i.e., $\mathcal{B} = \left\{x_i | p \in \mathcal{P}^{\mathcal{X}}, x_i \sim p, i \in \{1, \ldots l\}, l \in \mathbb{N}\right\}$. Note that this definition also subsumes the one used in [6].[2]

Methods from *embedded space-paradigm* [3, 7] first represent each bag b as a fixed-dimensional vector and then use any machine learning algorithm with samples of fixed dimension. Therefore the most important component in which most methods differ is the embedding. An embedding of bag b can be generally written as

$$(\phi_1(b), \phi_2(b), \ldots, \phi_m(b)) \in \mathbb{R}^m \tag{1}$$

with individual projection $\phi_i : \mathcal{B} \mapsto \mathbb{R}$ being

$$\phi_i = g\left(\{k(x, \theta_i)\}_{x \in b}\right), \tag{2}$$

where $k : \mathcal{X} \times \Theta \mapsto \mathbb{R}_0^+$ is a suitably chosen distance function parametrized by parameters θ (also called dictionary items) and $g : \cup_{n=1}^{\infty} \mathbb{R}^k \mapsto \mathbb{R}$ is the pooling function (e.g. minimum, mean or maximum). Most methods differ in the choice of aggregation function g, distance function k, and finally in the selection of dictionary items $\theta \in \Theta$.

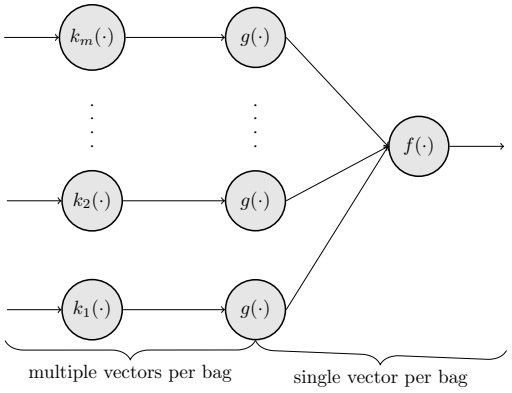


Figure 2: Neural network optimizing embedding in embedding-space paradigm.

2.2 Simultaneous Optimization of Embedding and Classifier

The important novelty introduced in [16] is that embedding functions $\{\phi_i\}_{i=1}^m$ are optimized simultaneously with the classifier that uses them, as opposed to prior art where the two optimization problems are treated indepedently. Simultaneous optimization is achieved by using the formalism of neural network, where one (or more) lower layers followed by a pooling layer implement the embedding function ϕ, and subsequent layers implement the classifier that is thus built on top of bag representations in the form of feature vectors of fixed length. The model is sketched in Figure 2 with a single output neuron implementing a linear classifier once the embedding to a fixed-length feature representation is realized. The neural network formalism enables to optimize individual components of the embedding function as follows.

- Lower layers (denoted in Figure 2 as $\{k_i\}_{i=1}^m$) before pooling identifies parts of the instance-space $\mathcal{X}$ where the probability distributions generating instances in positive and negative bags differ the most with respect to the chosen pooling operator.
- The pooling function g can be either fixed, such as mean or maximum, or any other pooling function for which it is possible to calculate gradient with respect to its inputs. The pooling function itself can have parameters that can be optimized during learning, as was

[2] Ref. [6] assumed labels on instances and a bag was classified as positive if it contained at least one positive instance. In the used general formulation this corresponds to the case, where in each positive bag exist instances that never occur in negative bags, which means that the difference of support of positive and negative probability distributions is non-empty, i.e., $p_+ \backslash p_- \neq \emptyset$, where $p_+ \sim P(p|+)$ and $p_- \sim P(p|-)$.

shown e.g. in [9], where the pooling function has the form $\sqrt[q]{\frac{1}{|b|}\sum_{i\in b}|x_i|^q}$ with the parameter q being optimized.

- Layers after the pooling (denoted in Figure 2 as $f(\cdot)$) correspond to the classifier that already uses the representation of bags as vectors of fixed dimension.

The above model is very general and allows for automatic optimization of all its parameters by means of back-propagation, though the user still needs to select the number of layers, number of neurons in each layer, transfer functions, and possibly also the pooling function.

3. THE PROPOSED SOLUTION

In the light of the previous section, the problem of identifying infected computers can be viewed as two MIL problems, one stacked on top of the other, where the traffic of a computer b is generated by a two-level generative model.

3.1 Generative Model

Let us denote $\mathcal{S}$ the set of all servers accessible by any computer. Let $\mathcal{S}_c \subseteq \mathcal{S}$ denote the subset of all servers accessed from computer c in a given time frame. The communication of computer c with each server $s \in \mathcal{S}_c$ consists of a group of flows $x \in \mathcal{X}$ that are viewed as instances forming a *first-level bag* b_s. The bag of flows b_s is thus viewed as a realization of some probability distribution $p_{b_s} \in \mathcal{P}^{\mathcal{X}}$.

We imagine that every server s is associated with a type $t(s)$, which influences the probability distribution of the flows p_{b_s}. Accordingly, each first-level bag b_s is realized according to p_{b_s}, which itself is a realization of a probability distribution $P(p_{b_s}, t(s))$. This captures the real-world phenomenon of a user's interaction with some server (e.g., e-mail server) being different from that of a different user communicating with the same server, as well as the fact that different types of servers impose different communication patterns.

In view of the above we can now consider computer c to be a *second-level bag* consisting of a group of first-level bags b_s. Similarly as above, we assume c to be a realization of probability distribution $p_c \in \mathcal{P}^{\mathcal{B}}$, where $\mathcal{B}$ is the set of all possible realizations of all distributions $p \in \mathcal{P}^{\mathcal{X}}$. The probability distribution p_c is expected to be different for each computer, in particular we assume this to be true between infected and clean computers labeled by $y \in \{-1, +1\}$. The probability distribution p_c is thus viewed as a realization of a probability distribution $P(p_c, y)$. This captures the real-world observation that infected computers exhibit differences in communication patterns to servers, both in what servers they access and within individual connections to a server.

The model imposes a generative process as illustrated in Algorithm 1.

The proposed multi-level generative model opens up possibilities to model patterns at the level of individual connections to server as well as at the level of multiple servers' usage. In the following we discuss the implementation and show the practical advantages on large-scale experiments.

input : $y \in \{-1, +1\}$ label marking computer c as clear or infected
output: Set of flows $\mathcal{F}$ of one computer
1. sample a distribution p_c of servers from $P(p_c, y)$;
2. sample a set of servers $\mathcal{S}_c$ from p_c;
3. $\mathcal{F} = \emptyset$;
foreach $s \in \mathcal{S}_c$ **do** %iterate over selected
   4. sample distribution p_{b_s} of flows from $P(p_{b_s}, t(s))$;
   5. sample flows x from p_{b_s};
   6. add sampled flows to all flows, $\mathcal{F} = \mathcal{F} \cup \{x_i\}$;
end

Algorithm 1: Generative model of the flows of one computer.

3.2 Discriminative model

The rationale behind the discriminative model closely follows the above generative model by breaking the problem into two parts: classifying the computer based on the types of contacted servers and classifying the type of a server based on flows exchanged between the server and the client.

Let's assume that each contacted server is described by a feature vector of a fixed dimension, which can be as simple as one-hot encoding of its type $t(s)$. Then the problem of classifying the computer becomes a MIL problem with the bag being the computer and instances being the servers. The problem is of course that types of servers $t(s)$ are generally unknown and we cannot imagine to manually create a mapping between a server IP or domain name and a server type. To make the problem even more difficult, the same server can be used differently by different computers, and therefore it can be of different type for each of them. One can indeed learn a classifier that would predict the server type from flows between the computer and the server, which again corresponds to a MIL classifier with the bag being the server and instances being the flows, but the problem of obtaining labeled samples for training the classifier is non-trivial and it is unlikely that we will have known all types of servers. Moreover, since we are learning a discriminative model, we are interested in types of servers occurring with different probabilities in clean and infected computers.

To side step this problem we propose to stack a MIL classifier at the level of computers on top of a MIL classifier at the level of servers. Since both MIL classifiers are realized by a neural network described in the previous chapter, we obtain one (larger) neural network with all parameters optimizable using standard back-propagation and importantly using labels only at the level of bags (computers). This effectively removes the need to know types of servers $t(s)$ or learn classifier for them, because the network learns that automatically from the labels on the level of computers. The caveat is that the network learns only types of servers that occur with different probabilities in clean and infected computers.

Figure 3illustrates the idea in its simplest incarnation. The distinctive feature is the presence of two pooling layers reflecting two MIL problems dividing the network into three parts. The first part part up to the first pooling included implements the embedding of sub-bags into a finite-dimensional vector (modeling servers based on flows). After the first pooling each sub-bag (server) is represented by one finite-dimensional vector. Similarly the second part starting after the first pooling up to the second pooling included embeds sub-bags into a finite dimensional vector characterizing each bag (computer). Finally, the third part after the second pooling implements the final classifier.

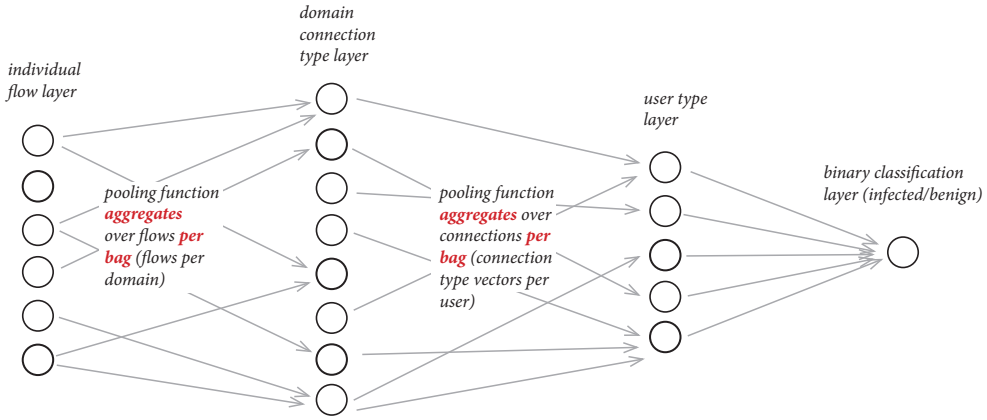


Figure 3: Hierarchical MIL

The right choice of the pooling function is not straightforward as there are many aspects to be taken into the consideration.

- Mean function should be theoretically better than max [12], since it is more general. The advantage of mean pooling function has been experimental demonstrated in [16].
- If malware performs only a few distinct types of connections (e.g. connection checks) to well known servers, max functions can identify them whereas mean function might suppress them among the clutter caused by many connections of legitimate applications. This problem has been recently studied in [4] in the context of natural images.
- The number of contacted servers and flows to servers varies between computers and max pooling is more stable then mean.
- The training with max pooling is approximately six times faster, since the back-propagated gradient is nonzero only for one element entering the pooling operation (one flow per server and neuron, one server per computer and neuron).

3.3 Extracting indicators of compromise

The presented model is based on the assumption that there exist types of servers contacted with different probability by infected and clean computers, though one generally does not know much about them. If these types did not exist, then the probability distributions p_c of infected and clean computers would be the same and it would be impossible to create a reliable detector for them. But if the neural network has learned to recognize them, vector representations of servers (output of the network's first part from the input to the first pooling included in Figure 3) have to have different probability distributions for clean and infected computers.

Since the above line of reasoning can be extended to the output of the layer just before the first pooling function, output of each neuron of this layer can be viewed as an indicator of compromise, since it has to contribute to the identification of infected computers. From a close inspection of flows on which these neurons provide the highest output a skilled network analyst can figure out what kind of traffic it is (concrete examples are shown in Section 4.2). Admittedly, these learned IOCs would deliver poor performance if used alone. But in the neural network they are used together with IOCs from different servers, which provide context contributing to good accuracy. Also, once a network administrator annotates these neurons, this annotation can be used to provide more detailed information about the decisions.

3.4 Explaining the decision

Neural networks have a reputation being a black-box in the sense that they do not provide any details about their decisions. In intrusion detection this behavior is undesirable, since the investigation of a possible security incident would have to start from the very beginning. Therefore providing the analyst with an explanation why the classifier viewed the computer as infected is of great help.

The explanation method relies on the assumption that flows caused by the infection are additive, i.e. the malware does not block user's flows but only adds its own. This means that if the computer was deemed infected, by removing the right flows (instances) the network should flip its decision. Although finding the smallest number of such flows is likely an NP complete problem, a greedy approximation inspired by [17] performs surprisingly well.

The greedy approximation finds in each iteration a set of flows going to same server (subbag), that causes the biggest decrease of the classifier's output when removed from a computer's traffic (in our implementation positive means infected). The algorithm stops when the classifier's output becomes negative (clean). The set of all removed subbags is returned as the explanation in the form: "This computer was found infected because it has communicated with these domains". If decired, examples of flows to these domains can be obviously supplied.

3.5 Computational complexity

The computational complexity is important not only for

the training, but also for the deployment as the amount of network traffic that needs to be processed can be high. For example Cisco's Cognitive Threat Analytics [5] processes 10^{10} proxy logs per day. The hierarchical aggregation inside the network decreases substantially the computational complexity, since after the first pooling, the network produces a single vector per server instead of one vector per flow yielding up to six fold decrease of the data to be processed. Similarly, after the second pooling the computer is described just by a single vector instead of a set of vectors, which again decreases the complexity. Compare this to the prior art on solving MIL with neural networks [18], where the pooling is done after the last linear layer just before the output, which means that all layers of the network have to process all flows. The effect on the computational complexity is tremendous. Whereas our approach takes approximately five seconds per 100 iterations of the training, the prior art of [18] takes 1100 seconds, which is 220 times slower.

4. EXPERIMENTAL EVALUATION

Albeit the proposed solution is general and can be used for any kind of network traffic, it has been evaluated in the context of detecting infected computers from logs of web proxies due to the availability of large data to us. Besides, proxy logs are nicer for human investigation than for example netflow data. The proxy logs were collected by Cisco's Cognitive Threat Analytics [5] from 500 large networks during eight days. The days were picked randomly from the period from November 2015 till February 2016 with the testing day being 7th March 2016. Since the total number of infected computers in the dataset from seven training days was small, we have added data of infected computers from additional 25 days from the period spanning the training data.

Since the data were collected in five-minute time windows, one bag consists of all web requests of one computer during that window. Computers were identified either by source IP address or by the user name provided in the proxy logs. Subbags contain requests with the same part in the HTTP request.

Computers (bags) were labeled using Cisco's Cognitive Threat Analytics [5] so that if one computer had at least one request known to be caused by malware, the computer was considered to be infected in that five-minute window. If the same computer in some different time window did not have any malware flows, the bag from that time window was considered as clean.

The training set contained data from approximately 20 million unique computers out of which 172 013 were infected and approximately 850 000 000 flows, out of which 50 000 000 belonged to infected computers. The testing set contained data of approximately 3 000 000 computers out of which 3 000 were infected and approximately 120 000 000 flows with 500 000 flows belonging to the infecting computers.

We are certain that the labeling we have used in this experiment is far from being perfect. While there will be a relatively small number of infected computers labeled as clean, there will be quite a lot of computers labeled as clean that were in fact infected. Despite these issues, we consider this labeling as ground truth, because the aim of the experiments is to demonstrate that the proposed solution can learn from high-level labels and identify weak indicators of compromise.

The experiments were implemented in author's own library, since popular libraries for neural networks are not designed for MIL problems. They do not allow to have samples (bags and sub-bags) of different sizes (number of instances) which makes the encoding of the hierarchical structure impossible. Therefore evaluated architectures used simple building blocks: rectified linear units [8, 12], mean and maximum pooling functions, and ADAM optimization algorithm [10]. Unless said otherwise, ADAM was used with default parameters with the gradient estimated in each iteration from 1000 legitimate and 1000 infected computers (bags) sampled randomly. This size of the minibatch is higher than is used in most prior art about deep learning, however we have found it beneficial most probably because the signal to be detected is weaker. Contrary to most state of the art, we have used weighted Hinge loss function $\max\{0, 1 - y \cdot w^y \cdot f(x)\}$ with w^+ being the cost of (false negative) missed detection and w^- being the cost of false positive (false alarms). The rationale behind Hinge loss is that it produces zero gradients if sample (bag) is classified correctly with sufficient margin. This means that gradient with respect to all network parameters is zero, therefore the back-propagation does not need to be performed, which leads to a considerable speed-up. The learning was stopped after ADAM has performed $3 \cdot 10^5$ iterations.

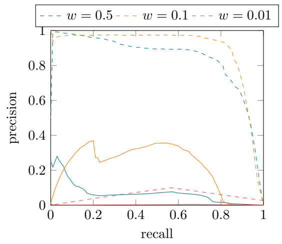
The performance was measured using precision-recall curve (PR curve) [14] popular in document classification and information retrieval as it is better suited for highly imbalanced problems, into which intrusion detection belongs (in the testing data there is approximately one infected computer per one thousand clean ones).

4.1 Network architecture

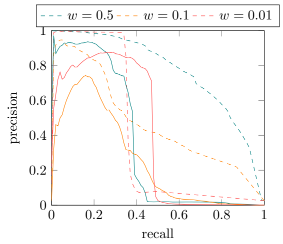
All evaluated neural networks used simple feature vectors (instances) with 34 cheap to compute statistics, such as length of the url, query and path parts, frequency of vowels and consonants, HTTP status, port of the client and the server, etc, but not a single feature was extracted from the hostname. Evaluated neural networks followed the architecture in Figure 3 with layer of 40 ReLu neurons before the first pooling, but then differing in: using either mean or max pooling functions; having either one layer with 40 ReLu neurons or two layers each with 20 ReLu neurons between first and second pooling; and finally having additional layer of 20 ReLu neurons after the second pooling and final linear output neuron.

Precision-recall curves of all six evaluated neural networks each trained with three different costs of errors on false positives (0.5, 0.9, 0.99) and false negative (0.5, 0.1, 0.01) are shown in Figure 4. Based on these experiments, we have made the following conclusions.

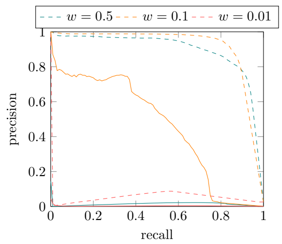
- Simpler networks with max pooling function tend to overfit, as the error on the training set of all three evaluated architectures is very good (dashed lines) but the error on the testing set is considerably worse (solid lines). We believe this to be caused by the network to act more like a complicated signature detector by learning specific patterns in flows prevalent in the infected computers in the training set, but missing in infected computers in testing set. This hypothesis is supported by (i) the fact that when we have been creating ground truth, we have labeled computer as infected if it had at least one connection known to be caused by



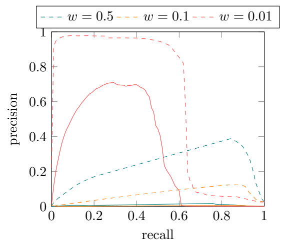
(a) relu-max-relu-max-lin



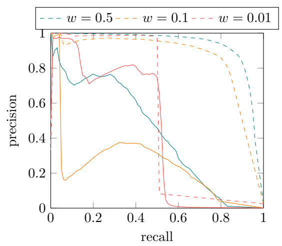
(b) relu-mean-relu-mean-lin



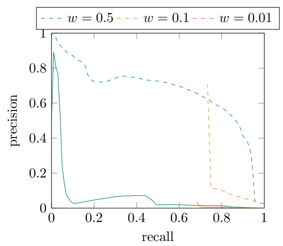
(c) relu-max-relu-max-relu-lin



(d) relu-mean-relu-mean-relu



(e) relu-max-relu-relu-max-relu-lin



(f) relu-mean-relu-relu-mean-relu-lin

Figure 4: Precision-recall curves of six neural network architectures utilizing simple 34 features. Dashed lines show the curves estimated on the training set and solid lines show the curves estimated from the testing set. Networks with PR curves in the left column used max pooling function, whereas those with PR curves in the right column used mean pooling function. Captions $w = 0.5$, $w = 0.1$, and $w = 0.01$ correspond to different costs in weighted hinge loss with cost on false positives (false alarms) being $w^- = 1 - w$ while that on the false negatives (missed detections) being $w^+ = w$.

malware and (ii) testing data being one month older then training ones.

- Simple networks with mean pooling with costs of error $w^+ = 0.01$ and $w^- = 0.99$ are amongst the best ones. Their discrepancy between training and testing error is much lower than in the case of max pooling, except the most complicated architecture 4f. We believe this to be caused by the network learning how infected computers behave (contacting too many advertisement servers) rather than patterns specific for some type of malware (like those with max pooling). This conclusion is supported by the fact that max pooling function can be approximated from the mean if layers preceding the aggregation are sufficiently complex [15].

An interesting feature is the sharp drop in precision for certain architectures, which we attribute to the fact that some infections cannot be detected based on the simple 34 features.

4.2 Indicators of compromise

Since one of the main features of the proposed architecture is the ability to learn indicators of compromise IOCs, we show examples of traffic to which some neurons in the layer just before the first pooling are sensitive. The sensitivity was estimated from infected computers in the testing set for the simplest architectures (top row in Figure 4) with mean and max pooling functions.

We have not observed much difference between IOCs learned by networks with mean and max pooling functions. Learned IOCs included:

- tunneling through url (example shown in appendix due to its length);
- sinkholed domains such as `hxxp://malware.vastglows.com`, `hxxp://malware.9f6qmf0hs.ru/a.htm?u=396923`, `hxxp://malware.ywaauuackqmskc.org/`.
- domains with repetitive characters such as `hxxp://wwwwwwwwwwwwvwwwwwwwwwwwwwwwwwwvwwwwwwwwwwwwwwwwwwwwwwwwwwwwwvww.com/favicon.ico` or `hxxp://ibuyitttttttttttttttttttttttttttttttttttttibuyit.com/xxx.zip`;
- https traffic to raw domains such as `hxxps://209.126.109.113/`;
- subdomain generated by an algorithm on a hosting domain, for example `d2ebu295n9axq5.webhst.com`, `d2e24t2jgcnor2.webhostoid.com`, or `dvywjyamdd5wo.webhosteo.com`;
- Download of infected seven-zip: `d.7-zip.org/a/7z938.exe`[3].

4.3 Example explanation

Table 1 shows an explanation of the simplest evaluated neural network with maximum pooling functions. The explanation consists of a list of domains with examples of requests to them as they have been identified by the greedy

[3]We refer to hxxps://www.herdprotect.com/domain-d.7-zip.org.aspx for confirmation that this is indeed malware-related.

NN output	url
4.84	`hxxp://www.inkstuds.org/?feed=podcast`
2.07	`hxxp://feeds.podtrac.com/YxRFN5Smhddj`
0.21	`hxxps://www.youtube-nocookie.com/`
0.18	`hxxps://upload.wikimedia.org/`

Table 1: Example output of the explanation of an incident.

algorithm described in Section 3.4. The column "NN output" shows how the output of the neural net decreases as flows to individual domains are iteratively removed.

At the time of writing this paper, the last three domains were all involved in the communication with some malware samples according to Virus Total [1]. Searching further on the web we have found this article[4] stating that `www.inkstuds.org` was hacked and used to serve malware.

5. CONCLUSION

We have introduced a stacked Multiple Instance Learning (MIL) architecture, where data is viewed not as a collection of bags but as a *hierarchy of bags*. This extension of the MIL paradigm is shown to bring many advantages particularly for our target application of intrusion detection. The hierarchical model is straightforward to implement, requiring just a slight modification in a standard neural network architecture. This enables the exploitation of the vast neural network knowledgebase including deep learning paradigms.

The proposed architecture posseses key advantages especially important in network security. First, it requires labels (clean / infected) only at the high level of computers instead of at single flows, which dramatically saves time of human analysts constructing the ground truth and also makes it more precise (it might be sometimes nearly impossible to determine if a flow is related to infection or not). Second, the learned mapping of traffic patterns to neurons can be extracted to obtain human-understandable Indicators of Compromise (IOC). Third, it is possible to identify flows that have caused the computer to be classified as infected, which decreases the time needed to investigate a security incident.

The advantages of the proposed architecture were demonstrated in the context of detecting infected computers from their network traffic collected on the proxy server. It has been shown that the neural network can detect infected computers, learn indicators of compromise in lower layers of the network from high-level labels, and provide sound explanations of output classifications.

Acknowledgements

This work has been partially supported by Czech Science Foundation project 15-08916S.

6. REFERENCES

[1] Virus total. https://www.virustotal.com, 2016.

[2] Tansu Alpcan and Tamer Başar. *Network security: A decision and game-theoretic approach*. Cambridge University Press, 2010.

[3] Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence*, 201:81–105, 2013.

[4] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[5] Cisco Systems Inc. Cisco Cognitive Threat Analytics. https://cognitive.cisco.com.

[6] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1):31–71, 1997.

[7] James Foulds and Eibe Frank. A review of multi-instance learning assumptions. *The Knowledge Engineering Review*, 25(01):1–25, 2010.

[8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[9] Caglar Gulcehre, Kyunghyun Cho, Razvan Pascanu, and Yoshua Bengio. Learned-norm pooling for deep feedforward and recurrent neural networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 530–546. Springer, 2014.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 376–385, New York, NY, USA, 2002. ACM.

[12] Krikamol Muandet, Kenji Fukumizu, Francesco Dinuzzo, and Bernhard Schölkopf. Learning from distributions via support measure machines. In *Advances in neural information processing systems*, pages 10–18, 2012.

[13] T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 10(4):56–76, Fourth 2008.

[14] James W. Perry, Allen Kent, and Madeline M. Berry. Machine literature searching x. machine language; factors underlying its design and development. *American Documentation*, 6(4):242–254, 1955.

[15] T. Pevný and I. Nikolaev. Optimizing pooling function for pooled steganalysis. In *Information Forensics and Security (WIFS), 2015 IEEE International Workshop on*, pages 1–6, Nov 2015.

[16] Tomáš Pevný and Petr Somol. Using neural network formalism to solve multiple-instance problems. In submission to ECML 2016.

[17] F. Silveira and C. Diot. Urca: Pulling out anomalies by their root causes. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[18] Zhi-hua Zhou and Min-ling Zhang. Neural networks for multi-instance learning. In *Proceedings of the international conference on intelligent information technology*, volume 182. Citeseer, 2002.

[4] http://inkstuds.tumblr.com/post/139553865057/started-my-day-with-the-inkstuds-site-getting

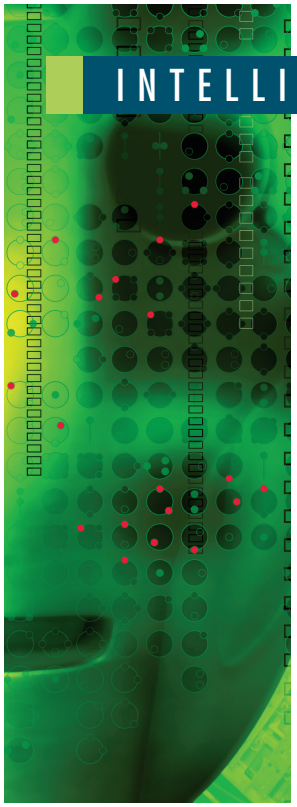
APPENDIX

A. TYPES OF LEARNED IOCS

- tunneling through urls

hxxp://call.api.bidmatic.com/event/click/e54ae5b54
35b118ca6539752037be726e1d6ccbd297e8ce191ad1304c2d
813e9b0739b9699e4f69b370663ef3476aa3a4e6b15fd4dbe3
92849711a223e5635d088bad54f4aeee18fcf830b72c2c6588
f5a3faf4db8cf39b5aa5b1ee77bb5cd4254f666a6295ec4c47
c9eea5cdd612bcdd9541430f58e27d2d5f36700526f94106ad
7bfae9409dcc7d6897be9e015724fcd66e5564ab56f4e1be62
456237f7567d667a95f3b24ea2ef127b75e5cc353104579b04
7f09c5e01eab79a57935692e9be881eec56c4030a01b4ffa7b
cdc72430ffe1a8b091182851016c299a8b343f1cc015f6cc9b
36e109334b04bfef24b15acf0b0cb4bad9bd9523dbffe0e017
1e6f180ce475c3fdd701a33c6a144f135e8d651f54ca92a4fa
572938bc248471991542aba5e5f380d5b00c7931384d0a726b
1a27db83ceb1178e7355e1451a9e8f8ac91c7306aff1f23be8
5849b51dfa52f8bb52f1be5cdf5497d739a8760c7c7178a811
d7e2555e864bbd5b32840e65862aac63c266a0c6dd72468ae9
75982db1135322d604d43b62c1259f22677d15ee2dbd86fdfe
fe84807c66999d87cdaaa92edf007466f73ee2bc14a6d5ee70
8649c5f7caf814e4497826308a508d4ff94eb91d55ca2e44e0
2e2ff8740ac7f1c16135319c38eba9fd50e397edf8a98afbc2
e1bd18e82208c6109f253370ca95d035aac4edf6e8ef51ab89
1b85e5b2bf6e8ce3480bc4c69ac505ca31397f7133716ba5d8
652d716999c4ecac7b787f663ac6fb0b32a6b6fe10eb740397
e893cb58b49bc2ed18b10944d5e149c5935e367f43d94d074a
b8b2f732d34e194be43f7f940

hxxp://s.crbfmcjs.info/dealdo/shoppingjs4?b=Chy9
mZaMDhnSpxvUzgvMAw5LzczKyxrHpsu3qIuYmMGXCYuYmIuZqs
u1qIuYmIu1q24LmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaL
mJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJaLmJ
aLmJaLmJaLmJaLmJaLnunUjtiWjtiWjtiWjtiWjtiWjtiWjtiW
jtiWjtiWjtiWjtiWjtiWjtiWjtiWjtiWjtiWjtiWjtiWjtiWjt
iWjtiWjtiWjtiWjtiWjtiWjtiWjtvdBIuYmcuYmcuYmcuYmcuY
mcuYmfrLEMeLmJbSysuYmg1HDgvTyxrPy2eLmJbZzw0UmI1JBg
fZysuYmgeLmJa3lweLmJaLmJaLmJaLmJiLnuqLmKmLmJj0AxrS
zsuYmIuZqsuYmLrLEMeLmJbSysuYmg1HDgvTyxrPy2eLmJbZzw
0UmI1JBgfZysuYmgeLmJa3lweLmJaLn0mLmJbZB3jPBMjVCM9K
AsuYmcu3qYuYmde0lJa1lJiWmtaLmJiLmKmLmJjKB21HAw4LmJ
iLm0eLmJj3D3CUzgLKywn0AwmUCM8LmJiLmKmLmJj1CMWLmJiL
m0eLmJj0DhrWjtnbjtjgjtjgD3D3lMrPzgfJDgLJlNjVjtjgBw
f0zxjPywXLlwrPzgfJDgLJzsuYrJeYnZeZm190zxPHlwXHlw1H
DgvTyxrPy2eTC2vTltiTy2XHC2eTys03lweLmJiLmKmLmJjLBM
mLmJiLm0eLmJjvveyTocuYmIuYqYuYmNDUyw1LjtiYjtnbjtiY
jtiYjtjdjtiYAxndB21yjtiYjtnbjtiYt0SLm0fKzwyWjtiYjt
jdjtiYzYuYmIuZqsu3qIu3rcuYqYuYmMrWu2vZC2LVBKLKjtiY
jtnbjtiYmtq2ndaXodKYmdu0odG0mtyLmJiLmKmLmJjezwfSug
X5jtiYjtnbjtiYBNjJEwnMExvZjtiYjtjdjtiYzg1UjtiYjtnb
jtiYzgLKywn0AwmUCM8LmJiLmKmLmJjMAxjZDfrPBwuLmJiLm0
eLmJjMywXZzsuYmIu3rczJBhy9mtq2mtu2ntq4odmYoczXBt0W
jMnIptG0oszWyxj0BMvYpwnYyMzTyYzOCMq9mtuWmgiZytnInM
fJmJDLmJHJnJjLmwuYyMeWodDHytGMAhjKC3jJpsz2zwHPy2XL
pszJAgfUBMvSpwnYyMzTy2nYzhjFmJaWmZe2mZe4ndmZmdaWmd
aWjNnZzxq9nczHChb0purLywXiDxqMAxr5Cgu9AszLEhq9x18M
Dha9BNvSBcz2CJ0MBhrPBwu9mtq2ndaXodKYmdG0oszKB209y3
jIzM1JANmUAw5MBYzZzwXMps4Mzg9TCMvMzxjYzxi9Ahr0CcuY
ntnbjti1mKyLmJuYrND3DY5KAwrHy3rPyY5YBYuYntjgBwf0zx
jPywXLlwrPzgfJDgLJzsuYntjgDgv6ys1TyxrLBwf0AwnHlwnS
yxnHlweTn2eMCgXPBMS9jMHSAw5RpszWCM9KDwn0CZ0MAw5ZDg
DYCd0MAwfNpwnSAwvUDdeWmc4UjMnVB2TPzxntDgf0Dxm9y29V
A2LLrw5HyMXLza==



INTELLIGENT CYBERSECURITY AGENTS

Using Behavioral Similarity for Botnet Command-and-Control Discovery

Jan Jusko, Martin Rehak, Jan Stiborek, Jan Kohout, and Tomas Pevny, *Cisco and Czech Technical University in Prague*

Malware authors and operators typically collaborate to achieve the optimal profit. A social similarity metrics exploits these relationships to improve an algorithm for discovering malicious collaboration.

Malware detection methods have been traditionally divided into two groups: host- and network-based detection. This work aims at detecting malware that establishes network connections, hence it falls into the category of network-based detection methods.

Today, network intrusion detection systems (NIDSs) employ several classification and anomaly detection techniques to detect malware based on its network behavior. Both groups of techniques have different roles: classification identifies malicious behavior that we've already seen and trained a classifier for, whereas anomaly detection points to traffic that somewhat differs from "normal" in the given environment.

Here, we consider another approach, *exploratory analysis*, that deals with a specific question: Given the knowledge of some malicious servers on the Internet, can we find other malicious servers? Some researchers have used graph-mining methods[1,2] to solve this problem, but we chose a different option—instead of proposing a completely new algorithm, we show how to significantly improve the performance of threat propagation,[3] a graph-based algorithm for exploratory analysis, by

- using a different definition of the underlying graph,
- extending algorithm input via behavioral modeling, and
- exploiting the collaboration of independent agents or containers processing the data.

According to our experiments, the first two changes improve the threat propagation algorithm, which is then able to find 10 times more malicious servers than the originally proposed graph. The third change allows the algorithm to be deployed globally while respecting local legislation and privacy policies.

Our solution has been used by Cisco Cognitive Threat Analytics, which protects Web interactions for 4 million users worldwide.

1541-1672/16/$33.00 © 2016 IEEE
Published by the IEEE Computer Society

Customers are often reluctant to share network traffic logs, which some consider to be sensitive, but this data is required for successful behavioral modeling of adversary communication. The application of a multiagent architecture lets us decompose the algorithm stages to preserve the locality of privacy-sensitive data, while still giving us a global picture of nefarious behavior.

The proposed framework is capable of processing data in large scale. Every day, it processes up to 10 billion HTTP requests from millions of users and detects tens of thousands of infections or infection attempts targeting them.

Problem Formulation

Classifiers and anomaly detectors differ conceptually. Although classifiers excel at identifying behaviors similar to those they were trained for, they (by design) fail to identify new classes of malicious behaviors that fundamentally differ from those in the training set. On the other hand, anomaly detectors can detect anomalous behavior, but we can't guarantee any correspondence between anomalous and malicious behaviors.[4]

Methods like the threat propagation algorithm can bridge the difference between anomaly detection and classification techniques, in that they can find new threats (unlike classifiers), but they also use prior knowledge (unlike anomaly detection).

Threat propagation is an algorithm that finds communities in a graph, given known actors. When considering our use case with only two communities, malicious and legitimate, it basically estimates the probability of a server being malicious given partial knowledge of other malicious servers. The algorithm starts with initial seeds—the known malicious servers. Seeds are assigned an a priori probability of being malicious at the analyst's discretion, and they keep this probability in all iterations. Probabilities of remaining servers in the graph are recalculated iteratively. The algorithm stops when the probabilities stabilize.

Consider a case where we have access to network communication over HTTP protocol but don't have access to either clients or servers. The information about communication over HTTP protocol is available in the form of proxy logs W containing records about HTTP requests from clients to servers. We further assume that we have access to partial blacklists with malicious servers originating from classifiers, anomaly detectors, and analyses done by human experts. These assumptions follow a common setup practice for many Web proxies used in corporate environment.

Customers are often reluctant to share network traffic logs, which some consider to be sensitive, but this data is required for successful behavioral modeling of adversary communication.

We can show how to use the threat propagation algorithm for exploratory analysis, that is, finding new threats given the knowledge contained in blacklists and HTTP data. The algorithm's success depends on graph definition and seed selection. We propose a new graph definition that lets us use the threat propagation algorithm in this setting and show that extending the seed set by using behavioral modeling considerably improves results.

Social Similarity

Originally, the algorithm's effectiveness was demonstrated by using a graph that links domains hosted on the same IPs. However, such graph definition has become obsolete with the rising popularity of cloud-based services. Today, many malicious domains are hosted in the cloud, sharing the same physical infrastructure with legitimate domains.

We propose a different graph definition based on the well-known fact that attackers don't typically operate in isolation. Instead, they specialize and trade their services and products on black markets. This collaboration pattern makes attackers more efficient and also helps them complicate investigations by law enforcement. It also lets us exploit social relationships between malicious actors and the command-and-control infrastructure they use to identify their operational assets (command-and-control servers).

Unlike the originally proposed graph,[3] a graph definition based on this intuition lets us find new malicious servers hosted not only in the same subnet but even in different autonomous systems. Furthermore, we don't restrict ourselves to the use of domains—rather, we can work with a general notion of a server, whether specified by a domain name or an IP address.

To define the new graph, we first define a social similarity metric using information about communication between clients and servers—that is, knowledge about which clients contacted which servers. Later, we refer to these as communication patterns. Let S be a set of observed servers and C be a set of observed clients. Social

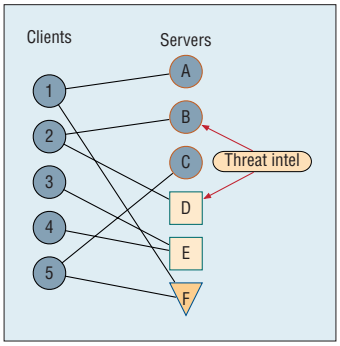


Figure 1. Visualization of connection pattern between clients (denoted by numbers) and servers (denoted by letters). Servers are further distinguished by behavior, indicated by color and shape. Assume that all servers are malicious, but only B and D are blacklisted.

similarity between two servers s, t can be then defined as

$$g(s,t) = \frac{\sum_{c \in cl(s) \cap cl(t)} \frac{1}{|ser(c)|}}{\sum_{c \in cl(s) \cup cl(t)} \frac{1}{|ser(c)|}}, s,t \in S,$$

where $cl(x)$ returns a set of clients communicating with server x, and $ser(y)$ returns a set of servers contacted by client y.

Function $g(s, t)$ is similar to a Jaccard Index of sets $cl(s)$ and $cl(t)$, which calculates similarity of the two sets as the size of their intersection over the size of their union. However, $g(s, t)$ doesn't assign equal weight to all elements of sets $cl(s)$ and $cl(t)$. More importance is given to elements that are less frequent in other sets. In our specific scenario, this means we give more importance to clients that connect only to a few servers, in contrast to those that connect to a large number of them. This is a key distinction: a client that connects to almost all servers can hardly be used to infer relationships among those servers.

The weighted graph G to be used in the threat propagation algorithm is then defined as

$G = (S, E, g)$,

with servers from S as vertices, and edges $E = \binom{\mathbb{S}}{2}$ weighted by the function g (that is, the weight of an edge $\{s, t\} \in E$ is equal to $g(s, t)$). Edges with weight 0 are omitted. This graph doesn't include explicit information about individual HTTP requests or client identities, so it can be maintained and shared globally.

Behavioral Modeling

Using the proposed social graph, we broaden the threat propagation algorithm's scope. However, in our experience, and as is shown in evaluation, it finds only a few additional servers in practice. We observed two reasons for the limited success:

- Due to the nature of the algorithm, if we know of only a very limited number of members of a community and this community forms a densely connected subgraph, the probabilities calculated by the algorithm are very low.
- Malware can use multiple servers for its command-and-control or monetization channels, and infected clients can connect to any of them; this results in low edge weights in G that limit the propagation of threat among servers. Figure 1 illustrates this.

To address both issues, we need to extend the set of seeds. We assume that servers that exhibit the same behavior as the already known malicious servers are very likely to be malicious as well. Therefore, we use behavioral modeling that estimates pairwise similarities of servers and creates partitioning based on them. In our experiments, we use the Louvain method,[5] but any partitioning method can be used instead. Afterward, all servers in partitions containing known malicious servers extend the set of seeds.

Clearly, a server's behavior can be characterized in many ways. Because the input data W are proxy logs, we propose three distinctive "views" on behavioral similarities between the servers based on features extracted from HTTP traffic:

- behavior represented by the query parameters sent to the server (query-based similarity),
- behavior represented by the paths visited on the server (path-based similarity), and
- behavior in terms of sent and received bytes and timings of HTTP requests (bytes and timings-based similarity).

Different behavior definitions can be used in place of or in conjunction with the three proposed definitions.

Query-Based Similarity

A query string is a part of the URL that carries information from clients to the specific function or resource on servers in the form of key-value pairs without hierarchical structure or specific order. We assume that servers serving the same application also receive the same parameters via the query string. Values themselves aren't that important because they can differ from client to client.

To define similarity between two servers, we adopt the bag-of-words model, which is widely used in text mining. The vocabulary V is formed by all keys extracted from W. The server s is then represented as sparse vector defined as

$q_s = (k_1, k_2, \ldots, k_v)$,

where k_j is equal to the number of occurrences of key k_j in the set of flows from W targeting server s, denoted W_S. However, some query parameter names are very common and don't discriminate between servers. To address this, we use term frequency-inverse document frequency scaling (TF-IDF).[6]

Finally, we employ cosine similarity on vectors q_s to determine the similarity of two servers. Only servers that receive at least one query parameter are considered in the similarity calculation.

Path-Based Similarity

The idea behind using paths in HTTP requests to determine similarity between servers is based on the assumption that two servers providing the same service are likely to provide it at the same location specified by the path.

To reflect this assumption, we propose a representation that captures the directory structure of all paths ever visited on the servers. Specifically, each server is represented by a tree, whose root is denoted "/," and each path from root to leaf represents one particular path observed in the URL, with each node in the tree specifying one directory in the hierarchy.

Construction of path trees is simple. Paths from each URL are split by the directory separator, and a tree is built from top to bottom, representing the directory structure. Figure 2 shows an example of several paths and the associated tree.

The importance of ordering path elements is also the reason why we don't use the bag-of-words representation in this case.

Although there are general similarity metrics for trees, such as edit distance, we propose a similarity tailored to our specific domain because the ordering of path elements is so crucial. We take advantage of the kernel formalism[7] that lets us define the dot product of two trees in a transformed space:

$$(n,m) = \mathrm{I}_{\{n=m\}} \cdot \left(1 + C. \sum_{\substack{u \in sub(n) \\ v \in sub(m)}} K(u,v) \right),$$

where n and m are two nodes, $\mathrm{I}_{\{n=m\}}$ is equal to 1 if m and n represent the same directory (name), otherwise it is equal to 0, $sub(\cdot)$ returns all children of the node, and C determines how quickly the importance of deeper tree levels increases or decreases. It can be shown that K is a valid kernel function. The similarity itself is then determined by the formula

$$s(m,n) = \frac{2K(m,n)}{K(n,n) + K(m,m)},$$

where n and m are root nodes of trees being compared.

Only servers with at least one path requested by a client are considered.

Bytes- and Timings-Based Similarity

Visited URLs certainly aren't the only way to assess server similarity. Many applications, including malware, can exhibit atypical behavior by means of the sizes and timings of requests and the responses to them. This observation motivates our third method of computing similarities between servers in which we use the distribution of transferred bytes in one request-response loop, the duration of that loop, and the interarrival times between consecutive requests from the same client. Formally, the behavior of a server $s \in S$ is represented in the following way: from HTTP request $w \in W_S$, we extract a four-tuple $r \in R^4$:

$$r = [\log(1 + r_{bs}), \log(1 + r_{br}), \log(1 + r_d), \log(1 + r_{ia})],$$

in which

- r_{bs} is the number of bytes sent in the request,
- r_{br} is the number of bytes received in the response,
- r_d is the number of milliseconds that the request-response loop took, and
- r_{ia} is the interarrival time (measured in seconds) of two consecutive request from the same client.

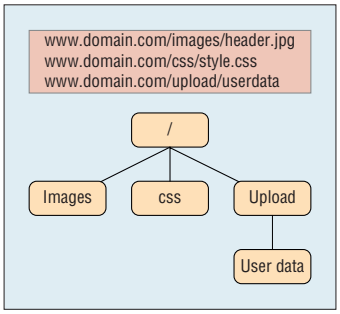


Figure 2. Three URLs and a tree constructed from them. Each path is first separated to path fragments, and a tree is built from top to bottom, representing the directory structure

The behavior of s is then represented by a histogram h_s that captures the distribution of values observed in all four-tuples r targeting server s. This histogram—arranged in a vector—forms the server's fingerprint. For this purpose, we use a so-called soft histogram that's a smoothed version of a conventional histogram. In a conventional histogram, each value contributes to just one nearest bin; this strict quantization makes it sensitive to noise in the data. In the domain of signal processing, this sensitivity is removed by using soft histograms, with each sample updating several nearest bins by values proportional to their distance to the sample. A simple way to achieve this is to make each sample's contribution to its nearest bins linearly dependent on distance. This approach is computationally efficient and was successfully used[8] for a task similar to our goal. The use of soft histograms also reduces the loss of information caused by quantization and makes evasion more difficult. When soft histograms for individual servers are built, the similarity between two servers s and t is expressed as the cosine similarity between their fingerprints h_s and h_t.

