FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Contextual Passive DNS Resolution
Student:	Bc. Olena Marchenko
Supervisor:	Mgr. Lukáš Bajer
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2021/2022

Instructions

Estimation of the most probable hostname from web traffic is a standard task in network traffic analysis. A standard approach for such tasks is based on prior probabilities from the frequencies in DNS or proxy logs. However, these data offer broader set of time or volumetric features and can be used for better predictions. The thesis will propose, implement and evaluate predictive models for such context-aware passive DNS resolution.

In particular, the student will

1) study the literature, 2) get familiar with the data (private, non-shareable data), 3) obtain public, or generate a simplified synthetic, DNS dataset, 4) implement and evaluate a context-aware passive DNS classifier of second-level domains, 5) suggest and implement additional feature extraction, 6) implement and evaluate a classifier working also on the new features, 7) implement and evaluate classifiers to the fully qualified domain names, 8) implement and evaluate classifiers for malware hostnames identification.

–

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman. 'The Elements of Statistical Learning'. Springer, 2009
[2] Torabi, Sadegh, Amine Boukhtouta, Chadi Assi, and Mourad Debbabi. 'Detecting Internet Abuse by Analyzing Passive DNS Traffic: A Survey of Implemented Systems'. IEEE Communications Surveys Tutorials 20, no. 4 (Fourthquarter 2018): 3389–3415.
[3] Bushart, Jonas, and Christian Rossow. 'Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS'. In 10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20). USENIX Association, 2020.

Electronically approved by Ing. Karel Klouda, Ph.D. on 2 February 2021 in Prague.

Master's thesis

Contextual Passive DNS

Bc. Olena Marchenko

Department of applied mathematics
Supervisor: Mgr. Lukáš Bajer, Ph.D

June 27, 2021

Acknowledgements

I would like to thank my family and friends for love and support during writing this thesis. I thank my supervisor Mgr. Lukáš Bajer, Ph.D for the time, advice and auspicious atmosphere of cooperation given to me while writing this work. I also thank the Cisco company for the opportunity to learn from the professionals.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. I further declare that I have concluded an agreement with the Czech Technical University in Prague, on the basis of which the Czech Technical University in Prague has waived its right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act. This fact shall not affect the provisions of Article 47b of the Act No. 111/1998 Coll., the Higher Education Act, as amended.

In Prague on June 27, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Marchenko, Olena. *Contextual Passive DNS*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Pasivní DNS je jeden z nejběžnějších nastrojů pro analyzu bezpečnostních incidentů z telemetrii, kde se vyskytují IP adresy. Bez aktivního dotazování DNS resolveru dává informaci o nejpravděpodobnějším doménovém jménu, které mohlo byt při přístupu na IP adresu použito. Tato práce navrhuje použití dodatečných informací obsažených v NetFlow telemetrii k extrakcí dodatečných přiznaků a použití metod strojového učení pro zlepšení přesnosti predikce nejpravděpodobnějšího doménového jména. Navržené řešení je porovnáno s řešením nejběžnejšího pDNS systému využívajícího pouze statisticky nejpravděpodobnejší hodnoty.

Klíčová slova Pasivní DNS, kybernetická bezpečnost, klasifikace, predikce hostu

Abstract

Passive DNS is one of the most common tools for analyzing security incidents from telemetry where IP addresses occur. Without active querying of the DNS resolver it gives information about the most likely domain name that could be used during accessing the IP address. This work proposes the use of additional information contained in NetFlow telemetry to extract additional

features and the use of machine learning methods to improve the accuracy of prediction of the most probable domain name. The proposed solution is compared with the solution of the most common pDNS system which uses only the statistically most probable values.

Keywords Passive DNS, cybersecurity, classification algorothms, hostame prediction.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and objectives

Determining the the most probable hostname that was requested on a particular IP address is a classic task in the field of network traffic analysis. The standard approach in this area is analysis of Passive DNS data. This approach is based on the calculation of prior probabilities based on the frequency of one or another hostname in the DNS logs. However, this data contains a much larger amount of information, rather than just the frequency tables. Consequently, this data may be invaluably useful in the context of traffic investigation and specifically in this work, in the problem of hostname prediction and second-level domain prediction.

1.2 Intrusion Detection Systems

Intrusion detection systems (IDS) are systems that collect information from various points of a protected computer system (computer network) and analyze this information to identify both attempted violations and real security violations (intrusions) [5]. Until recently, the the most common IDS structure was the model proposed by D. Denning [16].

In modern detection systems, the following basic elements are logically distinguished: an information collection subsystem, an analysis subsystem and a data presentation module [5].

- The information collection subsystem is used to collect primary information about the operation of the protected system.
- The analysis (detection) subsystem searches for attacks and intrusions into the protected system.
- The data presentation subsystem (user interface) allows the IDS user to monitor the state of the protected system.

Among the methods used in the analysis subsystem of modern IDS, two directions can be distinguished: one is aimed at detecting anomalies in the protected system and the other is aimed at finding abuses [5]. Each of these areas has its own advantages and disadvantages, therefore, in the most of the existing IDS, combined solutions based on the synthesis of the corresponding methods are used. The idea behind these methods is to recognize if the process that caused the changes in the system is an attacker's operation. Let's take a look at the the most common anomaly detection methods:

- Rules generation: During the learning process, the detection system generates a set of rules describing the normal behavior of the system. At the stage of searching for unauthorized actions, the system applies the received rules and, in case of insufficient compliance, signals an anomaly detection.

- Descriptive statistics: The training consists of collecting simple descriptive statistics of a set of indicators of the protected system into a special structure. To detect anomalies, the "distance" is calculated between two vectors of indicators - current and stored values. The state in the system is considered abnormal if the obtained distance is large enough.

- Neural networks: Training is performed with data representing the normal behavior of the system. The resulting trained neural network is then used to assess the anomalousness of the system. The output of the neural network indicates the presence of an anomaly.

The methods currently implemented in IDS are based on the general concepts of pattern recognition theory. In accordance with them, an image of the normal functioning of the information system is based on an expert assessment to detect an anomaly. This image acts as a set of values of the assessment parameters. Its change is considered a manifestation of the abnormal functioning of the system. After detecting the anomaly and assessing its degree, a conclusion is formed about the nature of the changes: whether they are a consequence of an intrusion or an acceptable deviation. An image (signature) is also used to detect abuse, but here it reflects the previously known actions of the attacker [4].

1.3 DNS

1.3.1 Definition and usage

The Domain Name System (DNS) is an Internet service that plays role of "phone book" of the Internet. It is used to translate IP addresses to human-readable representations, domain names.

In the seventies, the ARPAnet, predecessor of Internet, was a tight community of several hundred nodes. All information on nodes, in particular, necessary for mutual translation of names and addresses of ARPAnet nodes, was contained in a single HOSTS.TXT file. The HOSTS.TXT file was managed by the Stanford Research Institute (SRI) Network Information Center (NIC). ARPAnet administrators typically simply e-mailed the changes to the NIC and periodically synchronized their HOSTS.TXT files with the copy on the SRI-NIC node using FTP (file transfer protocol). The changes they submitted were added to the HOSTS.TXT file once or twice a week. However, as the ARPAnet grew, this scheme became inoperable. The file size grew in proportion to the number of ARPAnet nodes. The information flow associated with the need to update the file on the nodes grew even faster: the appearance of one new node led not only to the addition of a line to HOSTS.TXT, but also to the potential need to synchronize the data of each node with the SRI-NIC data. To cope with this problem, there was a need to create a scalable system that can be maintained in distributed manner. In 1983 the Domain Name System (DNS) was introduced and replaced HOSTS.TXT [42].