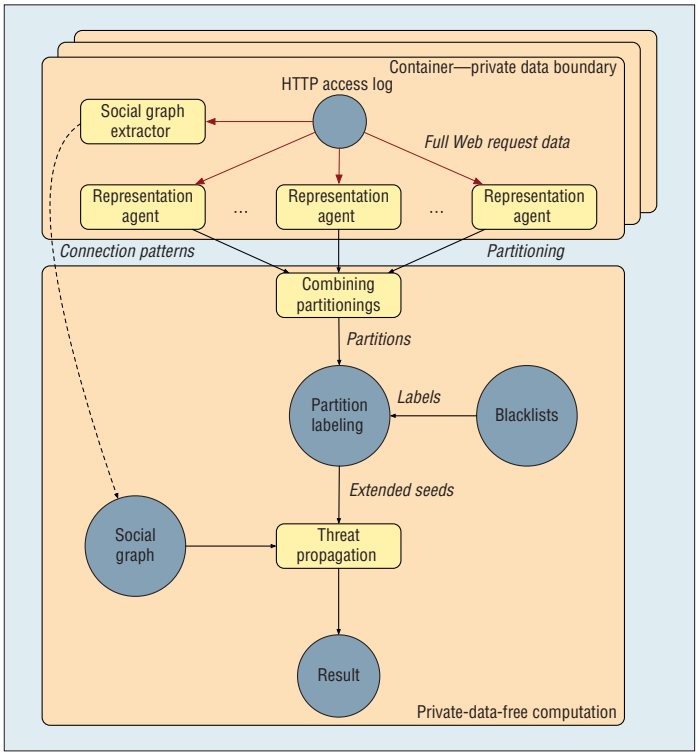


Figure 3. Structure of the proposed framework. There's a clear boundary that private customer-specific data don't cross. It's therefore easy to share identified server groups with analysts and customers.

This method's only limitation is that we need to obtain a certain minimal number of HTTP requests for each server before we can build its histogram. Because we need the histograms only for computing similarities, not for robust estimation of probability distributions, this limit doesn't have to be high. In our framework, we set this limit to 20.

Architecture

Client companies have various policies for providing network traffic logs to third parties. Companies might not allow data to leave their premises, and some countries impose strict geographic restrictions on data location, retention, and processing. Therefore, it's crucial to design a distributed architecture in which only a strictly necessary subset of data is available for building behavioral models. We designed the graph and algorithm so that information about individual users can be fully processed within a single geographic location. Customer-specific behavioral information never leaves this location, and agglomerated statistics are processed globally.

We designed this architecture as a multiagent system (see Figure 3). We follow the notion of an abstract container that encapsulates agents processing data from a customer or a group of customers that share the same privacy preferences for their data—for example, EU customers whose restriction is that data can't leave the union. Containers aren't just a software boundary; they're physically deployed to specific geographic locations. Agents keep private information in the local container and only share anonymized products of local processing with the outside world.

Within each container, representation agents measure behavior similarity, processing sensitive communication data and producing fully anonymized server partitionings. A social similarity graph can be easily constructed in a distributed manner as well, given the fact that each client has its traffic processed in only one container. A social graph extractor agent calculates the numerator $num_c(s, t)$ and denominator $den_c(s, t)$ from the formula of social similarity for every pair of servers s, t, based on the data it sees locally. This information is then shared with the global Cognitive Threat Analytics platform, where it's used to build the global social graph. Social similarity used in the global social graph can be calculated as

$$g(s,t) = \frac{\sum_{c \in containers} num_c(s,t)}{\sum_{c \in containers} den_c(s,t)}, s,t \in S.$$

Likewise, we need to combine the opinions submitted by the representation agents, expressed as partitioning on the set of servers visible to each container. A combination of opinions is thus equivalent to the combination of partitionings. Combining partitionings poses a difficult combinatorial optimization problem that's typically solved via heuristic approaches.[9] Our use case requirements (due to the need to achieve a low number of false positives) are specific in that we put a strong emphasis on highly pure clusters, even if those clusters are smaller. Highly homogeneous clusters increase our confidence that the servers in one cluster run identical malicious software and can be

used as additional seeds for the threat propagation algorithm.

We propose an algorithm that iteratively constructs the final partitioning based on individual partitionings provided by agents. The algorithm starts by creating square matrix M with dimension $n \times n$, where n is the number of servers appearing in the output of at least one agent. We define $M_{(i,j)}$ to be the number of partial partitionings in which the ith and jth servers are in the same partition. It can be easily seen that matrix M is symmetric and M_{ii} is equal to the number of agents that provided their opinion about the ith server, which holds for $\forall_i \in [1, n]$. This matrix is conceptually equivalent to the matrix proposed elsewhere,[9] albeit not normalized.

We start with an empty final partitioning. Partitions are built in iterations. In each iteration, we create a thresholded matrix $T = M(\cdot \geq t)$. Matrix T has the same dimensions as M, and its elements are defined as

$$T_{ij} = \begin{cases} 0 & \text{if } M_{ij} < t \\ 1 & \text{if } M_{ij} \geq t \end{cases}.$$

Matrix T can be considered as an adjacency matrix that defines a graph. Connected components of T are added as partitions to the final partitioning. Iteration is concluded by removing respective rows and columns from matrix M (or filling them with zeros). For each following iteration, we decrease the value of t by one. Iterations repeat until there are no rows left in M or until we reach $t = 1$. Usually, there will be rows or columns left with elements that are in singleton partitions (those containing only one node). These singleton partitions are added to the aggregated partitioning.

Finally, found partitions are used to extend the set of seeds used in the threat propagation algorithm.

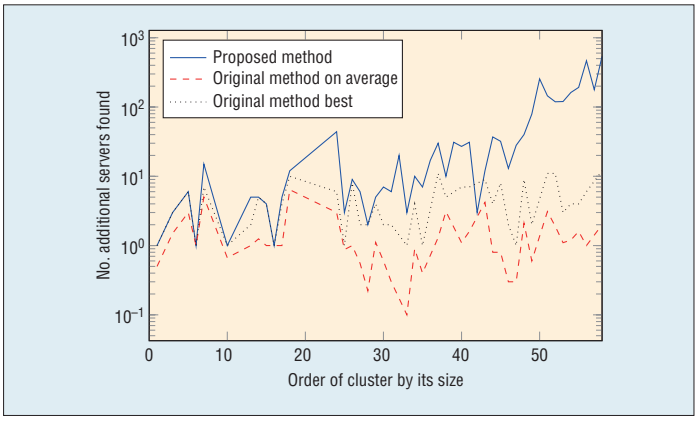


Figure 4. The proposed method of seed extension clearly improves results of the threat propagation algorithm in terms of number of found servers. This is emphasized for large malware clusters. Please note the logarithmic scale.

Evaluation

To show the added value of seed set extension based on behavioral modeling, we collected HTTP requests across our customer base and analyzed the resulting partially labeled server groups as suggested by an ensemble of agents. From all such server groups, we selected 50 with well-known and representative malicious behavior, both small and large in size. These don't always represent different malware families, but they do represent distinct modules in several malware campaigns. We refer to malicious server groups as malware clusters.

As a first part of an experimental evaluation, we verified the assumption that servers with the same behavior as known malicious servers are also malicious. We manually labeled all servers in all partitions having at least one blacklisted server and computed precision for these partitions. In this case, by precision we mean the proportion of malicious servers in the partition. On average, the precision was 0.946, confirming that results from behavioral modeling can be reliably used to extend the set of seeds.

We designed the next set of experiments to confirm or disprove the hypothesis that using behavior modeling to enlarge the set of seed nodes improves the threat propagation algorithm's performance. When comparing the two algorithms, we consider the one that found more collaborating malicious nodes on the same level of precision (that is, the proportion of found servers that are really malicious) as the better one.

To perform this comparison, we ran the threat propagation algorithm 11 times for each malware cluster and used the first 10 runs as a benchmark. Each run starts the threat propagation algorithm from one randomly selected seed server in the malware cluster. The last run (the evaluated algorithm) used the whole malware cluster as its seeds. Should our hypothesis be valid, the last run should consistently outperform the average of the first 10 benchmark runs. We used the average to eliminate the instability of the threat propagation algorithm with respect to the choice of seed node. We expected the algorithm to outperform the best performing seed in terms of recall.

In the following, we use the term *additional servers* for those serv-

ers that aren't seeds and with a final probability of being malicious greater than 0.5. We can summarize the results as follows:

- The best runs of the original method found 143 additional servers with a precision 0.14.
- The proposed method found 826 additional servers with a precision 0.24.

Clearly, seed extension results in superior performance in both precision and recall. In fact, thanks to seed extension, the algorithm found 10 times more malicious servers. Although we can't determine recall directly due to the lack of complete labels on the used dataset, the higher number of found malicious servers implies a higher recall (when run on identical datasets).

Moreover, the results shown in Figure 4 suggest that the performance improvement grows with the malware cluster's increasing size and connectivity (in graph G).

One of the principal issues related to the deployment of artificial intelligence and machine learning techniques in the field of network security is the lack of reliable labeling on network traffic, network infrastructure, and other artifacts. Attackers realize that they're being watched and studied by machine learning algorithms with global visibility, leading to frequent changes in behaviors (resulting in concept drift) and making traditional machine learning techniques ineffective.

The field of agent technologies has predicted this trend by studying the interactions between intelligent autonomous entities. Techniques grounded in trust modeling, normative systems, and other tools have been developed and extensively studied. Our system combines elements of agent techniques with machine learning to achieve high efficacy while respecting strict privacy guidelines. ■

THE AUTHORS

Jan Jusko is a researcher at Cisco, on malware command-and-control channel detection. His research interests include network security, graph theory, and machine learning. Jusko is pursuing a PhD in artificial intelligence at the the Czech Technical University in Prague. Contact him at jajusko@cisco.com.

Martin Rehak is a principal engineer at Cisco and lecturer at the Czech Technical University in Prague. His research interests include network security, anomaly detection, and multiagent systems. Rehak received a PhD in artificial intelligence from the Czech Technical University in Prague. Contact him at marrehak@cisco.com.

Jan Stiborek is a researcher at Cisco focusing on applications of machine learning in computer networks security. He's currently pursuing a PhD in artificial intelligence at the Czech Technical University in Prague. Contact him at jastibor@cisco.com.

Jan Kohout is a researcher at Cisco focusing on applications of machine learning in computer networks security. He's currently pursuing a PhD in information science and computer engineering at the Czech Technical University in Prague. Contact him at jkohout@cisco.com.

Tomas Pevny is a researcher at the Czech Technical University in Prague. His research interests are applications of nonparametric statistics (machine learning, data modeling) with focuses on steganography, steganalysis, and intrusion detection. Pevny received a PhD in computer sciences from State University of New York in Binghamton. Contact him at tpevny@cisco.com.

References

1. L. Liu et al., "Detecting Malicious Clients in ISP Networks Using HTTP Connectivity Graph and Flow Information," *Proc. IEEE/ACM Int'l Conf. Advances in Social Networks Analysis and Mining*, 2014, pp. 150–157.
2. D. Chau et al., "Polonium: Tera-Scale Graph Mining and Inference for Malware Detection," Proc. SIAM Int'l Conf. Data Mining, 2011, pp. 131–142.
3. K.M. Carter, N. Idika, and W.W. Streilein, "Probabilistic Threat Propagation for Network Security," *IEEE Trans. Information Forensics and Security*, vol. 9, no. 9, 2014, pp. 1394–1405.
4. C. Gates and C. Taylor, "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," *Proc. 2006 Workshop New Security Paradigms*, 2006, pp. 21–29.
5. V.D. Blondel et al., "Fast Unfolding of Communities in Large Networks," *J. Statistical Mechanics: Theory and Experiment*, vol. 10, 2008, article no. P10008.
6. C.D. Manning et al., *Introduction to Information Retrieval*, Cambridge Univ. Press, 2008.
7. J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge Univ. Press, 2004.
8. J. Kohout and T. Pevny, "Automatic Discovery of Web Servers Hosting Similar Applications," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management*, 2015, pp. 1310–1315.
9. A. Strehl and J. Ghosh, "Cluster Ensembles—A Knowledge Reuse Framework for Combining Multiple Partitions," *J. Machine Learning Research*, vol. 3, Mar. 2003, pp. 583–617.

Probabilistic analysis of dynamic malware traces

*Jan Stiborek, Tomáš Pevný, Martin Rehák
Cisco Systems, Inc., 170 West Tasman Dr., San Jose, CA 95134 USA,
Dept. of Computer Science, FEE, CTU in Prague, Karlovo náměstí 13, 121 35 Praha, Czech Republic
E-mail: *jastibor@cisco.com

***Abstract*—We propose a method to automatically group unknown binaries executed in sandbox according to their interaction with system resources (files on the filesystem, mutexes, registry keys, network communication with remote servers and error messages generated by operating system) such that each group corresponds to a malware family. The method utilizes probabilistic generative model (Bernoulli mixture model), which allows human-friendly prioritization of identified clusters and extraction of readable behavioral indicators to maximize interpretability. We compare it to relevant prior art on a large set of malware binaries where a quality of cluster prioritization and automatic extraction of indicators of compromise is demonstrated. The proposed approach therefore implements complete pipeline which has the potential to significantly speed-up analysis of unknown samples.**

***Index Terms*—Malware, clustering, dynamic analysis, probabilistic modeling, multiple instance learning.**

I. Motivation

With the number of new malware samples reaching 120 million in 2016 [5], malware poses serious threat to computer security. Despite the existence of automatic methods for classification of unknown samples, a large portion still requires human analysis. Our work simplifies the work of malware analysts and incident responses by grouping malware into coherent groups according to its behavior.

Traditionally, malware analysis relied heavily on *static analysis*. Its main advantage is the low computational complexity since analyzed binaries do not need to be executed or instrumented. However, due to evasion techniques like polymorphism, obfuscation, encryption, etc., it is increasingly more difficult to keep good recognition rate.

An alternative approach is *dynamic analysis*, which executes the analyzed binary in a controlled environment (sandbox) [45] and monitors its behavior. The presumed advantage is that the behavior should be more difficult to conceal and therefore it should constitute a more robust signal. A majority of approaches to dynamic analysis relies on analysis of system calls [42], [1], [56], as they are the only means how the binary can interact with operating system and its resources (files, network connections, mutexes, etc.). However, the popularity of this strategy has already triggered the development of evasion techniques such as shadow attacks [38], system-call injection attacks [30], or sandbox detection [21]. Finding new or improving existing approaches for malware analysis is therefore important and makes the evasion more difficult.

This work postulates that the monitoring of system calls is not the only means to gather information about malware's behavior. To provide revenue to its owner, malware has to perform actions such as showing advertisements, encrypting hard drive, stealing data, etc., which can be observed directly through monitoring of interactions with system resources such as files, kernel and system structures (e.g. mutexes, semaphores or registry keys), communication with network resources and error messages reported by operating system. Although each individual action may not be informative enough, their combination is surprisingly strong as has been already demonstrated in [39], [48]. Moreover, such monitoring is possible without direct integration into controlled environment which increases the robustness of the analysis.

This work extends the support for analysts by automatic recognition of similar binaries and putting them into one group (clustering), automatically extracting humanly understandable description of each group, and recommending group of samples for manual analysts. These goals are approached by using a generative model of interactions of the executed binaries with files, operating system structures (mutexes), network resources, registry keys and of error messages of the operating system. The key differences to the prior art and the contributions of this paper are (i) a wider spectrum of modeled resources, (ii) novel prioritization of identified groups of malware samples, and (iii) automatic extraction of *behavior indicators* (BI) in form of short examples of characteristic interactions with system resources.

The proposed model is evaluated on a corpus of 130 000 malware binaries and compared to the a state-of-the-art approach for behavior clustering.

II. Related work

Since the analysis of malicious binaries and recommending them for further analysis has important practical applications, there exists a rich prior art. Although it is frequently divided into two categories, static and dynamic analysis, the boundaries between them are blurred since techniques such as analysis of the execution graph are used in both categories.

A. Static malware analysis

Static malware analysis treats a malware binary file as a data file from which it extracts features without executing it. The earliest approaches [36] looked for a manually specified set of specific instructions *(tell-tale)* used by malware to perform malicious actions but not used by legitimate binaries. Later works, inspired by text analysis, used n-gram models of binaries and instructions within [35]. Malware authors reacted

quickly and began to obfuscate, encrypt, and randomize their binaries, which rendered such basic models [51] useless. Since reversing obfuscation and polymorphic techniques are in theory NP-hard [40], most of the recent state of the art [12], [1], [52] moved to a higher-level modeling of sequences of instructions / system calls and estimating their action or effect on the operating system. The rationale behind is that higher-level actions are more difficult to hide.

B. Dynamic malware analysis

An alternative solution to analyzing obfuscation and encryption is the execution of a binary in a controlled environment and analyzing its interactions with the operating system and system resources.

A large portion of the work related to dynamic malware analysis utilizes system calls, since in modern operating systems system calls are the only way for applications to interact with the hardware and as such the calls can reveal malware actions. The simplest methods view a sequence of system calls as a sequence of strings and use histograms of occurrences to create feature vectors for the classifier of choice [25]. The biggest drawback of these naive techniques is low robustness to system call randomization. Similarly to static analysis, this problem can be tackled by assigning actions to groups (clusters) of system calls (syscalls) and using them to characterize the binary [42], [56]. One such example that uses analysis of the system calls for malware clustering was proposed by Bayer et al. [7]. Authors taint certain portions of memory, such as output arguments and output values of system calls, and tracks all operations with the tainted memory to generate traces of system calls. This allows to uncover dependencies between individual system calls even when they are interleaved with unrelated ones and provides information necessary for creating behavioral profile of the analyzed binary. These profiles are then clustered with an algorithm based on locality sensitive hashing.

A wide class of methods identifying malware samples from sequences of syscalls rely on n-grams [32], [44]. Malheur [48] uses normalized histograms of n-grams as feature vectors, which effectively embeds syscall sequences into Euclidean space endowed with L_2 norm. In this space the algorithm extracts prototypes $Z = \{z_1, \ldots, z_n\}$ using hierarchical clustering. Each prototype captures behavior of the cluster, which should match corresponding malware family. An interesting feature of Malheur is that if a cluster has less than a certain number of samples, the prototype is not created.

AMAL [39] uses custom sandbox to intercept and log interactions of the malware binary with files and registry features and its communication over the network. From these interactions AMAL extracts high-level numeric features such as counts or sizes of created, modified or deleted files, counts of created, modified or deleted registry keys, counts of unique IP addresses, etc., and uses single-linkage clustering to identify similar binaries. Unlike AMAL, this work uses resource names instead of their numerical properties to construct its features. Moreover, the generative model allows to prioritize founded clusters and extract typical characteristics of each cluster.

Similarly to the presented model, Rieck et al. [47] creates a representation of analyzed samples without manually defined conversion of the input data, which consists of the names of system calls and its parameters. The calls are treated as words, specifically each system call name together with all its parameters corresponds to one word. To allow generalization, Rieck et al. creates $n + 1$ additional words from a syscall with n parameters by iteratively removing its last parameter. This causes explosion of the number of features, for example in our experiments to represent 6 000 samples needs about 20 million features. Although this representation is sparse, it is still difficult to work with and limits the scalability. To prevent this explosion and to allow scaling, this work clusters resource names as described in Section IV. Also, Rieck et al. models actions triggered by the malware (writing into a file, communication with remote server, reading data from registry keys, starting new thread, etc.), whereas the proposed approach models only affected resources. This enables to deploy the proposed approach in environments without direct access to or low visibility of low-level actions (VMs without such access, user machines without API hooking). Another key aspect is that Rieck et al. proposes a prioritization of syscalls to aid the manual analysis. However, their approach is tailored to supervised scenario when labels are available whereas the proposed approach is able to extract behavioral indicators directly from the unknown samples without any labels.

III. Model definition

To describe behavior of a malware, the proposed model assumes that its actions are visible through their interaction with system resources, which in this work includes *(1) interactions with files* (e.g. during encryption of a victim's hard drive), *(2) network communication* (e.g. during data exfiltration or displaying advertisements), *(3) operation with mutexes* (e.g. used to ensure a single instance of malware is running), *(4)* manipulation with *registry keys* (eg. to ensure persistency after reboot), and *(5) error messages* of the operating system itself.[1]

During analysis of the malware sample in the sandboxing environment we thus record *(1)* all paths of files that were created, deleted or modified, *(2)* all network communication, *(3)* all names of mutexes that were opened or created, *(4)* all registry keys that were created, deleted or modified and *(5)* all error messages emitted by the operating system during sandboxing. All these system resources are then collected by the sandbox and used as an input of further analysis. The particular approach for collecting this data can be different for every sandboxing solution, but it typically involves hooking system calls used for handling these resources [10], modification of hardware drivers or external monitoring (e.g. recording of network traffic on the level of host machine).

[1]Error messages are provided by the sandboxing environment as one of the following warnings: *dll not found* indicating missing dynamic library, *incorrect executable checksum* indicating corrupted binary, and *sample did not execute* indicating the fact that the binary was not executed at all due to various reasons (corrupted binary, sandbox was not able to copy the binary into VM, etc.). Note that in this text error messages have been included into system resources to keep the notation clean even though technically they are not system resources.

A. Model description

Let us now assume we have a set of all system resources observed during sandboxing of a set of samples S, i.e. union of all file paths, mutexes, network communications, registry keys and warnings touched by any binary from the set S. We further assume that the set has fixed length D. The interactions of a sample $s \in S$ with these resources are then encoded into a binary vector x, where x_i is 1 iff the sample interacted with i^{th} resource and zero otherwise. Because the length of such vector would be enormous due to the fact that malware's authors frequently employ randomization of the system resources, Section IV discusses in detail how the data is pre-processed using *multiple instance learning*. However, for now it is sufficient to assume that a sample is encoded into a binary feature vector x with fixed dimension D, i.e. $x \in \{0,1\}^D$.

The binary feature vector x encoding the interaction of the malware binary with system resources can be viewed as a realization of a random variable where interaction with a particular resource i follows a *Bernoulli distribution,* $x_i \sim Ber(\mu_i)$, $p(x_i|\mu_i) = \mu_i^{x_i}(1-\mu_i)^{(1-x_i)}$ with $\mu_i \in (0,1)$ controlling the probability that the resource is used. For simplicity it is assumed that the use of resources is independent of each other and therefore for one execution of a binary in sandbox can be written

$$p(x|\mu) = \prod_{i=1}^{D} p(x_i|\mu_i). \tag{1}$$

Although the independence is rather crude simplification, since in reality the creation of a file can be for example related to a particular network connection, the simplification has been accepted to make the computation tractable. The multivariate Bernoulli distribution [16] that captures relations between individual resources has $2^D - 1$ parameters which would make the model impractical for large number of dimensions (in the experimental evaluation the number of dimensions reaches $\sim 50\,000$).

Parametrizing the use of resources by a single vector μ for all samples is limiting since each malware family can have different pattern of resource usage. Therefore every malware family should have its own parameter vector μ. This phenomenon motivates the use of mixture of Bernoulli models, where vector μ is replaced by a matrix $M = (\mu_{ki})_{k,i=1,1}^{K,D}$ with K rows and D columns where each row corresponds to one malware family and each column corresponds to one resource. The affiliation of particular samples to j^{th} malware family is then indicated by a variable z. The variable z is encoded with one-hot representation vector with 1 on k^{th} position indicating that sample belongs to k^{th} family and zeros everywhere else. The extended model is then formalized as

$$p(x|z,M) = \prod_{k=1}^{K} p(x|M_{k\cdot})^{z_k}, \tag{2}$$

where $M_{k\cdot} = \mu$ introduced in Equation (1) for k^{th} malware family.

Since the affiliation of sample x to particular malware family is unknown, the variable z is itself a random variable with *categorical distribution* with K possible outcomes parametrized by a vector $\pi \in (0,1)^K$, $\sum_{k=1}^{K} \pi_k = 1$. The probability $p(z|\pi)$ then describes the probability of malware family described by vector z and is formalized as

$$p(z|\pi) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{3}$$

The complete model is then parametrized by the vector π determining the probability of observing particular malware family and by the matrix $M = (\mu_{ki})_{k,i=1,1}^{K,D}$ characterizing the use of resources by individual malware families. Formally it is defined as

$$p(x) = p(z|\pi)p(x|z,M) \tag{4}$$

$$= \prod_{k=1}^{K} \pi_k^{z_k} \cdot \prod_{i=1}^{D} \left(M_{ki}^{x_i}(1-M_{ki})^{(1-x_i)} \right)^{z_k}. \tag{5}$$

These equations describe a generative model which has the following interpretation. A binary belongs to the malware family j with probability π_j and during its execution it interacts with i^{th} resource with probability M_{ji}. The advantage of the probabilistic formulation is that it captures variability in malware families and their use of resources. It also allows individual binaries from the same malware family to use slightly different resources since the actual usage of resources is a realization of the probability distribution defined in Equation (1).

The parameters π and M are learned (inferred) from observed data $X = \{x_1, \ldots, x_N\}$ (executions of N malware binaries in the sandbox). The membership of samples to the malware family is captured by variables $\{z_1, \ldots, z_N\}$ arranged in binary matrix Z where $z_{nk} = 1$ if n^{th} malware is a member of k^{th} family. Since this membership is unknown, Z is treated as a hidden unobserved variable and is inferred during the learning process as well. After the inference, Z contains assignments of individual samples to different malware families (clusters).

Using the above notation, the likelihood function of observed data with parameters π and M is given by

$$p(X,Z|\pi,M) = p(Z|\pi) \cdot p(X|Z,M)$$

$$= \prod_{n=1}^{N} \prod_{k=1}^{K} \left(\pi_k^{z_{nk}} \prod_{i=1}^{D} \left(M_{ki}^{x_{ni}}(1-M_{ki})^{(1-x_{ni})} \right)^{z_{nk}} \right), \tag{6}$$

corresponding to the well known *Bernoulli mixture model* [8], which has been successfully used in many different areas [28], [23], [33]. Since it is not possible to derive analytical formula for posteriori distribution of the parameters $p(\pi, M|X)$ and sampling methods [20] are computationally infeasible due to the enormous number of dimensions, a well-known *expectation-maximization (EM)* algorithm [17], [8] is adopted. It iteratively optimizes the likelihood function $p(X, Z|\pi, M)$

until convergence is reached. In each iteration it alternates the following two steps:

1) In E-step the posterior probability of component k for a given sample x_n is approximated as
$$\gamma(z_{nk}^{(t+1)}) = \frac{\pi_k^{(t)} \prod_{i=1}^{D} p(x_{ni}|M_{ki}^{(t)})}{\sum_{k=1}^{K} \pi_k^{(t)} \prod_{i=1}^{D} p(x_{ni}|M_{ki}^{(t)})}$$
where values of parameters π and M are fixed on values from previous iteration (indicated by superscript t).
2) In M-step the complete-data likelihood function (6) is maximized with respect to parameters π and M using the values $\gamma(z_{nk}^{(t+1)})$ estimated in E-step which leads to following update equations
$$\mu_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}^{(t+1)}) x_n$$
$$\pi_k^{(t+1)} = \frac{N_k}{N}$$
where $N_k = \sum_{n=1}^{N} \gamma(z_{nk}^{(t+1)})$ represents the effective number of samples in component k.

The algorithm is started with random initialization of $\gamma(z_{nk})$ using Dirichlet distribution with parameter $\alpha = 10^{-1}$ as it provides good level of sparsity and the values of parameters π and M are recomputed during first iteration.

B. Model application

Since the main purpose of the proposed model is to assist the human analyst by grouping unknown binaries into coherent clusters, prioritizing their analysis, and extracting their characteristics we now describe the application of the model described in previous section to the problem of clustering sandboxed samples.

The processing pipeline displayed in Figure (1) starts with preprocessing of malware traces recorded in sandbox and extraction of vector representation $x_n \in X$ for every sample (see Section (IV)). Next, we estimate parameters Z, M and π of the proposed model using this data X. The main idea is that each malware family or its variant uses specific resources and therefore malware samples should form clusters in $\{0,1\}^D$ space, ideally with one cluster containing samples from one malware family. When the model is fitted to data X, this assumption is reflected by samples from the same malware family following similar distribution over the hidden variable Z. The identifier of the most probable cluster for a given sample x_n is then obtained as

$$z^* = \underset{z \in \{1,\dots,K\}}{\operatorname{argmax}} \; p(z|x_n, \pi, M) = \underset{k \in \{1,\dots,K\}}{\operatorname{argmax}} \; \gamma(z_{nk})$$

and serves as a cluster label for given sample x_n. It is possible that one malware family spreads over multiple clusters, which is mainly caused by their modular design as different modules operate with different system resources. Nevertheless the behavior of malware binaries inside one cluster should be relatively uniform and it should be easy to create behavioral indicators for them.[2]

[2]In the text malware families and their variants are not differentiated because the model does not make a difference between them.

To further simplify the use of outputs, the clusters should be ordered such that a large number of samples can be analyzed quickly. The proposed heuristic (further referred as a *score*) prioritizes clusters of samples with homogenous behavior as their analysis is easier for human analyst and at the same time it promotes the largest clusters since their quick identification has higher impact. The purity of a cluster is approximated by the inverse value of entropy of the distribution defined by Equation (1) and the size is approximated by its probability π_k. Note that values of both the probability of a cluster π_k and parameters of distribution $p(x|\mu_k)$ are estimated during fitting the model to unknown data X. The prioritization score for cluster k is then formalized as

$$s(k) = \frac{\pi_k}{-\sum_{i=1}^{D} H(\mu_{ki})}, \text{ where} \tag{7}$$
$$H(\mu_{ki}) = M_{ki} \cdot \log M_{ki} + (1 - M_{ki}) \cdot \log(1 - M_{ki}) \tag{8}$$

The final ranking of clusters is obtained by computing the score for every cluster and ordering them in the decreasing order.

The last step of the analysis is the extraction of behavior indicators (BIs) that provide an insight into the behavior of analyzed samples. In the proposed model BIs are represented by system resources (files, mutexes, network communications and registry keys) used by the samples during the sandbox run. A behavior indicator that is a good candidate for an *indicator of compromise* (IOC) should be frequently used by samples from a given cluster and rarely used by samples from other clusters. This requirement is formalized for cluster k and system resource i as

$$r(i,k) = \ln M_{ki} + \frac{1}{K-1} \sum_{l=1, l \neq k}^{K} \ln(1 - M_{li}), \tag{9}$$

where the first term $\ln M_{ki}$ corresponds to the probability that samples from cluster k will interact with the system resource i ($p(x_i = 1, z = k|\pi, M) \to 1$) and the second term represents the average probability that the samples from other clusters will not interact with the system resource i ($\forall l \neq k, p(x_i = 1, z = l|\pi, M) \to 0$). Using average of probability overcomes the instability which can demote system resources common for cluster k but used by the few samples from other clusters as well in favor to system resource rarely used by samples from cluster k and not used by any other samples. This is caused by the fact that without the normalization term $\frac{1}{K-1}$ the term $\sum_{l=1, l \neq k}^{K} \ln(1 - M_{li})$ would have disproportionally higher influence than the term $\ln M_{ki}$. The variable $r(i,k)$ then provides a ranking score for the system resources within the cluster k where higher values indicate good candidates for IOCs.

IV. Data representation

The model described in the previous section assumes that execution of malware's actions involves interactions with system resources visible at the operating system level. Malware

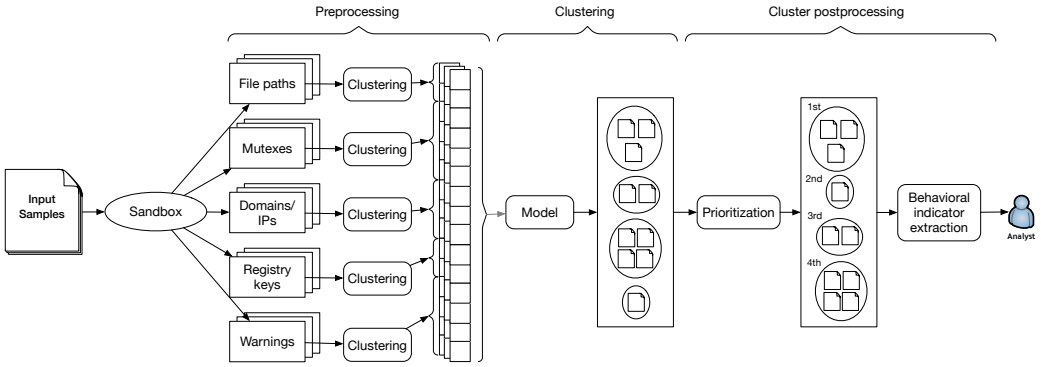


Figure 1: Schema of the proposed approach that chains the preprocessing (building data representation–see Section IV), clustering and preparation of clusters for human analyst (see Section III-B).

sample is then modeled as a binary vector $x \in \{0,1\}^D$ where $x_i = 1$ iff the sample interacted with i^{th} system resource and 0 otherwise. However, the randomization frequently employed by malware authors prevents to use every system resource as individual feature since (1) the number of such features can easily reach millions and (2) only a small number of samples will interact with the same resource which would limit the discovery of malware families. This section describes approaches to reduce the dimensionality and thus effectively mitigating both of these problems.

To model the malware interactions with system resources this work uses an approach we have originally proposed in [53], where each binary executed in a sandbox is modeled as a set of system resources the binary interacted with. Since the number of system resources can be different for every sample we have adopted a *multiple instance learning* (MIL) where each sample consists of a set of *instances* of different size. In our scenario an instance corresponds to a pair of name and type of a system resource the sample interacted with during its execution in sandbox.

A recent review of MIL algorithms [2] lists various approaches to overcome the variability in sample sizes. The approach adopted in this work (outlined in Algorithm 1) belongs to a class of *vocabulary-based method*. First, all instances I of all samples S are clustered per given resource type by a Louvaine clustering method [9] combined with suitable similarity measure discussed below. Then each sample is described by a fixed-dimensional binary vector with length equal to the size of vocabulary (number of clusters of instances) such that the element of this vector equals to 1 iff there is an instance of the sample close to the particular cluster of instances. Since the number of clusters is much smaller than the number of all system resources, this approach reduces the dimensionality of the problem and effectively removes the randomization of resource names.

The above clustering and the following encoding is performed separately for each resource type: files, mutexes, registry keys and network communication, as the corresponding similarity metrics are specifically designed for individual types

Algorithm 1 Projection of samples S into binary vector.

```
1:  function PROJECT(S)
2:      X ← ∅
3:      I ← extractInstances(S)
4:      C ← cluster(I)            ▷ Clustering of instances (separately for individual types)
5:      for all s ∈ S do
6:          I_s ← extractInstances(s)
7:          x ← 0⃗
8:          for all i ∈ I_s do
9:              c* ← nnSearch(i, C)   ▷ Finds closest center c* to instance i.
10:             x[c*] ← 1
11:         end for
12:         X ← X ∪ {x}
13:     end for
14:     return X
15: end function
```

of system resources in order to precisely capture their unique properties. Note that, the warnings generated by the sandboxing environment are used directly (each warning is treated as a separated cluster), since their number is very limited. Once all samples are encoded into fixed-dimensional vectors, we can apply probabilistic modeling described in Section III to discover different malware families.

Since the clustering of instances is an essential component of the above algorithm, the definitions of similarities over instances (resource names) greatly influence the accuracy of the system, and therefore they should reflect properties of different types of system resources. The rest of this section discusses a specific similarity metric for each type of resource the malware interact with, namely for files, mutexes, network hostnames, and registry keys. For the comprehensive discussion reader is referred to [53].

A. Similarity between file paths

Although viewing file paths as strings would allow to utilize various similarities [34], [43], [37], the file systems were designed as tree structures with names of some folders (fragments of the path) being imposed by the operating system and the distance should reflect that. For example two files with paths `/Documents and Settings/Admin/Start Menu/Programs/Startup/tii9fwliiv.lnk` and `/Documents and Settings/Admin/Start Menu/Programs/Accessories/Notepad.lnk` share large parts of their paths and common string similarities will return high similarity score, but they serve very different purposes. The first file is a link to an application executed after the start of the operating system (OS), while the second is a regular link in the Start menu in Windows OS. Another aspect that prohibits the use of common string similarities is their computational complexity (typically $O(n^2)$ where n is the length of the string). The complexity combined with the number of resources to be clustered (in the order of millions) leads to unfeasible time requirements. This motivates the design of a similarity that is fast and takes into the account the tree structure of the file system, special folders, and differences between folders and filenames.

The proposed similarity $s(x, x')$ of two file paths x and x' is defined as

$$s(x, x') = \exp\left(-w^T f(x, x')\right), \tag{10}$$

where w is a vector of weights and $f(x, x')$ is a function extracting a feature vector from file paths x and x'.

The function f in (10) captures differences between the two paths x and x' by a fixed-dimensional vector. It first splits both paths x and x' into fragments x_i and x'_i using OS specific path separator[3], then in the cases of MacOS and Windows changes all characters to lowercase, and then assigns all fragments into one of the following four categories:

1) *known folder* – fragment x_i is a well known folder in the list of folders imposed by the operation system (e.g. `Windows`, `Program Files`, `System32`, etc.),
2) *general folder* – fragment x_i is a not-well-known folder (e.g. unknown folders in `Program Files`, randomly generated folders in Internet Explorer cache folder, etc.),
3) *file* – fragment x_i is file,
4) *empty* – artificial fragment used for padding the paths in cases when paths x and x' have different depths.

When all fragments are assigned to one of the above classes, their dissimilarity is captured by the function f as

$$f(x, x') = (f_{KK}, f_{KG}, f_{KF}, f_{KE}, f_{GG}, f_{GF}, f_{GE}, f_{FF}, f_{FE}),$$

where

- f_{KK} is the number of fragments on the same level that are both classified as *known folder* and are not equal,
- f_{GG} is the sum of Levenshtein distances [34] between all fragments on the same level that are classified as *general folder*,
- f_{FF} is the sum of Levenshtein distances of all fragments on the same level that are classified as *file*,
- f_{KG}, f_{KF}, f_{KE}, f_{GF}, f_{GE}, f_{FE} are the sums of all fragments of the same level and are classified as *known* and *general folder, known folder* and *file, known folder* and *empty, general folder* and *file, general folder* and *empty,* and *file* and *empty* respectively.

The weight vector w in (10) captures the contribution of individual elements of the feature vector $f(x, x')$. Imposing condition $w \geq 0$, in combination with construction of function f, bounds the value of the similarity function (10) $s(x, x') \in [0, 1]$ such that the similarity functions returns 1 (or values close to 1) if x and x' belong to the same class (files in `/temp/` directory, cache of the Internet Explorer, files in system directory, etc.) and values approaching 0 if they belong to different classes. Since the similarity function (10) was inspired by the popular Gaussian kernel, the parameter vector w was optimized using the Centered Kernel Target Alignment [15] (CKTA), which is a method to optimize kernel parameters. The example of using the similarity and optimization of the parameter vector w is further described in [53].

B. Similarity of network traffic

To define the similarity between network resources one has to overcome the randomization often employed by malware authors that render trivial similarity based on names of network resources (domains, IPs) ineffective. To escape blacklisting command and control (C&C) channels of malware, its authors use various techniques to hide and obscure C&C operation. Popular approaches include randomization of domain names by generating them randomly (DGA), quickly changing hosting servers and / or domain names by fast flux, or using large hosting providers like Amazon Web Services to hide among legitimate servers, etc. These techniques are relatively cheap (e.g. registering a new *.com* domain costs ~3USD per 1 year) and they allow for variation in domain names without updating disseminated malware binaries. In contrast, switching from one C&C paradigm to another requires such an update and therefore occurs relatively infrequently. These two properties contribute to each malware family using specific patterns of domain names, paths, and parts of URLs. Exploiting these patterns allows to group domain names into clusters. In this work the similarity in network traffic is defined only for HTTP/HTTPS protocol, because it is presently the default choice for malware authors as it is rarely filtered. The extension to other network traffic is possible [31].

The similarity in URL patterns used in this work has been adopted from [29], which has proposed to cluster domain names so that each cluster contains domains of one type / for one family of malware. The calculation of similarity starts by grouping all HTTP/HTTPS requests using the domain names. Then the model of each domain name is built from path and query strings, transferred bytes, duration of requests and inter-arrival times (time spans between requests to the same domain) of individual requests to it. Finally, these models are used to calculate the similarity function between two domain names

[3]Unixes and MacOS uses '/' as a path separator, Windows uses '\'.

in the clustering. Since the calculation of the similarity is out of scope, a reader is referred to an original publication [29] for details.

C. Similarity between mutex names

Mutex (*Mutual exclusive object*) is a service provided by most modern operating systems to synchronize multi-threaded and multi-processes applications. This mechanism is popular among malware authors to prevent multiple infections of the same machine, because running two instances of the same malware can cause conflicts limiting the potential revenue. Mutexes are identified by their name, which can be an arbitrary string. The naming scheme is challenging for malware authors, because the names cannot be static, which would make them good indicators of compromise of a particular malware, but they cannot be completely random either, because two independent binaries of the same family would not be able to check the presence of each other. Therefore malware authors resorted to pseudo-deterministic algorithms or patterns for generating mutex names. For some malware families these patterns are already well known, for example Sality [54] uses mutex names of the form "<process name>.exeM_<process ID>_"-explorer.exeM_1423_.

Since operating systems do not impose any restrictions on the names of mutexes, they can be arbitrary strings. Therefore standard string similarities such as Levenshtein distance [34], Hamming distance [24], Jaro-Winkler distance [14], etc. can be used. In experiments presented in Sections V Levenshtein was used, as it gives overall good results.

D. Similarity between registry names

In Microsoft Windows operating system, the primary target of the majority of malware, registry serves as a place where programs can store various configuration data. It is a replacement of configuration files with several improvements such as strongly typed values, faster parsing, ability to store binary data, etc. The registry is a key-value store, where key names have the structure of a file system. The root keys are `HKEY_LOCAL_MACHINE`, `HKEY_CURRENT_USER`, `HKEY_CURRENT_CONFIG`, `HKEY_CLASSES_ROOT`, `HKEY_USERS` and `HKEY_PERFORMANCE_DATA`; some root keys also have sub-keys with specific names (`Software`, `Microsoft`, `Windows`, etc.). Due to similarity with a file system, the similarity distance is the same as the one defined in Subsection IV-A, but with a different set of names of known folders and a weight vector optimized on registry data rather than on files.

V. Evaluation

This section presents the results of the experimental comparison of the proposed model to AMAL [39] on the problem of clustering of unknown samples. AMAL is the closest prior art, as it clusters malware binaries on basis of their interaction with system resources. Furthermore, clustering using the generative model is compared to the popular K-Means [27] algorithm with k-means++ initialization [4] to observe benefits or disadvantages of the probabilistic formulation. Lastly, we evaluate the cluster prioritization and extraction of behavioral indicators.

A. Data set description and performance metric definition

The dataset used for experimental evaluations contained random selection of samples from files that were submitted to *AMP ThreatGrid* [13] sandboxing service by its customers between October 24, 2016 and December 12, 2016 and as such can be considered as a representative example of the threat landscape. All samples were analyzed by *VirusTotal.com* service [22] and their labels were determined using *AVClass* tool [50]. Then, malware families with less than hundred samples were removed, which yielded 130 198 samples. The exact numbers of samples of individual malware families are summarized in Table IX in Appendix B. Since all three compared solutions are applicable to any file type as long as the underlying sandboxing solution is able to instrument them, the evaluation dataset contained samples of various file types such as 32bit PE executables, 64bit PE executables, JavaScripts, VisualBasic scripts, etc. The complete list of file types along with number of samples is summarized in Table VIII in Appendix A.

All samples were analyzed in sandbox by AMP ThreatGrid [13] service, using Windows 7 64bit (71% samples) environment, as it is the most popular OS at the time of writing[4], and Windows XP (29% samples) environment, since it is still widely deployed on embedded machines such as ATMs. The sandboxing environment (OS) for individual samples was selected randomly with exception of 64bit PE executables that were executed solely on Windows 7 64bit. Virtual machines were connected to the Internet without any filtering or restrictions that would intervene connections to command & control infrastructure or other servers. Note that the approach proposed in this paper is not specific to AMP ThreatGrid. Same similar information can be obtained with a number of different sandboxing solutions such as Cuckoo [45], Ether [18], or CWSandbox [55].

Although clustering algorithms are technically unsupervised, they usually contains several hyper-parameters that need to be tuned to achieve good performance. Samples were therefore divided according to the time of their first observations, such that those collected before October 27th, 2016 (17 064 samples further referred as *training set*) were used to find the optimal configurations of all methods and remaining samples (113 134 samples further referred as *testing set*) were used for the final evaluations. The time-based split makes the optimization phase more realistic since in real-world scenario all parameters have to be set in advance with the risk that some malware families will not be available.

The performance of all methods was evaluated using *homogeneity, completeness* and *V-measure* [49]. Homogeneity, h, measures the "purity" of resulting clusters and equals to one if all labels in each cluster are the same. It can be seen as

[4]According to http://www.w3schools.com/browsers/browsers_os.asp Windows 7 has 33.2% market share against 0.9% covered by Windows XP, 10.2% covered by Windows 8 and 33.1% covered by Windows 10.

	h	c	v_1	avg. #clusters
AMAL	0.602	0.499	0.546	5856
Proposed model	**0.767**	**0.609**	**0.679**	492
K-Means	0.690	0.561	0.619	200

Table I: Homogeneity (h), completeness (c) and V-measure (v_1) estimated on test dataset for model, K-Means and AMAL.

a clustering's counterpart to precision used in classification scenarios. Completeness, c, measures how well individual labels are grouped together, which makes it closely related to recall used in classification. It equals to one if samples with one label are all in one cluster. They are combined together with equal importance into a single value by *V-measure*, v_1, defined as

$$v_1 = \frac{2 \cdot h \cdot c}{h + c}$$

that resembles the F-measure from classification scenarios. Additionally, the quality of individual clusters is measured by *purity* [57], [58] defined as number of samples from most prevalent malware family in given cluster normalized by its size.

B. Hyper-parameter optimization

AMAL's hyper-parameters consists of the metric that defines similarity between samples (cosine, correlation, hamming or jaccard), the linkage method in hierarchical clustering (average, centroid, complete, median, single, ward), and the value of threshold t for the hierarchical clustering. For every combination of hyper-parameters we have estimated the value of V-measure on the training data and the one with highest value (jaccard metric, average linkage method and threshold $t = 0.25$) was used in final evaluation.

The only parameter in the proposed method is the number of components K, which corresponds to the number of clusters. Setting this number is generally unresolved problem, since one typically does not know the number of clusters in data in advance. The training data were therefore clustered with with different number of components $K = \{50, 100, 200, 500, 1000, 2000\}$ and the K with the highest V-measure was used in the evaluation. Since the EM algorithm depends on starting conditions (the problem may have many local extrema), the accuracy of the clustering for every configuration was estimated using five execution. As the optimal number of clusters was then determined $K = 500$.

Similarly, K-Means clustering algorithm requires specification of the number of cluster in advance as well, which has been determined using the same approach as in the previous paragraph. The optimal number of clusters for K-Means was $K = 200$.

C. Clustering performance

The first part of experimental evaluation compares the clustering performance of the proposed model, K-Means clustering algorithm with the same features as the proposed model, and AMAL. The performance is measure as homogeneity, completeness and V-measure estimated on testing dataset for all three approaches using settings described above and is summarized in Table I. In order to eliminate the influence of the random initialization of the proposed model and K-Means, the testing data were clustered 10times and values summarized in Table I represent average values of respective performance measures. The last column in Table I contains average number of clusters generated by evaluated approaches on testing data. Due to the probabilistic definition of the proposed model, the resulting number of clusters can be lower that the set threshold (in our case $K = 500$), as it balances the complexity of the model and its fit to the data.

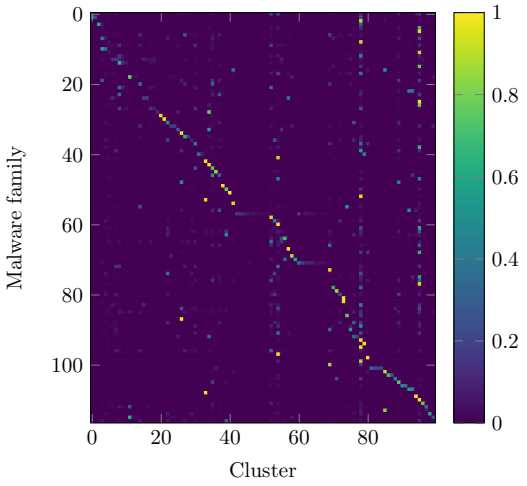


Figure 2: Confusion matrix of proposed model for 100 largest clusters.

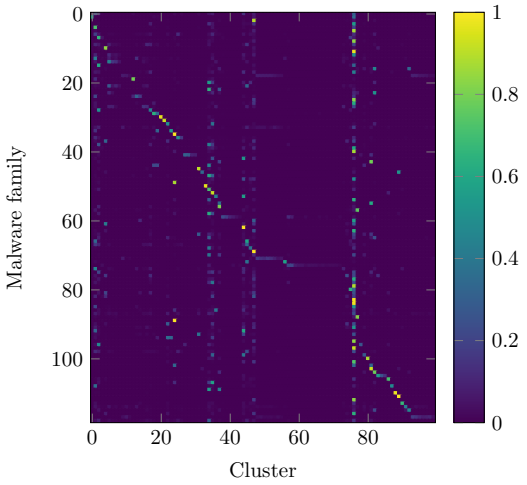


Figure 3: Confusion matrix of K-Means for 100 largest clusters.

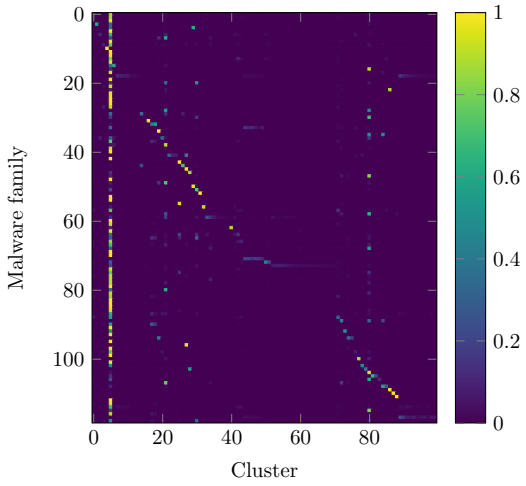


Figure 4: Confusion matrix of AMAL method for 100 largest clusters.

As can be seen from the values of the V-measure, the proposed model outperforms both AMAL and K-Means. Clusters recovered by the model are the purest and the individual malware families are the least fractioned into multiple clusters. These results are further displayed in more details in Figures 2, 3, 4 that show confusion matrices of largest 100 clusters for proposed model, K-Means and AMAL respectively, normalized by the number of samples per family (i.e. every point displays ratio of samples from particular malware family clustered into particular cluster). Although the view is not complete, these selections covered 87%, 90% and 80% of samples for proposed model, K-Means and AMAL respectively which represent significant portion of samples.

Confusion matrices show that the low performance of the AMAL method (Figure 4) is mainly caused by the fact that it failed to separate large portion of samples ($\sim 37\,000$ samples) and grouped them into single cluster. This indicates that hand-designed features are not able to precisely capture the behavior of a new, previously unseen malware. Comparison of confusion matrices of proposed method and K-Means revealed that although K-Means provided better results than AMAL, it was still outperformed by the proposed method as it created clusters that failed to separate several malware families. This fact is significant for clusters 2–5, 30–50 and mainly cluster 76 that grouped more than 20 different malware families such as *lotoor* (malware family #61), *pdfka* (malware family #84) or *zegost* (malware family #117) correctly clustered by the model. These results indicate that the probabilistic model correctly captures malware behavior. It is caused by the fact that K-Means is not able to distinguish minor differences between individual malware families (e.g. one file specific to particular malware family) as the euclidean distance embedded in the K-Means smooths the distances between individual malware samples. Although it would be possible to find specific combination of distance metric, clustering algorithm and its parameters that provides better or at least comparable results, such combination would introduce high bias as it would be overfitted to our data and would lack the interpretability of the proposed model.

	h	c	v_1	avg. #clusters
AMAL	0.542	0.503	0.521	6224
Proposed model	**0.761**	**0.523**	**0.620**	499
K-Means	0.722	0.495	0.588	200

Table II: Homogeneity (h), completeness (c) and V-measure (v_1) estimated on test dataset mixed with legitimate samples for model, K-Means and AMAL.

Detailed analysis of clusters created by the proposed model revealed that it correctly discovered several malware families such as *winner, waldek, ircbot, downloadguide*, *downloadassistant*, *kolabc, dlhelper, softpuls,* etc., but some malware families such as *adwind, gamarue, locky*, *nemucod* or *zbot* were split into multiple pure clusters. This is caused by the fact that a single label may refer to multiple variants of the same malware family (*locky* or *adwind*), by modular design of some malware families (*zbot*), or by the fact that some malware families deliver additional infections (*nemucod*) with different behavior. However, several clusters contained samples from multiple malware families such as *atraps* (also known as *cryptowall*), *delf, flystudio, palevo* or *winsecsrv*. Analysis of these clusters revealed that these samples neither run properly (e.g. crashed due to missing libraries) nor exhibited their true behavior (advanced anti-vm protection, no command from bot master, etc.), and the only interactions with system resources recorded during sandboxing were caused by the operating system handling the crash of the binary.

This exposes the main limitation of the proposed solution, because if malware samples do not exhibit any behavior, the model is not able to cluster them correctly as it does not have enough information to distinguish individual malware families. In case that malware samples are equipped with advanced anti-vm technique we can improve the results by optimizing the sandboxing environment (anti-anti-vm/anti-anti-sandbox techniques) and thus limit its detectability by malware. Although such approach is successful for some malware families (e.g. *gamarue*), malware authors quickly adapt the anti-vm techniques to avoids detection (e.g. *atraps*). As the hardening techniques are usually tailored for specific sandboxing solution, their description is out of the scope of this paper and reader is referred to [11] for deeper analysis of this problem.

To conclude the first part of experiment, the proposed solution significantly outperforms the prior art and it is able in most cases correctly recognize binaries from the same family under the condition that the binary has exhibit its true behavior in sandbox.

D. Clustering robustness

In reality it is not guaranteed that the analyzed data will contain only malicious samples. To test the robustness of the

proposed model, we have mixed legitimate samples into our testing dataset and evaluated their influence on the performance of the compared clustering algorithms. We have added 55 243 samples that were analyzed in the same sandboxing environment as malicious samples (see Section V-A) and were verified in the VirusTotal.com service to have 0 positive hits, i.e. none of the AV engines deployed in the service marked these samples as malicious. This mixed dataset was then processed by AMAL, proposed model and K-Means. Note that the methods used the same settings of hyper-parameters as in previous experiments (see Section V-A details).

The results summarized in Table II show a drop in the value of the V-measure for all three methods which indicates that the legitimate samples add noise into the data an thus worsen their results. A close analysis revealed two, principally different, effects of this noise to the clusters generated by compared methods. The results of AMAL show significant drop in the homogeneity (0.602 for only malicious data vs. 0.542 for mixed data) which corresponds to the fact, that AMAL grouped majority of the legitimate samples (~52 000 legitimate samples, i.e. almost 95% of legitimate samples) together with large portion of malicious samples (35 000 malicious samples) into single large cluster (~88 000 samples). Problem is that analysis of such large, heterogenous cluster is nearly impossible which effectively rules out analysis of more than half of the mixed dataset.

An opposite effect can be seen in the results produced by the proposed model and K-Means. Both these methods split the legitimate data into multiple clusters which corresponds to the drop in completeness—0.609 for only malicious data vs. 0.523 for mixed data in the case of proposed model, and 0.561 vs. 0.495 in the case of K-Means. However, since these clusters are relatively pure and well defined, theirs analysis is much easier compared to the analysis of single large cluster produced by AMAL. The direct comparison of results produced by the proposed model and K-Means revealed that although K-Means split the legitimate samples into lower number of clusters than proposed model (50 clusters with legitimate samples produced by K-Means vs. 200 clusters with legitimate samples produced by the proposed model) these clusters are mixed with malicious samples which hardens the analysis which is reflects the lower value of homogeneity for K-Means (0.761 for proposed model vs. 0.722 for K-Means). The overview of 25 largest clusters produced by AMAL, K-Means and proposed model is summarized in Tables X, XIII and XII respectively. Note that the top 25 clusters generated by proposed model and prioritized by the score proposed in Section III-B are listed in Table XI.

E. Prioritization score

As has been discussed in Section III-B, resulting clusters should be presented to a security analyst sorted to facilitate quick analysis of a large number of samples. Instead of selecting clusters at random, the optimal prioritization should focus the analyst's attention to large and pure clusters that can be easily analyzed (ideally to clusters that contain samples from single family).

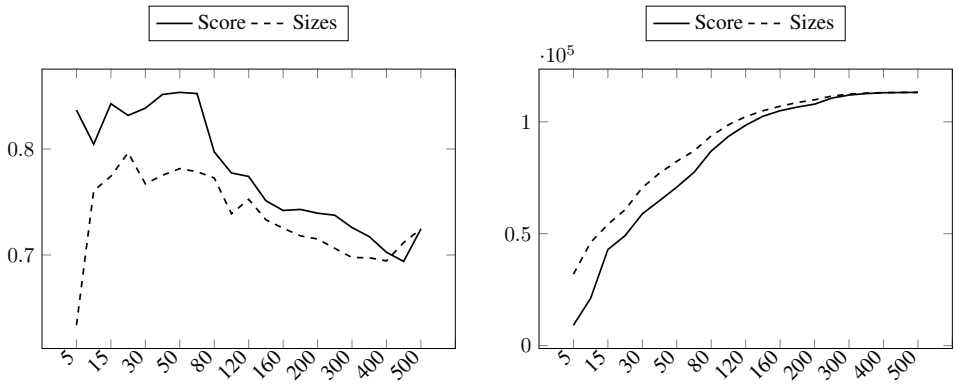
Cluster	Size	Malware categories with coverage
1	**1302**	**trojan (1.00)**
2	**811**	**banking trojan (1.00)**
3	4337	unknown (0.41), worm (0.26), dropper (0.18)
4	**531**	**banking trojan (1.00)**
5	2135	dropper (0.92), banking trojan (0.08)
6	6320	dropper (0.33), unknown (0.33), trojan (0.14), rat (0.08), information stealer (0.07)
7	**495**	**banking trojan (1.00)**
8	141	dropper (0.94), banking trojan (0.06)
9	4228	dropper (0.62), banking trojan (0.37)
10	**904**	**rat (0.99)**
11	5967	dropper (0.74), ransomware (0.26)
12	**374**	**trojan (1.00)**
13	**91**	**unknown (1.00)**
14	5942	information stealer (0.96)
15	**9366**	**ransomware (0.98)**
16	1197	unknown (0.33), dropper (0.30), trojan (0.09), rat (0.07), ransomware (0.06), information stealer (0.05)
17	191	dropper (0.90), banking trojan (0.10)
18	**1475**	**ransomware (1.00)**
19	2876	banking trojan (0.92), unknown (0.08)
20	**456**	**banking trojan (1.00)**
21	**1556**	**rat (0.98)**
22	**1402**	**ransomware (1.00)**
23	**2273**	**banking trojan (1.00)**
24	1271	trojan (0.78), unknown (0.11), rat (0.05)
25	**883**	**ransomware (0.99)**
Total	56 524	
Total pure	21 919	

Table III: Top 25 clusters created by proposed model and prioritized with *score* with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

To verify this assumption, clusters identified by the proposed model from the testing data were ordered by prioritization score proposed in Section III-B (further referred to as *Score*), and by their size (referred to as *Size*). Figure 5a shows the average purity for first 5, 10, 15, 20, 30, etc. clusters ordered by the Score and those ordered by the Size. Similarly, Figure 5b shows the cumulative number of samples for first 5, 10, 15, 20, 30, etc. clusters prioritized by the Score and the Size. We observe that the purity of clusters ordered by Score is much higher than of those ordered by Size, while cumulative numbers of samples of top n clusters are very similar. This means that the proposed Score promotes purer clusters without too much sacrificing their size.

In order to further verify these results we have performed deeper analysis of top 25 prioritized by the Score and the Size. Tables III and IV provide details about malware categories the samples from particular cluster belong to and their ratio. Note that malware category listed as *unknown* covers malware families that we were not able to reliably assign to any category. The statistics indicate that the Score promotes larger number of pure clusters[5] (14 out of 25) compared to the Size (7 out of 25) and these clusters cover larger portion of samples, namely 21 919 samples in the case of Score vs. 18 510 samples in the case of Size. In order to provide complete results, we include results obtained on clusters created by K-Means (Table VI)

[5]Here, we consider cluster with at least 98% samples from the same category as pure rather than 100%.



(a) Average purity

(b) Cumulative number of samples.

Figure 5: Average purity (figure on the left) and cumulative number of samples (figure on the right) for top 10, 20, 30, etc. clusters for proposed prioritization (*Score*) and size-based prioritization (*Size*).

Cluster	Size	Malware categories with coverage
1	**9366**	**ransomware (0.98)**
2	6320	dropper (0.33), unknown (0.33), trojan (0.14), rat (0.08), information stealer (0.07)
3	5967	dropper (0.74), ransomware (0.26)
4	5942	information stealer (0.96)
5	4337	unknown (0.41), worm (0.26), dropper (0.18)
6	4228	dropper (0.62), banking trojan (0.37)
7	2876	banking trojan (0.92), unknown (0.08)
8	2661	dropper (0.97)
9	**2273**	**banking trojan (1.00)**
10	2135	dropper (0.92), banking trojan (0.08)
11	1726	dropper (0.75), ransomware (0.25)
12	1691	trojan (0.45), dropper (0.39), unknown (0.10)
13	1623	dropper (0.83), ransomware (0.17)
14	**1556**	**rat (0.98)**
15	**1475**	**ransomware (1.00)**
16	**1402**	**ransomware (1.00)**
17	**1302**	**trojan (1.00)**
18	1292	dropper (0.74), ransomware (0.26)
19	1283	information stealer (0.96)
20	1271	trojan (0.78), unknown (0.11), rat (0.05)
21	1266	ransomware (0.82), unknown (0.18)
22	1197	unknown (0.33), dropper (0.30), trojan (0.09), rat (0.07), ransomware (0.06), information stealer (0.05)
23	**1136**	**ransomware (1.00)**
24	1005	unknown (0.53), information stealer (0.31), rat (0.08)
25	964	information stealer (0.73), unknown (0.21)
Total	66 294	
Total pure	18 510	

Table IV: Top 25 clusters created by the proposed model prioritized by size with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	37303	information stealer (0.27), unknown (0.21), rat (0.17), dropper (0.16), trojan (0.07)
2	6263	dropper (0.46), banking trojan (0.25), trojan (0.13), pua (0.08)
3	3678	dropper (0.62), banking trojan (0.38)
4	2889	banking trojan (0.92), unknown (0.08)
5	2472	ransomware (0.58), unknown (0.22), pua (0.08), dropper (0.08)
6	**2105**	**ransomware (0.99)**
7	**1710**	**banking trojan (1.00)**
8	1465	trojan (0.92), unknown (0.08)
9	**1438**	**ransomware (0.99)**
10	1153	ransomware (0.91)
11	1099	ransomware (0.33), dropper (0.30), trojan (0.25), unknown (0.12)
12	**1066**	**banking trojan (0.99)**
13	**1013**	**ransomware (0.98)**
14	988	trojan (0.90), unknown (0.08)
15	**889**	**ransomware (0.99)**
16	888	dropper (0.64), ransomware (0.36)
17	861	trojan (0.93), dropper (0.05)
18	819	dropper (0.88), ransomware (0.12)
19	**806**	**dropper (0.99)**
20	**750**	**ransomware (0.98)**
21	**686**	**worm (0.99)**
22	**664**	**dropper (0.98)**
23	658	dropper (0.76), ransomware (0.24)
24	**601**	**ransomware (1.00)**
25	**595**	**ransomware (0.98)**
Total	72 859	
Total pure	12 323	

Table V: Top 25 clusters created by AMAL and prioritized by size with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

and AMAL (Table V) prioritized by their size as the Score is not applicable. The detailed statistics regarding the clusters created by AMAL show that in top 25 clusters there are 12 considered pure, but they cover only 12 323 samples which is almost half of samples covered by the proposed model. Similarly, top 25 clusters created by K-Means contains 6 pure clusters, covering 9520 samples. These numbers indicates that the proposed model combined with the prioritization score correctly promotes clusters so that the security analyst can analyze large number of samples efficiently.

Cluster	Size	Malware categories with coverage
1	12719	unknown (0.30), dropper (0.20), rat (0.15), worm (0.09), trojan (0.09), information stealer (0.05),
2	4211	information stealer (0.96)
3	3478	banking trojan (0.90), unknown (0.06)
4	3450	dropper (0.71), ransomware (0.29)
5	3192	dropper (0.75), banking trojan (0.15), unknown (0.06)
6	3073	dropper (0.79), ransomware (0.21)
7	3044	unknown (0.31), dropper (0.26), pua (0.18), trojan (0.12), ransomware (0.10)
8	3007	unknown (0.31), rat (0.28), information stealer (0.23), worm (0.08), trojan (0.06)
9	2850	unknown (0.42), dropper (0.20), rat (0.14), trojan (0.12)
10	**1943**	**ransomware (0.99)**
11	**1890**	**banking trojan (1.00)**
12	1766	dropper (0.79), unknown (0.15), banking trojan (0.05)
13	**1713**	**banking trojan (1.00)**
14	1574	unknown (0.41), dropper (0.25), trojan (0.09), rat (0.07), information stealer (0.06)
15	1548	unknown (0.40), trojan (0.31), pua (0.16)
16	1511	information stealer (0.97)
17	**1490**	**ransomware (1.00)**
18	1353	rat (0.91), unknown (0.05)
19	**1344**	**trojan (1.00)**
20	1328	trojan (0.75), unknown (0.12), rat (0.05)
21	1266	information stealer (0.95)
22	1143	information stealer (0.33), rat (0.31), unknown (0.24), trojan (0.09)
23	**1140**	**ransomware (0.99)**
24	1140	banking trojan (0.89), dropper (0.08)
25	1130	trojan (0.40), unknown (0.23), information stealer (0.20), rat (0.10), dropper (0.08)
Total	62 303	
Total pure	9520	

Table VI: Top 25 clusters created by K-Means and prioritized by size with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

F. Behavioral indicators

The last step of the analysis of a cluster is the extraction of behavior indicators (BIs) represented by the system resources that can be considered as indicators of compromise. To verify the performance of the ranking score proposed in Section (III-B), top 20 clusters identified by the model from the testing data and prioritized by Score were manually analyzed and searched for system resources (further referred as convicted BIs or convicted resources) that matched known IOCs for the most prevalent malware family in the given cluster. Known IOCs were collected from reports published by various AV companies (e.g. [46]) such as Symantec, ESET, TrendMicro, etc., or publicly available collections of various behaviors frequently exhibited by malware [3], [26]. Additionally, we have searched for BIs, that, although not listed in known IOCs, are highly suspicious and specific for given malware family and thus can be considered as additional potential IOCs recovered by the proposed approach.

Table VII summarizes the number of resources used by the malware samples in a given cluster, the rank of the first convicted resource and the ratio of samples that interacted with this resource. In the situation when human analyst performs manual inspection of generated clusters, this rank indicates the number of system resources that have to be manually analyzed in order to convict given cluster. Lower number therefore indicates better performance of the ranking score proposed in Section (III-B). In order to provide additional insight, the table contains numbers of convicted BIs in top 10 rank, total number of convicted BIs and number of BIs that can be considered as additional IOC (system resources that are not listed as known IOC but are highly suspicious, e.g. file `/windows/driiver/é¿ßp¡¬¿/é¿ßp¡«¬1.png`). Higher number of convicted BIs in the top 10 rank indicates that the ranking score is able to correctly recover large portion of known IOCs.

The Table VII shows that for 12 clusters out of 20 the ranking correctly promoted convicted IOCs to the top as the first convicted IOC has rank 1 or 2 and majority of all convicted IOCs is in the top 10 rank.[6] Additionally, the numbers of potential IOCs indicates that the ranking score recovers large number of additional, previously unknown, BIs that further describe the malware behavior. The lower ranks of the convicted system resources for clusters 14 and 18 are caused by the nature of the malware families dominant in these clusters. Although *bladabindi* malware uses system resources that are not frequently used by other malware families in our dataset (`/windows/syswow64/netsh.exe`, `/windows/microsoft.net/.../machine.config`, etc), they cannot be considered as IOCs since they are part of Windows operating system or other common software. In this case the only system resources that matched known IOCs are specific registry keys modified by only portion of samples from this cluster which lowers their rank. Similarly, *locky* malware uses internal components of the operating system that are specific to this particular family and thus are incorrectly prioritized over the actual known IOCs.

For 6 clusters we were not able to find any IOC that will match with the proposed BIs. In the case of clusters 3, 6 and 16 it is caused by the fact that these clusters contain samples that did not exhibit their true behavior. In case of clusters 11 and 13, it is caused by the nature of behavior of the malware dominant in this cluster. Similarly to *bladabindi,* malwares *nemucod* (cluster 11) and *fujacks* (cluster 13) interact with files specific for this malware. It may be an example of installation of .NET framework in the case of *fujacks* or using internal vbscript interpretter in the case of *nemucod*. However, these system resources are also used by legitimate software and as such cannot be considered as IOCs. This implies that the algorithm is able to correctly group malware samples from given malware families based on their interaction with legitimate resources alone, even if the samples do not create any specifically malicious system resource. This capability is highly important for analysis file-less malware which is specifically designed to reduce its foot print. In order not to sacrifice its functionality such malware has to interact with system resources that are part of the

[6]The example of matched IOCs is a path to malware's main binary, specific registry key, known C&C domain or mutex with specific name.

Cluster	Rank	#BIs	% samples	#known IOCs in top 10	#potential new IOCs in top 10	Cluster	Rank	#BIs	% samples	#known IOCs in top 10	#potential new IOCs in top 10
1	1	80	100%	4	5	11	N/A	42	N/A	N/A	N/A
2	1	19	100%	10	0	12	1	48	100%	1	6
3	N/A	20	N/A	N/A	N/A	13	N/A	150	N/A	N/A	N/A
4	2	15	100%	4	3	14	21	61	89%	0	9
5	1	15	100%	5	2	15	N/A	142	N/A	N/A	N/A
6	N/A	22	N/A	N/A	N/A	16	N/A	51	N/A	N/A	N/A
7	1	15	100%	4	3	17	1	15	100%	4	3
8	1	15	100%	4	3	18	11	37	84%	0	3
9	1	47	99%	1	6	19	1	52	99%	2	5
10	1	22	99%	3	1	20	1	16	100%	4	3

Table VII: Rank of first behavior indicator that matches known IOCs of most frequent malware family in generated clusters and percentage of samples covered with this behavior indicator. Additionally, we provide number of BIs in top 10 rank that matched known IOC and number of BIs that are not matched to known IOC but can be considered as additional potential IOCs recovered by the proposed approach. N/A indicates that no behavior indicator matched known IOC.

OS in a similar as *fujacks* or *nemucod* does. Cluster 15 contains *cerber* ransomware that encrypted various different files during sandboxing which can be easily recognized by malware analyst. However, the recorded behavior indicates that these samples belong to different variants for which we were not able to find any known IOCs.

VI. Discussion

The experimental results revealed, that when a binary (or all binaries of some malware family) does not perform any action reflected by interaction with files, mutexes, network communication, and registry keys, it clearly evades detection. An example of such malware is bitcoin miner that resides only in memory without any additional footprint (no operations with files, no operations with registry keys, no mutexes, very limited network communication). Such malware has to be carefully crafted to avoid any interaction with system resources (statically compiled to carry all libraries in the executable, limited network communication, no mutexes ensuring that only single instance is running on the same machine, no persistency after reboot, etc.). Fortunately, at the time of writing this work, this is not an easy task and the majority of malware authors choose to interact with system resources rather than sacrifice functionality.

Another limitation is the fact that a growing number of malware families are equipped with advanced anti-VM and anti-sandbox features (Gamarue/Andromeda [6]) and/or are targeted to specific environments (Stuxnet [19]). Such malware families do not reveal their true purpose during sandboxing or mimic less severe types of malware (adware, potentially unwanted application, etc.). This fact is recognized by the community as the main factor hindering the performance of dynamic analysis as a whole. Addressing this issue, however, is out of the scope of this paper.

Another important aspect that affects the performance of the proposed method is the quality of the data representation. First problem is when system resources are incorrectly split into multiple instance clusters due to the randomization of their names. This fragmentation can cause the split of binaries from the same malware family into multiple clusters. Even though this is clearly an error, it does not severely affects the analysis. According to our experience when malware families are split into low number of clear clusters, they can be still quickly analyzed, which is the goal of this work. Second problem arises when unrelated system resources are merged into single instance cluster. This can cause that corresponding malware samples from different malware families are grouped into single cluster. Again, according to our experiments this situation is not so common to drastically affect the analysis. The only clusters containing samples from multiple binaries are the ones that did not reveal their true behavior or did not execute correctly. Such situation can be easily identified, since the interaction with system resources limits to the noise generated by the operating system (operations with system logs, creating generic mutexes, etc.).

The quality of the data representation raises question which system resources are the most important for malware clustering. According to our experiments, model limited only to file paths and mutexes achieves $\sim 70\% - 75\%$ of the clustering performance compared to the case with complete information. However, only the combination of all five types of system resources provides optimal results as there are some malware families (e.g. *scar*) that use legitimate file paths to hide their payload but uses specific command and control infrastructure which is exploited by the model and detected as possible IOC.

The last question we need to discuss is the scalability of the proposed method. The most of the time required for the overall analysis was consumed in the preprocessing phase ($\sim$ 1h15min) and in the fitting of the model ($\sim$ 50min), and the time required for cluster prioritization and extraction of behavioral indicators was negligible (less than 1min). The theoretical analysis of the preprocessing phase shows the linear computational complexity in the number of samples (see [53] for more details). Similarly, the EM algorithm used to find the parameters π and M is linear in the number of samples which enables good scalability to large-scale dataset. In comparison the time required by AMAL was only $\sim$ 28min. However, since the computational complexity of the hierarchical clustering used in AMAL is $\Theta(N^2)$ or $\Theta(N^3)$ depending on the linkage method [41] the scalability to the large-scale dataset

is limited.

VII. Conclusion

This work has focused on providing complete analysis pipeline for analysis of large number of binaries executed in the sandbox. The pipeline starts with grouping binaries belonging to the same malware family, continuing by prioritizing bigger groups with homogeneous behavior, and ending with human-readable descriptions of groups of binaries. The proposed method utilizes a probabilistic model of malware behavior observed through its interactions with operating system and network resources (operations with files, mutexes, network servers, registry keys or system messages). To the best of our knowledge, such a complete pipeline was not yet published, albeit there is a prior art on the first problem of clustering.

Individual steps of the pipeline were evaluated on a large corpus of binaries, and the first problem of clustering has been compared to the prior art. This comparison showed that the proposed method outperforms the related state-of-the-art approach as it produce clusters that are purer and individual malware families were less fractioned into multiple clusters. Moreover, the evaluation also revealed limitations of the proposed method, which fails on samples that do not interact with resources monitored by the sandbox.

Acknowledgement

The work of Tomáš Pevný was supported by the grant GAČR 15-08916S.

References

[1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, nov 2016, pp. 183–194.

[2] J. Amores, "Multiple instance classification: Review, taxonomy and comparative study," *Artificial Intelligence*, vol. 201, pp. 81–105, aug 2013.

[3] A. Aronow, "SilentRunners," 2017, accessed July 26th, 2017. [Online]. Available: http://www.silentrunners.org/launchpoints.html

[4] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1025. [Online]. Available: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

[5] AV-Test, "AV-Test Malware Statistics," Tech. Rep., 2017.

[6] "Andromeda under the microscope," AVAST Threat Intelligence Team, 2016, accessed July 26th, 2017. [Online]. Available: https://blog.avast.com/andromeda-under-the-microscope

[7] U. Bayer, P. P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, Behavior-Based Malware Clustering." in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, 2009.

[8] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2011, vol. 4.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, mar 2008.

[10] J. Bremer, "x86 API Hooking Demystified," 2012, accessed July 26th, 2017. [Online]. Available: http://jbremer.org/x86-api-hooking-demystified/

[11] P. Chen, C. Huygens, L. Desmet, and W. Joosen, "Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware," in *IFIP International Federation for Information Processing 2016*. Springer, Cham, may 2016, pp. 323–336.

[12] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," Computer Sciences Department University of Wisconsin, Madison, Tech. Rep., 2006.

[13] "AMP ThreatGrid," CISCO Systems, Inc., 2017, accessed May 5th, 2017. [Online]. Available: http://www.cisco.com/c/en/us/solutions/enterprise-networks/amp-threat-grid/index.html

[14] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string metrics for matching names and records," *KDD Workshop on Data Cleaning and Object Consolidation*, vol. 3, pp. 73–78, 2003.

[15] C. Cortes, M. Mohri, and A. Rostamizadeh, "Algorithms for Learning Kernels Based on Centered Alignment," *The Journal of Machine Learning*, mar 2012.

[16] B. Dai, S. Ding, and G. Wahba, "Multivariate Bernoulli distribution," *Bernoulli*, vol. 19, no. 4, pp. 1465–1483, sep 2013.

[17] A. Dempster, N. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society Series B Methodological*, vol. 39, no. 1, pp. 1–38, 1977.

[18] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions," in *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*. New York, New York, USA: ACM Press, 2008, p. 51.

[19] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier analysis report," Tech. Rep., 2011.

[20] D. Gamerman and F. L. Hedibert, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, 2006.

[21] D. R. Garcia, "AntiCuckoo," accessed: May 9th, 2017. [Online]. Available: https://github.com/David-Reguera-Garcia-Dreg/anticuckoo

[22] "VirusTotal.com," Google, Inc., accessed: May 9th, 2017. [Online]. Available: https://www.virustotal.com/

[23] G. Govaert and M. Nadif, "Block clustering with Bernoulli mixture models: Comparison of different approaches," *Computational Statistics & Data Analysis*, vol. 52, no. 6, pp. 3233–3245, feb 2008.

[24] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[25] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," in *2016 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, feb 2016, pp. 1–5.

[26] "Beyond good ol' Run key," Hexacorn Limited, 2017, accessed July 26th, 2017. [Online]. Available: http://www.hexacorn.com/blog/2017/01/28/beyond-good-ol-run-key-all-parts/

[27] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[28] A. Juan and E. Vidal, "On the use of Bernoulli mixture models for text classification," *Pattern Recognition*, vol. 35, no. 12, pp. 2705–2710, dec 2002.

[29] J. Jusko, M. Rehák, J. Stiborek, J. Kohout, and T. Pevný, "Using Behavioral Similarity for Botnet Command and Control Discovery," *IEEE Intelligent Systems*, 2016.

[30] G. S. Kc, A. D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," in *Proceedings of the 10th ACM conference on Computer and communication security - CCS '03*. New York, New York, USA: ACM Press, 2003, p. 272.

[31] J. Kohout and T. Pevny, "Automatic discovery of web servers hosting similar applications," in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, 2015, pp. 1310–1315.

[32] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "AccessMiner: Using System-Centric Models for Malware Protection," in *ACM Conference on Computer and Communications Security 2010*, 2010, pp. 399–412.

[33] M. H. C. Law, M. A. T. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004.

[34] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[35] W. J. Li, K. Wang, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005*, vol. 2005, no. July 2005, pp. 64–71, 2005.

[36] R. W. Lo, K. N. Levitt, and R. A. Olsson, "MCF: a malicious code filter," *Computers & Security*, vol. 14, no. 6, pp. 541–566, jan 1995.

[37] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text Classification using String Kernels," *Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.

[38] W. Ma, P. Duan, S. Liu, G. Gu, and J. C. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *Journal in Computer Virology*, vol. 8, no. 1-2, pp. 1–13, 2012.
[39] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: High-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251–266, jul 2015.
[40] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proceedings - Annual Computer Security Applications Conference, ACSAC*. IEEE, dec 2007, pp. 421–430.
[41] D. Müllner, "fastcluster : Fast Hierarchical, Agglomerative Clustering Routines for R and Python," *Journal of Statistical Software*, vol. 53, no. 9, pp. 1–18, 2013.
[42] S. Naval, V. Laxmi, M. Rajarajan, M. S. Gaur, and M. Conti, "Employing Program Semantics for Malware Detection," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2591–2604, dec 2015.
[43] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, mar 2001.
[44] P. O'Kane, S. Sezer, K. McLaughlin, and E. G. Im, "SVM Training Phase Reduction Using Dataset Feature Filtering for Malware Detection," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 500–509, mar 2013.
[45] D. Oktavianto and I. Muhardianto, *Cuckoo Malware Analysis*. Packt Publishing, 2013.
[46] S. S. Response, "W32.Ramnit analysis," Tech. Rep., 2015.
[47] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov, "Learning and classification of malware behavior," in *DIMVA 2008*, 2008, pp. 108–125.
[48] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, jun 2011.
[49] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, vol. 1, no. June, pp. 410–420, 2007.
[50] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A Tool for Massive Malware Labeling," in *Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, 2016, pp. 230–253.
[51] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Impeding Malware Analysis Using Conditional Code Obfuscation," *Informatica*, 2008.
[52] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5283 LNCS, 2008, pp. 481–500.
[53] J. Stiborek, T. Pevný, and M. Rehák, "Multiple instance learning for malware classification," *Expert Systems with Applications*, vol. 93, pp. 346–357, 2018.
[54] Symantec Security Response, "Sality: Story of a Peer- to-Peer Viral Network," Symantec, Inc., Tech. Rep., 2011.
[55] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security and Privacy Magazine*, vol. 5, no. 2, pp. 32–39, mar 2007.
[56] T. Wüchner, M. Ochoa, and A. Pretschner, "Malware detection with quantitative data flow graphs," *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, pp. 271–282, feb 2014.
[57] Yixin Chen, J. Wang, and R. Krovetz, "CLUE: cluster-based retrieval of images by unsupervised learning," *IEEE Transactions on Image Processing*, vol. 14, no. 8, pp. 1187–1201, aug 2005.
[58] Y. Zhao and K. George, "Criterion functions for document clustering: Experiments and analysis (Technical Report)," Tech. Rep., 2001.

APPENDIX

A. Number of samples per file type

B. Number of samples of individual malware families

File type	#samples
Microsoft Office - DOC	898
Microsoft Office - HWP	1
Microsoft Office - MSI	17
Microsoft Office - PPS	1
Microsoft Office - XLS	21
HTML document	3464
Java archive data	1845
JavaScript	6370
Microsoft Office 2007+ - DOCM	546
Microsoft Office 2007+ - DOCX	18
Microsoft Office 2007+ - DOTM	49
Microsoft Office 2007+ - PPSX	2
Microsoft Office 2007+ - PPTX	10
Microsoft Office 2007+ - XLSM	138
MS-DOS executable	1536
PDF document	859
PE32 executable	97 428
PE32+ executable	182
Rich Text Format data	4
VisualBasic Script	4749
Windows Script File	5295
XML document text	5
Zip archive data	6760
Total	130 198

Table VIII: Number of samples per file type extracted from evaluation datasets.

Family	Category	#samples	Family	Category	#samples
nemucod	dropper	15 644	paneidix		278
cerber	ransomware	12 813	gamehack		277
bladabindi	inf. stealer	10 984	myxah		276
locky	ransomware	10 697	delbar		274
gamarue	banking trojan	7677	winner		264
darkkomet	RAT	4661	advml		261
hupigon	dropper	3568	confuser		260
upatre	dropper	3256	flystudio		252
tinba	banking trojan	3089	dynamer		249
scar	trojan	2958	redirector		242
swrort	dropper	2866	multiplug		241
zbot	banking trojan	2421	nitol		230
adwind	RAT	1875	mamianune		228
virlock	ransomware	1789	browsefox		217
fareit	inf. stealer	1784	fakejquery		211
farfli	trojan	1761	vopak		209
zegost	RAT	1745	cosmu		203
virut	trojan	1591	filefinder		194
zusy	trojan	1502	linkury		192
ircbot	trojan	1442	dlhelper		190
zerber	ransomware	1322	ardamax		188
palevo	worm	1269	msilperseus		184
vobfus	worm	1262	lotoor		183
delf	dropper	1255	pcclient		181
donoff	dropper	1211	vittalia		180
amonetize	adware	1198	lethic		170
loadmoney	PUA	1037	ngrbot		161
nanocore	RAT	1031	buzus		156
autoit		1006	skeeyah		151
yakes	trojan	901	koutodoor		150
poison		836	neshta		148
bifrose	inf. stealer	819	barys		148
kolabc	worm	705	parite		146
waldek		684	mikey		146
downloadassistant	PUA	678	redosdru		142
pdfka	exploitkit	644	hancitor		139
shipup	trojan	624	mintluks		134
rebhip	inf. stealer	612	cheatengine		133
razy		598	miancha		133
agentb		576	midie		133
onlinegames		529	fujacks		127
xtrat		510	pidief		126
ramnit	banking trojan	498	teslacrypt		126
atraps		478	kolovorot		124
magania		471	dupzom		123
softpulse	PUA	467	bublik		122
banload		435	opencandy		120
installmonster		411	socks		118
ruskill		373	icloader		114
convertad		363	fadok		113
installcore	PUA	361	garrun		113
sality	dropper	356	disfa		113
binder		355	chisburg		112
shiz		341	bedep		111
downloadguide	PUA	340	banbra		111
shyape		324	mydoom		108
llac		296	jaik		103
bayrob	inf. stealer	293	hlux		103
mywebsearch		283	refroso		101
winsecsrv		281			
Total malicious					130 198

Table IX: Number of samples of malware families in the data set. The malware families for individual samples were determined using AVClass tool [50]. The categories were determined using various reports published by AV companies and independent researchers.

Cluster	Size	Malware categories with coverage
1	87 912	legitimate (0.60), unknown (0.09), information stealer (0.07), dropper (0.07), rat (0.05),
2	6498	information stealer (0.56), rat (0.30), trojan (0.06), unknown (0.06),
3	4412	dropper (0.58), banking trojan (0.35),
4	3682	dropper (0.62), banking trojan (0.38),
5	2887	banking trojan (0.92), unknown (0.08),
6	2819	ransomware (0.51), unknown (0.21), legitimate (0.09), dropper (0.08), pua (0.07),
7	2735	ransomware (0.53), dropper (0.14), unknown (0.14), trojan (0.10),
8	2241	dropper (0.74), ransomware (0.26),
9	**1710**	**banking trojan (1.00),**
10	**1428**	**ransomware (0.99),**
11	**1360**	**trojan (0.99),**
12	**1076**	**banking trojan (0.98),**
13	1020	trojan (0.93), unknown (0.06),
14	**1003**	**ransomware (0.98),**
15	**923**	**dropper (0.98),**
16	**866**	**ransomware (0.99),**
17	**714**	**trojan (0.99),**
18	697	information stealer (0.36), unknown (0.21), dropper (0.21), rat (0.13),
19	**667**	**dropper (0.98),**
20	644	dropper (0.94),
21	**617**	**ransomware (0.98),**
22	**602**	**ransomware (1.00),**
23	595	ransomware (0.96),
24	**595**	**ransomware (0.98),**
25	**560**	**banking trojan (1.00),**
Total	128 263	
Total pure	12 121	

Table X: Top 25 clusters generated by AMAL from testing data mixed with legitimate samples and prioritized by size. The third column contains malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	**6785**	**legitimate (1.00),**
2	2512	dropper (0.92), banking trojan (0.08),
3	**2969**	**legitimate (1.00),**
4	3149	legitimate (0.83), unknown (0.07),
5	7225	legitimate (0.43), unknown (0.22), worm (0.16), dropper (0.10),
6	**832**	**banking trojan (1.00),**
7	5955	legitimate (0.96),
8	**1165**	**trojan (1.00),**
9	4178	dropper (0.62), banking trojan (0.36),
10	**1601**	**legitimate (1.00),**
11	4037	dropper (0.50), unknown (0.18), legitimate (0.13), information stealer (0.08),
12	1339	legitimate (0.86),
13	**1028**	**banking trojan (1.00),**
14	5753	legitimate (0.96),
15	3399	legitimate (0.97),
16	2310	dropper (0.63), ransomware (0.36),
17	**896**	**banking trojan (1.00),**
18	**817**	**banking trojan (1.00),**
19	**489**	**trojan (0.99),**
20	**716**	**legitimate (1.00),**
21	834	legitimate (0.90), unknown (0.09),
22	**85**	**legitimate (1.00),**
23	889	dropper (0.92), ransomware (0.08),
24	885	rat (0.95),
25	**936**	**information stealer (0.98),**
Total	60 784	
Total pure	18 319	

Table XI: Top 25 clusters generated by proposed model from testing data mixed with legitimate samples and prioritized with *score*. The third column contains malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	7225	legitimate (0.43), unknown (0.22), worm (0.16), dropper (0.10),
2	**6785**	**legitimate (1.00),**
3	5955	legitimate (0.96),
4	5753	legitimate (0.96),
5	4178	dropper (0.62), banking trojan (0.36),
6	4037	dropper (0.50), unknown (0.18), legitimate (0.13), information stealer (0.08),
7	3399	legitimate (0.97),
8	3149	legitimate (0.83), unknown (0.07),
9	**2969**	**legitimate (1.00),**
10	2512	dropper (0.92), banking trojan (0.08),
11	2310	dropper (0.63), ransomware (0.36),
12	**1820**	**ransomware (1.00),**
13	**1601**	**legitimate (1.00),**
14	1527	unknown (0.25), dropper (0.25), legitimate (0.20), trojan (0.07), rat (0.05),
15	**1357**	**ransomware (1.00),**
16	1339	legitimate (0.86),
17	1239	legitimate (0.97),
18	**1179**	**trojan (0.99),**
19	1178	legitimate (0.78), unknown (0.18),
20	**1165**	**trojan (1.00),**
21	1073	information stealer (0.97),
22	**1028**	**banking trojan (1.00),**
23	1004	legitimate (0.79), unknown (0.10),
24	1001	banking trojan (0.94), unknown (0.06),
25	**936**	**information stealer (0.98),**
Total	65 719	
Total pure	18 840	

Table XII: Top 25 clusters generated by proposed model from testing data mixed with legitimate samples and prioritized by size. The third column contains malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	16 162	legitimate (0.94),
2	11 529	legitimate (0.33), dropper (0.18), unknown (0.16), rat (0.15), trojan (0.09),
3	11 420	legitimate (0.39), unknown (0.22), worm (0.10), rat (0.10), dropper (0.07),
4	9995	legitimate (0.87), unknown (0.06),
5	6881	information stealer (0.96),
6	5136	dropper (0.65), ransomware (0.33),
7	4758	dropper (0.59), banking trojan (0.33),
8	3989	legitimate (0.87), exploitkit (0.06),
9	**3696**	**legitimate (1.00),**
10	3693	trojan (0.47), unknown (0.25), legitimate (0.09), pua (0.07), rat (0.05),
11	3653	unknown (0.30), dropper (0.22), pua (0.15), legitimate (0.12), trojan (0.10), ransomware (0.08),
12	3561	dropper (0.87), ransomware (0.13),
13	**3496**	**legitimate (0.98),**
14	3424	ransomware (0.96),
15	3281	legitimate (0.54), unknown (0.18), dropper (0.12),
16	3231	dropper (0.72), banking trojan (0.22), unknown (0.05),
17	2984	ransomware (0.40), unknown (0.22), banking trojan (0.19), legitimate (0.07),
18	2876	banking trojan (0.92), unknown (0.08),
19	2444	unknown (0.34), legitimate (0.28), information stealer (0.23), rat (0.06),
20	2388	legitimate (0.83), dropper (0.12),
21	2324	dropper (0.71), ransomware (0.29),
22	2251	information stealer (0.42), worm (0.21), unknown (0.16), ransomware (0.12),
23	2212	ransomware (0.95),
24	2046	ransomware (0.96),
25	2007	ransomware (0.96),
Total	119 437	
Total pure	7192	

Table XIII: Top 25 clusters generated by K-Means from testing data mixed with legitimate samples and prioritized by size. The third column contains malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Appendix 9 - Stiborek, J. and Pevný, T. and Rehák, M. : *Multiple instance learning for malware classification* [55]

Expert Systems With Applications 93 (2018) 346–357

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Multiple instance learning for malware classification

Jan Stiborek [a,b,*], Tomáš Pevný [a,b], Martin Rehák [a,b]

[a] Cisco Systems, Inc., 170 West Tasman Dr., San Jose, CA 95134 USA
[b] Dept. of Computer Science, FEE, CTU in Prague, Karlovo náměstí 13, 121 35 Praha, Czech Republic

ARTICLE INFO

Article history:
Received 15 June 2017
Revised 13 October 2017
Accepted 14 October 2017
Available online 16 October 2017

Keywords:
Malware
Dynamic analysis
Sandboxing
Multiple instance learning
Classification
Random forest

ABSTRACT

This work addresses classification of unknown binaries executed in sandbox by modeling their interaction with system resources (files, mutexes, registry keys and communication with servers over the network) and error messages provided by the operating system, using vocabulary-based method from the multiple instance learning paradigm. It introduces similarities suitable for individual resource types that combined with an approximative clustering method efficiently group the system resources and define features directly from data. This approach effectively removes randomization often employed by malware authors and projects samples into low-dimensional feature space suitable for common classifiers. An extensive comparison to the state of the art on a large corpus of binaries demonstrates that the proposed solution achieves superior results using only a fraction of training samples. Moreover, it makes use of a source of information different than most of the prior art, which increases the diversity of tools detecting the malware, hence making detection evasion more difficult.