1.3.1.1 Basic concepts

Let us briefly introduce key terms and concepts within the DNS.

- Domain names - textual representation of an IP address, consists of several parts separated by dots. The structure of a domain name represents the hierarchical structure of DNS as follows: the right-the most suffix is called top-level domain, and usually formed after last dot (e.g., *.com).* The string preceding the top-level domain up to the second dot from right is called second-level domain (SLD). For simplicity and to follow standard terminology in the cybersecurity field, we will use the term SLD for the string of second-level domain **and** top-level domain together (e.g., in *calendar.google.com google.com* is a second-level domain). In case of multiple subdomains, they are named according to the order (n-th level domains) and separated with dots. In the following text we will use term "hostname" referring to a fully-qualified domain name.

- Name server is a server part in DNS client-server based systems. Each name server corresponds to some DNS namespace called zone and maintain the information about authoritative name servers that contain actual mapping records for a specified domain name.

- DNS Lookup - a process of finding out the IP address of a domain.

- DNS Zone files and records - a zone file is a text file that contains mapping information between domain names and IP addresses together

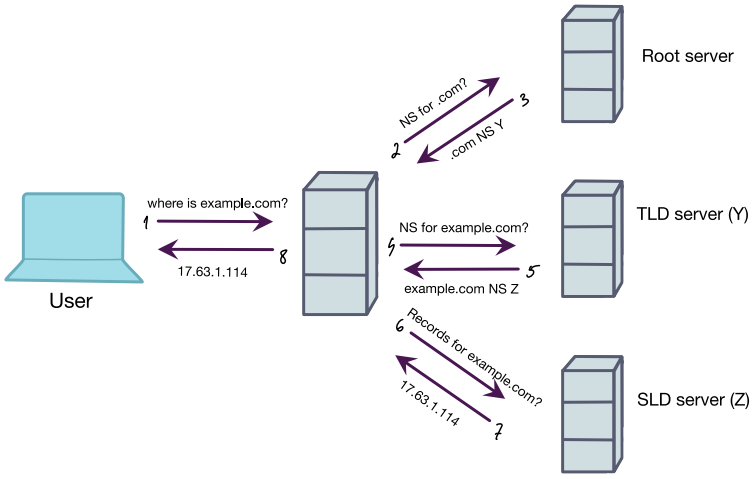


Figure 1.1: Resolution process

with additional information such as mail servers or status information about the zone file.

DNS resolution process often starts with user enters desired domain name in browser search panel, so that browser will send request to the recursive name server (resolver) [2]. If mapping record is in the resolver's cache, resolver will send it back to browser, which will use it to establish connection to website. Otherwise, the resolver will recursively send the request to authoritative name servers unless one of them provides it with required information. This process is visualized in Figure 1.1.

1.3.2 DNS from the security point of view

Despite all the benefits and improvements associated with the implementation of DNS, this system is a well-known attack vector for cybercriminals. DNS services have been abused in different ways to perform various attacks [8], [19]. An attacker can utilize a set of domains and IP addresses to perform sophisticated attacks, like spamming campaigns, phishing, and Distributed Denial of Service (DDoS) attacks. These attacks can result in a wide range of outcomes: receiving spam, stealing card credentials, service shut-downs, and privacy issues, etc.

for the purpose of attacks prevention, or minimizing their negative consequences, it is necessary to isolate malicious or compromised domains and hosts from the rest of the Internet (deny lists serve for this purpose). The

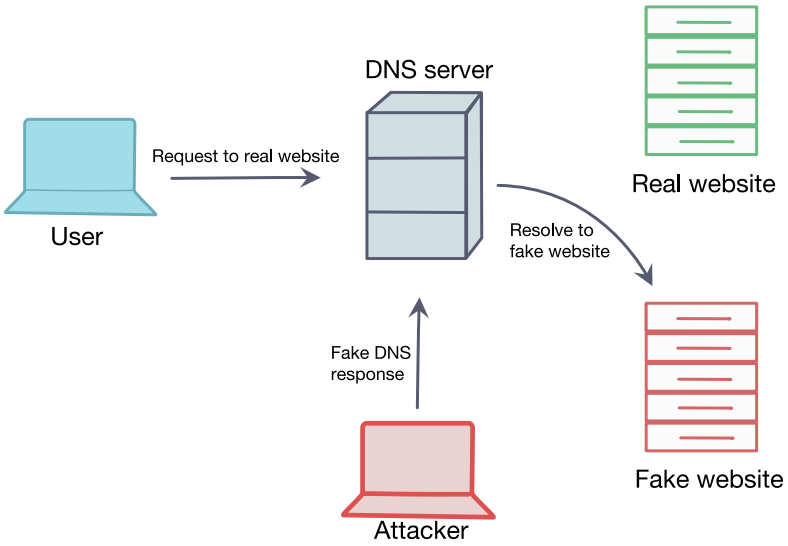


Figure 1.2: Spoofing

frequent question of modern studies in the field of cybersecurity is trying to give a response: how to determine malicious hosts and IP addresses in a huge traffic stream.

DNS activity is present in almost all communication sessions, regardless of the protocol. Thus, DNS logging data is a valuable source of information for security professionals, allowing them to detect anomalies or obtain additional data about the system under investigation [51].

These attacks are categorized into protocol attacks and server attacks. Protocol attacks are characterized by the usage of vulnerabilities in the implementation of DNS, while the server attack is associated with disabling a device that provides certain services, obtaining confidential information or an attempt to take control of the operating system of the device.

In the next paragraphs, the the most known attacks with utilizing of DNS will be described.

DNS spoofing or cache poisoning. DNS spoofing is a type of attack in which an attacker intercepts DNS request and returns the address that leads to its own server instead of the real address. Malicious actors can use DNS spoofing to launch a man-in-the-middle attack and direct the victim to a fake website that looks like the real one, or they can simply relay the traffic to the real website and stealthily steal the data. DNS cache poisoning is a method of DNS spoofing when the user's system logs the malicious IP addresses in local memory cache [48]. As a consequence, DNS remembers fraudulent website

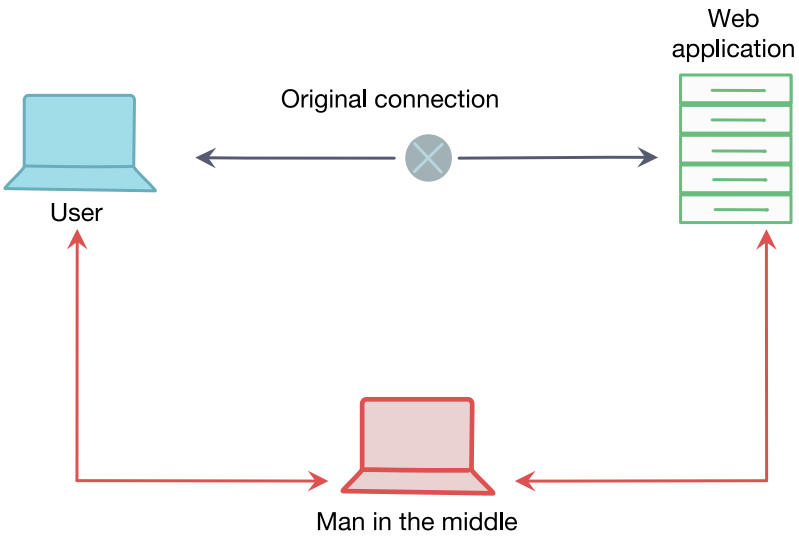


Figure 1.3: Man in the middle

specifically for a given user 1.2.