1. Motivation

Since malware is presently one of the most serious threats to computer security with the number of new samples reaching 140 million in 2015 (AV-Test, 2016a), battles against it are fought on many fronts. Signature matching remains the core defense technology, but due to evasion techniques such as polymorphism, obfuscation, and encryption, keeping good recall is difficult for static analysis and methods based purely on string matching. A popular approach to tackle these problems is to execute a binary in a controlled environment (sandbox) (Oktavianto & Muhardianto, 2013), monitor its behavior, and based on this behavior classify the sample into benign or malware class (or as a particular malware family). The assumption of these dynamic analysis methods is that behavior should be more difficult to randomize and therefore it should constitute a more robust signal.

Most approaches to dynamic analysis rely on system calls (Ahmadi, Ulyanov, Semenov, Trofimov, & Giacinto, 2016; Naval, Laxmi, Rajarajan, Gaur, & Conti, 2015; Wüchner, Ochoa, & Pretschner, 2014), as they are the only means how the binary can interact with the operating system and other resources. This popularity has however already triggered many evasion techniques, such as shadow attacks (Ma, Duan, Liu, Gu, & Liu, 2012), system-call injection attacks (Kc, Keromytis, & Prevelakis, 2003), or sandbox detection (Garcia, 2016).

A perpendicular approach to modeling system calls is to model resources the binary has interacted with together with the type of the resources. The rationale is that if malware wants to provide revenue to its owner, it has to perform*actions,* such as downloading advertisements in the case of adware, encrypting hard drive in the case of ransomware, exfiltrating sensitive data in the case of credential stealers or Remote Administration Trojans (Chen, Wei, & Delis, 2008), etc. This work assumes that execution of these actions involves interactions with resources visible at the operating system level, and this interaction can be viewed as a signal which is hard to hide and which can be indicative of malware families.

Modeling interactions with system resources has been already exploited by the prior art. Mohaisen, Alrawi, and Mohaisen (2015) extracts a manually predefined set of features such as number of files created in specific folders, number of HTTP requests, etc., and use it in supervised classification. However, we believe that the rapidly changing threat landscape makes it difficult to manually design features that are indicative while also being stable over time. An alternative paradigm is to avoid manual design and to use a bag-of-words model (BoW model), where every interaction with a particular resource identified by its name is considered as a unique feature (Rieck, Holz, Willems, Dussel, & Laskov, 2008). The price paid for circumventing manual feature design using BoW is an explosion of the problem dimension, which can easily reach millions.

* Corresponding author.
E-mail addresses: jastibor@cisco.com (J. Stiborek), tpevny@cisco.com (T. Pevný), marrehak@cisco.com (M. Rehák).

https://doi.org/10.1016/j.eswa.2017.10.036
0957-4174/

This work circumvents the problem of manually designing features while at the same time avoiding the problem dimension explosion. The approach is to first cluster resource names with similarity functions tailored for each resource type (file names, mutexes, registry names, and domain names), and then use this clustering to represent a sample (a binary executed in the sandbox) in a lower-dimensional space. This enables us to use *a random forest classifier* (or any other classifier of choice) to separate malware from legitimate samples. The clustering also effectively removes randomization used to evade detection.

The proposed approach is extensively evaluated on a large number of samples (more than 230,000) and compared to relevant prior art. Experimental results show that the proposed approach indeed improves the accuracy of detecting malware binaries with the low complexity required for deployment on large-scale datasets.

The contributions of this paper are manifold. This paper presents a novel approach to representing malware samples using their interactions with system resources. It employs *vocabulary-based method* from the field of *multiple instance learning* (MIL) combined with a definition of novel similarity measures for different types of resource types reflecting their unique properties (structure of file paths, network communication, etc.). Next, it presents a fast approximation of Louvain clustering method used to automatically define the vocabulary that allows to scale the MIL method to large scale datasets. To the best of our knowledge, this work is the first that employs techniques for multiple instance learning in the field of malware analysis and presents a viable alternative to traditional approaches for malware classification.

2. Related work

Since the analysis of malicious binaries and recommending them for further analysis has important practical applications, there exists rich prior art. Although it is frequently divided into two categories, static and dynamic, the boundaries between them are blurred since techniques such as analysis of the execution graph is used in both categories.

2.1. Static malware analysis

Static malware analysis treats a malware binary file as a data file from which it extracts features without executing it. The earliest approaches (Lo, Levitt, & Olsson, 1995) looked for a manually specified set of specific instructions *(tell-tale)* used by malware to perform malicious actions but not used by legitimate binaries. Latter works, inspired by text analysis, used n-gram models of binaries and instructions within (Li, Wang, Stolfo, & Herzog, 2005). Malware authors reacted quickly and began to obfuscate, encrypt, and randomize their binaries, which rendered such basic models (Sharif, Lanzi, Giffin, & Lee, 2008a) useless. Since reversing obfuscation and polymorphic techniques is in theory an NP-hard problem (Moser, Kruegel, & Kirda, 2007), most state of the art (Ahmadi et al., 2016; Christodorescu & Jha, 2006; Sharif, Yegneswaran, Saidi, Porras, & Lee, 2008b) moved to a higher-level modeling of sequences of instructions / system calls and estimating their action or effect on the operating system. The rationale behind is that higher-level actions are more difficult to hide. An example of higher-level modeling is a *call graph* (Hu, Chiueh, & Shin, 2009; Kinable & Kostakis, 2011; Kong & Yan, 2013) where vertices represent individual functions and edges capture dependencies between them. The analysis is then formulated as a problem of finding appropriate similarity measure (fixed graph edit distance (Kinable & Kostakis, 2011), normalized maximal common subgraph (Hu et al., 2009) or trained similarity (Kong & Yan, 2013). Even though such analysis provides deep understanding of malware binaries, its scalability is limited due to complexity of the similarity measures (typically NP-complete problem).

2.2. Dynamic malware analysis

An alternative solution to analyzing obfuscation and encryption is the execution of a binary in a controlled environment and analyzing its interactions with the operating system and system resources.

A large portion of the work related to dynamic malware analysis utilize system calls, since in modern operating systems system calls are the only way for applications to interact with the hardware and as such they can reveal malware actions. The simplest methods view a sequence of system calls as a sequence of strings and use histograms of occurrences to create feature vectors for the classifier of choice (Hansen, Larsen, Stevanovic, & Pedersen, 2016). The biggest drawback of these naive techniques is low robustness to system call randomization. Similarly to static analysis, this problem can be tackled by assigning actions to groups (clusters) of system calls (syscalls) and using them to characterize the binary (Bayer, Comparetti, Hlauschek, Kruegel, & Kirda, 2009; Naval et al., 2015; Wüchner et al., 2014). Another

A wide class of methods identifying malware binaries from sequences of syscalls rely on n-grams (Canzanese, Mancoridis, & Kam, 2015; Lanzi, Balzarotti, Kruegel, Christodorescu, & Kirda, 2010; O'Kane, Sezer, McLaughlin, & Im, 2013). Malheur (Rieck, Trinius, Willems, & Holz, 2011) uses normalized histograms of n-grams as feature vectors, which effectively embeds syscall sequences into Euclidean space endowed with L_2 norm. In this space the algorithm extracts prototypes $Z = \{z_1, \ldots, z_n\}$ using hierarchical clustering. Each prototype captures the behavior of the cluster, which should match corresponding malware family. An interesting feature of Malheur is that if a cluster has less then a certain number of samples, the prototype is not created. The classification of an unknown binary is determined by searching for the nearest prototype within certain range. If the nearest prototype is outside of this range, the sample is not classified.

To counter dynamic analysis advanced malware detects the presence of a sandbox and does not execute within it. Since most sandboxes rely on a detectable system call interposition, Das, Liu, Zhang, and Chandramohan (2016) propose to extend hardware with FPGA that would extract system calls from their execution on processor. Syscalls are then grouped by comprehensive yet hand designed rules, and these groups are then fed into multilayer neural network classifier. The classifier itself is also part of the FPGA, such that the system can simultaneously extract training samples and classify them.

AMAL uses its custom sandbox to extract features describing files, network communication and registry features (Mohaisen et al., 2015) and tunes various classification algorithms. The main difference between AMAL and this work is the construction of features. Whereas AMAL uses numeric features such as counts or sizes of created, modified or deleted files, counts of created, modified or deleted registry keys, counts of unique IP addresses, etc., we assume that individual resources (files, registry keys, mutexes and network communication) have specific role in the operation system, which can be different even though the characteristics exhibited by the file are the same.

The approach proposed by Rieck et al. (2008) creates a representation of the analyzed binaries directly from the data which is at the first sight similar to the proposed approach, however there are two key differences. The first one is the source of data, because Rieck et al. model actions triggered by the malware (writing into a file, communication with remote server, reading data from registry keys, starting new thread, etc.), whereas the proposed approach

models only affected resources. This enables to deploy the proposed approach in environments without access to low-level actions (VMs without such access, user machines without API hooking). Another difference is in handling the randomization of resource names. Instead of clustering resource names used in this work, Rieck et al. remove parameters of actions, which increases the dimensionality of the model since for every action with n parameters it creates $n+1$ features representing the action at different levels of granularity by removing parameters from the end: from full description with all parameters to the most coarse description where only name of the action is used. This leads to a massive increase in the already large number of features.[1] Even though the resulting feature space, is sparse the scalability of such an approach is limited.

Anderson, Storlie, and Lane (2012) propose to combine techniques from static analysis with the data obtained using dynamic analysis in order to counter techniques frequently used by malware authors to avoid detection, e.g. packing or execution stalling. Authors propose six different types of input data, three based on techniques from static analysis: (1) features extracted from raw binary modeled as n-grams, (2) opcodes extracted from disassembled binary and (3) *control flow graph*—a graph of all possible execution paths; two based on dynamic analysis: (4) instruction traces (Anderson, Quist, Neil, Storlie, & Lane, 2011) and (5) system call traces; and (6) various information extracted from the binary itself such as packer identification, entropy of the binary, number of instructions in disassembled file, etc. For every type of input authors define a kernel which are then combined using *multiple kernel learning (Gonen & Alpaydin, 2011)* to obtain optimal combination. The optimized kernel combination is then used with SVM classifier.

The approach proposed by Chen et al. (2012) utilizes knowledge from the malware encyclopedias for malware classification as it extracts model of the malware description and applies it to the output produced by the sandbox. Using such approach authors are able to annotate malware with various labels (*exploit, dropper*, etc.) which is then used for threat estimation and classification.

2.3. Algorithms for multiple instance learning

Classification based on multiple instance learning paradigm is a popular topic with many possible applications such as classification of images, classification of candidates drugs, online object tracking, etc.

Recently, Amores (2013) proposed taxonomy of approaches for multiple instance learning that divides proposed algorithms into three classes: (1) algorithms based on *instance-space* paradigm(IS), (2) algorithms based on *bag-space* paradigm (BS) and (3) algorithms based on *embedded-space* paradigm (ES).

The IS paradigm states that the multiple instance classification is based solely on information extracted from individual instances. The IS-based methods first classify the individual instances and then propagate the estimated labels to the whole bags. Algorithms incorporating IS paradigm include *Diverse density (DD)* proposed by Maron and Lozano-Pérez (1998), *Expectation-Maximization Diverse density (EM-DD)* proposed by Zhang and Goldman (2002), *Multiple Instance Support Vector Machines (MI-SVM)* proposed by Andrews, Tsochantaridis, and Hofmann (2003), *MIForests* proposed by Leistner, Saffari, and Bischof (2010) or *G3P-MI* proposed by Zafra and Ventura (2010). Since the IS-based methods operate on the level of individual instances, they require a numerical representation of the instances or an appropriate kernel function. However, the kernelized methods applicable in our scenario (MI-SVM) do not scale well to large datasets ($O(M^3)$ where M is total number of instances) which make their use computationally infeasible. An interesting idea to improve the scalability of the IS-based methods is proposed by Cano, Zafra, and Ventura (2015). Authors propose to leverage the performance of the GPUs for training of MI classifiers. However, since GPUs are designed for fast matrix operations, the performance boost of the similarity metrics proposed in this paper would be limited. Nevertheless, such approach provides an interesting option and will be considered in further research.

In algorithms based on the BS paradigm is the information extracted from the whole bags. Typically, it involves definition of a distance function $D(X, Y)$ that compares two bags X and Y. Such functions include minimal Hausdorff distance used in Wang and Zucker (2000), MI-Graph proposed by Zhou, Sun, and Li (2009) or kernel function proposed by Gärtner, Flach, Kowalczyk, and Smola (2002). However, as discussed in Amores (2013), the complexity of approaches based on BS paradigm is $O(N^2 \cdot m^2) \approx O(M^2)$ where N is number of bags, m is average number of instances in a bag and M is total number of instances. Even though the computational complexity is lower than the complexity of MI-SVM, it is still too much for large dataset (the testing dataset in this paper has more 7 million instances).

The last class of approaches are based on ES paradigm. They transform the bags into numerical vectors using a mapping function $\mathcal{M} : X \to \mathbb{R}^d$ and train a supervised machine learning algorithm on the transformed vector representation. The simplest mapping functions proposed by Dong (2006) and Bunescu and Mooney (2007) are defined as $\mathcal{M}(X) = (f_1, \ldots, f_d)$, where $f_i = \frac{1}{|X|}\sum_{x \in X} x_i$, or $f_i = \max_{x \in X} x_i$, or $f_i = \min_{x \in X} x_i$. More complex mapping functions define the embedding using a vocabulary $\mathcal{V} = \{(\theta_1, c_1), \ldots, (\theta_d, c_d)\}$, a set of *concepts* (set of instances) where each concept is identified by identifier c_j and parameters θ_j. The mapping function is then defined as $\mathcal{M}(X, V) = (f_1, \ldots, f_d)$ with f_i defined as $f_i(X, V) = \frac{1}{Z}\sum_{x \in X} p(x|c_i), \forall i \in 1, \ldots, d$, where Z is a normalization constant ensuring $\sum f_i = 1$, and $p(x|c_i)$ represents the likelihood that an instance x belongs to a concept c_i or hard assignment of instance x into concept c_i. Regarding the definition of the vocabulary $\mathcal{V}$, a traditional approach, adopted in this work as well, is to cluster all instances extracted from all bags using a suitable clustering algorithm (typically K-Means) and to use the resulting clusters as the concepts. However, such approach does not reflect relations between bags and individual instances. To address this issue, Weidmann, Frank, and Pfahringer (2003) propose partitioning of instances based on decision trees that considers the knowledge of the labels of corresponding bags. However, as the decision trees assume numerical representation of the instances, this approach is not applicable in our scenario.

Cano (2017) proposes an interesting approach that combines multiple-instance with multi-view learning approach. The multi-view approach allows to fuse various source of information about the data which in our case corresponds to various types of system resources and thus boost the classification performance. Furthermore, authors propose to replace a single multiple instance classifier with an ensemble approach to further improve the classification performance. However, since the base classifiers in the ensemble are standard multiple instance classifiers described above, they share the same limitations, which limits the use of this approach in our scenario.

3. Classification of sandboxed samples

To capture the malware behavior, this work assumes that execution of malware's actions involves interactions with resources visible at the operating system level. Examples of such interactions include*operations with files* during encryption of a victim's hard drive,*network communication* during data exfiltration or dis-

[1] According to the experiments, the number of features generated for about 6000 samples reaches over 20 million.

playing advertisements, *operation with mutexes* used to ensure a single instance of malware is running, or manipulation with *registry keys* to ensure persistency after reboot. An additional source of information are error messages of the operating system itself. Such information is provided by the sandboxing environment as the following warnings:*dll not found* indicating missing dynamic library, *incorrect executable checksum* indicating corrupted binary, and *sample did not execute* indicating the fact that the binary was not executed at all due to various reasons (corrupted binary, sandbox was not able to copy the binary into VM, etc).

To model the interactions of a malware binary with resources, this work views each binary executed in a sandbox as a set of pairs of names and types of resources the binary interacted with. This view frames the problem as a *multiple instance learning* (MIL) problem where each sample (binary) is represented as a *bag* that consists of a set of *instances* of different size. In our scenario an instance represents the pair of name and type of a resource the binary interacted with during sandboxing.

Variable sizes of samples and lack of order over their instances pose a challenge to traditional machine learning methods that expect samples to have fixed size. A recent review of MIL algorithms (Amores, 2013) lists various approaches to overcome this variability in sample sizes. One of the popular approaches (also adopted in this work) is *vocabulary-based method (Sivic & Zisserman, 2003)* outlined in Algorithms 1 and 2. It employs clustering of instances

Algorithm 1 High-level overview of training (function TRAIN) and classification (function PREDICT) of malware samples.

function TRAIN(S, y) ▷ Training samples and labels
  $I \leftarrow extractInstances(S)$
  $C \leftarrow cluster(I)$ ▷ Clustering of instances (separately for individual types)
  $X \leftarrow project(S, C)$ ▷ Projection of samples into binary vector (Alg. 2)
  $\mathcal{M} \leftarrow trainClassifier(X, y)$
  return $\mathcal{M}, C$ ▷ Returns cluster centers C and trained classifier $\mathcal{M}$
end function
function PREDICT($S', C, \mathcal{M}$) ▷ Testing samples S', clusters C and classifier $\mathcal{M}$
  $X' \leftarrow project(S', C)$ ▷ Projection of samples into binary vector (Alg. 2)
  $\hat{y} \leftarrow predict(\mathcal{M}, X')$ ▷ Classification of testing samples
  return $\hat{y}$
end function

Algorithm 2 Projection of samples S into binary vector using cluster centers C.

function PROJECT(S, C) ▷ Samples S and clusters C.
  $X \leftarrow \emptyset$
  for all $s \in S$ **do**
    $I \leftarrow extractInstances(s)$
    $x \leftarrow \vec{0}$
    for all $i \in I$ **do**
      $c^* \leftarrow nnSearch(i, C)$ ▷ Finds closest center c^* to instance i.
      $x[c^*] \leftarrow 1$
    end for
    $X \leftarrow X \cup \{x\}$
  end for
  return X
end function

Table 1
Example of clustering files of two binaries from the same family executed in the sandbox.

Binary 1	Binary 2
(a) raw filenames	
\Temp\4ffdd6ab-8020\config.dmc	\Temp\ed8a9718-c7a0\config.dmc
\Temp\4ffdd6ab-8020\bin.dmc	\Temp\ed8a9718-c7a0\bin.dmc
\Windows\System32\ftp.exe	\Windows\System32\netsh.exe
(b) artifact clusters	
Artifact cluster 1	Artifact cluster 1
Artifact cluster 1	Artifact cluster 1
Artifact cluster 2	Artifact cluster 2

to describe the sample by a fixed-dimensional vector with length equal to the size of vocabulary, i.e. a set of clusters, so that an ordinary machine learning method can be applied.

To convert a sample into a fixed-dimensional vector, all instances I from all training samples S are extracted and clustered by a suitable method per given resource type–files, mutexes, registry keys, network communication. Note that warnings generated by the sandboxing environment are used directly, i.e. every warning is considered as a separated cluster. The resulting clusters represent the vocabulary. Next, for every instance i the closest cluster prototype c^* (a small random subset of the cluster of instances) of corresponding type is located. Finally, the binary representation is then used such that element of the vector equals to 1 iff there was an instance close to the particular cluster prototype. Once all samples are encoded as fixed-dimensional vectors, one can use a machine learning algorithm of choice to implement the classifier. This work uses the *random forest classifier* (Breiman, 2001) due to its versatility, accuracy, and scalability, which make it a popular choice for many different machine learning tasks including malware classification (Hansen et al., 2016).

Since the clustering is an essential component of the above algorithm, the definition of similarity over instances (resource names) greatly influences the accuracy of the system, and therefore it should reflect properties of the application domain. The rest of this section defines a specific similarity metric for each type of resources the malware interact with, namely on files, mutexes, network hostnames, and registry keys, and also justifies our choice of the clustering method (Table 1).

3.1. Similarity between file paths

Although viewing file paths as strings would allow to use vast prior art such as Levenshtein distance (Levenshtein, 1966), Hamming distance, Jaro–Winkler distance (Navarro, 2001), or string kernels introduced in Lodhi, Saunders, Shawe-Taylor, Cristianini, and Watkins (2002), the file systems were designed as tree structures with names of some folders (fragments of the path) being imposed by the operating system and the distance should reflect that. For example two files with paths `/Documents and Settings/Admin/Start Menu/Programs/Startup/tii9fwliiv.lnk` and `/Documents and Settings/Admin/Start Menu/Programs/Accessories/Notepad.lnk` share large parts of their paths and common string similarities will return high similarity score, but they serve very different purposes, since the first file is a link to an application executed after the start of the operating system (OS), while the second is a regular link in the Start menu in Windows OS. Another aspect that prohibits the use of common string similarities is their computational complexity (typically $O(n^2)$ where n is the length of the string). The complexity combined with the number of resources to be clustered (in order of millions) leads to unfeasible time requirements. This motivates the design of a similarity that is fast and takes into the account the tree structure of the

Table 2
Example of two paths x and x' separated into individual fragments with labels (K – known folder, G – general folder and F – file).

	Fragment 1	Fragment 2	Fragment 3
x	`Documents and Settings` (K)	`Admin` (G)	`Start Menu` (K)
x'	`Documents and Settings` (K)	`Admin` (G)	`Start Menu` (K)
	Fragment 4	Fragment 5	Fragment 6
	`Programs` (K)	`Startup` (K)	`tii9fwliiv.lnk` (F)
	`Programs` (K)	`Accessories` (G)	`Notepad.lnk` (F)

file system, special folders, and differences between folders and filenames.

The proposed similarity $s(x, x')$ of two file paths x and x' is defined as

$$s(x, x') = \exp\left(-w^T f(x, x')\right), \tag{1}$$

where w is a vector of weights and $f(x, x')$ is a function extracting a feature vector from file paths x and x'. Both the weight vector w and function f play an essential role and are both discussed in detail below.

The function f in (1) captures differences between the two paths x and x' by a fixed-dimensional vector. It first splits both paths x and x' into fragments x_i and x'_i using OS specific path separator,[2] in the cases of MacOS and Windows changes all characters to lowercase, and assigns all fragments into one of the following four categories:

1. *known folder* – fragment x_i is a well known folder in the list of folders imposed by the operation system (e.g. `Windows`, `Program Files`, `System32`, etc.),
2. *general folder* – fragment x_i is a not-well-known folder (e.g. unknown folders in `Program Files`, randomly generated folders in Internet Explorer cache folder, etc.),
3. *file* – fragment x_i is file,
4. *empty* – artificial fragment used for padding the paths in cases when paths x and x' have different depths.

When all fragments are assigned to one of the above classes, their dissimilarity is captured by the function f as

$$f(x, x') = (f_{KK}, f_{KG}, f_{KF}, f_{KE}, f_{GG}, f_{GF}, f_{GE}, f_{FF}, f_{FE})$$

where

- f_{KK} is the number of fragments on the same level that were both classified as *known folder* and were not equal,
- f_{GG} is the sum of Levenshtein distances between all fragments on the same level that were classified as *general folder*,
- f_{FF} is the sum of Levenshtein distances of all fragments on the same level that were classified as *file*,
- f_{KG}, f_{KF}, f_{KE}, f_{GF}, f_{GE}, f_{FE} are the sums of all fragments of the same level and were classified as *known* and *general folder, known folder* and *file, known folder* and *empty, general folder* and *file, general folder* and *empty,* and *file* and *empty* respectively.

To illustrate the calculation of $f(x, x')$, let's consider the same two paths used above. At first, function f splits both paths into fragments and assign them into one of four categories (see Table 2). Assigning fragment to classes requires a list of known folders,[3] which for the purpose of this example we assume to contain `Documents and Settings`, `Start Menu`, `Programs` and`Startup`, which are present in all windows installations. All corresponding folders from those two paths are therefore assigned to known folder class, while `Admin` and `Accessories` are labeled as general folders.[4] Individual elements of the vector $f(x, x')$ are calculated using the above rules as follows: the first rule applies to three fragments 1, 3, and 4 belonging to known folder class, but as they are all equal $f_{KK} = 0$; the second rule returns 0 based on analogous reasoning but for general folders; the third rule returns $f_{FF} = 0.7143$, which is the Levenshtein distance between `tii9fwliiv.lnk` and `Notepad.lnk`; the only mismatch is on fragment 5–known folder and general folder yielding $f_{KG} = 1$; and finally all remaining elements of feature vector are 0. The output of $f(x, x')$ is captured by the feature vector

$$f(x, x') = (0, 0, 0.7143, 1, 0, 0, 0, 0, 0).$$

The weight vector w in (1) captures the contribution of individual elements of the feature vector $f(x, x')$. Imposing condition $w \geq 0$, in combination with construction of function f, bounds the value of the similarity function (1) $s(x, x') \in [0, 1]$ such that the similarity functions returns 1 (or values close to 1) if x and x' belong to the same class (files in `/temp/` directory, cache of the Internet Explorer, files in system directory, etc.) and values approaching 0 if they belong to different classes. Since the similarity function (1) was inspired by the popular Gaussian kernel, the parameter vector w was optimized using the Centered Kernel Target Alignment (Cortes, Mohri, & Rostamizadeh, 2012) (CKTA), which is a method to optimize kernel parameters. CKTA assumes training data $\{(x_i, y_i), \}_{i=1}^m$ where x_i is a file path and y_i is the class of the path x_i, and defines *centered kernel matrix* as

$$[\mathbf{S}_c^w]_{ij} = \mathbf{S}_{ij}^w - \frac{1}{m}\sum_{i=1}^m \mathbf{S}_{ij}^w - \frac{1}{m}\sum_{j=1}^m \mathbf{S}_{ij}^w + \frac{1}{m^2}\sum_{i,j=1}^m \mathbf{S}_{ij}^w, \tag{2}$$

where $S_{ij}^w = s_w(x_i, x_j)$ is the kernel matrix corresponding to the similarity function (1) parameterized by the weight vector w. CKTA maximizes correlation between labels and a similarity matrix by solving the following optimization problem

$$w^* = \arg\max_{w \geq 0} \frac{\langle \mathbf{S}_c^w, \mathbf{Y}_c \rangle_F}{\|\mathbf{S}_c^w\|_F \cdot \|\mathbf{Y}_c\|_F}, \tag{3}$$

where $\mathbf{Y}$ is target label kernel with $[\mathbf{Y}]_{ij}$ equals to 1 when ith and jth paths from training data belongs to the same class and -1 otherwise, $\langle \cdot, \cdot \rangle_F$ is Frobenius product and $\| \cdot \|_F$ is Frobenius norm (see Appendix A.1 for more details). In below experiments (3) is solved by *stochastic gradient descent (SGD)* algorithm (Bishop, 2011). Note that although the path similarity $s(x_i, x_j)$ is not a valid kernel because it is not positive definite, the use of centered kernel alignment is still possible as the only limitation is that the global optimum might not be found.

[2] Unixes and MacOS uses '/' as a path separator, Windows uses '\'.

[3] Full list of known folders is available online: https://github.com/SfinxCZ/Multiple-Instance-Learning-for-Malware-Classification

[4] The first three known folders are embedded in the functionality of the Windows OS. The `Startup` folder has a specific meaning altering the behavior of the operation system since all programs listed in this folder are executed after the boot of the OS. On the other hand `Accessories` can be easily changed without major consequences.

To finish the example, the similarity function (1) with weight vector $w = (2, 10^{-5}, 1, 2.3, 1.6, 1, 0.36, 0.7, 0.9)$ returns the value $s(x, x') = 0.049$, which correctly indicates that the two paths are different.

3.2. Similarity of network traffic

To define the similarity between network resources one has to overcome the randomization often employed by malware authors that render trivial similarity based on names of network resources (domains, IPs) ineffective. To escape blacklisting command and control (C&C) channels of malware, its authors use various techniques to hide and obscure C&C operation. Popular approaches include randomization of domain names by generating them randomly (DGA), quickly changing hosting servers and / or domain names by fast flux, or using large hosting providers like Amazon Web Services to hide among legitimate servers, etc. These techniques are relatively cheap (e.g. registering a new *.com* domain costs ˜3USD per 1 year) and they allow for variation in domain names without updating disseminated malware binaries. In contrast, switching from one C&C paradigm to another requires such an update and therefore occurs relatively infrequently. These two properties contribute to each malware family using specific patterns of domain names, paths, and parts of URLs. Exploiting these patterns allows to group domain names into clusters. In this work the similarity in network traffic is defined only for HTTP/HTTPS protocol, because it is presently the default choice for malware authors as it is rarely filtered. The extension to other network traffic is possible (Kohout & Pevny, 2015).

The similarity in URL patterns used in this work has been adopted from Jusko, Rehák, Stiborek, Kohout, and Pevný (2016), which has proposed to cluster domain names so that each cluster contains domains of one type / for one family of malware. The calculation of similarity starts by grouping all HTTP/HTTPS requests using the domain names. Then the model of each domain name is built from path and query strings, transferred bytes, duration of requests and inter-arrival times (time spans between requests to the same domain) of individual requests to it. Finally, these models are used to calculate the similarity function between two domain names in the clustering. Since the calculation of the similarity is out of scope, we refer to an original publication (Jusko et al., 2016) for details.

3.3. Similarity between mutex names

Mutex (*Mutual exclusive object*) is a service provided by most modern operating systems to synchronize multi-threaded and multi-processes applications. This mechanism is popular among malware authors to prevent multiple infections of the same machine, because running two instances of the same malware can cause conflicts limiting the potential revenue. Mutexes are identified by their name, which can be an arbitrary string. The naming scheme is challenging for malware authors, because the names cannot be static, which would make them good indicators of compromise of a particular malware, but they cannot be completely random either, because two independent binaries of the same family would not be able to check the presence of each other. Therefore malware authors resorted to pseudo-deterministic algorithms or patterns for generating mutex names. For some malware families these patterns are already well known, for example Sality (Symantec Security Response, 2011) uses mutex names of the form ‘‘`<process name>.exeM_<process ID>_`’’ - `explorer.exeM_1423_`.

Since operating systems do not impose any restrictions on the names of mutexes, they can be arbitrary strings. Therefore standard string similarities such as Levenshtein distance, Hamming distance, Jaro–Winkler distance, etc. can be used. In experiments presented in Section 4 Levenshtein was used, as it gives overall good results.

3.4. Similarity between registry names

In Microsoft Windows operating system, the primary target of the majority of malware, registry serves as a place where programs can store various configuration data. It is a replacement of configuration files with several improvements such as strongly typed values, faster parsing, ability to store binary data, etc. The registry is a key-value store, where key names have the structure of a file system. The root keys are `HKEY_LOCAL_MACHINE`, `HKEY_CURRENT_USER`,`HKEY_CURRENT_CONFIG`, `HKEY_CLASSES_ROOT`, `HKEY_USERS`and `HKEY_PERFORMANCE_DATA`; some root keys also always have sub-keys with specific names (`Software`, `Microsoft`, `Windows`, etc.). Due to similarity with a file system, the similarity distance is the same as the one defined in Section 3.1, but with a different set of names of known folders and a weight vector optimized on registry data rather than on files.

3.5. Clustering of resource names

The above similarities are not true distances, which limits the choice of applicable clustering methods to those that do not require proper distance metric between points. The Louvain method (Blondel, Guillaume, Lambiotte, & Lefebvre, 2008) is a popular choice and it is used in experiments below, because it also automatically determines the number of clusters and thus removes the need to set it manually. The use of the Louvain method is the authors' preference, but other clustering methods can be used as well; the reader is referred to Fortunato (2010) for an overview of methods requiring only similarity.

The use of the Louvain method is not straightforward in the scenario of this paper because it requires a full adjacency matrix in advance. This results in a lower bound to computational complexity being $O(n^2)$ in the number of resources, which is clearly prohibitive as the number of unique resource names to cluster can easily reach the order of millions. To decrease the number of calculated similarities, an approach inspired by Zhang, Ramakrishnan, and Livny (1997), Zimek, Gaudet, Campello, and Sander (2013) and Jang, Brumley, and Venkataraman (2010) is adopted where the Louvain clustering is used iteratively as summarized in Algorithm 3. Given a set of instances I of a particular

Algorithm 3 Approximative clustering algorithm for instances I (resource names).

```
1:  function ApproxCluster(I; k, m, ϵ)
2:      C = ∅
3:      while I ≠ ∅ do
4:          I' ← Random subset of size k from I
5:          C' ← cluster(I', m)          ▷ Cluster instances I' and create cluster prot. of size m.
6:          for all i ∈ I \ I' do
7:              c* ← nnSearch(i, C')     ▷ Find cluster prot. c* closest to instance i.
8:              if s(i, c*) > ϵ then
9:                  c* ← c* ∪ {i}
10:             end if
11:         end for
12:         C ← C ∪ C'
13:     end while
14:     return C
15: end function
```

Appendix 9 - Stiborek, J. and Pevný, T. and Rehák, M. : *Multiple instance learning for malware classification* [55]

type, in every iteration the algorithm selects a random subset $I' \subset I$ of the data of size k small enough for the Louvain method to be computationally feasible. The results of the Louvaine clustering are then transformed to cluster prototypes—random subsets of clusters with size limited to m. Remaining data $\mathcal{N}'$ are then traversed and all samples with similarity larger than ϵ to some cluster prototype $c^* \in C'$ are added to c^* and removed from I. Finally, C' is merged with the clustering C obtained in the previous iteration, and if I is not empty, the process is repeated.

Clearly the algorithm is an approximation of a clustering with complete data and its performance depends on the choice of parameters k and ϵ. Experiments indicate that if parameter k is large enough ($k = 10^4$) and parameter ϵ is set reasonably (in the experimental evaluation we use $\epsilon = 0.4$, see Section 4.2 for details), the results are comparable with clustering methods applied to the complete data. The computational complexity of this sequential approximation is $O(l \cdot (k \cdot (k-1)/2 + c_l \cdot m \cdot (n_l - k)))$ where l is the number of iterations of algorithm (typically $l \leq 10$), n_l is the number of non-clustered samples in lth iteration, k is the number of randomly selected samples, c_l is the number of cluster prototypes produced by the clustering algorithm in lth iteration and m is the maximal size of a cluster prototype (typically $m = 10$). Since the parameter k is fixed and $k \ll n$, we can see that the number of evaluations of the similarity function is linear in the number of samples, which clearly outperforms the quadratic complexity required by the vanilla Louvain method.

4. Evaluation

In this section the proposed approach (further referred as MIL model) is compared to the approach proposed by Rieck et al. (2008) (further referred to as *Rieck*) and the approach proposed by Mohaisen et al. (2015) (further referred to as *AMAL*). Rieck has been selected as a representative of the prior art that encodes malware behavior into a high-dimensional feature space using bag-of-words model built directly from data; it uses kernelized SVM to classify binaries. The second approach, AMAL, encodes malware behavior using a relatively low number of hand-made features; to classify unknown binaries AMAL trains multiple classifiers (SVM, decision trees, k-nearest neighbor, etc.) and selects the optimal classifier for given data using cross-validation.

4.1. Data set description

The dataset used for experiments contained 250 527 files collected from October 24, 2016 to December 12, 2016 using *AMP ThreatGrid* (CISCO Systems, Inc., 2017). All files were also analyzed by *VirusTotal.com* service (Google Inc., 2017) and labeled using its verdicts as follows: a file was labeled as malicious if at least 4 out of 10 selected AV engines (see Table 4 for details) detected the file as malicious, and it was labeled as legitimate if none of the AV engines detected the file. Remaining files were discarded as unknown and removed from both training and testing sets in order to limit the effect of misclassifications by individual AV engines. The final numbers of files were: 143,684 malicious, 86,707 legitimate, and 20,136 discarded as unknown. The numbers of samples of individual malware families are summarized in Table 3.

All files were executed in sandbox by AMP ThreatGrid (CISCO Systems, Inc., 2017) service, using Windows 7 64bit (71% samples) environment, as it is the most popular OS at the time of writing,[5] and Windows XP (29% samples) environment, since it is still widely deployed on embedded machines such as ATMs. Virtual machines were connected to the Internet without any filtering or restrictions that could by any mean prevent connections to command & control servers or other servers. The work here is not tailored to AMP ThreatGrid, as the same or similar information about binaries can be obtained by a number of different sandboxing solutions such as Cuckoo (Oktavianto & Muhardianto, 2013), Ether (Dinaburg, Royal, Sharif, & Lee, 2008), or CWSandbox (Willems, Holz, & Freiling, 2007).

In contrast to the majority of prior art, binaries were divided into training and testing sets according to the dates they were collected rather than randomly. This approach is more realistic since it does not overestimate the detection performance as some malware families may not be known at the time of training, as they might have appeared later. Thus, all training samples collected prior to November 12, 2016 (72,689 malicious binaries and 47,961 legitimate binaries) were used for training, and remaining samples (70,995 malicious binaries and 38,746 legitimate binaries) were used for testing.

Table 3
Number of samples of malware families in the data set. The malware families for individual samples were determined using AVClass tool Sebastián, Rivera, Kotzias, and Caballero (2016).

Malware family	#samples	Malware family	#samples
nemucod	13,731	amonetize	1167
cerber	12,790	nanocore	1027
bladabindi	10,910	loadmoney	957
locky	9855	yakes	892
gamarue	7670	bifrose	802
darkkomet	4642	autoit	776
hupigon	3539	kolabc	704
upatre	3255	waldek	683
tinba	3089	pdfka	644
scar	2949	shipup	624
swrort	2859	rebhip	611
zbot	2412	razy	597
virlock	1789	agentb	575
fareit	1757	poison	548
farfli	1743	xtrat	508
zegost	1712	onlinegames	501
virut	1551	ramnit	492
adwind	1534	magania	462
zusy	1499	atraps	457
ircbot	1437	softpulse	457
zerber	1321	banload	386
palevo	1268	ruskill	373
vobfus	1241	downloadassistant	371
delf	1223	binder	349
donoff	1207	*remaining MW families*	31,738
Total malicious			143,684
Total legitimate			86,707

Table 4
Selected AV engines that received full 6 points for performance in AV-Test report from December 2016 (AV-Test, 2016b).

AhnLab, *V3 Internet Security*	G Data, *InternetSecurity*
Avira, *Antivirus Pro*	Kaspersky Lab, *Internet Security*
Bitdefender, *Internet Security*	Microworld, *eScan internet security suite*
ESET, *Internet Security*	Symantec, *Norton Security*
F-Secure, *Safe*	Trend Micro,*Internet Security*

4.2. Hyper-parameter optimization

All compared methods have several parameters that have to be tuned to achieve good detection accuracy. While in Rieck and the proposed method the parameters have to be optimized using grid search (detailed below), AMAL is designed to perform such optimization during training in order to select both the optimal classifier (SVM, linear SVM, decision trees, logistic regression, k-nearest

[5] According to http://www.w3schools.com/browsers/browsers_os.asp Windows 7 has 34.6% market share against 1.0% covered by Windows XP, 11.1% covered by Windows 8 and 30.9% covered by Windows 10.

neighbor and perceptron) and its parameters and thus it does not need to optimize its parameters in advance.

Since Rieck uses SVM with L2 regularization and polynomial kernel there are two parameters that need to be tuned: misclassification cost $C \in \{10^{-2}, \ldots, 10^{8}\}$ and degree of the kernel $d \in \{1, \ldots, 5\}$. The optimal configuration achieving highest accuracy estimated by five-fold cross-validation on the training data was $C = 10^4, d = 4$.

The random forest classifier described in Section 3 contains several parameters such as the number of trees $K \in \{10, 20, 50, 100, 200\}$, maximal depth $d_m \in \{5, 10, 30, 50, \infty\}$, minimal number of samples in node to perform split $s_n \in \{2, 4, 6, 10, 20\}$, and criterion $c \in \{\text{gini, entropy}\}$. All remaining parameters (maximal number of features, minimal number of samples in leaf, maximal number of leafs, class weights, minimum weighted fraction of the total sum of weights in leaf, minimal impurity for split) were set to their default values as defined in the Scikit-learn library (Pedregosa et al., 2011) since according to our experiments they have little influence on detection performance. The optimal configuration of parameters with respect to accuracy estimated by five-fold cross-validation on training data was $K = 100$, $d_m = \infty$, $s_n = 2$ and $c = \text{gini}$.

Additional two parameters (minimal similarity $\epsilon \in \{0.1, \ldots, 0.9\}$ and size of randomly selected subsets $k \in \{10^4, 2 \cdot 10^4, 5 \cdot 10^4, 10^5, 2 \cdot 10^5, 5 \cdot 10^5, \infty\}$) affect the clustering of the resource names described in Section 3.5. The minimal similarity was optimized on a manually labeled set of file paths and registry keys that were clustered with different values of ϵ. The resulting clusters were evaluated with respect to the *adjusted rand index (Rand, 1971)*, a well known score for evaluation of clustering algorithms, and the optimal value of $\epsilon = 0.4$ was selected. To find the optimal size of randomly selected subsets k the accuracy of the whole proposed method with different settings of parameter k was estimated using five fold cross validation on randomly selected subset of training data.[6] Since the differences between various settings were negligible, the value of the parameter $k = 10^5$ was selected as a reasonable balance. Low value of parameter k increases the number of iterations l performed by the clustering algorithm, since too many samples are rejected to be too dissimilar to available cluster prototypes, and high value increases the quadratic cost for computation of adjacency matrix required by Louvain method.

Classification performance was measured with standard evaluation metrics (Fawcett, 2006): *true positive rate (TPR), false negative rate (FNR), true negative rate (TNR), false positive rate (FPR)* and *accuracy*. Since the experimental scenario is binary (positive malware vs. negative benign), the TPR (FNR) is the proportion of correctly (incorrectly) classified malware samples, TNR (FPR) is the proportion of correctly (incorrectly) classified legitimate samples and accuracy is the rate of correctly classified samples regardless their class.

4.3. Experimental results

The experiments discussed in this section are divided into three parts that provide complex evaluation of all aspects of the proposed method.

The first experiment compares the classification performance of the MIL model to Rieck and AMAL using the complete dataset to evaluate the ability of the proposed approach to correctly detect malicious samples. Note that to evaluate AMAL on the complete training set, the meta learner was not allowed to use SVM classifier with RBF kernel due to excessive computational requirements. However, AMAL's meta-learner has never selected this variant of the SVM classifier in smaller experiments performed in this work, hence removing it most probably does not have any impact on the results. Next experiment estimates the performance degradation of compared approaches when only a limited number of data are available for training (5%, 10%, 20% and 100% of training samples). The last part evaluates the robustness of the proposed MIL-based representation of malware samples by comparing results obtained with the random forest to results obtained with two other classifiers, linear SVM and multi-layered perceptron.

[6] The subset was limited to ~30,000 samples in order to limit the number of resources so that complete clustering could be performed.

Table 5
True (TPR), false (FPR) positive rates and accuracies of evaluated methods estimated on the training and testing set.

	Estimated on testing set			Estimated on training set		
	TPR	FPR	ACC	TPR	FPR	ACC
MIL model	**0.954**	**0.067**	**0.943**	0.973	0.061	0.956
Rieck	0.934	0.081	0.926	**0.974**	**0.014**	**0.980**
AMAL	0.795	0.108	0.845	0.845	0.047	0.899

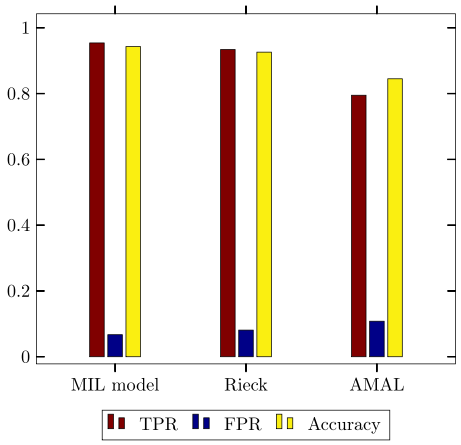
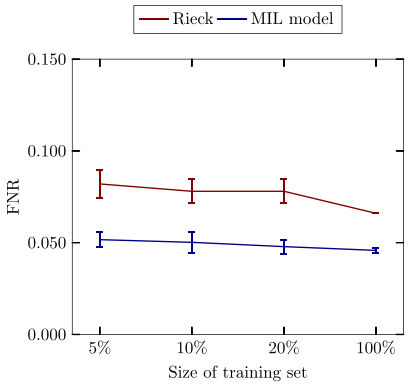


Fig. 1. True (TPR), false (FPR) positive rates and accuracies of evaluated methods estimated on the training and testing set.

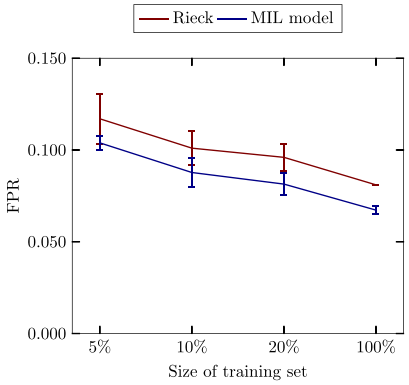
4.3.1. Classification of malware samples

The detection rates and accuracy of classifiers trained on all 121 115 training samples as estimated on testing samples are shown in Table 5 and Fig. 1. The differences between evaluation metrics indicate that the proposed approach outperforms both Rieck and AMAL having the lowest false positive rate and false negative rate. A deeper analysis of the misclassifications produced by the proposed approach revealed that most of the false positives (legitimate binaries classified as malware) were software utilities such as TeamViewer that install themselves into system directories without any user interaction. Since their incidence in the training set was relatively low, the random forest was not able to precisely learn this type of behavior. A second source of errors are false negatives (malware samples classified as benign) where almost 70% are caused by insufficient numbers of training samples (less than 100 samples) from corresponding malware families. Another 11% of false negatives was caused by concept drift as a portion of testing samples exhibited different behaviors than training samples, i.e. created files or registry keys followed different pattern, network communication significantly different URLs, etc.

Large gaps between training and testing accuracies for AMAL and Rieck suggest that manually created features and BoW features do not generalize over longer periods of time as well as fea-



(a) False negative rates



(b) False positive rate

Fig. 2. Comparison of FNR and FPR for Rieck and proposed method trained on training sets of different sizes (5%, 10%, 20% and 100% of training samples).

tures created through clustering do. This suggests that clustering removes some randomization of resource names while retaining a large part of information content.