There are known methods for spoofing and cache poisoning.

- Man in the middle DNS duping: This form of cyberattack uses techniques to intercept data to infiltrate an existing connection or communication process [22]. An attacker can be a passive listener in conversation, stealthily stealing information, or an active participant, altering the content of messages or impersonating the person or system user thinks he is talking to. An attacker goes between the user's web browser and the DNS server to infect both of them. The tool is used to infect the cache on the user's local device and infect the server on the DNS server at the same time. As a result, a redirection occurs to a malicious site hosted on the attacker's own local server 1.3.

- DNS server hijack: The attacker directly reconfigures the server to direct all requesting users to the malicious website. Once a fraudulent DNS record is entered into a DNS server, any IP request for a fake domain will result in a fake site [6].

- DNS cache poisoning via spam: The DNS cache poisoning code is often found in URLs sent via spam messages. These emails try to intimidate users into clicking the specified URL, which infects their computer. Banners and images, both in emails and on untrustworthy websites, can also direct users to this code. Once poisoned, a computer will redirect the

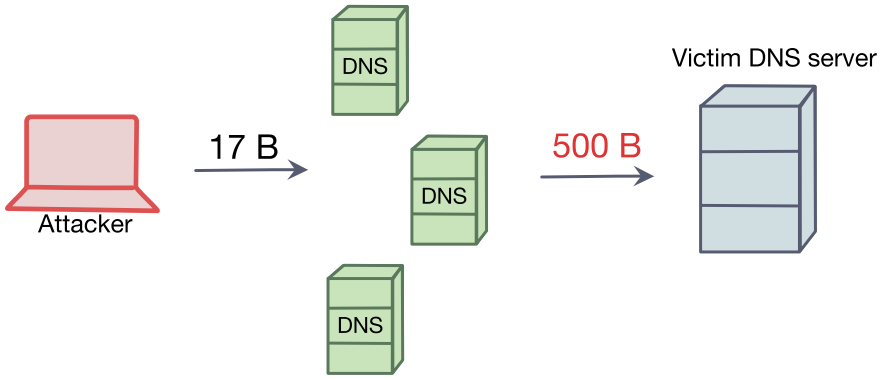


Figure 1.4: Amplification attack

user to fake websites that will be spoofed to make them look real. This is where the real threats emerge for user devices.

Denial of service (DoS) attack or distributed denial of service (DDoS) DoS (Denial of Service) attack is the bombardment of the victim's servers with separate packets with a forged return address. Failure, in this case, is the result of overflow of the band leased by the client or increased resource consumption on the attacked system. Attackers mask the return address to exclude the possibility of blocking by IP. If the attack is distributed and is carried out simultaneously from a large number of computers, we speak of a DDoS attack [21]. One of the the most popular types of DDoS attacks utilizing DNS is DNS amplification attack.

Amplification attack The essence of the amplification is that the attacker sends a (usually short) request to a vulnerable DNS server, which responds to the request with a much larger packet 1.4. In case of usage of the address of the victim's computer (IP spoofing) as the initial IP address when sending a request, the vulnerable DNS server will send a large number of unnecessary packets to the victim's computer until it completely paralyzes its work [27].

Other types of DNS abuse

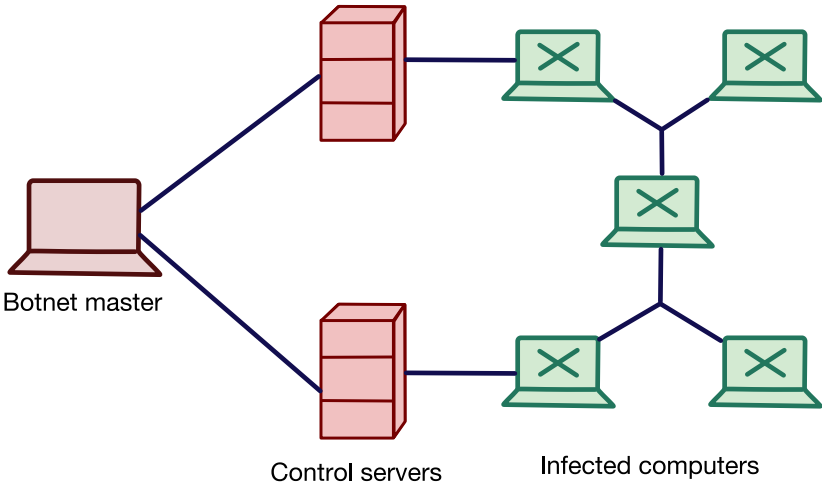


Figure 1.5: Botnet

- Cybersquatting. The essence of cybersquatting is that people involved in it register domain names containing the names of well-known companies, geographical names, organizations, goods and other recognizable things, and then resell them to companies interested in this domain name.

- Typosquatting. In fact, this method is a registration of the domain names that are similar in spelling to the addresses of popular sites. This is done with the expectation that the victim will incorrectly type the desired address and follow phishing and fraudulent links [36].

- Botnets

A botnet is a network of computers infected with malware [17]. Here, a cybercriminal acting as a botmaster uses viruses to compromise the security of multiple computers and connect them to the network for malicious purposes (Figure 1.5). Every computer on the network acts as a "bot" and is controlled by a fraudster to transmit malware, spam, or malicious content to launch an attack. In the structure of the "client-server" bot-system, a basic network is created in which one server acts as a botmaster. Botmaster controls the transfer of information from each client to set commands and control over client devices. Botnets are the mostly used for DDoS attacks, spam and traffic monitoring, stealing credentials

- Fast-flux domains

Fast-Flux networks are networks of compromised computer systems with public DNS records that are constantly changing, in some cases every 3 minutes. The ever-changing architecture makes it much more difficult to track and shut down criminal activity. Fast flux DNS is a technique that an attacker can use to prevent the IP address of their computer from being identified. By abusing DNS technology, a criminal can create a botnet with nodes, connect through them, and change them faster than law enforcement officials can trace [38].

Fast flux DNS uses a load balancing method built into the domain name system. DNS allows an administrator to register an arbitrary number of IP addresses with a single hostname. Alternative addresses are legitimately used to distribute internet traffic across multiple servers. Typically, the IP addresses associated with a host do not change very often, or even may never change.

However, criminals have found that they can hide key servers using the time to live (TTL) of the DNS resource record associated with the IP address and change them extremely quickly. Because system abuse requires the cooperation of a domain name registrar, the most Fast flux DNS botnets are believed to originate in developing countries or other countries without cybercrime laws.

From all of the above, we can conclude that DNS is a wide field of activity for different kinds of attackers, and therefore the development of methods for tracking their activities is a high priority task for ensuring security on the Internet.

1.3.3 Passive DNS in a broader sense

It is known that DNS zone records are not stale and change frequently. Furthermore, before the introduction of Passive DNS technology by Florian Weimer in 2004 [55], there was no possibility to track these changes. Users were unable to check DNS lookup history. This fact prevented researchers from investigating and analyzing compromised domains according to resolution data from the past.

In 2004, Florian Weimer proposed a logging method called Passive DNS. This concept allows restoring the history of changes in DNS data with the ability to index and search. It provides a wide range of opportunities for cybersecurity research and makes it easier to identify traffic anomalies, malicious domains and infected machines. It can provide access to the following data:

- Domain name
- IP address of the requested domain name
- Date and time of response
- Response type

- etc.

Data for Passive DNS is collected from recursive DNS servers by built-in modules (sensors). Despite the fact that this data may not represent exact DNS structure, using Passive DNS allows building relationships between domain names and IP addresses, consequently building maps of studied systems and track changes in such a map from the first detection to the current moment [25].

Scientific works with Passive DNS applications often use machine learning approach. For example, such an approach was used in papers of Bilge et al.[8] where the Exposure system was developed. A lot of attention was paid to the feature engineering part. The system attempts to identify malicious domains using data from Passive DNS dataset and thus classify domains using machine learning algorithms. Khalil et al. [28] suggested building host-domain graphs (also called user query behavior) to investigate more global patterns in comparison to local features in Exposure and to utilize graph algorithms to compute the malicious scores of domains.

1.4 NetFlow analysis

The essence of this thesis lays in the analysis of network traffic. By network traffic we understand data obtained with NetFlow, a proprietary session sampling protocol designed by Cisco Systems that allows extracting information about packet flows (a bunch of related packets, which have a common source and destination IP, IP protocols, etc) [30]. NetFlow extract critical information from packet flows such as source, destination, byte counts, and other metadata. Analysis of NetFlow is a powerful tool in the traffic investigation, network monitoring, troubleshooting, and anomaly detection [50].

NetFlow tools are divided into two parts, similar to the standard IDS (Intrusion Detection Systems) deployment architecture:

1. One or more flow generation devices (sensors) that monitor network traffic and make corresponding NetFlow records in real time;

2. Central component (collector and analyzer) collects stream records, stores them, and provides analysts with a set of tools for interaction; analysis tools usually work either through the command line, through a web interface, or both [30].

When network security analysts examine the IDS warning, they see something potentially negative about just one packet in one network session. Flow records for the source and destination hosts mentioned in this IDS alert give analysts the context they need. What other hosts did the attacker interact with? Did the victim have similar connections with other hosts after the IDS

was triggered (this would indicate the spread of the attack)? These questions can be answered using flow records and query tools.

So the task of this thesis can be formulated as follows:

Problem: How can we determine the destination of the user request (in terms of DNS) without exact knowledge of the fully qualified domain name (hostname)?

Goal: We want to predict which hostname did user look for and possibly to mark this communication as dangerous.

Solution: Apply machine learning technics to Passive DNS context data to predict the most probable hostnames for each user.

Also: second-level domains are also the area of interest since they may indicate an organization or geographical affiliation, which is useful in the context of security goals. As an illustration, if the URL is proven to be malicious there is a non-zero probability that all the hostnames with given SLD may be also malicious. Therefore, in this work we will concentrate on the SLD prediction as well as on the hostnames prediction.

Chapter 2

Methods and theoretical background

This section focuses on the theory behind the approaches used in this work. We define a typical task in the field of machine learning, consider the basic classification of these tasks. Also, we will highlight the main strategies for training models. We will describe the main types of data that analysts often have to work with. An important aspect is that when implementing a particular model, the expert needs to assess how well the model copes with the given task. Different metrics can be used to quantitatively evaluate the model performance. The metrics used in the analysis of the implemented models will be defined and briefly described. Finally, all the models used in the prediction will be described.

2.1 Supervised and unsupervised learning

Classically, machine learning problems can be classified into two classes: supervised learning and unsupervised learning [44].

Supervised learning This type of learning is used for determining the object belonging to some class or correspondence to some value from the training data. Training data consists of pairs of input objects (typically feature vectors) and the desired output (labels). There may be some relationship between inputs and target outputs, but its nature is unknown. Only a finite set of precedents is known, the input-output pairs called training set. Based on these data, it is required to restore the dependence (to build a model of the relationship suitable for forecasting), that is, to build an algorithm capable of giving a sufficiently accurate answer for any newly introduced object from the test set.

Unsupervised learning The main difference between supervised and unsupervised learning is that, in the case of unsupervised learning, we do not have target values for each feature vector. We have a dataset of vectors (samples) from which we want to extract some knowledge. A typical approach is called clustering, the process during which according to similarities (which is often represented as distance between data points in feature space), we assign input vector to some class.

Semi-supervised learning Semi-supervised learning can be understood as a "mixture" of supervised and unsupervised approaches, when we have some small amount of labeled data and a large amount of unlabeled data.

2.2 Input data

Feature space A feature is a mapping $f : X \to D_f$, where D_f is the domain of the attribute. If attributes $f_1, \ldots, f_n$ are given, then the vector $\mathbf{f_x} = (f_1(x), \ldots, f_n(x))$ is called a feature description of an object $x \in X$. Feature descriptions can be identified with the objects themselves $\mathbf{f_x} \equiv x$. In this case, the set $X = D_{f_1} \times \ldots \times D_{f_n}$ is called a feature space [52].

Depending on the set D_f, features are divided into the following types:

1. binary feature: $D_f = \{0, 1\}$
2. nominal or categorical feature: D_f is a finite set without any particular ordering of its members
3. ordinal attribute: D_f is a finite ordered set;
4. quantitative feature:D_f is a subset of real numbers R^n[52].

The input data can have various structure and, depending on the representation, the data is processed using different feature engineering or feature extraction technics. After extraction, a numerical vector can be submitted to the input of the model, and only then we will be able to work with feature vector. the most common features representations are:

- Attribute description is the the most common case. Each object is described by a vector corresponding to its characteristics, or features.
- Distance matrix between objects. Each object is described by the distance to all other objects of the training set. Few methods work with this type of input, in particular, the nearest neighbors method.
- A time series or signal is a sequence of measurements over time. Each dimension can be represented by a number, a vector, and, in the general case, by an indicative description of the object under study at a given moment in time.

2.3 Classification and regression

Classification and regression are representatives of supervised learning approach. A typical problem in machine learning looks as follows: having a vector of features, we want to predict, depending on the type of problem, either the class (e.g., which is in the image, an apple or a pear), to which the object described by this vector belongs, or to predict a value from a set of real numbers corresponding to a given input (e.g., stock price tomorrow).

2.3.1 Classification

Classification is the section of machine learning devoted to solving the following problem: a finite set of objects is given, for which it is known to which classes they belong. The class of the remaining objects is unknown. It is required to construct an algorithm capable of classifying an arbitrary object from the original set. To classify an object means to indicate the value (or name of the class) to which this object belongs [52].

Let $\mathcal{X}$ be the set of descriptions of objects, $\mathcal{Y}$ be a finite set of numbers (names, labels) of classes. There is an unknown target dependence or the mapping $f : \mathcal{X} \mapsto \mathcal{Y}$, the values of which are known only on the objects of the final training set $X = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. It is required to build a mapping $a : \mathcal{X} \mapsto \mathcal{Y}$, capable of classifying an arbitrary object $x \in X$.