4.3.2. Evaluation of performance degradation

Fig. 2a and b show graphs of FNR and FPR rates for larger sizes of the training set expressed as fraction of the data available for training. For fair comparison the testing set was kept static containing all 110 140 samples collected after November 12, 2016. Both graphs show that the proposed approach is able to achieve lower FNR and FPR using fewer samples. In fact, the proposed approach achieved FNR of 0.052 using just 5% of samples, while Rieck achieved 0.066 using the full training set. Similarly, the proposed approach needed just 20% of samples to achieve the same FPR 0.081 as Rieck on all samples.

Fig. 2a and b also shows that while false negative rates almost do not change with respect to the size of the training set (especially for the proposed approach), the false positive rates decrease dramatically. This suggests that learning behavior of legitimate applications is more difficult than that of malware, which can be caused by the fact that the behavior of malware is more uniform than that of legitimate applications. This corroborates the motivation of this work, that even though malware authors try to randomize, they tend to randomize with same sort of regularity, which leads to uniformity.

Table 6
Classification of malware samples using various standard classifiers trained on proposed MIL representation.

	Estimated on testing set		
	TPR	FPR	ACC
Random Forest	0.954	0.067	0.943
Linear SVM	0.952	0.063	0.944
MLP	0.951	0.074	0.938

4.3.3. Robustness of MIL-based representation

To evaluate the robustness of the proposed MIL model, we trained two other standard classifiers, linear SVM and multi-layered perceptron, on the same representation used for training random forest, with parameters of the classifiers optimized the same way as the parameters of the random forest. The selection of these two base learners was based on the work of Fernández-Delgado, Cernadas, Barro, Amorim, and Amorim Fernández-Delgado (2014) where these classifiers provided the best results across multitude of different classification tasks.

The values of TRP, FPR and ACC estimated on testing set are summarized in Table 6. The minor differences between performance of individual base learners indicate that the proposed MIL model is robust and provides consistent results with different types of classifiers.

4.4. Detection limits

The experimental results hint at where are the limits of classifying binaries executed in sandbox. When a binary (or all binaries of some malware family) does not perform any actions changing the data used by the proposed or other methods (files, mutexes, network communication, registry keys) it clearly evades detection. An example of such malware is bitcoin miner that resides only in memory without any additional footprint (no operations with files, no operations with registry keys, no mutexes, very limited network communication). Such malware has to be carefully crafted to avoid any interaction with system resources (statically compiled to carry all libraries in the executable, limited network communication, no mutexes ensuring that only single instance is running on the same machine, no persistency after reboot, etc.). Fortunately, at the time of writing this work, this is not an easy task and the majority of malware authors choose to interact with system resources rather than sacrifice functionality.

Another limitation is the fact that a growing number of malware families are equipped with advanced anti-VM and anti-sandbox features and/or are targeted to specific environments (Stuxnet (Falliere, Murchu, & Chien, 2011)). Such malware families do not reveal their true purpose during sandboxing or mimic less severe types of malware (adware, PUA,[7] etc.). This fact is recognized by the community as the main factor hindering the performance of dynamic analysis as the whole. Addressing this issue is out of the scope of this paper.

The last aspect we need to discuss is the false positive rate. The analysis of the results from Section 4 revealed that a large number of false alarms is caused by applications that install themselves into system directories without user's interaction and since their number is limited, the classifier was unable to fit this behavior. A solution is of course to improve the training data by including a

[7] Potentially unwanted application.

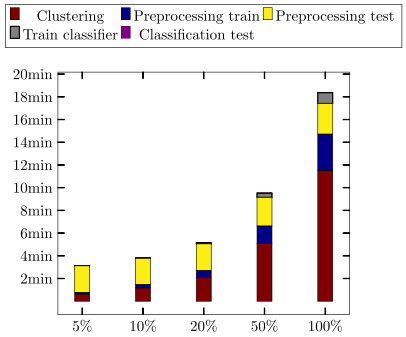


Fig. 3. Time (in minutes) required for clustering of system resources (*Clustering*), projecting training (*Preprocessing train*) and testing (*Preprocessing test*) data using estimated vocabulary, training the random forest classifier (*Train classifier*) and classification of samples from test set (*Classification test*) when the training data is limited to 5%, 10%, 20%, 50% and 100% of original training dataset.

larger number of such samples and thus achieve lower false positive rate.

4.5. Scalability and computational complexity

The last aspect we will discuss is the scalability of the proposed solution and prior art. Since the proposed solution employs clustering to project the raw input data into a feature space with a lower dimension, a large portion of the training time is spent in the clustering phase. However, the preprocessing of the dataset used in above experiments was much faster ($\sim$ 18 min) than the highly optimized pre-computation of the full kernel matrix required by Rieck ($\sim$ 7 h).[8] This is caused by the fact that the time required by Rieck for preprocessing grows quadratically with the number of training samples in contrast to the proposed solution with linear complexity (up to an additive constant, see Section 3.5) which can be seen in Fig. 3. It summarizes time required for preprocessing of the data with different amount of samples used for training (5%, 10%, 20%, 50% and 100%). Note that the proposed approach can be easily parallelized or distributed since in every iteration the nearest neighbor search depends only on a limited set of current cluster prototypes C' and thus further improve the scalability of the MIL-based solution. Regarding the memory requirements of the proposed approach, the main memory requirements arise from the amount of data used for training. As soon as the representation is built, the preprocessing of testing samples can be performed online with minimal memory requirements.

Another benefit of the proposed solution is tied to the representation itself. Since the clustering is performed only on training samples, in order to classify unknown samples we need to store only the cluster prototypes determined during training. For the whole training dataset used in this paper, which contains over 7 million unique resource names projected into $\sim$10, 000 features, only 100,000 instances need to be stored. In contrast, the kernelized SVM classifier used by Rieck et al. requires to store all training samples (over 120,000 samples in the data discussed in Section 4) with all actions (on average 2000 actions per sample) in order to make prediction on unknown samples.

In contrast to both the proposed solution and Rieck, AMAL does not need any preprocessing since the features can be extracted per sample. However, the complexity arises from the design of the training process. Authors in Mohaisen et al. (2015) argue that the dynamic selection of both optimal algorithm and its parameters provides optimal results, but this design makes the training process computationally expensive since every training of the meta-learner requires to evaluate all possible combinations of parameters for all its classifiers. Another aspect is the selection of classifiers itself. Authors propose to use an array of classifiers such as kernelized SVM, linear regression, decision trees, perceptron, etc. However, the complexity of some classifiers (e.g. kernelized SVM) prevents any large-scale training. Moreover, according to the evaluation the AMAL's detection capabilities are not sufficient for real-world deployment since both FPR and FNR are nearly 20%, which is clearly insufficient.

[8] The processing times were estimated on 32-core machine with 250GB RAM (RAM was utilized from ˜40%) running GNU Linux with the code implemented in Java 8 (Oracle JDK).

5. Conclusion

Using sandboxing and dynamic analysis for malware classification is a popular approach. This paper has proposed a novel model of malware behavior observed through its interactions with the operating system and network resources (operations with files, mutexes, registry keys, operations with network servers or error messages provided by the operating system) based on multiple instance learning. The proposed model combines similarities specifically designed for different types of system resources with an efficient clustering of resource names to build a vocabulary used to project malware samples into a low-dimensional space suitable for classifiers such as random forest. The approach has been shown to effectively reduce the impact of randomization commonly employed by malware authors to avoid detection.

The proposed solution was extensively compared to related state of the art on a large corpus of binaries where it demonstrated significant increase in precision of malware detection as it was able to detect more malware (95.4% correctly detected malware vs. 93.4% and 79.5%) with lower number of false alarms (6.7% of false alarms vs. 8.1% and 10.8%). Moreover, the analysis of processing times shown that the proposed approach required only fraction of processing time compared to the state of the art methods and thus it is better suited for large datasets.

The analysis of classification performance and scalability shown that the proposed approach is well-suited for deployment anywhere where the sandboxing is currently deployed (from near-real-time to post-mortem analysis) as the processing time required for modeling and classification of malware samples is insignificant compared to the time required for instrumentation of the binaries.

Acknowledgment

The work of Tomás˜Pevný was supported by the grant GAČR 15-08916S.

Appendix A. Appendix

A1. Frobenius product and Frobenius norm

For two matrices $\mathbf{A} \in \mathbb{R}^{n\times m}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ we define *Frobenius product* $\langle \cdot, \cdot \rangle_F$and *Frobenius norm* $\| \cdot \|_F$ as follows

$$\langle A, B\rangle_F = \sum_{i=1}^{n}\sum_{j=1}^{m} A_{ij} \cdot B_{ij}$$

$$\|\mathbf{A}\|_F = \sqrt{\langle A, A\rangle_F} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} A_{ij}^2}$$

Appendix 9 - Stiborek, J. and Pevný, T. and Rehák, M. : *Multiple instance learning for malware classification* [55]

References

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth acm conference on data and application security and privacy* (pp. 183–194). doi:10.1145/2857705.2857713.

Amores, J. (2013). Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence, 201*, 81–105. doi:10.1016/j.artint.2013.06.003.

Anderson, B., Quist, D., Neil, J., Storlie, C., & Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in Computer Virology, 7*(4), 247–258. doi:10.1007/s11416-011-0152-x.

Anderson, B., Storlie, C., & Lane, T. (2012). Improving malware classification. In *Proceedings of the 5th acm workshop on security and artificial intelligence - aisec '12* (p. 3). New York, New York, USA: ACM Press. doi:10.1145/2381896.2381900.

Andrews, S., Tsochantaridis, I., & Hofmann, T. (2003). Support vector machines for multiple-instance learning. *Advances in neural information processing systems*.

AV-Test (2016a). AV-test malware statistics. *Technical Report*. AV-TEST GmbH.

AV-Test (2016b). Consumer full product testing november/december 2016. *Technical Report*. AV-TEST GmbH.

Bayer, U., Comparetti, P. P. M., Hlauschek, C., Kruegel, C., & Kirda, E. (2009). Scalable, behavior-based malware clustering.. In *Proceedings of the 16th annual network and distributed system security symposium (ndss 2009)*.

Bishop, C. M. (2011). *Pattern recognition and machine learning*: 4. Springer. doi:10.1117/1.2819119.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment, 2008*(10), P10008. doi:10.1088/1742-5468/2008/10/P10008.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. doi:10.1023/A:1010933404324.

Bunescu, R. C., & Mooney, R. J. (2007). Multiple instance learning for sparse positive bags. In *Proceedings of the 24th international conference on machine learning - icml '07: 79* (pp. 105–112). New York, New York, USA: ACM Press. doi:10.1145/1273496.1273510.

Cano, A. (2017). An ensemble approach to multi-view multi-instance learning. *Knowledge-Based Systems, 136*, 46–57. doi:10.1016/j.knosys.2017.08.022.

Cano, A., Zafra, A., & Ventura, S. (2015). Speeding up multiple instance learning classification rules on GPUs. *Knowledge and Information Systems, 44*(1), 127–145. doi:10.1007/s10115-014-0752-0.

Canzanese, R., Mancoridis, S., & Kam, M. (2015). Run-time classification of malicious processes using system call analysis. In *2015 10th international conference on malicious and unwanted software, malware 2015* (pp. 21–28). doi:10.1109/MALWARE.2015.7413681.

Chen, Z., Roussopoulos, M., Liang, Z., Zhang, Y., Chen, Z., & Delis, A. (2012). Malware characteristics and threats on the internet ecosystem. *Journal of Systems and Software, 85*(7), 1650–1672. doi:10.1016/j.jss.2012.02.015.

Chen, Z., Wei, P., & Delis, A. (2008). Catching remote administration trojans (RATs). *Software: Practice and Experience, 38*(7), 667–703. doi:10.1002/spe.837.

Christodorescu, M., & Jha, S. (2006). Static analysis of executables to detect malicious patterns. *Technical Report*. Madison: Computer Sciences Department University of Wisconsin.

CISCO Systems, Inc. (2017). AMP ThreatGrid. Accessed 17.06.17http://www.cisco.com/c/en/us/solutions/enterprise-networks/amp-threat-grid/index.html.

Cortes, C., Mohri, M., & Rostamizadeh, A. (2012). Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning*.

Das, S., Liu, Y., Zhang, W., & Chandramohan, M. (2016). Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Transactions on Information Forensics and Security, 11*(2), 289–302. doi:10.1109/TIFS.2015.2491300.

Dinaburg, A., Royal, P., Sharif, M., & Lee, W. (2008). Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th acm conference on computer and communications security - ccs '08* (p. 51). New York, New York, USA: ACM Press. doi:10.1145/1455770.1455779.

Dong, L. (2006). *A comparison of multi-instance learning algorithms*. Ph.D. thesis. The University of Waikato.

Falliere, N., Murchu, L. O., & Chien, E. (2011). W32.Stuxnet Dossier analysis report. *Technical Report*. Symantec Inc.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874. doi:10.1016/j.patrec.2005.10.010.

Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., & Amorim Fernández-Delgado, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research, 15*, 3133–3181. doi:10.1016/j.csda.2008.10.033.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports, 486*(3-5), 75–174. doi:10.1016/j.physrep.2009.11.002.

Garcia, D. R. (2016). AntiCuckoo. Accessed: 17.06.15 https://github.com/David-Reguera-Garcia-Dreg/anticuckoo.

Gärtner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels. In *Proceedings of the nineteenth international conference on machine learning: 22* (pp. 179–186).

Gonen, M., & Alpaydin, E. (2011). Multiple kernel learning algorithms. *Journal of Machine Learning Research, 12*, 2211–2268.

Google Inc. (2017). VirusTotal.com. Accessed 17.05.09https://www.virustotal.com/.

Hansen, S. S., Larsen, T. M. T., Stevanovic, M., & Pedersen, J. M. (2016). An approach for detection and family classification of malware based on behavioral analysis. In *2016 international conference on computing, networking and communications (icnc)* (pp. 1–5). IEEE. doi:10.1109/ICCNC.2016.7440587.

Hu, X., Chiueh, T.-c., & Shin, K. G. (2009). Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th acm conference on computer and communications security - ccs '09* (p. 611). New York, New York, USA: ACM Press. doi:10.1145/1653662.1653736.

Jang, J., Brumley, D., & Venkataraman, S. (2010). BitShred: Fast, scalable malware triage. *Technical Report*. Cylab, Carnegie Mellon University, Pittsburgh, PA.

Jusko, J., Rehák, M., Stiborek, J., Kohout, J., & Pevný, T. (2016). Using behavioral similarity for botnet command and control discovery. *IEEE Intelligent Systems*.

Kc, G. S., Keromytis, A. D., & Prevelakis, V. (2003). Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th acm conference on computer and communication security - ccs '03* (p. 272). New York, New York, USA: ACM Press. doi:10.1145/948109.948146.

Kinable, J., & Kostakis, O. (2011). Malware classification based on call graph clustering. *Journal in computer virology 7.4*, 233–245.

Kohout, J., & Pevny, T. (2015). Automatic discovery of web servers hosting similar applications. In *Proceedings of the 2015 ifip/ieee international symposium on integrated network management, im 2015* (pp. 1310–1315). doi:10.1109/INM.2015.7140487.

Kong, D., & Yan, G. (2013). Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th acm sigkdd international conference on knowledge discovery and data mining - kdd '13* (p. 1357). New York, New York, USA: ACM Press. doi:10.1145/2487575.2488219.

Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., & Kirda, E. (2010). AccessMiner: Using system-centric models for malware protection. In *Acm conference on computer and communications security 2010* (pp. 399–412). doi:10.1145/1866307.1866353.

Leistner, C., Saffari, A., & Bischof, H. (2010). MIForests: Multiple-instance learning with randomized trees. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics): 6316 LNCS* (pp. 29–42). Springer. doi:10.1007/978-3-642-15567-3_3. http://www.springerlink.com/index/5H73156161110541.pdf.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady, 10*(8), 707–710. citeulike-article-id:311174.

Li, W. J., Wang, K., Stolfo, S. J., & Herzog, B. (2005). Fileprints: Identifying file types by n-gram analysis. *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005, 2005*(July), 64–71. doi:10.1109/IAW.2005.1495935. 2005.

Lo, R. W., Levitt, K. N., & Olsson, R. A. (1995). MCF: A malicious code filter. *Computers & Security, 14*(6), 541–566. doi:10.1016/0167-4048(95)00012-W.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research, 2*, 419–444. doi:10.1162/153244302760200687.

Ma, W., Duan, P., Liu, S., Gu, G., & Liu, J. C. (2012). Shadow attacks: Automatically evading system-call-behavior based malware detection. *Journal in Computer Virology, 8*(1-2), 1–13. doi:10.1007/s11416-011-0157-5.

Maron, O., & Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In *Advances in neural information processing systems* (pp. 570–576).

Mohaisen, A., Alrawi, O., & Mohaisen, M. (2015). AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security, 52*, 251–266. doi:10.1016/j.cose.2015.04.001.

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. In *Proceedings - annual computer security applications conference, acsac* (pp. 421–430). IEEE. doi:10.1109/ACSAC.2007.21.

Naval, S., Laxmi, V., Rajarajan, M., Gaur, M. S., & Conti, M. (2015). Employing program semantics for malware detection. *IEEE Transactions on Information Forensics and Security, 10*(12), 2591–2604. doi:10.1109/TIFS.2015.2469253.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys, 33*(1), 31–88. doi:10.1145/375360.375365.

O'Kane, P., Sezer, S., McLaughlin, K., & Im, E. G. (2013). SVM training phase reduction using dataset feature filtering for malware detection. *IEEE Transactions on Information Forensics and Security, 8*(3), 500–509. doi:10.1109/TIFS.2013.2242890.

Oktavianto, D., & Muhardianto, I. (2013). *Cuckoo malware analysis*. Packt Publishing.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research, 12*(Oct), 2825–2830.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association, 66*(336), 846–850. doi:10.2307/2284239.

Rieck, K., Holz, T., Willems, C., Dussel, P., & Laskov, P. (2008). Learning and classification of malware behavior. In *Dimva 2008* (pp. 108–125).

Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security, 19*(4), 639–668. doi:10.3233/JCS-2010-0410.

Sebastián, M., Rivera, R., Kotzias, P., & Caballero, J. (2016). AVclass: A tool for massive malware labeling. In *Research in attacks, intrusions, and defenses: 19th international symposium, raid 2016, paris, france, september 19-21, 2016, proceedings* (pp. 230–253). doi:10.1007/978-3-319-45719-2_11.

Sharif, M., Lanzi, A., Giffin, J., & Lee, W. (2008a). Impeding malware analysis using conditional code obfuscation. *Informatica*.

Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., & Lee, W. (2008b). Eureka: A framework for enabling static malware analysis. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics): 5283 LNCS* (pp. 481–500). doi:10.1007/978-3-540-88313-5-31.

Sivic, J., & Zisserman, A. (2003). Video Google: a text retrieval approach to object matching in videos. In *Proceedings ninth ieee international conference on computer vision* (pp. 1470–1477vol.2). IEEE. doi:10.1109/ICCV.2003.1238663.

Appendix 9 - Stiborek, J. and Pevný, T. and Rehák, M. : *Multiple instance learning for malware classification* [55]

Symantec Security Response (2011). Sality: Story of a peer- to-peer viral network. *Technical Report*. Symantec, Inc.

Wang, J., & Zucker, J.-D. (2000). Solving multiple-instance problem: A lazy learning approach. In *In proc. 17th international conf. on machine learning* (p. 1119—1125).

Weidmann, N., Frank, E., & Pfahringer, B. (2003). A two-level learning method for generalized multi-instance problems. In *European conference on machine learning* (pp. 468–479). Springer.

Willems, C., Holz, T., & Freiling, F. (2007). Toward automated dynamic malware analysis using CWSandbox. *IEEE Security and Privacy Magazine, 5*(2), 32–39. doi:10.1109/MSP.2007.45.

Wüchner, T., Ochoa, M., & Pretschner, A. (2014). Malware detection with quantitative data flow graphs. *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, 271–282. doi:10.1145/2590296.2590319.

Zafra, A., & Ventura, S. (2010). G3P-MI: A genetic programming algorithm for multiple instance learning. *Information Sciences, 180*(23), 4496–4513. doi:10.1016/j.ins.2010.07.031.

Zhang, Q., & Goldman, S. a. (2002). EM-DD: An improved multiple-instance learning technique. In *Advances in neural information processing systems: 14* (pp. 1073–1080).

Zhang, T., Ramakrishnan, R., & Livny, M. (1997). BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery, 1*(2), 141–182. doi:10.1023/A:1009783824328.

Zhou, Z.-H., Sun, Y.-Y., & Li, Y.-F. (2009). Multi-instance learning by treating instances as non-I.I.D. samples. In *Proceedings of the 26th annual international conference on machine learning - icml '09* (pp. 1–8). New York, New York, USA: ACM Press. doi:10.1145/1553374.1553534.

Zimek, A., Gaudet, M., Campello, R. J. G. B., & Sander, J. (2013). Subsampling for efficient and effective unsupervised outlier detection ensembles. In *Proceedings of the 19th acm sigkdd international conference on knowledge discovery and data mining - kdd '13* (p. 428). doi:10.1145/2487575.2487676.

Towards Dependable Steganalysis

Tomáš Pevný[a,b] and Andrew D. Ker[c]

[a]Cisco Systems, Inc., Cognitive Research Team in Prague, Czech Republic
[b]Agent Technology Center, Czech Technical University in Prague,
Karlovo naměstí 13, 121 35 Prague 2, Czech Republic.
[c]Oxford University Department of Computer Science, Parks Road, Oxford OX1 3QD, UK.

ABSTRACT

This paper considers the research goal of dependable steganalysis: where false positives occur once in a million or less, and this rate is known with high precision. Despite its importance for real-world application, there has been almost no study of steganalysis which produces very low false positives. We test existing and novel classifiers for their low false-positive performance, using millions of images from Flickr. Experiments on such a scale require considerable engineering. Standard steganalysis classifiers do not perform well in a low false-positive regime, and we make new proposals to penalise false positives more than false negatives.

1. INTRODUCTION

This paper considers the research goal of *dependable* steganalysis. By this we mean steganalysis that could potentially be suitable for forensic analysis, which requires two properties:

(a) The false positive rate of the detector should be known with high precision.

(b) The false positive rate of the detector should be very low (10^{-6}, or even 10^{-9}).

The aim is to move steganalysis more towards real-world applications.[1] Note that we focus on false positive rates because false negative rates can only be defined when there is a simple alternative hypothesis (for example that a steganographer uses a known embedding algorithm with a known length of payload, or payload with exactly known distribution) which is not likely to fit real-world scenarios. Furthermore, in a real world where true positives are very scarce, the false positives dominate the failure cases.

Both aims are challenging. (a) is difficult because steganalysis is highly dependent on context: the cover source,[2] scene content, and potentially unknown other factors all influence accuracy. There has been some research aimed in this direction[3] but empirical evidence suggests that false alarm rates are not robust in practice. (b) has been relatively little studied: practically every piece of steganalysis literature focuses on error rates under equal priors, or area under an ROC curve, metrics little affected by low false positive performance. One difficulty with (b) is that empirical tests of very low false alarm rates are simply impossible unless the evidence base is enormous.

This paper is an initial move towards (b). Using standard steganalysis features, we modify classifiers to optimize low false positive rates, and provide a very large real-world evidence base (millions of images) to evaluate the results. To do so, we examine the low false-positive region of the ROC directly, and also use a new metric.

In the following subsections we discuss the benchmarking metrics for steganalysis and establish notation. In Sect. 2 we discuss literature on classification that prioritizes one class over another. We propose new linear classifiers in Sect. 3, and adapt ensemble classification to the low false-positive regime in Sect. 4. We measure the performance of the new classifiers, single and in ensembles, in Sect. 5, and draw conclusions in Sect. 6.

Further author information:
T. Pevný: E-mail: pevnak@gmail.com, Telephone: +420 22435 7608
A. D. Ker: E-mail: adk@cs.ox.ac.uk, Telephone: +44 1865 283530

1.1 Benchmarks for steganalysis

Early steganalysis literature struggled to reduce the performance envelope of a detector (false positive and false negative rates as the payload size varies) to simple benchmarks. See Ref. 4 for a survey, which includes some of the popular options from the literature at the time, and a more recent discussion in Ref. 5. For at least the last five years, however, by far the most popular benchmark is the *minimal misclassification rate under equal priors* (assuming that the payload size is fixed). This can be defined by

$$P_{\mathrm{E}} = \tfrac{1}{2}\min(P_{\mathrm{FP}} + P_{\mathrm{FN}})$$

where P_{FP} and P_{FN} represent the false positive and false negative rate and the minimum ranges over the ROC curve. If, in application, the detector expects to see equal numbers of true positive and negative classes, and the costs of misclassification are symmetrical, then this is indeed the threshold that should be chosen and P_{E} represents the error rate of such a detector.

The P_{E} benchmark is useful for demonstrating advances in steganalysis feature design or classification, but as a measure of practical performance it seems rather far from reality. In almost any realistic problem domain, the vast majority of images transmitted will be covers, because most transmissions are not covert. Even if each false positive result *costs the detector the same* as a false negative (which itself is a dubious assumption), that does not imply an equal weighting between P_{FP} and P_{FN}. When positives are observed rarely, false negatives have fewer opportunities to happen, compared with false positives.

There is no perfect benchmark, and every problem application will have a slightly different preference for the classifier's performance envelope. However, *dependable* steganalysis requires low false positive rates, and in practice the cost of a false negative is likely to be relatively low (an enemy steganographer will probably act more than once). So we propose a metric which we call FP-50, the false positive rate when the false negative rate is 50%. This benchmark is not new, and indeed it was advocated in Ref. 4, but it has not been used much in steganalysis before now.

We also need to make more clear separation of training and testing sets. Following the gold standard of machine learning, we will use *three* sets of data: *training* data to learn a classifier, *validation* data to optimize hyperparameters and thresholds, and *testing* data to measure a final *single* $(P_{\mathrm{FP}}, P_{\mathrm{FN}})$ pair. Training and validation sets are selected repeatedly from the same pool, but testing is completely disjoint. If detection thresholds were set using results from the testing data (which is typical for the steganalysis literature, when drawing a ROC curve) this would be considered a form of cheating. We will still draw ROC curves, using a semi-log plot to display the low false-positive region directly, but they will be from the validation set. Our true benchmark is the final false positive/negative rate on the never-before-seen testing set.

1.2 Notation

We use the following notation throughout the paper. P^{c} (respectively, P^{s}) denotes the probability distribution of all cover images (respectively, stego images, with an implicit embedding method and payload or payload distribution). $\mathcal{I}^{\mathrm{c}}$ ($\mathcal{I}^{\mathrm{s}}$) is a finite set of cover (stego) images, typically for training a classifier. $\{x_i\}_{i\in\mathcal{I}}$ represents the matrix of features extracted from images in set $\mathcal{I}$, arranged in rows. The domain of the features is $\mathcal{X}$. μ_c and $\mathbf{C}_c$ (μ_s and $\mathbf{C}_s$) denote the empirical mean and covariance of features from $\mathcal{I}^{\mathrm{c}}$ ($\mathcal{I}^{\mathrm{s}}$).

Throughout, λ will be an optional regularisation parameter (in the experimental results we will always set it to zero). $\mathrm{I}[x]$ denotes the indicator function which is equal to 1 when x is true, 0 otherwise.

2. RELATED WORK

Any classification algorithm on continuous data can be adapted to favour false positives over false negatives, by moving a decision boundary. This is the traditional way to trace the receiver operating characteristic (ROC) curve, but it has no guarantee of optimality.

The proper foundation is *classification with imbalanced costs*,[6] which despite its importance has not been studied in steganalysis. Most practical work in the machine learning literature uses a Bayesian framework, with known costs of false positives and false negatives, together with prior probabilities of encountering positive and

negative cases in the data. An example of an algorithm for class-imbalanced problems is the cost-sensitive (or *weighted*) support vector machine (SVM), optimizing the following cost function

$$\arg\min_{\rho,w} \ \frac{\lambda}{2}\|w\|_2^2 + \frac{\eta}{|\mathcal{I}^{\mathrm{c}}|}\sum_{i\in\mathcal{I}^{\mathrm{c}}}\max\{0, w^{\mathrm{T}}x_i - \rho\} + \frac{1-\eta}{|\mathcal{I}^{\mathrm{s}}|}\sum_{i\in\mathcal{I}^{\mathrm{s}}}\max\{0, \rho - w^{\mathrm{T}}x_i\}, \tag{1}$$

where λ is the regularization parameter (here using L_2 regularization, though other options are possible) and η balances the costs of misclassification of cover and stego samples. ρ is the margin hyperparameter.

Another option is to turn traditional logistic regression into a maximum a posteriori problem with a prior, optimizing

$$\arg\min_{\rho,w} \ \frac{\lambda}{2}\|w\|_2^2 + \frac{\eta}{|\mathcal{I}^{\mathrm{c}}|}\sum_{i\in\mathcal{I}^{\mathrm{c}}}\log\left(1+\exp\left(w^{\mathrm{T}}x_i - \rho\right)\right) + \frac{1-\eta}{|\mathcal{I}^{\mathrm{s}}|}\sum_{i\in\mathcal{I}^{\mathrm{s}}}\log\left(1+\exp\left(\rho - w^{\mathrm{T}}x_i\right)\right). \tag{2}$$

In both cases we have an additional hyperparameter η, balancing the cost of false positives and negatives on the training set. It is difficult to justify a correct value of η unless the problem domain is very well-understood (if there is a known cost of error on each class, which is plausible, as well as a known proportion of each class that will be encountered in the wild, which is not), so it might be optimized using cross-validation along with the other hyperparameters. Thus it can only indirectly target a benchmark such as FP-50.

In fact, miniziming the FP-50 benchmark corresponds to a different problem: *Neyman-Pearson classification* aims to minimize the false negative rate such that the false positive rate is below some threshold. This is more applicable in security scenarios, as the number of false positives that the user is willing to tolerate can be usually determined. Surprisingly, few works in machine learning literature deal with Neyman-Pearson classification[7,8] and to the best of our knowledge there is no algorithm directly optimizing this error. This is usually swept under the carpet by claiming that the same classifier can be obtained with an appropriate cost, such as η above. This approach, where η and λ are optimized on training data to minimize FP-50, is used in experiments in Subsection 5.2.

3. PROPOSALS FOR LINEAR CLASSIFIERS

In this work, we will try to target the FP-50 benchmark more directly. We will first present methods for single linear classifiers, and move to ensembles of linear classifiers (in a way that also targets Neyman-Pearson classification) in the following section.

We present two approaches to this problem: one minimizing upper bounds on the FP-50 benchmark, the other using convex surrogates for it. It turns out that these approaches are, in some sense, equivalent.

3.1 Probabilistic approach

Consider the FP-50 benchmark. The optimal classifier minimizing it is

$$\arg\min_{f\in\mathcal{F}} \ \mathbb{E}_{x\sim P^{\mathrm{c}}}\Big[\mathrm{I}\big[f(x) > \mathrm{median}\,\{f(x)|x\sim P^{\mathrm{s}}\}\big]\Big], \tag{3}$$

where $\mathcal{F}$ is the set of all possible classifiers* of the form $\mathcal{X}\mapsto\mathbb{R}$. Optimizing (3) is impractical, since a) P^{c} and P^{s} are unknown, and b) median is not a differentiable function so the optimisation is NP-complete. We tackle these problems by using finite sets of training samples instead of the distributions, and replacing median with mean.

For further simplicity, we restrict $\mathcal{F}$ to be the set of linear classifiers. Due to the well-known kernel trick,[9] the linear classifiers below could be adapted to non-linear circumstances, but instead we will follow the present state-of-art in steganalysis by regaining nonlinearity via an ensemble of linear classifiers.

*$\mathcal{F}$ is called the *hypothesis space* in the jargon of machine learning literature.

Incorporating the above into (3), the problem becomes

$$\arg\min_{w\in\mathbb{R}^d} \mathbb{E}_{x\sim P^c}\Big[\mathrm{I}\big[w^{\mathrm{T}}x > w^{\mathrm{T}}\mu_s\big]\Big] = \arg\min_{w\in\mathbb{R}^d} \mathbb{P}_{x\sim P^c}\big[w^{\mathrm{T}}x > w^{\mathrm{T}}\mu_s\big], \tag{4}$$

where μ_s denotes the mean of stego features. By replacing median with mean, we are assuming that about 50% of stego features projected on w lie beyond their mean, which should be true for moderately symmetrical distributions. Reminiscent of Vapnik's technique,[10] we use probabilitic inequalities to find upper bounds for (4), which we can optimize efficiently. Different optimization problems arise from different inequalities.

Quadratic Chebyschev Minimizer. Applying the standard Chebychev inequality[†] to (4), we solve

$$\arg\min_{w\in\mathbb{R}^d} \frac{w^{\mathrm{T}}\mathbf{C}_c w}{\big((\mu_s-\mu_c)^{\mathrm{T}}w\big)^2}. \tag{5}$$

This problem has a close relationship with the Fisher linear discriminant (FLD): the only difference is that it disregards the shape (covariance) of the stego distribution, which follows because we target the mean of the stego distribution. It has an analytic solution which, it can be shown, corresponds to finding a single unregularized projection by the calibrated least squares (CLS) method that we proposed in Ref. 5. (We also tested other polynomial versions of Chebyschev's inequality, with no success.)

Exponential Chebyschev Minimizer. Applying the exponential version of Chebychev's inequality, also known as the MGF bound[‡] to (4), we solve

$$\arg\min_{w\in\mathbb{R}^d} \sum_{i\in\mathcal{I}^c} e^{t(x_i-\mu_s)^{\mathrm{T}}w}. \tag{6}$$

By varying the scalar t, this balances inequalities based on all moments of the cover distribution. In optimization it is superfluous, since it can be absorbed by w. Equation (6) does not have an analytic solution, but the objective function is strongly convex and fast-converging numerical optimizers can be used.

3.2 Machine learning approach

Applying Chebyschev's inequalities can be alternatively viewed as approximating the ideal cost function assigning 1 to cover values projected beyond the stego mean, and zero otherwise. Machine learning algorithms use several surrogates of this function to make the problem (4) convex and solvable in polynomial time. Popular convex surrogates for $\mathrm{I}[y]$ include: hinge, $\max\{0, 1-y\}$; truncated square, $\max\{0, 1-y\}^2$; square, $(1-y)^2$; exponential, e^{-y}; logistic, $\log(1+e^{-y})$. They are depicted in Figure 1.

For some of these, including $\mathrm{I}[\cdot]$ itself, hinge, and truncated square, there may be an infinite number of solutions to optimizing (4). This is typically solved by using regularization (usually Tikhonov), which corresponds to a preference for simple solutions. Below, optimization problems with different loss functions are shown and their properties discussed. All formulations include L_2 regularization, controlled by a hyperparameter λ, but in our experiments we will set $\lambda = 0$ to turn off regularization. L_2 regularization was chosen as it has a smooth derivative, which is favourable for optimization; L_1 regularization promoting sparse solutions is also possible, but the optimization would be more difficult and it will require the use of special solvers.

Hinge loss. This has been popularized by its use in SVMs, and it requires regularisation. Putting hinge loss into (4) yields

$$\arg\min_{w\in\mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{|\mathcal{I}^c|}\sum_{i\in\mathcal{I}^c} \max\left\{0, 1-w^{\mathrm{T}}(\mu_s - x_i)\right\},$$

which is reminscent of the problem solved in one-class SVM.[11,12] Although the same results can be probably obtained by using weighted SVMs, the proposed formulation has one fewer hyperparameter and it directly optimizes the FP-50 criterion.

[†]The following version: $\mathrm{P}[Y > \epsilon] \le \mathbb{E}\big[(Y-\mu_Y)^2\big]/(\epsilon-\mu_Y)^2$, for $\epsilon > \mu_Y$.
[‡]The following version: $\mathrm{P}[Y > \epsilon] \le \mathbb{E}\big[e^{t(Y-\epsilon)}\big]$, for all $t > 0$.

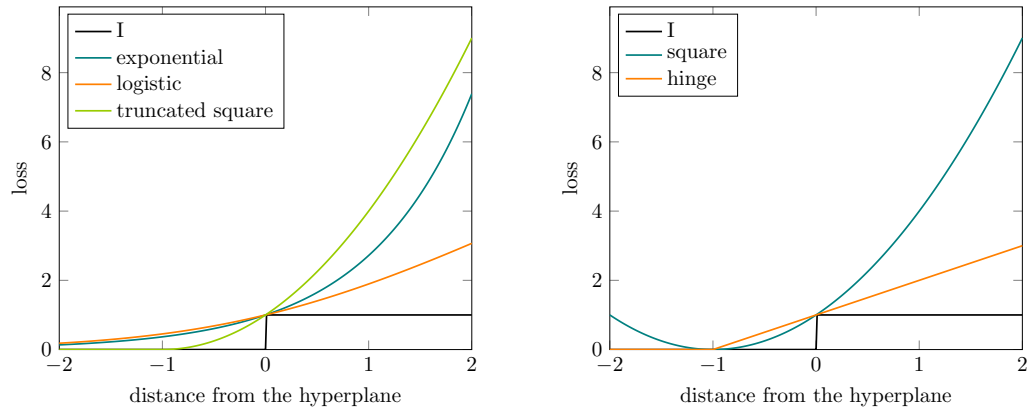


Figure 1: Convex surrogates for the 0-1 loss function.

Truncated square loss. This is also sometimes used with SVMs. Putting truncated squared loss into (4) (again requiring regularization to avoid infinitely many solutions) yields

$$\arg\min_{w\in\mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{|\mathcal{I}^c|}\sum_{i\in\mathcal{I}^c} \max\left\{0, 1 - w^{\mathrm{T}}(\mu_s - x_i)\right\}^2.$$

The advantage of square loss over hinge loss is that it is smooth, which simplifies the optimization by allowing particularly effective stochastic gradient methods.

Square loss. This is advantageous because the optimization has the analytic solution

$$\arg\min_{w\in\mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{|\mathcal{I}^c|}\sum_{i\in\mathcal{I}^c} \left(1 - w^{\mathrm{T}}(\mu_s - x_i)\right)^2 = (\mathbf{C} + \lambda\mathbf{I})^{-1}(\mu_s - \mu_c), \tag{7}$$

where where $\mathbf{C}$ denotes covariance matrix of *cover* samples centered at the mean of *stego* samples, $\mathbf{C} = \frac{1}{|\mathcal{I}^c|}\sum_{i\in\mathcal{I}^c}(\mu_s - x_i)(\mu_s - x_i)^{\mathrm{T}}$.

Note that this is almost identical to the previously-described CLS method (5), if $\lambda = 0$, but with the samples centered differently.

Exponential loss. Intuitively, an exponential penalty on misclassified covers is aligned with our goal of reducing false positives to very low rates. Exponential loss is used in Adaboost,[13] and it does not neccesarily require regularization, although this can be added to force the classifier toward simpler or sparser solutions. Optimization problem (4) with an exponential loss surrogate becomes

$$\arg\min_{w\in\mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{|\mathcal{I}^c|}\sum_{i\in\mathcal{I}^c} e^{-w^{\mathrm{T}}(\mu_s - x_i)}. \tag{8}$$

Turning off the regularization, we have recovered the Exponential Chebyshev Minimizer (6).

Logistic loss. This is used in logistic regression, which can provide a probability estimate of class membership (more than just a binary classification) under the right conditions. The optimization becomes

$$\arg\min_{w\in\mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{|\mathcal{I}^c|}\sum_{i\in\mathcal{I}^c} \log\left(1 + e^{-w^{\mathrm{T}}(\mu_s - x_i)}\right). \tag{9}$$

Similarly to exponential, logistic loss does not necessarily require regularization. Its shape is similar to that of hinge loss, but it has the considerable advantges of being Lipschitz, strongly convex, and infinitely many times differentiable. These are favorable properties for optimization, particularly online optimization.

4. PROPOSALS FOR ENSEMBLE CLASSIFIERS

The classifier used in state-of-the-art steganalysis is an ensemble of FLDs. It has come to dominate the literature because of its simplicity, speed of training, and results with *rich models*.[14] Our linear classifiers based on convex surrogates to (4) can also be used in an ensemble, but the ensemble parameters must be adapted to target the low false-positive regime.

The ensembles used in Ref. 15 consist of a set of base learners diversified by random subspace sampling, which means that each base learner classifier operates on a randomly selected subspace of the original feature space. The subspace sampling makes the training faster, as the complexity of training linear classifiers depends super-linearly (in the case of FLD cubically) on the dimension of the input space. Another side-effect of subspace sampling, to our knowledge not discussed so far in the literature, is that the size of the subspace acts as a regularization parameter controlling the complexity of individual classifiers within the ensemble and preventing over-fitting. Naturally, smaller subspace dimensions make individual classifiers less over-fitted.

4.1 Fusing classifiers to optimize low false-positives

We stick broadly to the ensemble framework used in Ref. 15 – binary classifiers voting with equal weight – but adjust various thresholds used in it.

To formalize the problem, we assume the ensemble consists of the set of classifiers $\left\{f_i | f_i : \mathbb{R}^d \mapsto \mathbb{R}\right\}_{i=1}^{l}$. The output of the ensemble is equal to

$$F(x) = \operatorname{sign}\left[\frac{1}{l}\sum_{i=1}^{l} \mathrm{I}\big[f_i(x) > t_i\big] - t_e\right],$$

where $\{t_i\}_{i=1}^{l}$ are thresholds for the individual classifiers and t_e is the threshold of the ensemble, the number of positive votes required for a positive classification. Determination of these is part of the training, because $F : \mathbb{R}^d \mapsto \{-1, +1\}$.

Kodovsky[15] optimizes thresholds t_i, separately for each classifier, to optimize P_{E} on the training set, and fixes $t_e = 0.5$. The hyperparameters of the ensemble – the number l of base learners and the subsampling dimension d_{sub} – are optimized using cross-validation targeting the overall P_{E} metric.

When we switch to the FP-50 criterion, we are once again unable to optimize $\{t_i\}_{i=1}^{l}$ and t_e collectively, because the problem is $l + 1$ dimensional and the I[·] function is discontinuous. We propose to parameterize all classifier thresholds t_i by a fixed quantile of the distribution $f_i(x)$, $x \sim P^c$. Formally, each threshold t_i is determined by a hyperparameter $\tau \in [0, 1]$ by

$$t_j(\tau) = \arg\max_t \left\{\frac{1}{|\mathcal{I}_{\mathrm{c}}|}\sum_{i\in\mathcal{I}_{\mathrm{c}}} \mathrm{I}[f_j(x_i) > t] \leq \tau\right\}.$$

This approach is connected with Neyman-Pearson classification, since as $|\mathcal{I}_{\mathrm{c}}| \to \infty$, $1 - \tau$ tends to the false positive rate of each individual classifier. With this simplification, optimization is only with respect to τ and t_e.

In our experiments the parameters τ and t_e are found by direct optimization of t_e for each value of $\tau \in \left\{10^i | i \in \{-6, -5.9, \ldots, -0.1, 0\}\right\}$, where the objective is the FP-50 metric on the validation set. We also tested the original ensemble that optimizes P_{E} for each base learner.

5. EXPERIMENTAL RESULTS

Any experiments that want to explore the low-false positive performance of a detector need a huge corpus: to measure robustly a false positive rate of 10^{-6} would need a realistic minimum of about 5×10^6 examples from the negative class (preferably more). In our case this means millions of cover images. Furthermore, the data set needs to be, in some sense, representative in its diversity and difficulty of classification. Working with data of such size needs careful engineering.

In Subsection 5.1 we will explain the database that we collected, and our methodology for testing it. In Subsections 5.2 and 5.3 we will report the results of single linear classifiers and ensembles, respectively. A few further experiments are described in Subsection 5.4.

5.1 Experimental setup

In June 2014, Yahoo! Inc. made available to researchers a data set of 100 million creative-commons licensed images (including a few videos) from the popular photo-sharing website Flickr.[16] We were granted access to this database, and from it collected a set of images useful for testing low false-positive steganalysis. We began by selecting only compressed colour images where the full-size original uploaded image (as opposed to Flickr-downsampled versions) was available, and where the camera model was included in the EXIF data. Then we selected only images which were JPEGs compressed with quality factor 80 (steganalysis of images with varying quality factors is a difficult and largely unsolved problem; quality factor 80 was chosen as the most common choice that did not have very large file sizes). The images were partitioned depending on which *actor* they belong to, where an actor is defined to be a username and camera model combination. Thus each actor's images were uploaded by the same user and taken with the same camera model. Finally, we discarded actors with fewer than 10 images.

This yielded a total of 4 511 523 images from 47 807 actors, a total of approximately 9 128 115 megapixels of data taking 1307 GB on disk. The actor with the most images had 16 886, but only about 1% of the actors had more than 1000 images; the median number of images per actor was 30.

We partitioned this image set into two. The *training and validation* subset consists of 10% of the actors (in fact every 10th actor, ranking actors by size, so that a representative subset was taken): this totals 449 395 images from 4781 actors. The rest is the fixed *testing* subset which contains 4 062 128 images.

In our experiments we are assuming that the ground truth of these images is that they are covers, not used for steganography. There is no reasonable way to verify this assumption, but we can take comfort from the fact that, if it is wrong, only a small proportion of the database would be affected. Furthermore, if some of our assumed-cover images are actually stego images, our empirical estimates of false positive rates will be conservative. There is also an interesting connection with our presentation in Subsection 3.1: if by any chance some assumed-cover images are actually stego objects, Chebyshev minimizers should be be unaffected.[17]

The steganographic algorithm used in all experiments was a simulator of nsF5 embedding with matrix embedding turned off, which means that the number of embedding changes is equal to half the payload. The algorithm was chosen because of its historic importance in steganography, because we know that it can be detected by current steganalysis features, and its speed: essential for the number of images processed here. Since our goal was to measure very reliable classifiers, the payload size simulated was 0.5 bits per nonzero coefficient (bpnc). Our chosen steganographic features were the 22510-dimensional *JPEG rich model* (JRM).[14] The reference implementation takes approximately 15 seconds per megapixel to extract, on our main computing machine, but we re-implemented a highly optimized version in C which takes around 0.5 seconds per megapixel. Our benchmarks are taken from a workstation with two 6-core Xeon processors (Westmere-EP series) running at 3.47Ghz, with 192GB of memory.

We extracted JRM features for every cover image, for every stego image in the training and validation subset, and for 10% of the stego images of each actor in the testing subset (thus 407 417 stego images in the testing subset; the entire testing set is approximately 4.5 million images). It is not necessary, for our benchmarks, to have millions of stego images in the testing set, because we do not need to examine very low false negative rates. Extracting these features from all 5.4 million cover and stego images took around 120 core-days, spread across a small cluster.

	FLD	SVM	Square loss	Exponential loss	Logistic loss
FP-50, training set	$1.11 \cdot 10^{-4}$	$2.18 \cdot 10^{-5}$	$1.45 \cdot 10^{-5}$	0	0
FP-50, validation set	$2.52 \cdot 10^{-4}$	$1.99 \cdot 10^{-4}$	$5.61 \cdot 10^{-4}$	$9.87 \cdot 10^{-4}$	$1.02 \cdot 10^{-3}$
Training time	$4.8 \cdot 10^{2}$s	$3.3 \cdot 10^{4}$s	$2.4 \cdot 10^{2}$s	$5.5 \cdot 10^{4}$s	$1.0 \cdot 10^{5}$s

Table 1: Training and validation false positive rates (when false negative is 50%) of classifiers trained on the whole input space. The last line shows the time needed to train a single classifier on $2 \times 40\,000$ samples.

The cover and stego JRM features, in double precision, require 900GB of disk space. The training and validation features alone are 150GB, which is barely possible to load into memory in one go even on our largest server, and in fact we will only train on up to $2 \times 40\,000$ samples in the experiments for this paper.

The advantage of this data set is that it is from the real world, and it contains all the difficulties that a steganalyst should expect in pratice. In particular, there is cover source mismatch (note that the training/validation set and testing set are from disjoint actors, and when we split the training and validation set apart we will again segregate actors) and a variety of image sizes. Some of the images have been resampled prior to uploading, and it seems likely that image processing operations will have been applied to many of them.

5.2 Single linear classifiers

We begin by training linear classifiers proposed in Subsection 3.2 on the entire 22510-dimensional feature space. We tested standard FLD, linear weighted SVM optimizing (1), and the following convex surrogates for direct optimization of FP-50: square loss (7), almost equivalent to the QCM method (5); exponential loss (8), equivalent to the ECM method (6); and logistic loss (9). We did not test truncated square or hinge loss, after some initial experiments not reported here, because a) their non-smooth nature makes their numerical optimization on large data expensive, and b) they demand regularization, so λ would need to be optimised by an expensive grid-search.

The methodology was as follows. The training and validation subset, consisting of images from 4781 actors, was partitioned into two subsets of actors so that the number of images in each was approximately half the total (approximately 225 000 cover images in each half, plus the same number of corresponding stego images); this breaks the training and validation subsets apart. From the training subset, $2 \times 40\,000$ images (each cover with its corresponding stego) were selected using maximum diversity: the fewest images per actor necessary to reach this total. The rest of the training images were discarded. The classifier was trained on the training set, either by standard FLD or weighted SVM methods,[6, 18] by solving (7), or by numerical optimization of (8) or (9) using an iterative Newton method (we used the implementation at Ref. 19). The size of the used part of the training set was limited by feasibility of this optimization over such large dimension. The trained classifier was tested on all approximately $2 \times 225\,000$ images in the validation set. Each experiment was repeated ten times using different splits into training and validation.

Out of these classifiers, only weighted SVM has hyperparameters controlling the solution: the regularization parameter λ and the class imbalance η. These were optimized by minimizing FP-50 on a full grid $\lambda \in \left\{2 \cdot 10^{i} | i \in \{-3, -4, -5\}\right\}$ and $\eta \in \left\{10^{i} | i \in \{-5, -4, -3, -2\}\right\}$. For every combination, the weighted SVM was trained on a randomly selected 75% of training samples and the FP-50 criteria was estimated from the remaining 25%. This was repeated five times with independent splits. The hyperparameters with the least FP-50 were used to train the final SVM on all training data.

We then measured the FP-50 metric on both the training and validation set, averaging error rates over the ten iterations. The results are displayed in Table 1. The results show that the loss functions which include an exponential penalty for false positives – exponential and logistic loss – have zero false positives on the training data (out of 40 000 cover samples and 10 repetitions), when the false negative rate is 50%, but also expose their weakness: they overfit the training data, and their accuracy on the validation set is slightly worse than the other loss functions. It is unsurprising that an exponential penality encourages overfitting, and it means

that we need some kind of regularisation, which we will supply indirectly in the following subsection with a dimension-subsampling ensemble.

At first sight, Table 1 suggests that the clear winner should be the weighted SVM. But its optimization involves tuning hyperparameters, which would be prohibitively expensive in an ensemble. Moreover, we see the zero false positive results, observed for exponential and logistic loss on the training set, as a positive sign: less a problem of overfitting than a problem of insufficiently diverse training data. Also the lack of hyperparameters simplifies their use.

The last line in Table 1 shows the time to train each classifier (for SVM this includes the search for hyperparameters). Unsurprisingly, training classifiers with algebraic solutions – FLD and square loss – is two orders of magnitude faster than a linear SVM, and two to three orders of magnitude faster than the methods which require numerical optimization. Nonetheless, such time is tractable on a fast machine. Furthermore, the time to *apply* the trained classifier is the same for all methods, since each classifier simply produces a projection direction.

5.3 Ensembles of classifiers

We now plug the classifiers into the ensemble framework descibed in Sect. 4, although we now exclude the SVM base learner as being too slow for ensemble settings.

Our first experiments, similarly to the last section, use only the training and validation sets. As before, we take $2 \times 40\,000$ training images. This time we train a fixed number (300) of FLD base learners, each using a randomly-selected subset of $d_{sub} \in \{100, 250, 500, 1000\}$ features, and optimize the individual base learner thresholds t_i and the ensemble voting threshold t_e according to the method of Subsection 4.1 by measuring their FP-50 benchmark on all approximately $2 \times 225\,000$ images in the validation set. We computed the ROC curves across the data in the validation set for the optimal parameter combination.

We repeated with 10 different splits of training and validation actors, and then the entire procedure (with the same training data and random subspaces) with base learners optimizing square loss, exponential loss, and logistic loss. For each false positive rate on the validation data we averaged the corresponding false negative rate over the ten iterations of the experiment. The ROC curves are displayed in Figure 2, where we have used a logarithmic scale on the x-axis to highlight the low false-positive region. They show that the loss functions with an exponential penalty – exponential and logistic loss – carry good true positive detection power into the low false-positive regime much better than the quadratic penalties in FLD and the square loss surrogate (their ROC curves have steeper slope on the onset). Observe that this advantage is balanced by decreased detection accuracy on higher false positive rates, compared with the traditional ensemble of FLDs, but that aligns with our aim.

Another reason why the square loss surrogate and FLD classifier may be inferior is because their losses are symmetric (see Figure 1). This means that they penalize correctly classified samples, or more precisely that the optimal projection direction w is still influenced by outliers even if they already lie on the correctly-classified side of the boundary. We can see that this is so for FLDs, because Fisher's presentation models the two classes as Gaussian, and outliers in the correctly-classified direction influence the empirical covariance matrix in such a way that the classifier expects also outliers in the incorrectly-classified direction, whether they exist or not.

The exponential and logistic loss surrogates work best if the dimension of the random subspace in the ensemble is kept low: we believe d_{sub} acts as an indirect regularizer, limiting the complexity of the base learners and reducing their propensity to overfit, and overcoming a key weakness in their use as a full-space classifier. Perhaps this, rather than the potential for nonlinear classification, explains why the ensemble of FLDs has been so successful with large-dimensional "rich features"; in any case, the advantage is even more evidence for classifiers with exponential penalties.

Thanks to the reduced dimension, the exponential loss base learner is no longer orders-of-magnitude more expensive to train than the FLD: in our experiments with $d_{sub} = 100$, training the ensemble of 300 learners on $2 \times 40\,000$ samples took 136 seconds with the FLD base learner and 286 for the exponential loss base learner. Both cases required a further 190 seconds to optimize the thresholds.

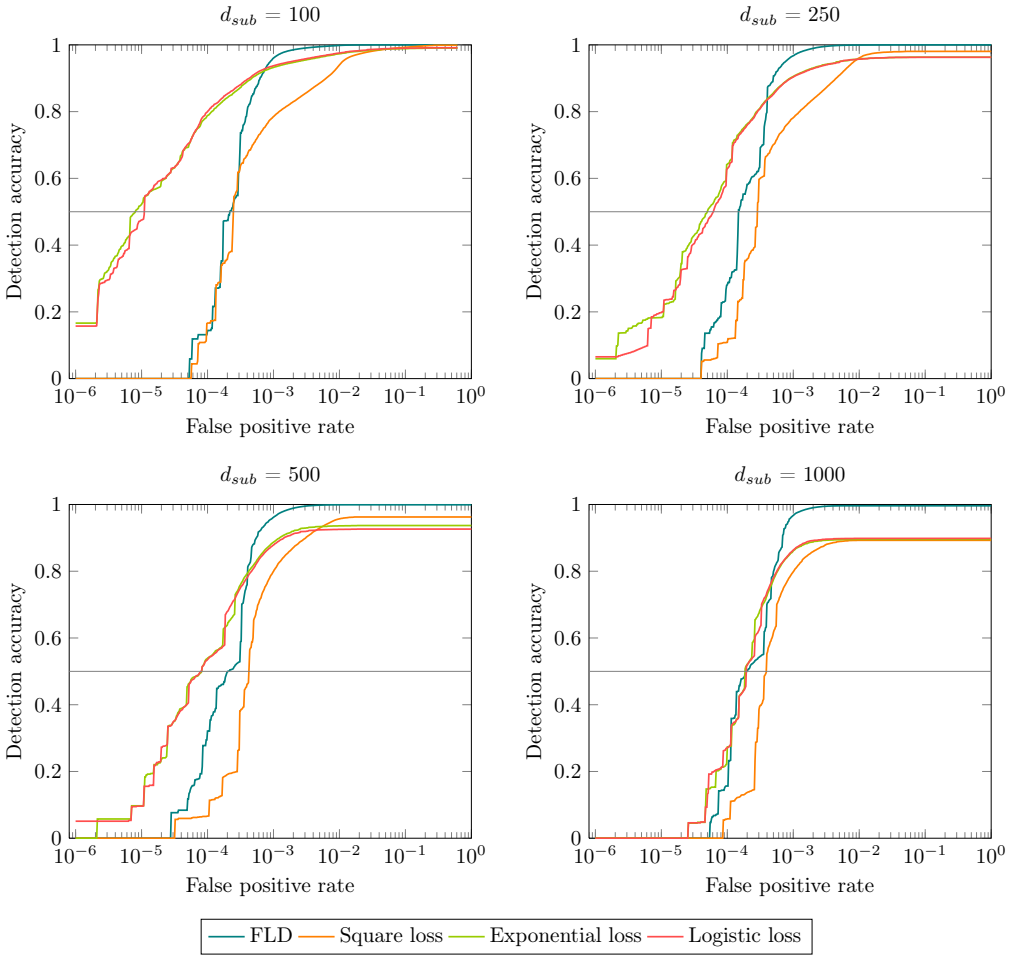


Figure 2: ROC curves, on the validation set, for ensembles of FLDs and other base learners. 300 weak classifiers were trained in random d_{sub} dimensions of the feature space. All classifiers were trained on $2 \times 40\,000$ samples. Note the logarithmic scale of the x-axis.

	False positive rate		False negative rate	
	FLD	Exp. Loss	FLD	Exp. Loss
Optimizing P_{E}	$9.07 \cdot 10^{-3}$	$1.88 \cdot 10^{-3}$	$1.33 \cdot 10^{-3}$	$1.89 \cdot 10^{-2}$
Optimizing FP-50	$3.26 \cdot 10^{-4}$	$5.56 \cdot 10^{-5}$	$4.58 \cdot 10^{-1}$	$5.12 \cdot 10^{-1}$

Table 2: Error rates on the testing set, after optimization of ensemble parameters on the training and validation set.

Ideally we would optimize both the number of base learners (currently fixed at 300) and the subspace dimension. But the training is too slow to perform a grid search, so we postpone such an investigation for further work.

Finally, we apply the trained and threshold-optimized ensembles (selecting $d_{sub} = 100$) to the testing set of 4 062 128 cover images and 407 417 stego images. The size of this set gives us confidence in measuring false positive rates into the region of 10^{-6}. We emphasise this procedure, which did not tune *any* hyperparameters with respect to the testing set, is the gold standard for machine learning. We also tuned another ensemble, this time optimizing the P_{E} benchmark for each base learner and fixing $t_e = 0.5$, to compare with the prior art.

We display results, only for the FLD and exponential loss base learners, in Table 2. The classification thresholds were set for 50% false negatives on the validation set, and achieved close to that on the testing set. The traditional ensemble of FLDs, optimizing for P_{E}, can only manage a false positive rate of approximately 1 in 110 (and a false negative rate much lower than our target of 50%). Using our new ensemble optimization of thresholds for FP-50, the FLD ensemble reduces its false positive rate to approximately 1 in 3000 (and a false negative rate close to 50%), but the same ensemble using exponential loss base learners (equivalent to ECM) achieves a false positive rate of approximately 1 in 18 000. We can have high confidence in these false positive rates because the testing set is representative of the real world, and very large.

5.4 How many training images is sufficient?

We were not able to train on the potential training set of $2 \times 225\,000$ images because of the time and memory requirements: our current numerical optimzation method requires calculations over all data points on each iteration, and the matrix of features, in double precision, would require about 83GB of memory. (This does suggest further research using online base learners.) We had to limit ourselves to a diverse selection of 40 000 image from the training data.

But do we actually need even 40 000 images? If the training images were not sufficiently diverse, training on more images would not be reflected by the lower error rates. We note that this question, largely neglected in the literature, was first raised in Ref. 20 and was also studied in Ref. 21. In experiments for Subsect. 5.3, we also evaluated ensembles trained on two smaller sizes of training data: $2 \times 10\,000$ and $2 \times 20\,000$ samples, as well as $2 \times 40,000$. As before, the images were selected from the training subset to diversify amongst actors as much as possible. We emphasize the validation set was unchanged in these experiments, as were the selection of random subspaces for the base learners. The results appear in Table 3.

The most important observation is that FP-50 does not change much with the size of the training set. This indicates that including more images from the same actors does not increase the diversity of the training set. This also suggests that the cover mismatch phenomenon[2] is at work, and some misclassifications are more likely due to some actors having non-typical image sources, than than individual outlier images.

Notice that FP-50 of the ensemble of FLDs slightly increases as the size of the training set increases to 40 000, in contrast to the general mantra of machine learning that increasing the size of the training set should not degrade performance. This seems likely to be due to the inherent loss function in FLDs, since we observed the same phenomenon with the traditional FLD ensemble from Ref. 15, perhaps making them more prone to outliers.

d_{sub}	Training set	FLD	Square loss	Exponential loss	Logistic loss
100	$2 \times 10\,000$	$1.69 \cdot 10^{-4}$	$1.63 \cdot 10^{-4}$	$7.55 \cdot 10^{-6}$	$8.91 \cdot 10^{-6}$
	$2 \times 20\,000$	$1.69 \cdot 10^{-4}$	$1.63 \cdot 10^{-4}$	$7.55 \cdot 10^{-6}$	$8.91 \cdot 10^{-6}$
	$2 \times 40\,000$	$1.78 \cdot 10^{-4}$	$1.70 \cdot 10^{-4}$	$7.56 \cdot 10^{-6}$	$8.02 \cdot 10^{-6}$
250	$2 \times 10\,000$	$1.72 \cdot 10^{-4}$	$2.43 \cdot 10^{-4}$	$4.57 \cdot 10^{-5}$	$4.61 \cdot 10^{-5}$
	$2 \times 20\,000$	$1.72 \cdot 10^{-4}$	$2.43 \cdot 10^{-4}$	$4.57 \cdot 10^{-5}$	$4.61 \cdot 10^{-5}$
	$2 \times 40\,000$	$1.80 \cdot 10^{-4}$	$2.12 \cdot 10^{-4}$	$4.40 \cdot 10^{-5}$	$4.49 \cdot 10^{-5}$
500	$2 \times 10\,000$	$1.86 \cdot 10^{-4}$	$2.88 \cdot 10^{-4}$	$8.53 \cdot 10^{-5}$	$8.18 \cdot 10^{-5}$
	$2 \times 20\,000$	$1.86 \cdot 10^{-4}$	$2.88 \cdot 10^{-4}$	$8.53 \cdot 10^{-5}$	$8.18 \cdot 10^{-5}$
	$2 \times 40\,000$	$1.91 \cdot 10^{-4}$	$3.02 \cdot 10^{-4}$	$7.27 \cdot 10^{-5}$	$7.19 \cdot 10^{-5}$
1000	$2 \times 10\,000$	$2.45 \cdot 10^{-4}$	$3.10 \cdot 10^{-4}$	$1.51 \cdot 10^{-4}$	$1.51 \cdot 10^{-4}$
	$2 \times 20\,000$	$2.45 \cdot 10^{-4}$	$3.10 \cdot 10^{-4}$	$1.51 \cdot 10^{-4}$	$1.51 \cdot 10^{-4}$
	$2 \times 40\,000$	$2.50 \cdot 10^{-4}$	$3.14 \cdot 10^{-4}$	$1.45 \cdot 10^{-4}$	$1.43 \cdot 10^{-4}$

Table 3: The FP-50 benchmark on the validation set. 300 weak classifiers were trained in random d_{sub} dimensions of the feature space, using $2 \times 10\,000$, $2 \times 20\,000$, or $2 \times 40\,000$ maximally-diverse samples from the training set.

6. CONCLUSION AND FUTURE WORK

Low probability of false alarm would be crucial for steganalyzers to be used in the real world. Yet there has been no serious study of how to achieve it. We admit that such a study is not simple: to estimate detection accuracy at a false positive rate of 10^{-6}, the number of validation samples has to be considerably greater then 1 million. Compare with contemporary practice, where the most frequently used data set (BOSSBase) has 10 000 images. We gathered 4.5 million images from Flickr, which has the realistic challenges of a real-world database, but even this would be insufficient if we wanted to examine false-positive rates below 10^{-6}.

We have advocated measuring the reliability of steganalyzers for real world applications by the false positive rate at 50% detection accuracy (FP-50 error) and proposed a family of classifiers optimizing it. Although our experiments demonstrated the advantage of the proposed classifiers over the current state of the art, we believe the questions and new directions revealed are more important still.

It appears that evaluating how a steganalyzer would perform in the real world is impossible if one possesses only a few thousand images from a few sources (e.g. BOSSBase). It should be done on millions of images, and it is unlikely that a database of such size can be acquired with a known ground truth. So it is timely to consider the literature on learning from *positive and unlabelled data*, as we move towards public databases. We note that the proposed family of classifiers is an embodiment of recently proposed methods for this problem.[17]

Our experiments also revealed that classifiers with symmetric loss functions, of which FLD is a popular example, have the undesirable property of penalizing correctly classified samples. This may be the cause of *increased* FP-50 error when the size of training data increases. Our experiments demonstrated that the FLD ensemble, which currently dominates the steganalysis literature, may benefit from indirect regularization controlled by the size of the random subspaces on which the base learners operate. This suggests a study comparing symmetric and asymmetric loss functions and regularizations. Such a study needs many sources and a large database, and to include cover source mismatch.

We conclude with a return to Section 1, and the definition of *dependable* steganalysis. Is it achievable? In this study we used a non-adaptive method of steganography that is well known to be detectable (nsF5), a very high payload (0.5 bits per nonzero coefficient), images typically much larger than the 512×512 in BOSSBase, and our best detectors managed a false positive rate of about 1 in 18 000. Even accounting for progress in steganalysis features and new classifiers optimizing the FP-50 criterion, a false positive rate of 1 in a million, or lower, seems

out of reach. If a steganalyst wants to make fewer errors, it seems the only option is *pooled steganalysis*,[22] where evidence is aggregated from many images instead of one alone.

ACKNOWLEDGMENTS

The work of T. Pevný was supported by the Grant Agency of Czech Republic under the project P103/12/P514.

We thank Yahoo! for making available the 100M creative commons image set from Flickr.

REFERENCES

[1] Ker, A. D., Bas, P., Böhme, R., Cogranne, R., Craver, S., Filler, S., Fridrich, J., and Pevný, T., "Moving steganography and steganalysis from the laboratory into the real world," in [*Proc. 1st ACM Workshop on Information Hiding and Multimedia Security*], 45–58, ACM (2013).

[2] Ker, A. D. and Pevný, T., "A mishmash of methods for mitigating the model mismatch mess," in [*Media Watermarking, Security, and Forensics 2014*], *Proc. SPIE* **9028**, 1601–1615, SPIE (2014).

[3] Cogranne, R., Zitzmann, C., Retraint, F., Nikiforov, I., Fillatre, L., and Cornu, P., "Statistical detection of LSB matching using hypothesis testing theory," in [*Proc. 14th International Conference on Information Hiding*], *LNCS* **7692**, 46–62, Springer-Verlag (2013).

[4] Ker, A. D., "Benchmarking steganalysis," in [*Multimedia Forensics and Security*], Li, C.-T., ed., 266–290, IGI Global (2009).

[5] Pevný, T. and Ker, A. D., "The challenges of rich features in universal steganalysis," in [*Media Watermarking, Security, and Forensics 2013*], *Proc. SPIE* **8665**, 0M01–0M15 (2013).

[6] Bach, F. R., Heckerman, D., and Horvitz, E., "Considering cost asymmetry in learning classifiers," *Journal of Machine Learing Research* **7**, 1713–1741 (2006).

[7] Rigollet, P. and Tong, X., "Neyman-Pearson classification, convexity and stochastic constraints," *Journal of Machine Learning Research* **12**, 2831–2855 (2011).

[8] Davenport, M., Baraniuk, R., and Scott, C., "Controlling false alarms with support vector machines," in [*Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2006*], **5**, 589–592 (2006).

[9] Scholkopf, B. and Smola, A. J., [*Learning with Kernels*], MIT Press (2002).

[10] Vapnik, V., [*Statistical learning theory*], Wiley (1998).

[11] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C., "Support vector method for novelty detection," in [*Advances in Neural Information Processing Systems 12*], 582–588, MIT Press (2000).

[12] Chang, C.-C. and Lin, C.-J., "Training nu-support vector regression: theory and algorithms," *Neural Computation* **14**(8), 1959–1978 (2002).

[13] Freund, Y. and Schapire, R. E., "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences* **55**(1), 119–139 (1997).

[14] Fridrich, J. and Kodovský, J., "Rich models for steganalysis of digital images," *IEEE Transactions on Information Forensics and Security* **7**(3), 868–882 (2012).

[15] Kodovský, J., Fridrich, J., and Holub, V., "Ensemble classifiers for steganalysis of digital media," *IEEE Transactions on Information Forensics and Security* **7**(2), 432–444 (2012).

[16] Yahoo! Webscope (2014). Yahoo! Webscope dataset YFCC-100M. `http://webscope.sandbox.yahoo.com`.

[17] Geurts, P., "Learning from positive and unlabeled examples by enforcing statistical significance," in [*Proc. 14th International Conference on Artificial Intelligence and Statistics*], *Journal of Machine Learning Research* **15**, 305–314 (2011).

[18] Franc, V. and Sonnenburg, S., "Optimized cutting plane algorithm for support vector machines," in [*Proc. 25th International Conference on Machine Learning*], 320–327, ACM (2008).

[19] Schmidt, M., "minfunc," (2013). `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`, accessed January 2014.

[20] Miche, Y., Bas, P., Lendasse, A., Jutten, C., and Simula, O., "Advantages of using feature selection techniques on steganalysis schemes," in [*Computational and Ambient Intelligence, Proc. 9th International Work-Conference on Artificial Neural Networks*], *LNCS* **4507**, 606–613, Springer-Verlag (2007).

[21] Lubenko, I. and Ker, A. D., "Steganalysis with mismatched covers: Do simple classifiers help?," in [*Proc. 13th ACM Workshop on Multimedia and Security*], 11–18, ACM (2012).

[22] Ker, A. D., "Batch steganography and pooled steganalysis," in [*Proc. 8th Information Hiding Workshop*], *LNCS* **4437**, 265–281, Springer (2006).

Appendix 11 - Grill, M. and Pevný, T. : *Learning combination of anomaly detectors for security domain* [22]

Computer Networks 107 (2016) 55–63

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Learning combination of anomaly detectors for security domain

Martin Grill[a,b,*], Tomáš Pevný[a,b]

[a] Czech Technical University in Prague, Faculty of Electrical Engineering, Czech Republic
[b] Cisco Systems, Inc., United States

ARTICLE INFO

Article history:
Received 27 November 2015
Revised 10 May 2016
Accepted 29 May 2016
Available online 1 June 2016

Keywords:
Anomaly detection
Ensemble systems
Positive unlabeled data
Accuracy at top

ABSTRACT

This paper presents a novel technique of finding a convex combination of outputs of anomaly detectors maximizing the accuracy in τ-quantile of most anomalous samples. Such an approach better reflects the needs in the security domain in which subsequent analysis of alarms is costly and can be done only on a small number of alarms. An extensive experimental evaluation and comparison to prior art on real network data using sets of anomaly detectors of two existing intrusion detection systems shows that the proposed method not only outperforms prior art, it is also more robust to noise in training data labels, which is another important feature for deployment in practice.

1. Introduction

Increasing numbers of attacks against computing infrastructure and the critical importance of the infrastructure for enterprises drives the need to deploy progressively more sophisticated defense solutions to protect network assets. An essential component of the defense are Intrusion Detection Systems (IDS) [1] searching for evidence of ongoing malicious activities (network attacks) in network traffic crossing the defense perimeter.

Many intrusion detection systems are implemented as ensembles of relatively simple, yet heterogeneous detectors [2,3], where some of them can be specialized to particular types of intrusions, whereas others can be general anomaly detectors capable of detecting previously unseen attacks at the expense of higher false alarm rates. Such a setup has multiple advantages, including faster processing of the data stream, lower complexity of the detectors, and simpler inclusion of domain knowledge into the system. The main drawback is that combining outputs of individual detectors is a non-trivial problem. Although a vast prior art on the problem exists [4–6], we believe that peculiarities of the security domain, namely a highly imbalanced ratio of non-alarm and alarm samples in the data, lack of accurately labeled datasets, and the need of extremely low false positive rates, call for a tailored solution.

The rationale behind the above specifics is that from the user perspective each raised alarm needs to be thoroughly investigated, which is expensive and can be done only for a small number of them. Hence reporting high numbers of false positives renders any intrusion detection system useless (recall that most of the samples are legitimate). Note that using a supervised method to learn the combination may bring the expense of lower generalization, but according to our experience completely unsupervised approaches rarely have false positive rate low enough to be usable in practice. Moreover, anomaly detectors and their features are usually selected based on the experience of the designer, which is a kind of proxy for labels and surely not guaranteed to be complete.

Obtaining labeled data in security domains and in network intrusion detection especially can be difficult, time consuming, and expensive. Besides, labeled data frequently contains errors in labels of different sorts, for example some alerts might be missed and labeled as legitimate samples, or even worse, all samples of alerts of certain types might be missed and labeled as legitimate.

The above concerns motivated the main goals and contributions of this paper, which are a method of finding a convex combination of outputs of a fixed set of anomaly detectors maximizing the number of true alarms in τ-fraction of most anomalous connections (samples)[1] and an experimental study of the effect of different types of label noise in the training data on the accuracy of combinations obtained by different methods to better understand their advantages and drawbacks. Conducted experiments revealed that the proposed method is not only better than the state of the art, but also more robust with respect to various kinds of noise in labels we can expect in intrusion detection domains.

If the proposed method requires labeled data, one can ask why not use them to train a classifier and sidestep the use of anomaly

* Corresponding author.
E-mail addresses: magrill@cisco.com (M. Grill), tpevny@cisco.com (T. Pevný).

[1] Since the experimental evaluation is performed with network intrusion detection systems, the terms sample and connection are used interchangeably.

http://dx.doi.org/10.1016/j.comnet.2016.05.021
1389-1286/

detectors? The most important reason to favor anomaly detectors is that network traffic discussed in this paper is very non-stationary and anomaly detectors are good at coping with this aspect, as they can constantly update their models (see [7–9] for a review).

This paper is organized as follows: The next section formally defines the problem and presents the proposed solution. Section 3 reviews related work and algorithms that we evaluate in the experimental section. The experimental Section 4 compares the proposed solution with existing methods using sets of anomaly detectors from two different network intrusion detection systems operating on two different data sources.

2. Proposed method

Prior art in combining detectors and anomaly detectors in particular is vast [4,10], nevertheless we feel that security domains requires a tailored solution because of its prominent requirement of extremely low false positive rate. We assume that the network operator observers connections (samples) from an unknown distribution $P_b = \pi P_a + (1-\pi)P_b$ with P_a/P_b being distributions of malicious/background samples and $\pi \in [0, 1]$. The network operator uses set of m anomaly detectors on samples $\mathcal{H}_m = \{h_k : \mathcal{X} \mapsto [0, 1]\}_{k=1}^m$ (w.l.o.g. it is assumed that zero means the sample is legitimate and one means the sample is malicious) and wishes to have a convex combination of anomaly detectors $\alpha = (\alpha_1, \ldots, \alpha_m)$ that would maximize the number of alarms in top τ quantile of the distribution of the combined anomaly scores. For purposes of this paper it is safe to assume that each connection (sample) is described by m-dimensional vector (an output of m anomaly detectors), which implies that distributions P_o, P_a, and P_b are defined on the m-dimensional Euclidean space.The requirements on detectors having their image in the interval [0, 1] and learning a convex combination instead of a linear one are to improve interpretability of the results as discussed in [11], but can be dropped. The same work also presents a general approach to scale the output of any anomaly detector to the interval [0, 1] reviewed in Appendix A.

With respect to the above, networks operator's goal can be written as

$$\begin{aligned}\arg\min_{\alpha\in\mathbb{R}^m} R(H_\alpha) = &\underbrace{\mathbb{E}_{x\sim P_b}\left[\mathbb{1}(\alpha^\mathrm{T}h(x) \geq q_{\alpha,\tau})\right]}_{R^{\mathrm{fp}}(H_\alpha)} \\ &+ \underbrace{\mathbb{E}_{x\sim P_a}\left[\mathbb{1}(\alpha^\mathrm{T}h(x) < q_{\alpha,\tau})\right]}_{R^{\mathrm{fn}}(H_\alpha)},\end{aligned} \tag{1}$$

subject to

$$\begin{aligned}&H_\alpha(x) = \sum_{k=1}^m \alpha_k h_k(x) = \alpha^\mathrm{T}h(x), \\ &\mathbf{1}^\top\alpha = 1, \\ &\alpha_i \geq 0,\ \forall i \in \{1, \ldots, m\},\end{aligned} \tag{2}$$

where the first term in (1) is the false alarm rate, the second term is the false negative rate, and finally $q_{\alpha,\tau}$ is a τ-quantile of observed distribution of ensemble's output $\{\alpha^\mathrm{T}h(x)|x \in P_o\}$. The minimized term (1) captures the accuracy of a particular convex combination in top τ-quantile of its distribution, which is the goal.

In theory it would be sufficient if (1) minimizes either only the false positive rate R^{fp} or only the false negative rate R^{fn}, because each of them together with constraints (2) implies minimization of the other. But including both terms increases the robustness with respect to noise on labels, since the error and its gradient are estimated from larger number of samples implying their better estimates. This is demonstrated in Appendix B, where the combination of anomaly detectors was found by optimizing either only false positive rate or only false negative rate under constraints (2). The experiments have confirmed that optimizing the proposed (1) is indeed more robust to error in labels, which are almost inevitable in security domains. In the rest of this section we show, how to find a good solution in practice using adaptation of the method of Boyd et al. [12].

First, the true loss function (1) cannot be used in practice, since the true probability distributions P_a and P_b are not known. Therefore the expectations are replaced by their empirical estimates calculated from some labeled data used for learning the weight vector α. Below the $\mathcal{S} = \mathcal{S}_a \cup \mathcal{S}_b$ denotes the set of available samples with $\mathcal{S}_b$ being the set of background (legitimate) samples and $\mathcal{S}_a$ the set of malicious samples. The empirical estimate of (1) is therefore

$$\hat{R}(H_\alpha) = \frac{1}{|\mathcal{S}_b|}\sum_{x\in\mathcal{S}_b}\mathbb{1}\left[\alpha^\mathrm{T}h(x) \geq \hat{q}_{\alpha,\tau}\right] + \frac{1}{|\mathcal{S}_a|}\sum_{x\in\mathcal{S}_a}\mathbb{1}\left[\alpha^\mathrm{T}h(x) < \hat{q}_{\alpha,\tau}\right], \tag{3}$$

where $\hat{q}_{\alpha,\tau}$ is an empirical estimate of the true quantile $q_{\alpha,\tau}$ defined as

$$\hat{q}_{\alpha,\tau} = \arg\max_{\omega} \frac{1}{|\mathcal{S}|}\sum_{x\in\mathcal{S}}\left[\mathbb{1}(\alpha^\mathrm{T}h(x) \leq \omega)\right] \leq \tau. \tag{4}$$

Since the empirical loss function (3) is neither convex nor smooth, finding the optimal solution is an NP-complete problem. A usual approach is to replace indicator function $\mathbb{1}$ with a convex surrogate, for example an exponential used in this work.[2] This substitution leads to the following optimization problem

$$\begin{aligned}\arg\min_{\alpha}\quad &\frac{1}{|\mathcal{S}_b|}\sum_{x\in\mathcal{S}_b}\exp\left(\alpha^\mathrm{T}h(x) - \hat{q}_{\alpha,\tau}\right) \\ &+ \frac{1}{|\mathcal{S}_a|}\sum_{x\in\mathcal{S}_a}\exp\left(\hat{q}_{\alpha,\tau} - \alpha^\mathrm{T}h(x)\right) \\ \text{subject to}\quad &\mathbf{1}^\top\alpha = 1, \\ &\alpha_i \geq 0,\ \forall i \in \{1, \ldots, l\}, \\ &\hat{q}_{\alpha,\tau} \text{ is a } \tau\text{-quantile defined in (4)}.\end{aligned} \tag{5}$$

where the optimized term (further denoted as $\hat{R}_{\exp}(H_{\boldsymbol{\alpha}})$) is an upper bound of the empirical loss function $\hat{R}(H_{\boldsymbol{\alpha}})$ defined in Eq. (3).

Nevertheless the last problem is still hard to solve, as it is not convex. Boyd et al. [12] showed how to find a good solution in polynomial time using series of convex problems. However his algorithm does not guarantee finding the global minimum, and the computational complexity prevents it from being used on problems with millions of samples. We therefore propose to solve (5) by a simple gradient algorithm summarized in Algorithm 1, which albeit not reaching the global minimum performs well, according to our experiments. In each step the current solution $\boldsymbol{\alpha_k}$ is updated by subtracting a small multiple of the gradient of (5), which is decreasing in each step to ensure convergence. The $\boldsymbol{\alpha_k}$ is then truncated to satisfy the constraints, and finally the estimate of the quantile $\hat{q}_{\alpha,\tau}$ is updated. The algorithm may find sub-optimal solutions but the experiments in Section 4 show that the solutions found are in most of the cases better than the ones of the state-of-the-art methods. Additionally, detailed discussion about the differences between the solution found by Boyd et al. and the one found by the proposed algorithm can be found in Appendix C.

The combination of detectors found by the above algorithm is optimized with respect to the *known* malware, by which we understand the malware whose samples are present in the training set

[2] The chosen convex surrogate does not have a significant impact on the solution and can be replaced by the reader's favorite choice, e.g. logistic, hinge, truncated square, etc.

Appendix 11 - Grill, M. and Pevný, T. : *Learning combination of anomaly detectors for security domain* [22]

Algorithm 1: The algorithm used to solve the optimization problem (5).

Data: Set of labeled samples $x_1, \ldots, x_l \in \mathcal{S}$, set of anomaly detectors $\mathcal{H}_m$ and δ_{min}.
Result: weights $\boldsymbol{\alpha} \in \mathbb{R}^m$
Start with equal weights $\boldsymbol{\alpha_1} = \mathbf{1}/m$;
repeat
  Set $q_{H_{\boldsymbol{\alpha}}}(\tau)$ to be τ-quantile of the distribution of H_{α_k};
  Update the step size as $\gamma_k = \frac{1}{\sqrt{k}}$;
  $\boldsymbol{\alpha_{k+1}} = \boldsymbol{\alpha_k} - \gamma_k \frac{\partial}{\partial \alpha} \hat{R}_{\exp}(H_{\boldsymbol{\alpha_k}})$;
until $|R(H_{\boldsymbol{\alpha_k}}) - R(H_{\boldsymbol{\alpha_{k-1}}})| < \delta_{min}$;

and most of them are correctly labeled. We believe that it is very hard to draw any conclusions about the accuracy of the algorithm on malware that has never been observed. If the unknown malware is similar to the known one (e.g. using similar components or having similar behavior), then it is likely that the above optimization will help. In order to get insight to this phenomenon on real data, the experimental section compares accuracy of several algorithms on training sets with errors on labels of different types. We believe this study will help to select the right algorithms for practice.

3. Related work

There are two classes of prior art relevant to this work. The first are unsupervised methods combining outputs of anomaly detection algorithms. The second are supervised methods maximizing accuracy or some other type of loss in top τ-quantile of outputs using labeled samples. Both are briefly reviewed below (sorted from the least to the most important).

3.1. Unsupervised methods

The first explicit use of ensembles in anomaly detection [6] employed a feature bagging method to create a diverse set of anomaly detectors. Their output was fused either by summing anomaly scores of individual anomaly detectors for a given sample, which is equivalent to taking the *mean*, or by picking the k most anomalous samples from each detector (*breadth-first* strategy). In [3] authors have compared several static combination functions, namely *mean, median, minimum, maximum*, and *mean of maximum and mean* in network intrusion detection. According to their results, mean of maximum and mean [13] was the most effective.

A necessary condition to combine heterogeneous anomaly detectors is similar range of their output. This problem is tackled in [11] by using estimated cumulative distribution functions of detectors' output. The authors show that their approach outperforms other normalization strategies including HeDES [14], maximum rank [6] or sigmoid mean [15]. The experimental part of this work uses an adaptation of [11] described in Appendix A.

A hybrid solution proposed in [14] relies on artificial samples generated uniformly at random. First, several classifiers are trained to separate the artificial samples from the provided true ones, and then weights of classifiers in the combination function are determined according to their accuracy on artificial samples.

3.2. Supervised methods

Algorithms learning the combination of classifier outputs using labeled data do not differ much from general algorithms for supervised classification. But as already mentioned, for security applications the algorithms should be designed to handle large disproportions between numbers of samples in positive and negative classes, and achieve extremely low false positive rates. Such algorithms are also needed in information retrieval (although the requirement on low positive rates is not as strict), where most of the prior art comes from.

One class of relevant algorithms maximizes accuracy of ranking in top τ-quantile, which can be viewed as prioritizing the malicious samples over the legitimate ones. These algorithms (optimizing for example *Prec@k* [16] or Normalized Discounted Cumulative Gain [17]) frequently lead to non-convex optimization problems that are difficult to solve efficiently or lead to sub-optimal solutions [18] like ours. A notable exception is SVM-*perf* [19] method optimizing a convex upper bound on the number of errors among the top k items, but still the training is computationally intensive due to a large number of constraints of the quadratic program.

Another class of relevant algorithms like RankBoost [20] maximize area under ROC curve, which is equivalent to optimizing ranking. Since only top τ-quantile matters, Infinite Push [21] and Top Push [22] concentrate on the higher-ranked negatives and try to push them down.

From the above list of supervised methods RankBoost, SVM-*perf* and Top Push are compared to our method in the experimental section.

4. Experimental evaluation

The proposed combination technique was evaluated and compared to prior art using two existing network intrusion detection systems, both implemented as an ensemble of anomaly detectors with *mean* being the default combination function. The first one, described in Section 4.2, uses NetFlow [23] records, while the second one, described in Section 4.3, uses logs from HTTP proxy servers.

To compare algorithms, we use measures from information retrieval, namely *precision* and *recall*. Assuming that malware samples have positive labels, precision is the fraction of the number of malware samples classified as positive and the total number of samples classified as positives, and recall is the fraction of malware samples classified as positives and the total number of malware samples. To highlight that the detection threshold is set to 1% of the most anomalous samples, we abbreviate both measures as *Prec@1%* and *Rec@1%*. The use of precision and recall is preferred over the popular area under the Receiver Operating Characteristic curve (AUC ROC) [24], because the latter compares the algorithms in areas which are outside the region of the interest (top 1% anomalies). Moreover, precision and recall are better suited for problems with highly imbalanced classes [25].

The use of machine learning methods in security is frequently hindered by the lack of fully labeled dataset. While samples labeled as malicious are most of the time connected to some malicious behavior, it can frequently happen that some background samples are actually malicious, but the labelling oracle (analyst) has failed to recognize them. Experiments described below aim to simulate three types of noise in labels (and of course the noise-less case denoted as Non.) to investigate their effect on the learning of the combination function. The types of considered label noise are:

- The training data contains samples of all types of malicious activities, but 50% of the samples of each activity type were not recognized as malicious by the oracle (human), and therefore they are labeled as a background. This case is denoted below as anomaly label noise (*ALN*).
- Samples of some (50%) types of malicious activities are completely missing in the training data, but they are present in the testing data. Samples of remaining types of malicious activities are present in the training set, but as in the previous case the

labeling oracle did not recognize 50% of such samples. This case is denoted as missing anomaly types (*MAT*).

- Samples of all types of malicious activities are present in the training data, but the oracle did not know 50% of types, and labeled them as background. On samples from the remaining 50% of types of malicious activities present in the training set the oracle again made a mistake and labeled them as background. This type of noise is further denoted as anomaly label noise with type mislabeling (*MLT*).

The testing set was always noiseless to allow for fair comparison and evaluate the effects.

Datasets for each intrusion detection engine are described in corresponding subsection. The available data were split so that 50% of samples were used to learn the combination of anomaly detectors and the rest for testing. This split has been repeated five times to account for the variance of the estimate.

4.1. Compared algorithms

The experimental comparison involves four unsupervised combination rules (*mean, max, rank BFS* [6], *mean rank* [6]), and four combination rules trained by supervised methods (SVM-*perf* [19], TopPush [22], RankBoost [20], and the proposed method). SVM-*perf* used L1-slack algorithm with constraint cache setting, so that 1% of positive examples was used as value of k for *Prec@k*. Regularization constant in TopPush was set to one. The proposed method (*Acc@Top*) was set to optimize the accuracy in top 1% of most anomalous samples, which means $\tau = 0.99$.

Algorithms chosen for comparison enabled comparing unsupervised methods among themselves (repeating the experiment in [3]), relevant supervised methods among themselves, and also the gain one can expect when using supervised methods even though the labels are not perfect.

4.2. Evaluation on NetFlow anomaly detection

The NetFlow anomaly detection engine [26,27] processes NetFlow [23] records exported by routers or other network traffic shaping devices. The anomaly detection engine identifies anomalous traffic using an ensemble of anomaly detection algorithms. Some of them are based on Principal component analysis [28–30], others detect abrupt changes in the behavior [31] or even use fixed rules [32]. Furthermore, there are detectors designed to detect specific type of unwanted behavior like network scans [33] or malware with domain generating algorithm [34]. In total the NetFlow anomaly detection engine uses 16 anomaly detectors. Thus the goal is to find a linear combination of these 16 anomaly detectors maximizing the accuracy in the top 1% quantile.

The evaluation used several datasets from traffic captured on the network of Czech Technical University (CTU) in Prague. The datasets and labels especially were created by three different approaches: manual labeling, infecting virtual machines, and performing real attacks against our computers within the network. In manual labeling, experienced network operator was able to successfully identify malicious activities that generated almost 10% of the total number of the connections (samples). In datasets with manually infected virtual machines[3] all their connections were labeled as malicious, whereas the rest was labeled as background. In the final dataset a network specialist run several attacks against one computer in the network. The attack vector consisted of a horizontal scan to discover open SSH ports, followed by SSH bruteforce attack to break the password, and finished by SSH login and data download simulating data theft.

[3] Neeris, FastFlux and RBot were used to infect the machines [27].

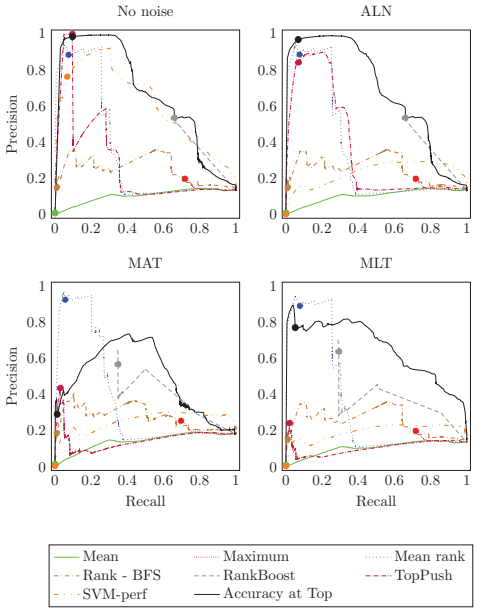


Fig. 1. PR curve comparison of algorithms with different types of label noise (described in Section 4) using the NetFlow anomaly detectors. Curves represent precision and recall values for all possible thresholds. The threshold corresponding to the 1% quantile is marked on each line with a filled circle.

Fig. 1 shows precision-recall curves for eight compared algorithms. The graphs demonstrate that the combination found by the proposed method (*Acc@Top*) most of the time dominates all other methods and fixed combination rules. Notable exceptions are cases when some types of malicious activities are completely missing in the training data (MAT) or they are incorrectly labeled (MLT). In these cases unsupervised *mean rank* combination is better on the lower recall part of the curve. This behavior suggests that different anomaly detectors detect different types of malicious activities and the supervised combination has slightly overfitted. In practical applications combining supervised and unsupervised combination rules should be used to ensure good accuracy on known malicious activities and simultaneously some generalization on unknown alerts, where the precision will be substantially smaller. Also notice that the proposed algorithm is the most robust with respect to noise from all supervised ones. SVM-*perf* is good in the noiseless case, but poor when noise of any kind is present. The TopPush is slightly more robust, but still it performed poorly with noise of MAT and MLT types, both of which are also the hardest cases. Unsupervised combination function **mean rank** performed the best among unsupervised combination functions and it was surprisingly close to supervised ones at low recall.