Class types:

1. Two-class (binary) classification. The the most technically simple case, which serves as the basis for solving more complex problems.

2. Multi-class classification. We talk about multi-class classification when we have three or more classes. When the number of classes reaches many thousands, the task of classification becomes much more difficult.

3. Non-overlapping classes.

4. Overlapping classes. An object can belong to several classes at the same time.

5. Fuzzy classes. It is required to determine the degree of belonging of an object to each of the classes, usually, it is a real number from 0 to 1 [52].

2.3.2 Regression

The regression task is a forecast based on a sample of objects with different characteristics. The output should be a real number, for example, the price of an apartment, the cost of a security after six months, the expected income of the store for the next month.

We can define regression problem as having a set of objects descriptions $\mathcal{X}$, but now for regression task $\mathcal{Y}$ is not a finite set of nominal variables but a numeric value from R^n. There is still unknown dependence $f : \mathcal{X} \mapsto \mathcal{Y}$, but now it is required to build a mapping $b : \mathcal{X} \mapsto \mathcal{Y}$, capable of predicting the value (real number) for an arbitrary input sample.

2.4 Model performance measurement

Each machine learning problem poses the question of evaluating the results of models.

Without criteria, it is impossible either to assess the "success" of the model, nor to compare two different algorithms with each other. That is why it is important to take into account the correct choice of metrics for the task at hand, although the many existing metrics can be confusing and, ultimately, lead to a sub-optimal solution.

2.4.1 Binary classification

In case of binary classification, the data is divided into two classes. Their labels are usually designated as 1 and 0. The metrics we are considering are based on the use of the following outcomes: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). False positive and false negative outcomes are also called errors of Type I and Type II, respectively. Let us explain the nature of these errors in the example.

Let's take a look at the task of detecting malicious Internet traffic. Traditionally, if the packet is indeed malicious, then that would be a positive class. If it is safe, the class is negative. The result of the model's work can be a determination: whether the package should be "suspected" (then the result is true) or not (then the result is false). Let some set of markers of a certain behavior be characteristic of an attack. If our model correctly identified and assigned a positive class, then this is a True Positive outcome, but if the model puts a negative class label, then this is a False Negative outcome. If the traffic turns out to be safe, and if the model classifies the feature vector as a positive class, then we are talking about a False Positive outcome (the model "said" that there is an attack, but in fact the traffic is safe), and vice versa, if the model recognizes the record as a negative class, then this is a True Negative outcome.

Accuracy One of the simplest and therefore common metric is accuracy. It shows the number of correctly assigned class labels (true positive and true negative) over the total amount of data and is calculated as follows [41]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Nevertheless, accuracy is criticized due to the fact that it can be misleading in case of imbalanced classes [3].

Let's go back to the internet traffic example. If a model's accuracy is 80%, we can say that, on average, out of 100 packets the model will correctly identify a malicious packet only in 80 cases, while another 20 will be either false negative or false positive.

It is worth paying attention to the fact that, in some tasks, it is necessary to identify all infected objects and researcher can even neglect false positive outcomes, since they can be eliminated at the next stages of the study, consequently, it is necessary to add one more metric, which could estimate the required priority.

Precision This metric shows the number of truly positive outcomes from the entire set of positive labels and is calculated using the following formula [41]:

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

The importance of this metric is determined by how high the "cost" of a false positive result is for the problem under consideration. If, for example, the cost of further checking for an attack is high and we simply cannot check all false positives, then it is worth maximizing this metric, because, if Precision = 50% out of 100 positively determined infected packets, only 50 of them will be really malicious.

Recall (true positive rate) This metric measures the number of true positives among all class labels that were determined to be "positive" and is calculated using the following formula [15]:

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

It is necessary to pay special attention to this metric when the error of non-recognition of the positive class is high.

F1-Score If Precision and Recall are equally significant, we can use their harmonic mean to obtain an estimate of the results [18]:

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{2.4}$$

False Positive Rate

$$FPR = \frac{FP}{FP + TN} \tag{2.5}$$

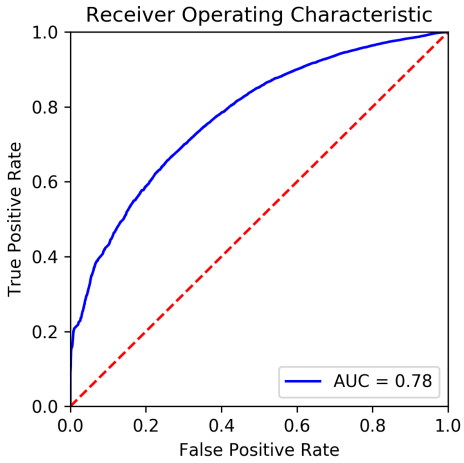


Figure 2.1: ROC curve example

It's the probability that a false alarm will be raised, the case when a positive result will be returned when the actual value is negative.

ROC In addition to listed metrics, there are also graphical methods that can assess the quality of a classification.

ROC (receiver operating characteristic) is a graph showing the dependence of correctly classified objects of a positive class on incorrectly classified objects of a negative class for different discrimination thresholds of a classification. In other words, it is a plot of True Positive Rate (TPR, Recall) versus False Positive Rate (Figure 2.1).

The ideal classifier curve would pass through the top left point (TPR = 1, a FPR = 0). Curves that lie below TPR=FPR line indicate that classifier predictions are worse than flipping a coin, or that it is necessary to reverse class labels.

Using the ROC curve helps to compare the models, as well as their parameters to find the the most optimal (in terms of TPR and FPR) combination. ROC curve, on the other hand, has a big disadvantage: it is sensitive to imbalanced data. Methods aimed to choose optimal operating point is an attempt to cope with this disadvantage [20].

AUC (Area Under Curve) As a numerical estimate of the ROC curve, it is a common practice to take the area under this curve (AUC value in Fig. 2.1). AUC also has a statistical interpretation: it shows the probability that a randomly selected instance of a negative class will be less likely to be recognized as a positive class than a randomly selected positive class.

2.4.2 Multi-class classification

All the metrics discussed above relate only to a binary problem, but, often, there are more than two classes. This makes it necessary to generalize the considered metrics. One possible way is to calculate the average metric for all classes [18]. In this approach, each class i is iteratively taken as the positive class, and all the others are taken as the negative (one versus all approach).

In this case, the formulas for the metrics will look like this:

$$Accuracy_{avg} = \sum_{i=1}^{k} Accuracy_i \tag{2.6}$$

$$Precision_{avg} = \sum_{i=1}^{k} Precision_i \tag{2.7}$$

$$Recall_{avg} = \sum_{i=1}^{k} Recall_i \tag{2.8}$$

$$F1_{1avg} = \sum_{i=1}^{k} F1_i \tag{2.9}$$

where k is number of classes. This approach is called macro-averaging, it will compute the metric independently for each class and then take the average (treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average metric.