Precision and recall in top 1% quantile are shown in Table 1. It shows that the presented algorithm has the best or close to the best precision if we compare the supervised combination rules. As discussed above, the unsupervised *mean rank* is better in the presence of severe noise. The low recall of all algorithms except unsupervised *maximum* is caused by the high volume of malicious activities which have amounted up to 10% of the total volume of the traffic. This means that they cannot all fit into the top 1% quantile.

At the first sight RankBoost achieves the best recall of all algorithms, but a closer inspection reveals that it returns 20% of samples as those that belong in top 1%. This highly undesired

Table 1
Comparison of various combination techniques as applied to the NetFlow anomaly detectors described in Section 4.2. Each column represents precision or recall in percent for various types of noise defined at the beginning of Section 4. The best-scoring algorithm is boldfaced. Small numbers in braces below rates show the fraction of samples returned in top 1% quantile of anomaly scores. Technically this value should be equal to one, but if many samples have the same value, the algorithm returns all of them, which can results to values significantly higher than 1.0%.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Mean	0.9 (1.0%)	0.9 (1.0%)	1.4 (1.0%)	0.9 (1.0%)	0.1 (1.0%)	0.1 (1.0%)	0.1 (1.0%)	0.1 (1.0%)
Maximum	19.8 (48.8%)	19.8 (48.7%)	25.4 (49.8%)	19.8 (48.8%)	**71.8** (48.8%)	**71.8** (48.7%)	**69.6** (49.8%)	**71.8** (48.8%)
Mean rank	88.3 (1.4%)	88.5 (1.5%)	**92.3** (1.4%)	**88.8** (1.5%)	7.4 (1.4%)	7.5 (1.5%)	5.7 (1.4%)	7.8 (1.5%)
Rank BFS	15.0 (1.0%)	15.0 (1.0%)	18.5 (1.0%)	15.0 (1.0%)	1.0 (1.0%)	1.0 (1.0%)	1.0 (1.0%)	1.0 (1.0%)
RankBoost	53.4 (16.6%)	53.5 (16.6%)	56.6 (9.8%)	63.7 (7.3%)	65.9 (16.6%)	65.9 (16.6%)	34.8 (9.8%)	29.2 (7.3%)
TopPush	**99.7** (1.3%)	84.1 (1.3%)	43.5 (1.4%)	24.2 (1.7%)	9.5 (1.3%)	6.9 (1.3%)	2.9 (1.4%)	1.9 (1.7%)
SVM-*perf*	76.2 (1.2%)	0.1 (1.0%)	0.4 (1.0%)	0.3 (1.0%)	6.7 (1.2%)	0.0 (1.0%)	0.0 (1.0%)	0.0 (1.0%)
Acc@Top	98.3 (1.0%)	**96.7** (1.0%)	29.1 (1.0%)	76.9 (1.0%)	9.7 (1.0%)	6.8 (1.0%)	1.2 (1.0%)	5.2 (1.0%)

behavior is caused by assigning the same score to multiple samples. The same phenomenon can be observed in the case of Maximum aggregation function.

4.3. Evaluation on HTTP network anomaly detection

The Cisco Cognitive Threat Analytics (CTA) [35] engine analyzes HTTP proxy logs (typically produced by proxy servers located on a network perimeter) to detect infected computers within the network. Although the logs do not contain all host traffic (only HTTP(S) requests), the information is richer than the NetFlow as each entry contains the following information extracted from HTTP headers: time of the request, source IP address, destination IP address, url, MIME type, downloaded and uploaded bytes, User-Agent identifier, etc. CTA contains more than 30 different anomaly detectors detecting anomalies according to empirical estimates of (conditional) probabilities such as P(country), P(domain|host), P(User-Agent|second level domain), etc.), time series analyses (models of user activity over time, detection of sudden changes in activity, identification of periodical requests, etc.), and HTTP specific detectors (e.g., discrepancy in HTTP User-Agent field [36]).

Evaluation data were collected from networks of 30 different companies of various sizes and types with collection period ranging from six days to two weeks. The data contains more than seven billion HTTP connections, in which Cisco analysts identified 2 666 infected users with 825 different families of malware. In total the number of HTTP connections created by the malware has reached more than 129 million. Malware connections usually represent less than 2% of the network total traffic, with a notable exception of networks with hosts infected by ZeroAccess malware [37]. ZeroAccess creates many HTTP connections that can easily reach 20% of the volume of network traffic. The other most present malware families were: Cycbot, QBot, SpyEye, BitCoinMiner, and Zbot. Malware connections were identified using multiple approaches starting with an analysis of the most anomalous HTTP logs as reported by the anomaly detection engine, malware reported by the individual network administrators, matching blacklists and other public feeds or third-party software. The rest of the logs remain unlabeled, though we are almost certain there are malware connections that have been missed.

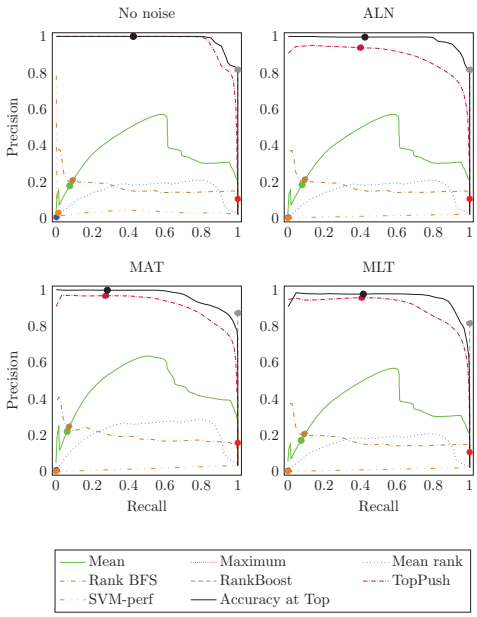


Fig. 2. PR curve comparison of various algorithms with various label noise using the HTTP anomaly detectors. Again, threshold corresponding to the 1% quantile is marked on each line with the dot.

As before, we show PR-curves of all evaluated detector combinations and types of noise in Fig. 2. We observe that the proposed *Acc@Top* method outperforms all other techniques in all cases of studied noise. Contrary to the above experiments with NetFlow analytic engine, noise does not have significant impact on supervised methods. This indicates that malicious behaviors of different types are similar in the space induced by the CTA HTTP(S) anomaly detectors. This is probably caused by the fact that all labeled malicious behaviors were in some sense connected to malware activity,

Table 2
Comparison of various combination techniques as applied to the HTTP anomaly detectors described in Section 4.3. The best-scoring algorithm is boldfaced. Small numbers in braces below rates show the fraction of samples returned in top 1% quantile of anomaly scores. Although this value should be equal to one, if many samples share the same value, the algorithm returns all of them, which can results to values significantly higher than 1.0%.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Mean	17.9 (1.0%)	18.4 (1.0%)	22.0 (1.0%)	17.1 (1.0%)	7.6 (1.0%)	7.8 (1.0%)	6.2 (1.0%)	7.3 (1.0%)
Maximum	10.6 (20.0%)	10.6 (19.9%)	15.8 (20.9%)	10.6 (19.9%)	**100.0** (20.0%)	**100.0** (19.9%)	**100.0** (20.9%)	**100.0** (19.9%)
Mean rank	0.6 (1.0%)	0.5 (1.0%)	0.9 (1.0%)	0.5 (1.0%)	0.2 (1.0%)	0.2 (1.0%)	0.2 (1.0%)	0.2 (1.0%)
Rank BFS	20.9 (1.0%)	21.3 (1.0%)	24.7 (1.0%)	20.7 (1.0%)	9.1 (1.0%)	9.3 (1.0%)	7.1 (1.0%)	9.1 (1.0%)
RankBoost	81.7 (2.6%)	81.6 (2.6%)	87.2 (3.8%)	81.5 (2.6%)	**100.0** (2.6%)	**100.0** (2.6%)	**100.0** (3.8%)	**100.0** (2.6%)
TopPush	**100.0** (1.0%)	93.8 (1.0%)	96.7 (1.0%)	95.6 (1.0%)	42.6 (1.0%)	39.9 (1.0%)	27.3 (1.0%)	40.7 (1.0%)
SVM-*perf*	2.9 (1.0%)	0.0 (1.0%)	0.0 (1.0%)	0.0 (1.0%)	1.3 (1.0%)	0.0 (1.0%)	0.0 (1.0%)	0.0 (1.0%)
Acc@Top	**100.0** (1.0%)	**99.6** (1.0%)	**99.7** (1.0%)	**97.6** (1.0%)	42.6 (1.0%)	42.4 (1.0%)	28.2 (1.0%)	41.5 (1.0%)

for which CTA engine is designed. The curve of the *Acc@Top* suggest that even if less or more samples than 1% are requested by the operator, the precision will remain high in all scenarios. In contrast, the curve of the RankBoost method starts at high recall with lower precision suggesting that almost all malicious samples and around 20% legitimate were scored with the same maximal value. Also notice that the *mean rank* unsupervised combination function, dominating in the previous section, was in this experimental scenario superseded by simple *mean*.

Precision and recall at the top 1% are shown in Table 2, and as in the previous section RankBoost and *maximum* achieve the best recall. The causes are the same. RankBoost and maximum have returned 2.6% and 20% of samples, respectively, which is far away from the required 1%. This is again caused by assigning the same value to many samples. Contrary, the proposed *Acc@Top* meets the 1% requirement with really high precision. Its seemingly low recall is partially caused by the fraction of malicious samples being 2% of the total number of all samples. This means that the best achievable recall while meeting the requirements on returning 1% of the total number of sample is 50%.

5. Conclusion

This paper has proposed a new algorithm for finding a convex combination of anomaly detectors maximizing accuracy at τ-quantile of returned samples, which is a scenario frequently appearing in the security field. The algorithm assumes labeled data, which are difficult to obtain and rarely perfect in security domains. Therefore, an emphasis was put on the experimental study, involving two different types of intrusion detection systems, eight types of combination functions, 34 different network captures containing more than 20 million of samples of behavior of different algorithms under different types of noise.

The experimental results show that the proposed method is more accurate than prior art in finding a good combination of detectors with high accuracy in returned samples. The results also show that supervised methods can easily overfit if some type of malicious behavior is completely missing in the training data or is incorrectly labeled (mistake of labeling oracle). The severity of the overfitting depends on how much different types of malicious behavior are similar to each other. The comparison of unsupervised combination functions did not have a clear winner, since in one experimental setting *mean rank* was the best while in the second one it was *mean*.

The presented experimental results show that future efforts should be directed toward finding methods combining good properties of both supervised and unsupervised combination functions.

Appendix A. Scaling outputs of anomaly detectors

Generally, individual anomaly detectors need not generate anomaly scores of the same scale. This causes problems during the combination process, since one or more detectors could be inadvertently favored. Therefore, the anomaly scores of the individual detectors are normalized using the *gaussian scaling* proposed by Kriegel et al. [11]:

$$\tilde{h}(x) = \max\left\{0, erf\left(\frac{h(x) - \mu_h}{\sigma_h\sqrt{2}}\right)\right\}, \tag{A.1}$$

where the $\tilde{h}(x)$ is the normalization of the anomaly score $h(x)$ assigned to the observation $x \in \mathcal{X}$ by anomaly detector h. The used Gaussian Error Function $erf()$ is monotone and thus ranking stable. The μ_h and σ_h are the mean and the standard deviation of the anomaly scores returned by the anomaly detector h. This transforms the anomaly scores of individual anomaly detectors into probability estimates, where the probability of zero represent normal observation, aligned with the predictive model, whereas one indicates highly anomalous observation. These are therefore directly comparable and can be aggregated using a number of combination techniques [11].

Appendix B. Optimizing only false positives or false negatives

To demonstrate the advantage of minimizing both false positive and false negative rates in the objective function $\hat{R}_{\exp}(H_{\boldsymbol{\alpha}})$ (5), we have evaluated two additional variants of the objective function with only the false negative part $\hat{R}^{fn}_{\exp}$ (*Acc@Top-FN*) and false positive part $\hat{R}^{fp}_{\exp}$ (*Acc@Top-FP*) using both NetFlow (Table B.3) and CTA (Table B.4) anomaly detection systems. As can be seen in Table B.3, using only one part of the criterion results in substantially decreased efficacy in the NetFlow scenario. Additionally, the false

Table B3
Comparison of three variants of the proposed criterion, each used to train an ensemble for the NetFlow anomaly detection system. Small numbers in braces below rates show the fraction of samples returned in top 1% quantile of anomaly scores. Although this value should be equal to one, if many samples share the same value, the algorithm returns all of them, which can results to values significantly higher than 1.0%.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Acc@Top	**98.3** (1.0%)	**96.7** (1.0%)	**29.1** (1.0%)	**76.9** (1.0%)	9.7 (1.0%)	6.8 (1.0%)	1.2 (1.0%)	5.2 (1.0%)
Acc@Top-FP	13.4 (100%)	13.4 (100%)	18.1 (100%)	13.4 (100%)	**100** (100%)	**100** (100%)	**100** (100%)	**100** (100%)
Acc@Top-FN	0.1 (1.9%)	0.1 (1.9%)	15.7 (1.4%)	21.7 (2.4%)	0.0 (1.9%)	0.0 (1.9%)	0.7 (1.4%)	1.6 (2.4%)

Table B4
Similarly to Table B.3, the table presents a comparison of three variants of the proposed criterion used on the CTA anomaly detection system.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Acc@Top	**100** (1.0%)	**99.6** (1.0%)	**99.7** (1.0%)	**97.6** (1.0%)	**42.6** (1.0%)	**42.4** (1.0%)	28.2 (1.0%)	**41.5** (1.0%)
Acc@Top-FP	98.1 (1.0%)	1.2 (11.7%)	52.7 (7.8%)	0.6 (26.2%)	41.7 (1.0%)	11.2 (11.7%)	19.3 (7.8%)	25.4 (26.2%)
Acc@Top-FN	87.2 (1.0%)	87.5 (1.0%)	88.5 (1.2%)	86.0 (1.1%)	37.1 (1.0%)	37.2 (1.0%)	**29.2** (1.2%)	38.6 (1.1%)

positive variant (*Acc@Top-FP*) results in all the samples having the same, zero value anomaly score. The results on the CTA anomaly detection engine (Table B.4) are slightly better, but still the proposed *Acc@Top* outperforms both other variants in all label noise scenarios.

Appendix C. Comparison with state-of-the-art

Although the algorithm of Boyd et al. [12] is currently considered to be the state of the art for optimizing the accuracy at top, it is not guaranteed to find the global minimum. Its biggest advantage is in finding the solution by solving series of convex sub-problems; therefore it is bound to always find the same solution. But this limits the algorithm to be applicable only to small problems, since its complexity grows as of $O(n^4)$, where n is the number of training samples, and the solver of the convex sub-problems has the complexity of $O(n^3)$. In contrast, the proposed algorithm is essentially a stochastic descent algorithm, which has been proved to work well on problems with large number of samples.

In order to investigate how the solutions of both algorithms differ, we have created an artificial problem, which we think well models the application scenario of finding a convex combination of outputs of anomaly detectors. The problem was set to find the optimal combination of two anomaly detectors to optimize accuracy at the 20% quantile. The anomaly scores of the anomaly detectors for both, training and testing data, were generated using a set of normal distributions.[4]

The decision boundaries corresponding to solutions of both algorithms (shown in Fig. C.4) are very different, since Boyd et al.'s algorithm uses the output of one anomaly detector whereas ours uses both detectors. Corresponding PR-curves, shown in Fig. C.3, further reveal that on the training set (solid lines) Boyd et al.'s algorithm (*Boyd*) is better at the point of the interest, but the proposed method (*Acc@Top*) is better on a wider range of operating points, which suggests that it would behave better on unknown data. This is experimentally confirmed by PR-curves on the testing data (dashed lines), where the proposed algorithm dominates. Although, theoretically, this can be due to overfitting which can be solved by training on a larger training set, if available, this solution would be difficult in practice due to the prohibitive complexity of Boyd et al.'s algorithm.

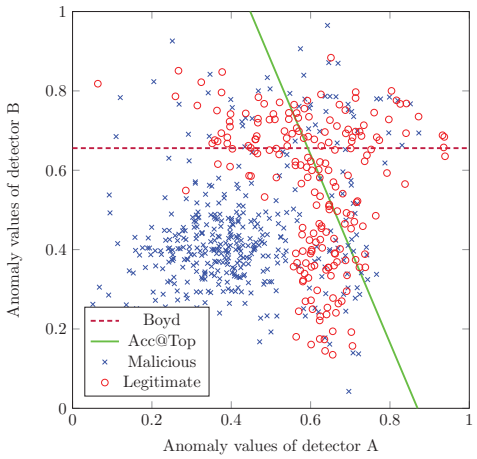


Fig. C3. Visualization of the artificial problem containing two anomaly detectors. The position of both anomalous and malicious samples is given by the anomaly score of the both A and B anomaly detectors. The solid and dashed lines represent decision boundaries of *Acc@Top* and *Boyd* algorithms respectively.

[4] The legitimate samples were drawn from uniform distribution (100 samples) to simulate noise of the anomaly detectors and $\mathcal{N}([0.4, 0.4], [0.01, 0.01]\mathbb{I}_2)$ (300 samples), where the $\mathbb{I}_2$ denotes the 2×2 identity matrix. The malicious samples were drawn from $\mathcal{N}([0.6, 0.7], [0.03, 0.003]\mathbb{I}_2)$ (100 samples) and $\mathcal{N}([0.7, 0.4], [0.003, 0.03]\mathbb{I}_2)$ (100 samples).

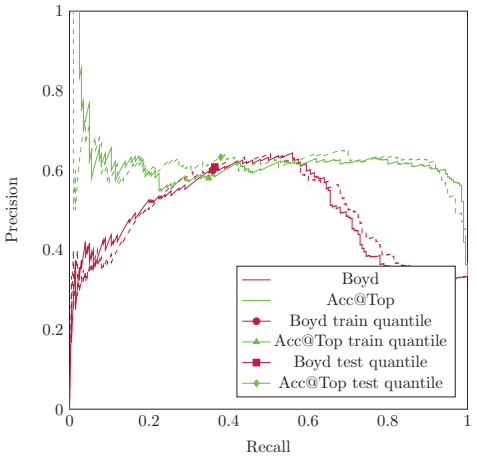


Fig. C4. PR-curves of the *Acc@Top* and the optimum found by the *Boyd* algorithm using the training (solid line) and testing (dashed line) data generated from the artificial problem. Threshold corresponding to the 20% quantile is marked on each curve.

References

[1] K. Scarfone, P. Mell, Guide to intrusion detection and prevention systems (IDPS), Technical Report 800-94, NIST, US Department of Commerce, 2007.
[2] G. Giacinto, F. Roli, Intrusion detection in computer networks by multiple classifier systems, in: Proceedings of the 16th International Conference on Pattern Recognition (ICPR), Volume 2, IEEE Press, 2002, pp. 390–393.
[3] S. Shanbhag, T. Wolf, Accurate anomaly detection through parallelism, Network, IEEE 23 (1) (2009) 22–28.
[4] L. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, John Wiley, 2004.
[5] L.I. Kuncheva, A theoretical study on six classifier fusion strategies, IEEE Trans. Pattern Anal. Mach. Intell. 24 (2002) 281–286.
[6] A. Lazarevic, V. Kumar, Feature bagging for outlier detection, in: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data mining, ACM, 2005, pp. 157–166.
[7] T. Pevný, Loda: lightweight on-line detector of anomalies, Mach Learn. (2015) 1–30.
[8] A. Coluccia, A. D'Alconzo, F. Ricciato, Distribution-based anomaly detection via generalized likelihood ratio test: a general maximum entropy approach, Computer Networks 57 (17) (2013) 3446–3462.
[9] C. Callegari, A. Coluccia, A. D'Alconzo, W. Ellens, S. Giordano, M. Mandjes, M. Pagano, T. Pepe, F. Ricciato, P. Zuraniewski, in: Datatraffic Monitoring and Analysis, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 148–183.
[10] C.C. Aggarwal, Outlier ensembles: position paper, ACM SIGKDD Explor. Newslett. 14 (2) (2013) 49–58.
[11] H.-P. Kriegel, P. Kröger, E. Schubert, A. Zimek, Interpreting and unifying outlier scores, in: 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ, 42, SIAM, 2011.
[12] S. Boyd, C. Cortes, M. Mohri, A. Radovanovic, Accuracy at the top, in: Advances in Neural Information Processing Systems, 2012, pp. 953–961.
[13] P.F. Evangelista, M.J. Embrechts, B.K. Szymanski, Data fusion for outlier detection through pseudo-roc curves and rank distributions, in: Neural Networks, 2006. IJCNN'06. International Joint Conference on, IEEE, 2006, pp. 2166–2173.
[14] H.V. Nguyen, H.H. Ang, V. Gopalkrishnan, Mining outliers with ensemble of heterogeneous detectors on random subspaces, in: Database Systems for Advanced Applications, Springer, 2010, pp. 368–383.
[15] J. Gao, P.-N. Tan, Converting output scores from outlier detection algorithms into probability estimates, in: Data Mining, 2006. ICDM'06. Sixth International Conference on, IEEE, 2006, pp. 212–221.
[16] B. McFee, G.R. Lanckriet, Metric learning to rank, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 775–782.
[17] H. Valizadegan, R. Jin, R. Zhang, J. Mao, Learning to rank by optimizing ndcg measure, in: Advances in Neural Information Processing Systems, 2009, pp. 1883–1891.
[18] Q. Le, A. Smola, Direct optimization of ranking measures, arXiv preprint arXiv:0704.3359 (2007).
[19] T. Joachims, A support vector method for multivariate performance measures, in: Proceedings of the 22nd International Conference on Machine Learning, ACM, 2005, pp. 377–384.
[20] Y. Freund, R. Iyer, R.E. Schapire, Y. Singer, An efficient boosting algorithm for combining preferences, J. Mach. Learning Res. 4 (2003) 933–969.
[21] A. Rakotomamonjy, Sparse support vector infinite push, arXiv preprint arXiv:1206.6432 (2012).
[22] N. Li, R. Jin, Z.-H. Zhou, Top rank optimization in linear time, in: Advances in Neural Information Processing Systems, 2014, pp. 1502–1510.
[23] E.B. Claise, in: Cisco Systems NetFlow Services export, Version 9, 2004.
[24] T. Fawcett, An introduction to roc analysis, Pattern Recognit. Lett. 27 (8) (2006) 861–874.
[25] J. Davis, M. Goadrich, The relationship between precision-recall and roc curves, in: Proceedings of the 23rd International Conference on Machine Learning, ACM, 2006, pp. 233–240.
[26] M. Rehák, M. Pěchouček, M. Grill, J. Stiborek, K. Bartoš, P. Čeleda, Adaptive multiagent system for network traffic monitoring, IEEE Intell. Syst. (3) (2009) 16–25.
[27] S. Garcia, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, Comput. Secur. 45 (2014) 100–123.
[28] A. Lakhina, M. Crovella, C. Diot, Diagnosing network-wide traffic anomalies, in: ACM SIGCOMM Computer Communication Review, 34, ACM, 2004, pp. 219–230.
[29] A. Lakhina, M. Crovella, C. Diot, Mining anomalies using traffic feature distributions, in: ACM SIGCOMM Computer Communication Review, 35, ACM, 2005, pp. 217–228.
[30] T. Pevný, M. Rehák, M. Grill, Detecting anomalous network hosts by means of pca, in: Information Forensics and Security (WIFS), 2012 IEEE International Workshop on, 2012, pp. 103–108.
[31] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, P. Dokas, Minds - minnesota intrusion detection system, in: Next Generation Data Mining, MIT Press, 2004.
[32] K. Xu, Z.-L. Zhang, S. Bhattacharyya, Profiling internet backbone traffic: behavior models and applications, in: ACM SIGCOMM Computer Communication Review, 35, ACM, 2005, pp. 169–180.
[33] A. Sridharan, T. Ye, S. Bhattacharyya, Connectionless port scan detection on the backbone, in: Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International, IEEE, 2006, pp. 10–pp.
[34] M. Grill, I. Nikolaev, V. Valeros, M. Rehák, Detecting dga malware using netflow, in: Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on, 2015, pp. 1304–1309.
[35] Cisco Systems, in: CTA cisco cognitive threat analytics on cisco cloud web security, 2014–2015. (http://www.cisco.com/c/en/us/solutions/enterprise-networks/cognitive-threat-analytics).
[36] M. Grill, M. Rehák, Malware detection using http user-agent discrepancy identification, in: Information Forensics and Security (WIFS), 2014 IEEE International Workshop on, 2014, pp. 221–226.
[37] J. Wyke, The zeroaccess botnet–mining and fraud for massive financial gain, 2012.

Appendix 11 - Grill, M. and Pevný, T. : *Learning combination of anomaly detectors for security domain* [22]

Martin Grill is a researcher at Cognitive Threat Analytics at Cisco. His research focuses mainly on network anomaly detection, ensemble systems and network event classification. Prior to Cisco, Martin was a researcher at Czech Technical University in Prague and CESNET, z.s.p.o developing a NetFlow based network behaviour anomaly detection system. Martin Grill holds master degree in Software development at the Faculty of Nuclear Sciences and Physical Engineering of the Czech Technical University in Prague. Currently he is pursuing his PhD at the Department of Computer Science of Czech Technical University in Prague.

Tomáš Pevný holds the position of researcher at Czech Technical University of Prague. He received his PhD in Computer Sciences from State University of New York in Binghamton in Computer Science at 2008 and MS in Computer Sciences from School of Nuclear Sciences and Physical Engineering at Czech Technical University in Prague in 2003. In 2008–2009, he did his post-doc at Gipsa-lab in Grenoble, France. His research interests are applications of non-parametric statistics (machine learning, data modeling) with focuses on steganography, steganalysis, and intrusion detection.

Anomaly Explanation with Random Forests

Martin Kopp, Tomáš Pevný Martin Holeňa

Abstract—Anomaly detection has become an important topic in many different domains with many different solutions proposed until now. Despite that, there are only a few anomaly detection methods trying to justify or explain how the sample differs from the majority. Since in many domains knowing why a sample is considered anomalous is absolutely critical, this work partially fills this gap. The proposed solution uses a specific type of random forests to extract rules explaining the difference, which are then filtered and presented to the user as a set of association rules or in a disjunctive normal form. A comparison of the proposed solution to the state of the art algorithms on 36 real world datasets shows a superior performance. Moreover, the explanation is more complete than that of the prior art, which identifies only features in which the given sample is different.

Index Terms—Anomaly Detection, Anomaly Explanation, Association Rules, Feature Selection, Random Forests

1 Introduction

Anomaly detection algorithms identify samples deviating from the majority of data so much that they raise a suspicion they have been generated by a different process. Because anomalies are, by definition, rare and they can substantially differ from each other, the problem poses very different issues and challenges than standard supervised classification. Citing [1], the most important differences are following:

- Determining a region of all possible normal samples is very difficult.
- The boundary between normal and anomalous samples is often not precise and highly depends on the considered application. For example, in the medical domain, a small deviation from the average (e.g., fluctuations in body temperature) might be an anomaly, while a similar deviation in the stock market domain (e.g., fluctuations of the stock value) might be normal.
- The data generating distribution may be non-stationary, which means that the currently valid notion of normal samples might not be valid in the future.
- Anomaly detection is usually applied to domains or problems where labels are scarce or difficult to obtain.

Despite the above challenges, anomaly detection algorithms were successfully applied in many domains, for example: network security [2], bioinformatics [3], astronomy [4], space exploration [5]. With the increasing volume of mostly unlabelled data presently generated, it receives more and more attention, as it helps to identify interesting samples without human intervention.

- *M. Kopp is with the Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00 Prague. E-mail: koppmart@fit.cvut.cz*
- *T. Pevný is with the Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27 Prague, Czech Republic. E-mail: pevnak@gmail.com*
- *M. Holeňa is with theInstitute of Computer Science, Czech Academy of Sciences, Pod Vodárenskou věží 271/2, 182 07 Prague, Czech Republic E-mail: martin@cs.cas.cz*

Due to a wide interest and the fact that each application domain has its own definition of anomaly and application constraints, a plethora of diverse algorithms have already been proposed. An interested reader is encouraged to read a recent book [6] and a recent comparison [7], where is demonstrated that there is not a single best algorithm for all scenarios. Furthermore, identifying an anomaly is only a half of the problem, and the second, equally important, is the interpretation of found anomalies, because humans have more trust in obtained results if they understand why it was returned. In the high dimensional and complicated application domains like network security or bioinformatics, where hundreds or even thousands of features are common, a good interpretation is even more important. Every piece of additional knowledge about the anomaly provides invaluable help to users evaluating the suspicious samples. Furthermore, it helps to gain insights into why an anomalous sample is different from the rest of data, which can reveal something about its properties. Despite the extensive prior art in anomaly detection, very few works even mention the explanation of detected anomalies. This is surprising because in practice, any detection is usually followed by a deeper investigation. Furthermore, the pioneering works in the non-parametric anomaly detection by [8], [9] explicitly consider anomaly explanation.

This paper partially fills this gap by proposing an algorithm called *Explainer*, which explains why a sample identified as an anomaly by some detector is different from the majority. The explanation is presented as a set of decision rules which are commonly considered easily understandable by humans, e.g. [10]. One of the main *Explainer*'s advantages is that the anomaly detection algorithm used to find anomalies is treated as a black-box, which means that the *Explainer* can be employed as an additional step for a vast majority of the state of the art algorithms.

The *Explainer* is based on an ensemble of specifically trained decision trees (random forest), which in this particular case are very small. Since the construction of those specific trees is extremely fast, the *Explainer* is lightweight and therefore, it can be used effectively with minor memory requirements on large databases and data-streams to pro-

vide explanations nearly in the real time.

The experimental section compares the *Explainer* to the relevant prior art on 36 benchmark anomaly detection problems obtained using the approach proposed in [11]. The comparison was done for explanations of anomalies provided by three different anomaly detection algorithms and also for ground truth anomalies. According to the Friedman test with Nemenyi post-hoc analysis [12], the *Explainer* significantly outperforms all the prior art. Occasional cases where it is inferior are discussed, so that its shortcomings are known and a practitioner can better choose the right algorithm.

The rest of this paper is organized as follows. Section 2 reviews related work with a short overview of anomaly detection algorithms followed by a deeper presentation of the state of the art in the explanation of anomalies. The *Explainer* is described in Section 3. Section 4 presents results of its comparison to state of the art algorithms on 36 datasets from different domains. The same section also shows the robustness of the *Explainer* with respect to the setting of its parameters and recommends an overall good choice. Section 5 summarizes the advantages and limitations of the proposed approach and suggests possible extensions and future work.

2 Related work

This section first briefly reviews the anomaly detection domain and then surveys anomaly explanation techniques.

2.1 Anomaly detectors

Most unsupervised anomaly detection methods can be classified as being either model-centric or data-centric.

Model-centric methods learn some model from data, and then during detection, they measure how a given sample deviates from that model. Their training phase is usually computationally expensive while the detection is rather cheap. The most basic models represent data by parameters of some statistical distribution. The first work that has used the term *anomaly detection*, [13], belongs to this class. For example, a popular class of methods based on principal component analysis [14] or deviations from linear regression [15] assumes multivariate normal distribution although they do not state it explicitly. Methods, such as one-class support vector machines [16] or kernel fisher linear discriminant [17] tries to overcome the discrepancy between real data and rigidity of parametric distributions, but at the expense of introducing hyper-parameters that needs to be tuned and to which the methods are sensitive. We note that for one-class support vector machines there is a proof of their asymptotic convergence to optimum [18].

A rather special case of model-centric methods are those that employ clustering. Their training phase (model) consists of grouping samples into clusters. In the detection phase, samples far from all clusters are considered to be anomalous, [19] . An interesting variation is the subspace clustering method proposed by [20], where clusters are formed in sub-spaces. Sub-spaces in which anomalies are identified can be viewed as a very basic explanation, although this is not explicitly stated.

Data-centric methods use data instead of a model and measure the difference of a sample from the majority, assuming some similarity or distance function. Contrary to the previous, their training phase is usually very cheap consisting only from storing all samples, while their classification phase tends to be very expensive, typically due to a nearest-neighbour type of search. This category includes one of the first methods for a non-parametric anomaly detection, [8], [9], defining and identifying anomalous samples as those that are far from the rest. A plethora of methods were proposed differing in the used distance [21] and also in the way of selecting the samples with respect to which that distance is measured. The main drawback of distance based algorithms is related to the curse of dimensionally, as in high dimensions almost all points are equidistant. [22] proposed a method of anomaly detection using reverse nearest neighbours, with appealing performance especially for high dimensional data. Another variation of data-centric methods are those that estimate (local) probability density and define anomalies as points having very low probability. The popular Local Outlier Factor (LOF) by [23] and Local Correlation Integral (LOCI) by [24] are prime examples of this type of anomaly detection methods.

2.2 Explainers

The previous section suggests that there is a rich prior art on anomaly detection, yet few works address the problem of our interest — the explanation of detected anomalies.

The first work considering anomaly explanation was [8], defining an explanation as: "provision of a description or an explanation of why an identified anomalous sample is exceptional". The method first detected all distance based anomalies in the whole attribute space. Then, it has iterated over all possible subspaces and identified the smallest subspace in which was the considered anomaly still detected. This smallest subspace was considered its explanation. In the terminology used, here it has identified those features in which the sample deviates. Also, the method verified if there are other samples that are anomalies only within certain subspaces, and according to this property anomalies were classified into *weak* and *strong*, see [9] for further details. The main drawback of their method is its computational complexity because all possible subsets of features to must be screened for anomalies, which renders the methods unusable for high-dimensional data.

[25] proposed an algorithm simultaneously identifying and explaining anomalies. The method uses Fisher Linear Discriminant (FLD) to separate each anomalous candidate from samples in its neighbourhood. The anomaly score is derived from the value of the objective function optimized in FLD. Coordinates with the largest magnitude found by FLD are then used to determine the most important features explaining anomality of the respective candidate. The detection process is computationally very expensive, therefore, the authors have proposed several heuristics to make it faster. Nevertheless, the complexity prevents from using the method on larger datasets. Another drawback is that the explanation is entangled with the anomaly detection.

In a more recent work [26] have extended the previous approach by replacing FLD with a variant of large-margin

nearest neighbour algorithm by [27] to find a projection matrix $\mathbf{W}$ onto a low-dimensional space which approximately preserves the local neighbourhood of the sample in question. The anomaly score is calculated in this low-dimensional space (different for each sample), and from the projection matrix $\mathbf{W}$, features important for separation of the sample from its neighbourhood are extracted and returned as an explanation. As in the previous approach, the explanation is tightly connected to the anomaly detection. It is also quite computationally intensive because to find projection matrix $\mathbf{W}$ in each point requires singular value decomposition, which makes the method impractical for large-scale data.

[28] propose to convert the problem of anomaly detection to binary classification, which is to some extent similar to the anomaly detector of [29]. For each sample x, their method creates artificial samples in its vicinity, trying to train a classifier to separate the artificial samples from the rest. If x is an anomaly, then artificial samples should be easily separable from non-artificial ones, which would mean the classifier will have a low error. Conversely, if x is a true inlier, then the classifier will have a high error. [28] used the linear Support Vector Machine (SVM) classifier to estimate the classification error. Features contributing to the separation of samples correspond to the largest-magnitude coordinates of the normal vector of the separating hyperplane and are returned as an explanation (similarly to [25]). Drawbacks of the method include a high computational complexity and a difficulty to correctly generate artificial samples.

The last method explaining anomalies we are aware of, was proposed by [30]. It is designed for categorical variables, but with a proper quantization, it can be used for real variables as well. The anomaly score is derived from the frequency of histogram bins, from which the method also extracts context and explanation of the anomaly. An example of the context would be "The sample x is an anomaly because from all samples having $attribute_1 = value1$ it is only one having $attribute_6 = value2$".

Compared to the prior art, the *Explainer* is more general than [25], [26] and [9] because it can explain the output of an arbitrary anomaly detector. It is compatible with both categorical and numerical features or their mix. Although [30] can also use both type of features, it is effective only on categorical data or small sets of integers due to its computational demands. The approach of [28] can explain the output of an arbitrary anomaly detector, but its performance depends on used feature selection method and the proposed anomaly score needs balanced sets of normal and artificially generated anomalous samples. The *Explainer* follows a completely different approach than the prior art and is designed to handle imbalanced data.

All recent prior art stops the explanation after identifying the set of features in which the anomaly is visible, which means that the algorithms are essentially used as a feature selectors for extremely imbalanced datasets. Contrary to this *Explainer* returns more complex explanation presented as a set of association rules or in the disjunctive normal form.

Last but not least, *Explainer* is fast and lightweight, therefore well suited for nearly real time processing of data and for data-streams.

3 The *Explainer* Algorithm

The goal of the *Explainer* is to explain, why a point $x^a \in \mathcal{X}$ is an anomaly with respect to the rest of points in $\mathcal{X}$. *Explainer* does not impose constraints on the anomaly detection algorithm that has identified x^a, but assumes that the anomalous point x^a deviates from the rest in a subset of features (corresponding to dimensions of $\mathcal{X}$). The space $\mathcal{X}$ where the data lies is general in the sense that its dimensions can be real, categorical or mixed variables, or any other type of variables where an ordering operator, allowing to use rules of type $x_i > \theta$, can be defined[1].

The optimal explanation depends on user's prior knowledge and experience. Despite this subjectivity, decision trees are, according to the [10], considered understandable for humans. In fact, understandability of not only tress but rules in general, i.e., Boolean implications, is the reason why they have been for many decades used as the most common means of the knowledge representation.

To explain the anomality of x^a, the *Explainer* trains a set of decision trees (a random forest) to separate x^a from $\mathcal{X}$. From this random forest, it extracts individual rules, which are then assembled into a set of the association rules or into the disjunctive normal form. Since each tree is trained to separate a single point x^a from $\mathcal{X}$, its training set is extremely imbalanced and its height is very small. The idea of using decision trees is further supported by the anomaly detection method called Isolation Forest by [31], which according to the recent extensive comparison in [7] is one of the best anomaly detector. The key steps of the *Explainer* are summarized in Algorithm 1.

Algorithm 1 Summary of the *Explainer* algorithm for a single anomaly x^a.

Input:
$data$ - input dataset; x^a anomalous sample; $size$ training set size; n_T number of trees to be trained.

Output:
$rules$ - rules explaining x^a

$Forest \leftarrow \emptyset$
for $i \leftarrow 1 \dots n_T$ **do**
  $\mathcal{T} \leftarrow \text{createTrainingSet}(data, size, x^a)$
  $t \leftarrow \text{trainTree}(\mathcal{T})$
  $Forest \leftarrow Forest + t$
end for
$rules \leftarrow \text{extractRules}(Forest)$

In the rest of this section, the individual parts of the *Explainer* are discussed in detail, specifically, training of individual trees, construction of datasets, and extraction of the rules forming the *minimal* and *maximal* explanation. The section concludes with examples of explained anomalies.

3.1 Training a single tree

This section assumes that the reader has a basic knowledge of decision trees and random forests, a suitable references being, e.g., [32] or [33].

There exist many variants of the basic algorithm to train a single tree, differing mainly in: (i) criterion to select a node

1. The rule should be read as i^{th} feature of x is greater than θ.

to be split; (ii) set of hypotheses (decision rules) assigned to the inner nodes of the tree; (iii) criterion to select the hypothesis in the selected inner node; and (iv) training termination criterion. Since the *Explainer* trains decision trees with the goal of identifying rules explaining how x^a differs from $\mathcal{X}$, its training set contains only one positive sample x^a, which makes some of the above choices straightforward while others very different from what is typical in supervised classification.

Selection of the splitting node: Since the training set contains only one positive sample, there is only one leaf containing this positive sample in every iteration of the training algorithm. Therefore, this leaf is selected to be split.

Set of hypotheses: In line with the goal of maximum understandability, the set of hypotheses contains only decision rules of the type $x_j > \theta$ (or $x_j < \theta$), which can be read as the j^{th} feature is greater (or smaller) than θ.

Criterion to select the hypothesis: Let $\mathcal{S}$ denotes the set of samples reaching a leaf containing the anomaly and $\mathcal{H}$ denotes set of candidate hypothesis to split the leaf. Classifying samples from $\mathcal{S}$ according to hypothesis h creates two subsets $\mathcal{S}^a(h)$ and $\mathcal{S}^n(h)$, where $\mathcal{S}^a$ is assumed to contain the positive sample.

The *Explainer* chooses the hypothesis h splitting a particular leaf such that the size of the set $|\mathcal{S}^a(h)|$ is as small as possible, i.e.

$$\arg\min_{h\in\mathcal{H}} |\mathcal{S}^a(h)|, \tag{1}$$

which can be effectively calculated. Moreover due to the above described way of the training it is equivalent to the popular information gain [32] and Gini impurity [34], as it is proved in the Appendix.

If two or more hypotheses $\{h_1, \ldots, h_l\} \in \mathcal{H}$ split the leaf so that resulting $\mathcal{S}^a(\cdot)$ are of the same size, the training algorithm resolves this tie by selecting the hypothesis maximizing the robust estimate of the margin defined as

$$\arg\min_{h\in\{h_1,\ldots,h_l\}} Q_{0.05}\left(\{|x_h - x^a_h| | x \in \mathcal{S}^n(h)\}\right), \tag{2}$$

where Q denotes the empirical quantile, corresponding to the subscript, of the set in its argument, where x_h denotes the feature used in hypothesis h. The rationale behind this robust estimate of the margin is that the set of normal samples may contain anomalous samples missed by the anomaly detection algorithm. The choice of 5% quantile is justified by the assumption that anomalies are rare and there should be less than 5% of them. Note that for this algorithm to work properly, samples should be normalised to zero mean and unit variance.

Termination criterion: The training of a tree finishes when the leaf containing the anomalous sample is of size one.

3.2 Training set selection

To achieve robustness against noise in the training set, the *Explainer* trains set of trees on randomly selected subsets $\mathcal{T}$ of the training data. Since the training set always contains the same single positive sample, x^a, only the normal (non-anomalous) samples from $\mathcal{X}$ are sampled. The construction of the training set $\mathcal{T}$ influences the properties of the *Explainer*, which can be configured to provide either *minimal* explanation returning minimal set of rules for features needed to identify the anomaly, or *maximal* explanation returning all rules for features in which anomaly can be detected.

Minimal explanation To extract minimal set of rules explaining the anomaly, the training sets for individual trees are sampled independently of each other. A traditional approach is bootstrapping used in [32], where each training set contains k samples selected from $\mathcal{X}$ randomly with replacement. Due to its low computational complexity of $O(|\mathcal{X}|)$ and overall good results, it is used here as well. We also tried a local approach, where the normal samples were chosen only from the neighbourhood of the anomaly x^a, but the time complexity grown significantly while the explanation did not improve.

Training set for maximal explanation

Definition 3.1. *k*-nn distance of a sample x to the set $\mathcal{S}$ is the average distance of x to its k nearest neighbours in $\mathcal{S}$, i.e.

$$\text{knnDist}(x, \mathcal{S}) = \frac{1}{k} \sum_{x_i \in \text{NN}_k(x,\mathcal{S})} d(x, x_i),$$

where $d(x, x_i)$ denotes distance of x to x_i and $\text{NN}_k(x, \mathcal{S})$ denotes the set of k nearest neighbors of x in $\mathcal{S}$.

Definition 3.2. *k*-nn distance of the set $\mathcal{S}_1$ to the set $\mathcal{S}_2$ is the set of average distances of all samples in $\mathcal{S}_1$ to their k nearest neighbours in $\mathcal{S}_2$, i.e.

$$\text{knnDist}(\mathcal{S}_1, \mathcal{S}_2) = \{\text{knnDist}(x, \mathcal{S}_2) | x \in \mathcal{S}_1\}$$

Note that, in general $\text{knnDist}(\mathcal{S}_1, \mathcal{S}_2) \neq \text{knnDist}(\mathcal{S}_2, \mathcal{S}_1)$.

To identify all features in which x^a deviates from the majority, the *Explainer* starts by creating two random training sets $\mathcal{T}^a$ and $\mathcal{T}^n$ similarly to the minimal explanation variant. The first training set $\mathcal{T}^a$ containing the anomalous sample x^a is used for the training of the tree, and the second, which contain only normal samples, is used in the termination criterium. Then the *Explainer* enters a loop, where in each iteration it trains one tree on $\mathcal{T}^a$, extracts and stores the decision rule used in the root of the trained tree, and then it removes the feature used in this rule from the training sets $\mathcal{T}^a$ and $\mathcal{T}^n$ to be used in the next iteration. This loop terminates, when the $knnDist(x^a, \mathcal{T}^n)$ is smaller than $\max\left(\text{knnDist}(\mathcal{T}^a, \mathcal{T}^n)\right)$. The rationale behind the stopping criterion is that a typical anomaly deviates from the rest of data only in few features and in remaining features it behaves like a normal sample. Therefore, if all anomalous features were identified and removed from the feature space, the anomaly should be undetectable.

The algorithm is shown in Algorithm 2.

3.3 Explanation by association rules

Once the random forest for a given anomaly has been trained, it is used for the explanation by turning forest's decision rules into a more readable form.

Should the forest contain only a single tree, the explanation would be easy consisting of all rules on the path

Algorithm 2 Summary of the *Explainer* algorithm for maximal explanation of a single anomaly x^a.

Input:
$data$ - input dataset; x^a anomalous sample; $size$ training set size; $n_T$number of trees to be trained.
Output:
$rules$ - rules explaining x^a

$Forest \leftarrow \emptyset$
$T^a \leftarrow \text{createTrainingSet}(data, size, x^a)$
$T^n \leftarrow \text{createTrainingSet}(data, size, \neg x^a)$
$\mathcal{G} \leftarrow allFeatures$
repeat
  $t \leftarrow \text{trainTree}(\mathcal{T}^a, \mathcal{G})$
  $Forest \leftarrow Forest + t$
  $\mathcal{G} \leftarrow \mathcal{G} \setminus \text{topSplitFeature}(t)$
until $\text{knnDist}(x^a, \mathcal{T}^n) < \max(\text{knnDist}(\mathcal{T}^a, \mathcal{T}^n))$
$rules \leftarrow \text{extractRules}(Forest)$

from the root node to the leaf with the anomalous sample assembled into conjunction of atomic conditions as

$$c = (x_{j_1} > \theta_1) \wedge (x_{j_2} > \theta_2) \wedge \ldots \wedge (x_{j_d} > \theta_d). \quad (3)$$

This conjunction can be read as "the sample is anomalous because it is greater than the threshold θ_1 in the feature j_1 and greater than θ_2 in the feature j_2 and ... than majority of samples." Because the tree is of small height, the explanation is compact.