2.5 Classifiers used in this work

2.5.1 Logistic Regression

Logistic regression is a method for constructing a linear classifier that allows estimating the posterior probabilities of objects belonging to classes [29]. Logistic regression is used to predict the likelihood of a certain class based on the feature set. For this, a dependent variable y is introduced, $y \in \{0, 1\}$ and a set of independent variables $X = \{x_1, ...x_n\}$, based on the values of which we are going to calculate the probability of accepting one or another value of the dependent variable y. Having training set $X = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ for the two-class classification we have $Y = \{-1, 1\}$. In the logistic regression the classification algorithm calculates its prediction as:

$$a(x) = sign\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right) \tag{2.10}$$

where w_j is a j-th feature weight, w_0 is a decision threshold. The task of training a linear classifier is to adjust the weight vector w from the samples

$\{x_1, \ldots, x_m\} \in X^n$. To do so in logistic regression the problem of empirical risk minimization is solved with a special type of loss function:

$$Q(w) = \sum_{i=1}^{n} \ln(1 + \exp(-y_i (x_i \cdot w))) \to \min_w \tag{2.11}$$

After appropriate w is found, it is possible not only to compute classification $sign(x \cdot w - \theta)$ for any object x, where θ is a threshold that we can use to adjust the tpr/fpr according to the requirements, but also to estimate posterior probabilities its belonging to classes:

$$P(y|x_i) = \sigma(y(x_i \cdot w)) \tag{2.12}$$

where $y \in Y$ and $\sigma(z) = \frac{1}{1+\exp(-z)}$ is sigmoid function [53].

Multinomial logistic regression This method generalizes binary logistic regression to the multiclass problem, i.e. with more than two target outcomes. To extend binary model to multinomial model, one can imagine, for K possible classes, we run K-1 independent binary logistic regression models, in which one class K is fixed as pivot and then the other K-1 classes are separately regressed against chosen pivot outcome [**?**]. For instance, if outcome K_p s chosen as the pivot, we will calculate for each other class $K \in \{0, \ldots, K-1\}$:

$\ln\left(\frac{P(Y_i=K)}{P(Y_i=K_p)}\right) = \beta_i \cdot X_i$,where X_i, β_i are set of regression coefficients (each set β_i corresponds to each possible outcome) and feature vector, respectively.After exponentiating both sides and from the assumption that probabilities sum up to one [1], we get:

$$P(Y_i = K_p) = \frac{\exp(\beta_K \cdot X_i)}{1 + \sum_{k=1} \exp(\beta_K \cdot X_i)} \tag{2.13}$$

.

2.5.2 K-Nearest Neighbors Classifier

The k-nearest neighbors method is an algorithm for feature classification or regression. In the case of using the method for classification, the object is assigned to the class that is the most common among the k neighbors of the given element, the classes of which are already known, so we assume that close objects in feature space are similar [13]. The algorithm can be applied to samples with a large number of attributes (multidimensional). Before the application, the distance function need to be chosen: Euclidean metric is a classical choice.

KNN is a distance-based classifier, meaning that we assume that the smaller the distance between two points, the more similar they are. This is a supervised learning algorithm, i.e. it each data point must have a corresponding label

Fit step: KNN stores all the training data points and their labels. No distances are calculated at this stage.

Predict step: KNN takes a point for which we want to predict its class, and computes the distances between that point and every other one in the training set. It then finds the k closest points or neighbors (according to selected metrics), and examines each neighbor's label. Then prevalent class across the neighbors is assigned to the data point.

Changing the value for k and distance function can affect the performance of the model, so the question is what is the the best value to use for k.

Choice of the number of neighbors *k* For $k = 1$, the nearest neighbor algorithm is unstable to outliers: it incorrectly classifies not only on the outlier objects themselves, but also on objects of other classes nearest to them. For $k = m$, where m is a number of points in the dataset, on the contrary, the algorithm is overly stable and degenerates into a constant. Thus, extreme values of k are undesirable. In practice, the optimal value of the parameter k is determined by the cross-validation criterion, the most often by the means of decreasing number of neighbors one by one (leave-one-out cross-validation) [54].

Choice of the metric The choice of the the best metric is depending on the essence of data and typically is chosen experimentally [12]. In this work following metrics were used:

- Manhattan distance [9] $d(x, y) = \sum_i |x_i - y_i|$
- Euclidean distance [34] $d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$
- Chebyshev distance also known as chessboard distance $d(x, y) = \max_{i=1,\ldots,n} |x_i - y_i|$

All of these metrics are special cases of Minkowski distance [47] with varying exponent c.

$$d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^c)^{\frac{1}{c}} \tag{2.14}$$

if $c = 1$, we obtain Manhattan metric
if $c = 2$, we obtain Euclidean metric
if $c = \infty$, we obtain Chebyshev metric.

- Canberra distance [26], a weighted version of Manhattan distance $d(x, y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$

Very large datasets The method of the nearest neighbors is based on the explicit storage of all training objects. Large datasets create technical problems: it is necessary not only to store a large amount of data, but also to be able to quickly find among them the nearest neighbors of an arbitrary data point [54].

The problem is solved in two ways:

- Dataset reduction by removing non-informative objects;
- Efficient data structures are used to quickly find the nearest neighbors (inter alia, KD trees) [56].

2.5.3 Multilayer Perceptron Classifier

2.5.3.1 Definition

Perceptron is an attempt to simulate the biological neuron behavior suggested by Frank Rosenblatt in 1958 [43].

A multilayer perceptron (MLP) is a class of feedforward artificial neural networks consisting of at least three layers: input, hidden, and output. Except for the input neurons, all neurons use a non-linear activation function. MLP training uses supervised learning and backpropagation algorithm based on the chain rule. There are various options to choose activation functions. Historically, sigmoidal ones were used: logistic or hyperbolic tangent [24]. Nowadays, ReLu and its derivatives are the common choice.

MLP has shown the ability to find approximate solutions to extremely complex problems. Since classification can be considered as a special case of regression, when the output variable is categorical, classifiers can be built based on MLP.

Multilayer perceptron classifier learns the approximation function $f : R^n \rightarrow R^m$, where m, n are input and output vectors dimensions. Given input feature vector $\mathbf{x} = x_1, ..., x_n$ and output labels $y = y_1, ..., y_m$, by optimizing its weights, classifier learns approximation of the function that minimizes preselected loss $\mathcal{L}$.

Each connection has its own weights $\mathbf{w} = \{w_1, ..., w_p\}$ and each neuron in hidden layer takes as input outputs from the previous layer and transforms their weighted sum with activation function. So output of i-th neuron will be:

$$\boldsymbol{x}^i = \sigma \cdot \left(W^i \boldsymbol{x}^{i-1} + \boldsymbol{b}^i \right) \tag{2.15}$$

where σ is devoted to activation function and b is a bias of i-th layer.

According to the Universal Approximation Theorem, proved by George Cybenko in 1989 [14], claims that an artificial neural network of feedforward (in which connections do not form cycles) with one hidden layer can approximate **any** continuous function of many variables with any accuracy.

The conditions are:

1. a sufficient number of neurons in the hidden layer,
2. continuous bounded activation function (which was later generalized to a larger class of functions [49])

2.5.3.2 Activation function

The activation function determines the output value of the neuron depending on the result of the weighted sum of the inputs and the threshold value. Let's take neuron x^i from equation 2.15. Expression in parentheses ω can be in range $\omega \in \{-\infty, +\infty\}$. How do we decide whether a neuron should be activated? We are considering an activation pattern since we can draw an analogy with biology.

Step function The simplest approach is to use a threshold to decide whether to activate the neuron. If the ω value is greater than a certain threshold value t, the neuron is considered activated [46]. Otherwise, we say that the neuron is inactive.

$$s(\omega) = \begin{cases} 1, & \text{if } \omega > t \\ 0, & \text{if } \omega \leq t \end{cases} \tag{2.16}$$

Despite the simplicity, this function is not applicable in most of the cases since it can not be used with optimization methods demanding calculation of the gradient (and they are majority), thus this means that we can not learn such network effectively.

Linear function A linear function $l(\omega) = A\omega$ is proportional to the input ω [46].

This choice of the activation function allows one to obtain a range of values, and not just a binary response.

Sigmoid

$$\sigma(\omega) = \frac{1}{1 + \exp(-\omega)} \tag{2.17}$$

is called sigmoid function [46]. It is a popular choice of the activation because of the following advantages.

First, the sigmoid is nonlinear in nature, and the combination of such functions also produces a nonlinear function (possibly more complex), so, we can build more complex functions by stacking layers. This can't be said concerning linear activation since combination of linear functions is still linear function, consequently, no matter how many layers we have, the output is linearly dependent on input.

The sigmoid is a smooth function, thus it has a well-defined gradient.

Sigmoid compresses the $(-\infty, \infty)$ range to the $(0, 1)$ range, i.e. forces data to have a "reasonable scale", in contrast to the linear function.

Mentioned earlier universal approximation theorem was proven for sigmoid function.

The sigmoid is still one of the the most frequent activation functions in neural networks, however, it has drawbacks that are worth paying attention to.

The sigmoid is bounded function, thus its derivative is close to zero since:

$$\lim_{x \to \infty} \sigma(x) = 1$$

$$\lim_{x \to -\infty} \sigma(x) = 0$$

This, in turn, leads to vanishing gradient problem. Since the gradients are multiplied by the sigmoid derivative at each layer, the gradients for layers farther from the input layer become very close to zero. In other words, only a few last layers of the neural network are actually learn.

Hyperbolic tangent Another commonly used activation function is the hyperbolic tangent.

The hyperbolic tangent is very similar to the sigmoid. Indeed, it is an adjusted sigmoid function [46].

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.18}$$

Therefore, this function has the same characteristics as the sigmoid discussed earlier. Its nature is non-linear, it works well for a combination of layers, and the range of values of the function is $(-1, 1)$. However, it is worth noting that the gradient of the tangential function is greater than that of the sigmoid (the derivative is steeper). Like the sigmoid, the hyperbolic tangent still prone to vanishing gradient problem.

ReLu

$$\text{relu}(x) = \max(0, x) \tag{2.19}$$

ReLu returns x if x is positive, and 0 otherwise [46]. The ReLu range is $[0, \infty)$, that is, activation can “explode”. Due to the fact that ReLu output is constant for negative values of x, the gradient on this part is 0. Because the gradient is zero, the weights will not be adjusted during gradient descent. This means that neurons in this state will not react to changes in the error/input and will always output 0. This problem is known as Dying ReLu problem. However, modifications of ReLu that can help overcome this problem exist. For example, it makes sense to replace the function for $x < 0$ with a small slope (Leaky ReLu [46]). For instance, the expression for a linear function is given by $y = 0.01x$ for $x < 0$, the line deviates slightly from 0, thus, we get

a non-zero gradient and the neurons will preserve the ability to update their weights.

2.5.3.3 Learning

By the learning process we understand adjusting weights of neurons connections in order to minimize loss function. The the most popular approach is called gradient descent. It is based on the idea that the sign of the derivative shows whether the original function increases (the derivative is positive) or decreases (negative). If the derivative exists and is equal to zero, then we are at the extremum (saddle point, minimum or maximum). Gradient descent is based on this property of the derivative. The gradient of a function of several variables is the vector of its partial derivatives with respect to each of these variables:

$$\nabla L(w_1, w_2, \ldots, w_n) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_n}\right] \tag{2.20}$$

In this case, L is the loss function of our machine learning model, and $\{w_1, ...w_n\}$ are the internal parameters (weights) of the model that should change during its training. The loss function measures the "quality" of the model; it can be very different depending on the task. For example, for a model predicting the value of one variable, this might be the square of the difference between the true value of that variable and the predicted value.

We need to find the global minimum of the loss function in the space of weights - that is, such values of the weights for which the model will be optimal. The gradient descent method at each step calculates the gradient at a given point in the n-dimensional space of the weights and moves to the next point in the opposite direction of the gradient vector.

Since the majority of the modern neural networks consist of several layers, and the input of each layer of the network, except for the first one, is the output of the previous layer, feedforward network is a complex function in which activation function of neurons of each layer is applied to the output of the activation function of the previous layer.

To train a neural network, we need to calculate the gradient of its loss function, that is, a set of derivatives of this function over all the weights of the network. As it is shown in Figure 2.2, the chain rule allows to represent the derivative with respect to each weight in the form of a product of simpler elements (derivatives of the loss function and neuron activation functions by their parameters), which we can already calculate. This method of calculating the gradient is called backpropagation.

2.5.4 Decision Tree Classifier

Decision trees are one of the the most powerful predictive analytics tools for solving classification and regression problems.

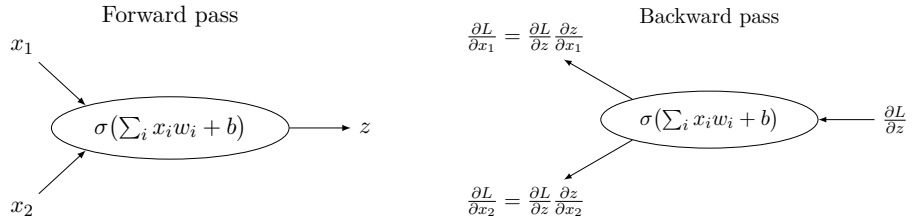


Figure 2.2: Backpropagation. Forward pass and backward pass.

They are hierarchical tree structures consisting of decision rules like "If ... then". The rules are automatically generated during the learning process on the training set and, since they are formulated almost in natural language, decision trees are more verbalized and interpretable than, i.e., neural networks [45]. Each feature vector must correspond to a target value since decision trees are supervised learning models. Moreover, if the target variable is discrete (class label), then the model is called a classification tree, and if it is continuous, then a regression tree. Actually, the decision tree itself is a method of representing decision rules in a hierarchical structure consisting of two types of elements - nodes and leaves. Decision rules are found in the nodes and the examples are checked for compliance with this rule by any attribute of the training set.

In the simplest case, as a result of the check, the set of examples found in the node is divided into two subsets, one of which contains examples that satisfy the rule, and the other does not.

Then the rule is applied to each subset again and the procedure is recursively repeated until some condition for stopping the algorithm is reached. As a result, in the last node there is no check and splits, and it is declared a leaf. The leaf determines the solution for each example that falls into it. For the classification tree, this is the class associated with the node, and for the regression tree, the interval of the target variable corresponding to the leaf. Thus, unlike a node, a leaf does not contain a rule, but a subset of objects that satisfy all the rules of a branch that ends with a given leaf.

Obviously, to get into a leaf, an example must satisfy all the rules that lie on the way to that leaf. Since the path in the tree to each leaf is unique, then each example can fall into only one leaf, which ensures the uniqueness of the solution.