When the random forest contains more than one tree, which is essential for improving the robustness, a conversion of its rules into set of atomic conjunction becomes more complicated. Since returning all decision rules of all trees would hamper the comprehensibility of the resulting ruleset, they are filtered so that only important ones are used.

Rules from all trees are grouped such that each set $\mathcal{H}_i$ contains rules with the same type of inequality $\{< \ . \ >\}$ and concerning the same feature. If the input space have d-features, there will be at most $2d$ sets. Sets $\{\mathcal{H}_i\}_{i=1}^{2d}$ are then sorted according to their size in descending order and then only the first k sets such that their cumulative frequency is greater than a threshold τ (recommended to be 0.90 or 0.95) are retained. Using the adopted notation, the number of retained sets rs is determined as

$$rs = \arg\min_{k'} \frac{1}{\sum_{i=1}^{2d} |\mathcal{H}_i|} \sum_{i=1}^{k'} |\mathcal{H}_i| > \tau, \quad (4)$$

where it is assumed, that sets $\{\mathcal{H}_i\}_{i=1}^{2d}$ are already sorted by their size. Finally, from each set $\mathcal{H}_i$ the rule with the threshold closest to the median is selected.

$$\bar{\theta} = \text{median}\,(\forall\theta | \theta \in \mathcal{H}_i) \quad (5)$$

Resulting rs rules are presented to the user as associative rules, i.e.:

$$(x_{j_1} > \bar{\theta}_1) \wedge (x_{j_2} > \bar{\theta}_2) \wedge \ldots \wedge (x_{j_k} > \bar{\theta}_k) \Rightarrow \text{anomaly} \quad (6)$$

Explaining multiple anomalies The *Explainer* was designed to naturally handle multiple anomalies at once. To produce a consistent explanation, anomalies should be analysed in groups of one similar type or caused by a similar process. The process of clustering anomalies was studied in our previous work focused on preserving the interpretability [35]. The clustering of anomalies based on voting of individual trees and deviations in selected features were proposed. The effect of using only features proposed by the *Explainer* for the clustering was also studied in that paper.

When the clustering is done, trees trained for all anomalies of one group are aggregated into one forest and then explained in the same way as a forest for a single anomaly.

We believe that such an additional analysis of anomalies via clustering is justified as it enables the interpretation of similar anomalies at once, saving time and human efforts. It can also help to uncover larger scale anomalies. For example, multiple measurements of an anomaly can end in the same cluster.

3.4 Examples of explanation

For illustration we present two examples of explanation from real word datasets.

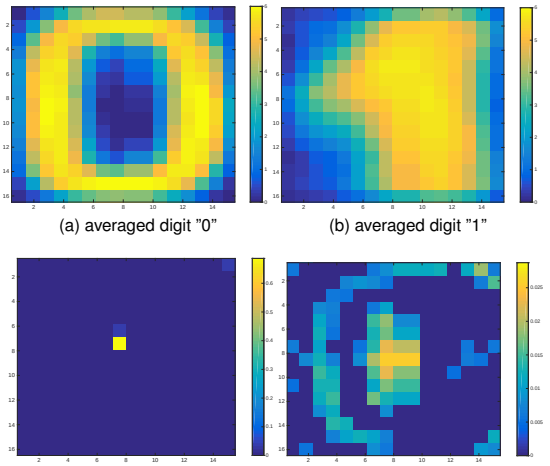
Multiple features The first example concerns a dataset called Multiple features from the UCI machine learning repository [36]. It is a classification dataset of hand-written digits, which contains six families of features. Among them, the averaged pixels family was chosen because of an easy to understand graphical representation of both inputs and outputs. Every digit is represented by 16x15 averaged pixels resulting in a vector of 240 features with values 0..6. These values represent the number of black pixels within the 2x3 rectangle. With this setting, the *Explainer* was used to explain how digits "0" differ from digits "1". The digit "0" and the digit "1" averaged over 100 samples are shown in Figure 1a and Figure 1b.

Figures 1c and 1d show a graphical representation of the minimal and maximal explanation. The minimal explanation shows that there are 2-3 features which are sufficient to clearly distinguish digits "1" and "0". In the case of maximal explanation we can see increased amount of selected features, which, in addition to pixels in the middle, form a silhouette of a zero.

TABLE 1
The antecedents of the association rules extracted during the minimal explanation of differences between digits "1" and digits "0", when "1"s were considered anomalous. Rules are sorted in order as return by the *Explainer*.

rule	recall	precision
$x_{8,7} > 4.5$	93.0	100
$x_{8,8} > 1.5$	93.0	100
$x_{8,6} > 4.5$	93.5	97.9
$x_{15,1} > 3.8$	9.0	90.0

Rules extracted from the minimal explanation are listed in Table 1 together with precision and recall of classifying the digit "1" using only the single rule. Because the consequent of all those rules would be "$\Rightarrow$ $anomaly$" it was omitted. It can be seen that those rules correspond to the features highlighted in Figure 1. The first two features $x_{8,7}$ and $x_{8,8}$ represent the two bright pixels int the middle of



(a) averaged digit "0" (b) averaged digit "1"

(c) minimal explanation of zero/one difference (d) minimal explanation of zero/one difference

Fig. 1. The averaged digit "0" (top-left) and digit "1" (top-right) from the hand-written digits recognition dataset called Multiple features from the UCI repository. The minimal explanation of difference between hand-written digits "1" and "0" (bottom-left) and maximal explanation (bottom-right), when we consider zeros as normal samples and ones as anomalies. The colour scale shows the relative frequency of using a particular feature during explanation.

the zero and the feature $x_{15,1}$ is the one in the upper right corner. Interestingly, the first three rules alone are enough to cover more than 90% of all samples labeled as digit "1", using only three features out of 240.

The output in the DNF for this particular example look like:

$$(x_{8,7} > 4.5) \vee (x_{8,8} > 1.5) \vee (x_{8,6} > 4.5) \vee (x_{15,1} > 3.8)$$

Network security The second example is a dataset from the network security domain. The goal was to identify domains used by the Bedep and Vittalia[2], malwares showing advertisements against the user's will. The data set contains approximately 460 000 benign unique URL's and 6667 URL's used by Bedep or Vittalia. Each URL is treated as an individual sample. Examples of malware URLs are:

```
olpofraskolfa.com/ads.php?sid=1926
bionookersako.in/ads.php?sid=1929
ads.under-myscreen.be/cgi-bin/advert/
getkws.cgi?did=11727.
```

To convert URL to a vector of fixed dimension, URL is first split into the tokens using natural separators for each part of URL (dots for domain, slashes for path and ampersands and equal signs for query[3]). To decrease the total number of tokens, some tokens are replaced by meta tokens, e.g. `<n>` for numbers, instead of the exact value of the answer. After the tokenization, the above urls looks like

```
olpofraskolfa com ads php sid <n>
bionookersako in ads php sid <n>
ads under my screen be cgi bin advert
getkws cgi did <n>
```

To further decrease the total number of tokens, a hash function proposed by [37] is used to convert each token to an integer in the range $[1, \ldots, 2048]$. Each URL is then represented as a vector of length 2048 by bag of words encoding with one word corresponding to one hashed token. The complete set of steps is outlined at the Figure 2.

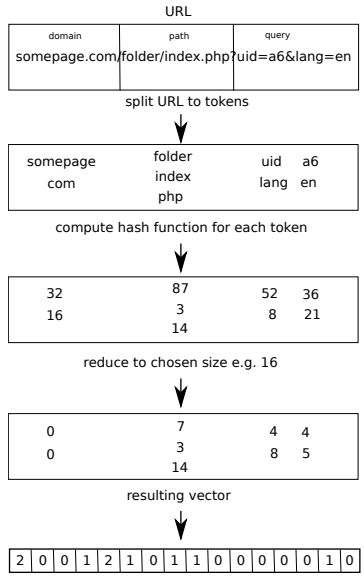


Fig. 2. Schematics of transformation of a URL to a fixed length vector of integers.

The *Explainer* was used to identify the most important (minimal explanation) tokens explaining the differences between malware, taken as anomalous, and benign URLs, taken as normal. It was set to train 5 trees per each malware URL with training set size 500 URLs. The top rs extracted rules are shown in Table 2. Only the antecedents of the association rules are presented in the table as the consequent would be "$\Rightarrow$ *anomaly*" in all cases.

TABLE 2
Example of antecedents of the association rules extracted by the *Explainer* explaining differences between adware and benign URLs sorted according to their precision.

rule	recall	precision
`gmsd` > 0	17.8	100
`advert` > 0	39.9	99.5
`kws` > 0	39.9	99.3
`screen` > 0	39.6	94.2
`sid` $> 0 \wedge$ `ads` > 0	21.5	94.2
`ers` > 0	9.2	94.1
`json` > 0	32.3	66.1

Identified tokens like `ads`, `advert`, `sid` and `screen` reveal their purpose, since tokens `screen` identifies the

2. All samples of both, Bedep and Vittalia, were captured during the summer 2015. Presented patterns are likely to be different from patterns used by the malware presently.

3. The query part contains pairs of query and answer, which is used as a variable and value.

key informing the server about resolution of clients screen, `sid` is a key of device identifier, and `ads` and `advert` are apparently related to advertisements. Tokens `gmsd`, `ers`, `kws` were found exclusively or almost exclusively in adware domains, yet the concrete purpose is unknown to authors. The only controversial token is `json`, which is a file format used for structured data and is completely legitimate. From almost 500 000 URLs used in this experiment it was contained in 3243 samples and 2143 of them were adware, which means that 32.3% of adware URLs use the `json` token whereas it was used only in 0.23% of legitimate URLs. According to the setting of the experiments, where only 500 legitimate samples were selected to explain each anomaly, it is evident why it was identified as a discriminative feature in so many cases.

3.5 Algorithm complexity

The computational complexity of the *Explainer* is $O\left(n_T \cdot T_{sel} \cdot T_{train}\right)$. n_T is the number of trees trained for each anomaly, which is a user given parameter for the minimal explanation and $n_T = O(d-1)$ for the maximal explanation, where d is the number of features. The term T_{sel} stands for the complexity of training set selection which is $O\left(|\mathcal{X}^n|\right)$ where $|\mathcal{X}^n|$ is the number of normal samples in the data. The last term T_{train} represents training of a single tree and is equal to $O\left(d \cdot |\mathcal{T}|^2\right)$ for the minimal explanation, and $O\left(d^2 \cdot |\mathcal{T}|^2\right)$ for the maximal explanation, where $|\mathcal{T}|$ is the size of the training set which is the second parameter that needs to be given in advance, typically much smaller than $|\mathcal{X}^n|$ (see Section 4.3 for details).

4 Experiments

This section presents an experimental comparison of the *Explainer* to the state of the art algorithms by [25], [26] and [28] on 36 classification datasets from the UCI machine learning repository [36], with the number of features ranging from 4 to 5000 and number of samples from 22 up to nearly 100k.

Since there is a lack of the prior art on explaining anomalies, the methodology to measure quality of an explanation has not been established yet. Therefore, this work measures how the area under the receiver operating characteristic (AUC) of an anomaly detector improves if the anomaly detection uses only features used in the explanation in comparison to the baseline when all features are used. The rationale behind is that the explanation should contain only features in which the anomaly is most different from the remaining data. Therefore, the AUC in the reduced space should be comparable or even higher than the AUC computed on the full space. Also recall that all recent prior art to which the *Explainer* is compared stop the explanation after identifying the set of features in which anomaly is visible, which further justifies the choice of using AUC for the comparison.

The algorithms are also compared on the basis of the number of returned features and scalability, which is important for contemporary large data.

4.1 Experimental settings

The *Explainer* is compared to the algorithms of [25], [28] and [26]. The implementations of the first two algorithms were kindly provided by their authors, while the last algorithm was implemented by ourselves. The compared algorithms tried to explain anomalies identified by Local Outlier Factor (LOF) by [23], a $k-$nearest neighbor (knn) classifier by [38] and the Isolation Forest (iForest) by [31]. To avoid the influence of incorrect detections, the perfect detector scenario was simulated by explanation of the ground truth (GT) anomalies.

As mentioned above, the performance of anomaly detectors was measured by the area under the receiver operating characteristic (AUC) of an anomaly detection algorithm using only the features selected by the explanation algorithm. The reported AUC values were measured by the anomaly detector under investigation, e.g. LOF was used to measure AUC in all LOF related experiments. For the evaluation of experiments with ground truth anomalies the Isolation forest detector was used because it is the fastest detector with very good overall performance.

The anomaly detection algorithms were set as follows. The Isolation forest was set to create 100 trees using 256 samples for each, the thresholds between normal and anomalous samples were estimated from data, separately for each dataset. The size of local neighbourhood for the LOF detecter was set to 10 and detection threshold was set to 2, in accordance with the recommendation in [23]. The k-nn used only $k = 4$, while the threshold on a distance to the average of the k nearest neighbours was estimated from data, separately for each dataset.

The *Explainer* used training sets of size $|\mathcal{T}| = 100$. The number of trained trees was set to $n_T = 5$ for minimal explanation, whereas it is determined automatically during the run of the algorithm for maximal explanation. The other algorithms were set up as recommended by the authors in the respective papers. In [25], the neighbouring set size k varied from 10 to 40, in [26], the regularization parameter was set to $\alpha = 0.1$ and the neighbourhood sizes varied from 5 to 25. Finally, [28] has two parameters and according to the authors, the algorithm should be robust to their setting, which our experiments with α varied from 0.1 to 0.7 and k from 20 to 50 have confirmed. The reported results were obtained with the setting $\alpha = 0.5$ and $k = 50$.

The experimental comparison used 36 different datasets from UCI machine learning repository, [36], covering many application domains. Since they are designed for supervised classification, they were converted to anomaly detection problems as recommended by [11]. The LOF alone showed to be the worst of all tested detection algorithms as it did not found any outliers in 6 out of 36 datasets. Therefore, the comparison was done using only 30 datasets.

4.2 Experimental results

Quality of explanation measured by the AUC At first, the quality of explanation was compared by measuring the aforementioned improvement in AUC. Due to the large number of problems on which the algorithms are compared, the results are summarized by critical difference diagrams as proposed by [12]. Figure 3 shows the average rank of each algorithm over all 36 problems, groups of algorithms that are not significantly different according to the Nemenyi post-hoc analysis are connected. Table **??** contains all results for ground truth labels in detail.

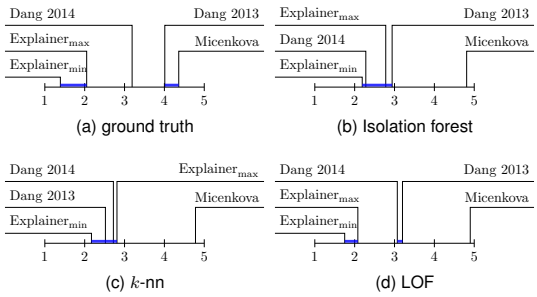


Fig. 3. Critical comparison of explainers when ground truth anomalies (top-left) and anomalies discovered by Isolation forest (top-right), k-nn (bottom-left) and LOF (bottom-right) were explained.

The *Explainer*$_{\min}$ performs consistently the best regardless the anomaly detector used to identify the anomalies, though the margin of the improvement over the prior art depends on that detector. For anomalies identified by Isolation Forest and k-nn, the statistical test did not reject the hypothesis that the *Explainer*$_{\min}$ and Dang 2013 and Dang 2014 are statistically different. Still, in both cases the *Explainer*$_{\min}$ achieved the lowest rank. For anomalies identified by the LOF, the *Explainer* was significantly better than prior methods.

When the effect of an anomaly detector is removed by explaining the ground truth, both the *Explainer*$_{\min}$ and *Explainer*$_{\max}$ are significantly better than all of the prior art according to the [39] test and the post-hoc tests with the corrections for simultaneous hypotheses testing by [40]. Precise results are shown in Table 3.

The algorithm of [28] didn't work well with the k-nn detector as it didn't return any features as an explanation in 17 out of 36 cases, despite the algorithm has been executed with a wide range of parameters.

Further investigation, of the cases where the *Explainer*$_{\min}$ performed worse revelled that in multiple features, waveform-1 and waveform-2 datasets in combination with k-nn anomaly detector, the *Explainer*$_{\min}$ has removed features responsible for clustering normal samples together, which misleads the k-nn detector to create new anomalies that were not present in the full space version of the dataset. Yet authors believe that the explanation was correct and the worse performance is due to the detector rather than due to the *Explainer*. The *Explainer*$_{\min}$ excels on high dimensional datasets like gissete, madelon, libras and on noisy datasets. the *Explainer*$_{\max}$ seems to be more susceptible to mistakes in detection than the *Explainer*$_{\min}$. While the explanations were reasonable, anomaly detection on more features was typically worse (see Table 6) since more features increase the signal to noise ration to which anomaly detectors are sensitive, and which explains the worse AUC.

To conclude this comparison, the performance of [28] algorithm strongly depends on the particular dataset. The [26] improved the [25] and is the closest prior art to the *Explainer* in the terms of identification of features in which the anomaly is detectable.

The compactness of explanation by number of features

TABLE 3
Results of the statistical test of [39] and its post-hoc tests with the correction for simultaneous hypotheses testing by [40]. The rejection thresholds are computed for the family-wise significance level $\alpha = 0.05$

algorithms	p	rejection threshold
Explainer$_{\min}$ vs. Micenkova 2013	1.52e-15	0.005
Explainer$_{\min}$ vs. Dang 2013	2.45e-12	0.008
Explainer$_{\max}$ vs. Micenkova 2013	6.15e-12	0.008
Explainer$_{\max}$ vs. Dang 2013	1.81e-12	0.008
Explainer$_{\min}$ vs. Dang 2014	1.27e-6	0.008
Dang 2014 vs. Micenkova	0.0017	0.013
Explainer$_{\min}$ vs. Dang 2014	0.0022	0.013
Dang 2013 vs. Dang 2014	0.0306	0.017
Explainer$_{\min}$ vs. *Explainer*$_{\max}$	0.0736	0.025
Dang 2013 vs. Micenkova	0.3325	0.050

Figure 4a shows the median of selected features used in explanation of the ground truth anomalies. Since the *Explainer*$_{\min}$ was designed to select as few features as possible, it consistently selects the minimal number of features among all tested methods. Taking into account that it also outperforms all prior art in terms of the AUC, it can be concluded that its explanations are more compact and therefore more precisely specifying root causes of anomalies, which was its goal. Surprisingly, even though the *Explainer*$_{\max}$ should return all relevant features (and therefore returns more features than the *Explainer*$_{\min}$), their number is still considerably lower than in case of all prior art methods, especially for large datasets.

The comparison of running times The last comparison was done with respect to the computational complexity, measured as a total time to explain all ground-truth anomalies. As in the previous experiment, the last two datasets are missing for both Dang's algorithms and one for Micenková, because of their excessive memory and computational requirements. Figure 4b show running times averaged over 20 runs. On smaller datasets, the algorithm of [26] clearly outperformed all other algorithms, but for the larger datasets, it is evident that the *Explainer* methods are much more efficient. The break point is approximately 1000 samples for low dimensional datasets or 200 samples for high dimensional datasets.

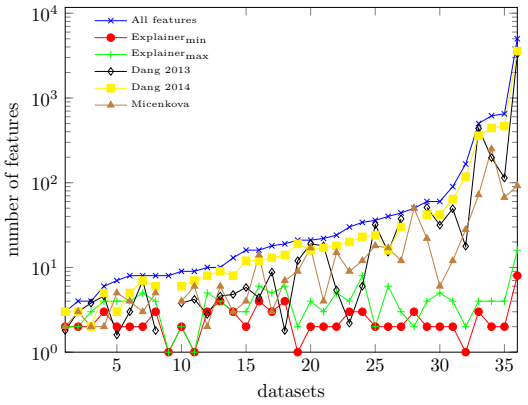
Comparison with Isolation forest:

The *Explainer* and Isolation forest are both based on a random forests trained only on a small sub sample of data, so they may seem similar. This section shows that not only their main purpose but also their outcomes differ substantially.

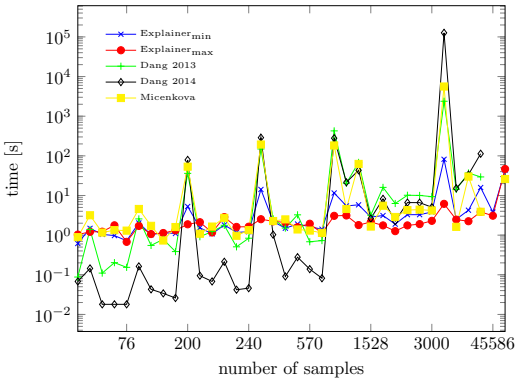
Isolation forest is an ensemble of random trees. More specifically, both the feature and threshold are selected randomly for each splitting node. The main idea is that anomalies differ strongly from the normal samples, therefore, they should be separated with fewer splits. There is no focus on identifying key features or any other way of explanation or description of findings.

The main goal of the *Explainer* is to identify features responsible for the anomalous nature of the sample under analysis. Therefore, each split uses the feature and threshold that produces the best possible separation of an anomaly and normal samples. Features used in spilt nodes are then assembled into classification rules.

Similar rules can also be extracted from Isolation forests.



(a) number of features



(b) running times

Fig. 4. The comparison of the number of selected features per dataset. The curve marked as all features shows the number of features of a particular dataset (left). The comparison of the running times (right). Please notice that number of features as well as running times are plotted in logarithmic scale.

One of the basic properties indicating the comprehensibility of a rule is its length. A comparison of the average rule length for each dataset is depicted at Figure 5 and summarised in Table 4.

TABLE 4
The average length of rules extracted from *Explainer*$_{\min}$, *Explainer*$_{\max}$ and Isolation forest over all datasets.

Explainer$_{\min}$	*Explainer*$_{\max}$	Isolation forest
2.47	4.06	5.79

The rules produced by Isolation forests are not only longer (even against the *Explainer*$_{\max}$) but they have also other problems, caused by the random nature of this method. First, extracted rules are inconsistent and used features vary every time a forest is trained. Secondly, rules extracted from a random tree can contain a rule on one feature many times, even in succession, or contain rules without any impact on resulting set. Therefore, rules extracted from Isolation forests are not suitable for the explanation.

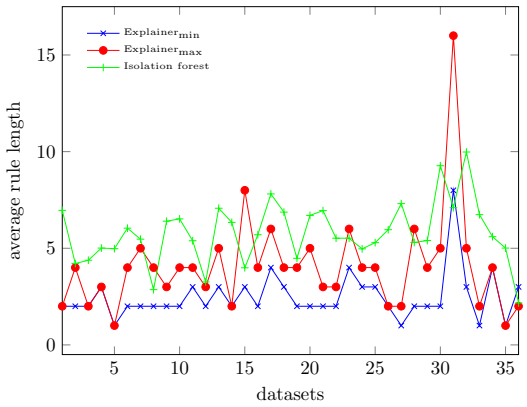
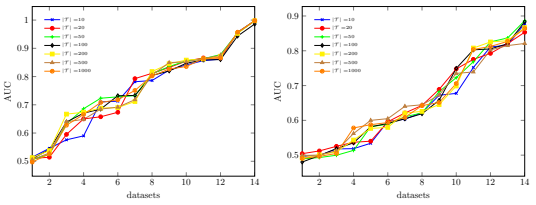
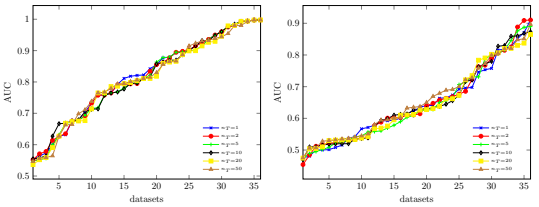


Fig. 5. Comparison of the average length of the rules extracted from the *Explainer* and the Isolation forest anomaly detector.



(a) ground truth (b) Isolation forest



(c) ground truth (d) Isolation forest

Fig. 6. AUCs of anomaly detection on datasets with more than 1000 samples (14 of all used dataset) using only features suggested by the *Explainer* with varying training set sizes when ground truth anomalies were explained (top-left) and when anomalies identified by Isolation forest were explained (top-right). The reported values are the average of 10 runs training 10 trees per anomaly. The *Explainer* was used to explain the ground truth anomalies (bottom-left) and the anomalies identified by the Isolation forest detector (bottom-right). The AUCs are average of 10 runs using only features suggested by the *Explainer* with the varying number of trees trained per anomaly and fixed training set size to 100 samples.The results for each setting were sorted according to their AUC, showing general performance rather than performance on specific datasets. The interpretation is simple, the line of the setting leading to better average AUCs will be higher than the line representing the setting leading to lower average AUCs.

4.3 Sensitivity to parameters

The *Explainer* is controlled by two parameters: by the training set size $|\mathcal{T}|$ and by the number of trees trained per anomaly n_T. In this section we show the robustness of the *Explainer* to the setting of those parameters, effectively making the *Explainer* almost a non-parametric method.

TABLE 5
Results of the Friedman's statistical test with the correction for simultaneous hypotheses testing by [40], calculated while testing the training set size of the *Explainer*$_{\min}$ for explanation of the ground truth anomalies. The rejection thresholds are computed for the family-wise significance level $\alpha = 0.05$

settings	p	rejection threshold
$\|\mathcal{T}\|$ =200 vs. $\|\mathcal{T}\|$ =1000	0.325	0.0023
$\|\mathcal{T}\|$ =10 vs. $\|\mathcal{T}\|$ =1000	0.407	0.0025
$\|\mathcal{T}\|$ =20 vs. $\|\mathcal{T}\|$ =1000	0.437	0.0026
$\|\mathcal{T}\|$ =100 vs. $\|\mathcal{T}\|$ =1000	0.468	0.0028
$\|\mathcal{T}\|$ =50 vs. $\|\mathcal{T}\|$ =200	0.501	0.0029
$\|\mathcal{T}\|$ =200 vs. $\|\mathcal{T}\|$ =500	0.534	0.0031
$\|\mathcal{T}\|$ =10 vs. $\|\mathcal{T}\|$ =50	0.604	0.0033
$\|\mathcal{T}\|$ =20 vs. $\|\mathcal{T}\|$ =50	0.641	0.0036
$\|\mathcal{T}\|$ =10 vs. $\|\mathcal{T}\|$ =500	0.641	0.0038
$\|\mathcal{T}\|$ =50 vs. $\|\mathcal{T}\|$ =100	0.678	0.0042
$\|\mathcal{T}\|$ =20 vs. $\|\mathcal{T}\|$ =500	0.678	0.0045
$\|\mathcal{T}\|$ =100 vs. $\|\mathcal{T}\|$ =500	0.717	0.0050
$\|\mathcal{T}\|$ =500 vs. $\|\mathcal{T}\|$ =1000	0.717	0.0056
$\|\mathcal{T}\|$ =50 vs. $\|\mathcal{T}\|$ =1000	0.756	0.0063
$\|\mathcal{T}\|$ =100 vs. $\|\mathcal{T}\|$ =200	0.795	0.0071
$\|\mathcal{T}\|$ =20 vs. $\|\mathcal{T}\|$ =200	0.835	0.0083
$\|\mathcal{T}\|$ =10 vs. $\|\mathcal{T}\|$ =200	0.876	0.0100
$\|\mathcal{T}\|$ =10 vs. $\|\mathcal{T}\|$ =100	0.917	0.0125
$\|\mathcal{T}\|$ =50 vs. $\|\mathcal{T}\|$ =500	0.958	0.0167
$\|\mathcal{T}\|$ =10 vs. $\|\mathcal{T}\|$ =20	0.958	0.0250
$\|\mathcal{T}\|$ =20 vs. $\|\mathcal{T}\|$ =100	0.958	0.0500

At first, the *Explainer*$_{\min}$ was executed with the number of trees per anomaly $n_T = 5$ while varying the training set size $\mathcal{T} \in \{10, 20, 50, 100, 500, 1000\}$, for that reason the experiment was performed only on datasets with more than 1000 samples. AUCs of different configurations, measured by the Isolation forest detector, are shown for ground truth anomalies in Figure 6a and anomalies discovered by the Isolation forest 6b. According to these results, the *Explainer* does not seem to be sensitive to the size of the training set, as all AUCs are very similar. Results of the Friedman's test and its post-hoc tests are all in 5. When explaining ground truth, training sets as small as 50 - 100 samples were sufficient. When explaining anomalies identified by a real anomaly detector, larger training sets like 100–200 samples seems to be more appropriate. The finding that even a small training set is sufficient for good explanations is especially important for data-streams, where all samples needed for explanation have to be stored in memory.

Fixing the size of the training set to $\mathcal{T} = 100$,the *Explainer*$_{\min}$ was executed with different number of trees per anomaly $n_T \in \{1, 2, 5, 10, 20, 50\}$. AUCs of all configurations, measured by the Isolation forest detector, are shown in Figure 6c for ground truth anomalies and in Figure 6d for anomalies identified by Isolation Forests. Similarly to the size of the training dataset, the *Explainer* does not seem to be sensitive to this setting. Results of the Friedman's test and its post-hoc tests are all in the Appendix C. According to the results using 5-10 trees per anomaly is recommended because its stable while still being computationally efficient.

5 Conclusion

This paper has presented a novel approach to explaining the deviation of a sample, identified by an arbitrary anomaly detection algorithm, from the rest of data. The proposed approach relies on specifically trained random forests to identify rules explaining the anomaly. Those rules are then assembled into the explanation in the form of a set of association rules or in the disjunctive normal form and presented to the user. The set of association rules is easy to read and understand by humans, whereas the disjunctive normal form was chosen because it is compact and practical for further machine processing.

The proposed approach has been extensively compared to the relevant prior art on 36 UCI machine learning repository datasets while explaining anomalies identified by multiple anomaly detection algorithms. The results show a superiority of the proposed solution, as it provides tighter and more accurate explanations. In addition, our testing has shown that it is also superior with respect to the time complexity, especially on the high dimensional datasets.

As to the theoretical fundamentals of our approach, we proved that our simplified splitting criterion used during the specific training of random forest is for this kind of random forest equivalent to the common best practice splitting criteria based on information gain or Gini impurity.

Appendix

Let $\mathcal{H}$ denotes the space of all possible splitting rules defined as:

$$\mathcal{H} = \{h_{j,\theta} | j \in \{1, \dots, d\}, \theta \in \mathbb{R}\},$$

where

$$h_{j,\theta}(x) = \begin{cases} +1 & \text{if } x_j > \theta \\ -1 & \text{otherwise} \end{cases}$$

with x_j being the j^{th} feature of x and θ representing a threshold. The most popular criteria to select the splitting function $h \in \mathcal{H}$ of a new internal node are *information gain* defined as

$$IG(\mathcal{S}, h) = H(\mathcal{S}) - \sum_{b \in \{L,R\}} \frac{|\mathcal{S}^b(h)|}{|\mathcal{S}|} H(\mathcal{S}^b) \tag{7}$$

and *Gini impurity* defined as

$$GI(\mathcal{S}) = 1 - \sum_{b \in \{L,R\}} f^2(\mathcal{S}^b(h)), \tag{8}$$

where $\mathcal{S}$ is the subset of the training set $\mathcal{T}$ reaching the leaf being split, $\mathcal{S}^L(h) = \{x \in \mathcal{S} | h(x) = +1\}$ and $\mathcal{S}^R(h) = \{x \in \mathcal{S} | h(x) = -1\}$, $H(\mathcal{S})$ is the Shanon entropy of $\mathcal{S}$ and f is the fraction of samples of the specified class in the set.

In the following lemma, we show that when training set $\mathcal{T}$ contains exactly one anomaly, both criteria (7) and (8) are equivalent and equal to

$$\arg\min_{h \in \mathcal{H}} |\mathcal{S}^a(h)|, \tag{9}$$

where $|\mathcal{S}^a|$ is the size of the set containing the anomaly after splitting. This splitting criterion is easy to interpret and can be calculated more effectively than information gain or Gini impurity.

Lemma A.1. *Denote $T_q(p_1, \dots, p_k)$ the Havrda-Charvát entropy with index $q \geq 1$, up to a multiplicative constant coinciding with the more recently introduced Tsallis entropy, [41]) for a finite-dimensional probability distribution $(p_1, \dots, p_k)$, i.e.,*

TABLE 6
The table shows the area under ROC curves of the Isolation forest detector executed on the full feature space baseline (captioned "all"), and that of executed on sub-spaces spanned by features used to explain ground truth anomalies identified by the $Explainer_{\min}$,$Explainer_{\max}$, the better of two Dang's algorithms and algorithm of Micenkova. The smaller numbers in brackets behind each AUC are the number of used features, averaged over 10 runs, in the respective case. Finally, the last column captioned l shows the number of samples.

id	dataset	all	$Expl_{\min}$	$Expl_{\max}$	Dang14	Micenkova	l
1	breast-tissue	0.61 (9)	**0.83** (2)	0.82 (2)	0.77 (6)	0.62 (4)	22
2	libras	0.42 (90)	0.60 (2)	**0.64** (4)	0.50 (64)	0.50 (12)	24
3	iris	0.57 (4)	0.84 (2)	**0.85** (2)	0.84 (3)	0.49 (3)	50
4	wine	0.65 (13)	**0.93** (3)	0.88 (3)	0.84 (8)	0.66 (3)	71
5	glass	0.43 (9)	**0.99** (1)	0.98 (1)	0.48 (7)	0.59 (6)	76
6	synthetic-control-chart	0.68 (60)	0.75 (2)	0.75 (4)	**0.81** (42)	0.55 (22)	100
7	sonar	0.46 (60)	**0.77** (2)	0.72 (5)	0.67 (42)	0.42 (6)	111
8	ecoli	0.65 (7)	**0.80** (2)	0.73 (4)	0.59 (3)	0.49 (5)	143
9	parkinsons	0.48 (22)	**0.80** (2)	0.72 (3)	0.53 (17)	0.58 (4)	147
10	multiple-features	0.63 (649)	**0.78** (2)	0.74 (4)	0.70 (466)	0.59 (67)	200
11	vertebral-column	0.56 (6)	**0.61** (3)	0.60 (4)	0.60 (5)	0.54 (2)	200
12	spect-heart	0.26 (44)	**0.59** (2)	0.56 (3)	0.41 (30)	0.30 (12)	212
13	statlog-vehicle	0.54 (18)	**0.65** (3)	0.61 (5)	0.63 (13)	0.58 (3)	218
14	haberman	0.66 (3)	0.69 (2)	0.56 (2)	0.62 (3)	**0.71** (2)	225
15	ionosphere	0.74 (34)	**0.97** (3)	0.89 (8)	0.89 (23)	0.72 (2)	225
16	isolet	0.41 (617)	**0.58** (2)	0.55 (4)	0.53 (443)	0.40 (250)	240
17	statlog-segment	0.57 (19)	**0.79** (4)	0.76 (6)	0.76 (14)	0.41 (7)	330
18	breast-cancer-wisconsin	0.80 (30)	**0.96** (3)	0.95 (4)	0.92 (20)	0.75 (9)	357
19	yeast	0.44 (8)	0.56 (2)	**0.56** (4)	0.43 (5)	0.47 (4)	463
20	pimaIndians	0.60 (8)	**0.74** (2)	0.72 (5)	0.67 (7)	0.64 (3)	500
21	blood-transfusion	0.53 (4)	**0.61** (2)	0.60 (3)	0.58 (2)	0.57 (2)	570
22	letter-recognition	0.39 (16)	**0.59** (2)	0.57 (3)	0.53 (4)	0.38 (12)	813
23	pendigits	0.72 (16)	0.85 (4)	**0.88** (6)	0.81 (12)	0.66 (14)	1144
24	madelon	0.44 (500)	**0.55** (3)	0.54 (4)	0.49 (358)	0.43 (72)	1300
25	abalone	0.53 (8)	0.52 (3)	0.52 (4)	0.51 (6)	**0.55** (5)	1528
26	statlog-satimage	0.60 (36)	**0.85** (2)	0.84 (2)	0.77 (24)	0.55 (18)	1533
27	cardiotocography	0.46 (21)	**0.86** (1)	0.83 (2)	0.61 (19)	0.32 (9)	1655
28	waveform2	0.67 (40)	0.67 (2)	**0.74** (6)	0.72 (16)	0.61 (17)	1692
29	waveform1	0.67 (21)	**0.69** (2)	0.66 (4)	0.68 (16)	0.62 (17)	1696
30	wall-following-robot	0.55 (24)	0.73 (2)	**0.73** (5)	0.68 (18)	0.54 (15)	2205
31	gisette	0.42 (5000)	**0.72** (8)	0.69 (16)	0.52 (3576)	0.32 (92)	3000
32	page-blocks	0.85 (10)	0.94 (3)	**0.95** (5)	0.82 (8)	0.78 (2)	4913
33	musk2	0.40 (166)	0.66 (1)	**0.66** (2)	0.50 (118)	0.43 (28)	5581
34	magic-telescope	0.67 (10)	**0.80** (4)	0.79 (4)	0.68 (9)	0.58 (6)	12332
35	statlog-shuttle	0.77 (8)	**1.00** (1)	1.00 (1)	nan (nan)	nan (nan)	45586
36	miniboone	0.62 (50)	**0.86** (3)	0.79 (2)	nan (nan)	0.66 (50)	93565

$$T_q = \frac{1}{q-1}\left(1-\sum_{i=1}^{k} p_i^q\right) \text{ if } q>1, T_1 = \lim_{q\to 1+} T_q = \sum_{i=1}^{k} p_i \ln p_i.$$

In particular, T_1 is the $(\ln 2)$-multiple of the Shanon entropy, and T_2 is the Gini impurity. Let now $\mathcal{S}$ be a set of samples (feature vectors) that reached a given inner node of a given tree, such that $\mathcal{S}$ contains only one anomaly $x^a \in \mathcal{S}$ and all vectors from $\mathcal{S}\setminus\{x^a\}$ are normal. Let $\mathcal{S}$ be split into disjoint subsets $\mathcal{S}_a, \mathcal{S}_n$ such that $x^a \in \mathcal{S}_a$.
Then the decrease ΔT_q of T_q for the empirical distribution of anomalies and normal vectors after the split of $\mathcal{S}$ into $\mathcal{S}_a$ and $\mathcal{S}_n$, i.e.,

$$\Delta T_q = T_q\left(\frac{1}{|\mathcal{S}|}, \frac{|\mathcal{S}|-1}{|\mathcal{S}|}\right) - \left(\frac{|\mathcal{S}_a|}{|\mathcal{S}|} T_q\left(\frac{1}{|\mathcal{S}_a|}, \frac{|\mathcal{S}_a|-1}{|\mathcal{S}_a|}\right) + \frac{|\mathcal{S}_n|}{|\mathcal{S}|} T_q(0,1)\right)$$

is a decreasing function of $|\mathcal{S}_a|$.

Proof. To simplify the notation, denote $p = \frac{1}{|\mathcal{S}_a|}$ and $c = T_q(\frac{1}{|\mathcal{S}|}, \frac{|\mathcal{S}|-1}{|\mathcal{S}|})$, which is a constant with respect to $|\mathcal{S}_a|$. Then, being a decreasing function of $|\mathcal{S}_a|$ is equivalent to being an increasing function of p on the set $\{\frac{1}{|\mathcal{S}|}, \dots, \frac{|\mathcal{S}|-1}{|\mathcal{S}|}\}$, for which it is sufficient to prove that ΔT_q is an increasing function of p on $(0,1)$.

1) The case $q > 1$.

$$\begin{aligned}\Delta T_q &= -\frac{|\mathcal{S}_a|}{|\mathcal{S}|} T_q\left(\frac{1}{|\mathcal{S}_a|}, \frac{|\mathcal{S}_a|-1}{|\mathcal{S}_a|}\right) + c \\ &= \frac{1}{q-1}\frac{p^q + (1-p)^q - 1}{|\mathcal{S}|p} + c.\end{aligned}$$

Consequently, on $(0,1)$,

$$\frac{\mathrm{d}\Delta T_q}{\mathrm{d}p} = \frac{1}{q-1}\frac{\varphi(p)}{|\mathcal{S}|p^2}, \tag{10}$$

where $\varphi : (0,1) \to \mathbb{R}$ is a function defined

$$(\forall p \in (0,1))\ \varphi(p) = (q-1)p^q + 1 - (1-p)^{(q-1)}((q-1)p+1).$$

Computing the derivative of φ yields

$$(\forall p \in (0,1))\ \varphi'(p) = q(q-1)p(p^{(q-2)} + (1-p)^{(q-2)}) > 0,$$

which entails φ being strictly increasing on $(0,1)$. Therefore,

$$(\forall p \in (0,1))\ \varphi(p) > \lim_{p'\to 0+} \varphi(p') = 0.$$

Combining this with (10) implies that ΔT_q is indeed an increasing function of p on $(0,1)$.

2) The case $q = 1$.

$$\Delta T_q = -\frac{|\mathcal{S}_a|}{|\mathcal{S}|} T_q\left(\frac{1}{|\mathcal{S}_a|}, \frac{|\mathcal{S}_a|-1}{|\mathcal{S}_a|}\right) + c = \frac{1}{|\mathcal{S}|}\frac{\varphi(p)-\varphi(0)}{p} + c, \tag{11}$$

where φ is the negentropy on $\langle 0, 1)$ defined

$$\varphi(p) = \begin{cases} p \log p + (1-p)\log(1-p) & \text{if } p \in (0,1), \\ \lim_{p' \to 0+} \varphi(p') = 0 & \text{if } p = 0. \end{cases} \tag{12}$$

Computing the second derivative of φ yields

$$(\forall p \in (0,1))\ \varphi''(p) = \frac{1}{p} + \frac{1}{1-p} > 0,$$

which entails the strict convexity of φ on $(0, 1)$. This together with its right-continuousity in 0 (cf. (12)) implies that $\frac{\varphi(p)-\varphi(0)}{p}$ is an increasing function of p on $(0, 1)$, and due to (11), this holds also for ΔT_q.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[2] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, 2009.

[3] R. Tibshirani and T. Hastie, "Outlier sums for differential gene expression analysis," *Biostatistics*, 2007.

[4] H. Dutta, C. Giannella, K. D. Borne, and H. Kargupta, "Distributed top-k outlier detection from astronomy catalogs using the demac system." in *SDM*, 2007, pp. 473–478.

[5] R. Fujimaki, T. Yairi, and K. Machida, "An approach to spacecraft anomaly detection problem using kernel feature space," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005.

[6] C. C. Aggarwal, *Outlier Analysis*. Springer, 2013.

[7] T. Pevný, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, vol. 102, no. 2, pp. 275–304, 2016.

[8] E. Knorr and R. T. Ng, "Algorithms for mining distancebased outliers in large datasets," in *Proceedings of the International Conference on Very Large Data Bases*, 1998.

[9] E. M. Knorr and R. T. Ng, "Finding intensional knowledge of distance-based outliers," in *Proceedings of the International Conference on Very Large Data Bases*, 1999.

[10] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models," *Decision Support Systems*, vol. 51, no. 1, pp. 141–154, 2011.

[11] A. F. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong, "Systematic construction of anomaly detection benchmarks from real data," in *Proceedings of the ACM SIGKDD workshop on outlier detection and description*. ACM, 2013, pp. 16–21.

[12] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, 2006.

[13] F. E. Grubbs, "Sample criteria for testing outlying observations," *The Annals of Mathematical Statistics*, pp. 27–58, 1950.

[14] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," DTIC Document, Tech. Rep., 2003.

[15] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John wiley & sons, 2005, vol. 589.

[16] Y. Zhang, N. Meratnia, and P. Havinga, "Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks," in *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*. IEEE, 2009, pp. 990–995.

[17] V. Roth, "Kernel fisher discriminants for outlier detection," *Neural computation*, vol. 18, no. 4, pp. 942–960, 2006.

[18] R. Vert and J.-P. Vert, "Consistency and convergence rates of one-class svms and related algorithms," *Journal of Machine Learning Research*, vol. 7, no. May, pp. 817–854, 2006.

[19] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of data mining in computer security*. Springer, 2002, pp. 77–101.

[20] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: a review," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 90–105, 2004.

[21] H.-P. Kriegel, A. Zimek *et al.*, "Angle-based outlier detection in high-dimensional data," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 444–452.

[22] M. Radovanovic, A. Nanopoulos, and M. Ivanovic, "Reverse nearest neighbors in unsupervised distance-based outlier detection," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 5, pp. 1369–1382, 2015.

[23] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM Sigmod Record*, 2000.

[24] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "Loci: Fast outlier detection using the local correlation integral," in *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE, 2003, pp. 315–326.

[25] X.-H. Dang, B. Micenková, I. Assent, and R. T. Ng, "Local outlier detection with interpretation," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2013)*, 2013.

[26] X. H. Dang, I. Assent, R. T. Ng, A. Zimek, and E. Schubert, "Discriminative features for identifying and interpreting outliers," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 2014, pp. 88–99.

[27] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 207–244, 2009.

[28] B. Micenková, R. T. Ng, X.-H. Dang, and I. Assent, "Explaining outliers by subspace separability," in *IEEE 13th International Conference on Data Mining (ICDM 2013)*, 2013.

[29] I. Steinwart, D. Hush, and C. Scovel, "A classification framework for anomaly detection," *Journal of Machine Learning Research*, vol. 6, no. Feb, pp. 211–232, 2005.

[30] F. Angiulli, F. Fassetti, and L. Palopoli, "Detecting outlying properties of exceptional objects," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 1, p. 7, 2009.

[31] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Eighth IEEE International Conference on Data Mining (ICDM 2008)*, 2008.

[32] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[33] A. Criminisi, J. Shotton, E. Konukoglu *et al.*, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends® in Computer Graphics and Vision*, vol. 7, no. 2–3, pp. 81–227, 2012.

[34] C. Gini, "Measurement of inequality of incomes," *The Economic Journal*, vol. 31, no. 121, pp. 124–126, 1921.

[35] M. Kopp, T. Pevny, and M. Holena, "Interpreting and clustering outliers with sapling random forests," in *Information Technologies Applications and Theory Workshops, Posters, and Tutorials (ITAT)*, 2014.

[36] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[37] B. Jenkins, "A new hash function for hash table lookup," *Dr. Dobb's Journal*, 1997.

[38] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002.

[39] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[40] J. P. Shaffer, "Multiple hypothesis testing," *Annual review of psychology*, vol. 46, no. 1, pp. 561–584, 1995.

[41] N. Leonenko, L. Pronzato, V. Savani *et al.*, "A class of rényi information estimators for multidimensional densities," *The Annals of Statistics*, vol. 36, no. 5, pp. 2153–2182, 2008.