Algorithms for constructing decision trees are classified as so-called greedy algorithms. Greedy algorithms are those that assume that locally optimal solutions at each step (partitions at nodes) lead to an optimal final solution. In the case of decision trees, this means that if an attribute has been selected once and partitioned into subsets, then the algorithm cannot go back and select another attribute that would give a better final partition. Therefore, at the stage of construction, it cannot be said whether the selected attribute will

ultimately provide the optimal partitioning.

the most popular algorithms for learning decision trees are based on the divide and conquer principle. Algorithmically, this principle is implemented as follows. Let a training set X be given, containing n examples, for each of which a class label $C_i, i \in \{1, \ldots, k\}$ is given, and m attributes $A_j, j \in \{1, \ldots, m\}$ which are supposed to determine the belonging of an object to a particular class. Then three cases are possible:

1. All examples of the X set have the same class label C_i (i.e. all training examples refer to only one class). Obviously, learning in this case does not make sense.

2. Set Xcontain no examples at all, i.e. is an empty set. In this case, a leaf will also be created for it (it is pointless to apply the rule to create a node to an empty set), the class of which will be selected from another set (for example, the class that is the most often found in the parent set).

3. Set X contains training examples for more C_i classes. In this case, it is required to split the X set into subsets associated with the classes. For this, one of the attributes A_j of the X set is selected, which contains two or more unique values $(a_1, a_2, ..., a_p)$, where p is the number of unique values for the feature. Then the X set is split into p subsets $(X_1, X_2, ..., X_p)$, each of which includes examples, containing the corresponding attribute value. Then the next attribute is selected and the partitioning is repeated. This procedure will recursively repeat until all the examples in the resulting subsets are of the same class.

It is worth noting that this is not the only way to divide the training dataset, random division is often used in which subsets of the train set can intersect, in particular, this approach is implemented in the sklearn library [40], the decision tree from which was used in this work.

The procedure described above is the basis of many modern algorithms for constructing decision trees.

- ID3 (Iterative Dichotomizer 3) is the algorithm that allows working only with a discrete target variable, therefore, decision trees built using this algorithm are classifying. The number of descendants in a tree node is not limited. Can't work with missing data [7].

- C4.5 is an improved version of the ID3 algorithm, which adds the ability to work with missing attribute values [31].

- CART (Classification and Regression Tree) is a decision tree learning algorithm that allows using both discrete and continuous target variables, that is, to solve both classification problems and regression. The algorithm builds trees that have only two children at each node [11].

2.5.5 Ensemble methods

The next two classifiers are representatives of ensemble methods, a class of algorithms that are built according to the idea that multiple weak classifiers can give a good prediction. In this work, we used these methods with decision trees as weak learners.

2.5.5.1 Bagging Classifier

Bagging (bootstrap aggregating) is a classification technology that uses an ensemble of algorithms, each of which learns independently [10]. The classification result is determined by voting. Bagging allows reducing the classification error in the case when the variance of the error of the base method is high.

Subspace Bagging Classification Algorithm:

1. It is necessary to divide the feature space into subsets, that is, each object will be characterized not by one m-dimensional vector of parameters, but by several vectors $x_{i,1}...x_{i,l}$, and the sum of the dimensions of these vectors cannot exceed m , that is, subspaces cannot intersect. For this, they resort to expert opinion, the expert identifies semantic subspaces based on his experience.

2. Each elementary classifier (each algorithm defined on its subspace) is independently trained. The main sample is classified on each of the subspaces (also independently).

3. The final decision is made about the belonging of the object to one of the classes.

Described approach is only one of the possibilities of generating feature subspaces. Others may include random selection of the features so subsets may intersect.

If all elementary classifiers assigned the same label to an object, then we assign the object to the selected class. The final decision on whether an object belongs to a class can be made, for example, by one of the following methods:

1. Plurality: consensus is very rare, therefore the simple majority method is the most often used. Here, the object is assigned a label of the class that the most elementary classifiers have defined for it. Should be mentioned that according to Leung et. al. plurality voting is not always the the best option [33].

2. Weighing of classifiers: if there is an even number of classifiers, then the votes can be equally divided, it is still possible that for experts one of the groups of parameters is more important, then they resort to weighing the classifiers. That is, when voting, the vote of the classifier is multiplied by its weight [39].

2.5.5.2 Gradient Boosting Classifier

Boosting is a procedure for sequentially building a composition of machine learning algorithms. In this composition each subsequent algorithm seeks to compensate the error of the composition of all previous algorithms. Boosting is a greedy algorithm and similarly to bagging uses multiple weak estimators to find a good solution [37]. These algorithms can be chosen from a wide variety of models, such as decision trees, regression, classifiers, etc.

One of the disadvantages of boosting is that it can lead to cumbersome compositions consisting of hundreds of algorithms. Such compositions exclude the possibility of meaningful interpretation, require large amounts of memory for storing basic algorithms and a significant amount of time to compute classifications.

2.5.6 Complexity estimations

Logistic regression classifier Training complexity for logistic regression with gradient-based optimization: $\mathcal{O}[(D + 1)cNE]$, where D is number of features, c number of classes, N size of a dataset, E number of epochs in gradient descent [35].

KNN According to [40], there are three variants of the algorithm that use different data structures. The first one uses brute force and has $\mathcal{O}[DN]$ time complexity, where D is dimension and N number of samples. For the sake of performance enhancement, there are variants of KNN using different data structures. With the help of trees, the feature space is divided so there is no need to calculate all the distances for each point. Since tree construction takes much less time, it significantly improves performance. For our task, the "auto" parameter was chosen, for which the the best variant of the algorithm is automatically selected based on the characteristics of the dataset. According to the documentation, if the dimension D is greater than 15, then it is too high for the tree-based algorithms and the classifier chooses the Brute force option, the execution time of which is not significantly affected by the number of samples or the number of lines. Based on all of the above, the time complexity of the KNN is $\mathcal{O}[DN]$.

Decision Tree Classifier Scikit-learn uses an optimized version of the CART algorithm. This algorithm complexity is defined by sorting algorithm complexity according to [23]. So in case of using Quick Sort, CART complexity would be on average $\mathcal{O}[DlogD]$, where D is dimension.

Bagging and boosting These algorithms are representatives of meta-learning and their complexity is defined by the complexity of weak learners, in our case, decision trees.

Multilayer perceptron classifier Time complexity of MLP [40] classifier depends on number of training samples N, number of features D, number of hidden layers k, each containing h neurons (for simplicity), and output o neurons. The time complexity of the backpropagation algorithm employed in neural network training is $\mathcal{O}[nmh^k oi]$, where i is the number of iterations. Thus, for given neural network architecture time complexity depends linearly on input dataset size.

Chapter 3

Implementation and evaluation

3.1 Data

NetFlow is a technology introduced by Cisco that allows to monitor the IP network traffic [30]. A dataset contains data from NetFlow sensors obtained from Cisco. Each record of the dataset corresponds to a flow at a given timestamp.

By flows we understand semantically grouped bunch of packets, characteristics of the user session.

By train set we understand data from six-hour observation period.

By test set we understand the data obtained by three hours of observation during the following day.

Initial dataset contains features described in Table 3.1 and explained in detail further.

Let's explain some features more precisely.

cidh feature indicates customer ID. Customer in context of a given system is a particular company, a client of Cisco Systems. Since it is sensitive data, the value is hashed.

cuih is a user ID. User is an employee or a device in customer's network and is based on the IP address or username used on a device.

sip server IP and our baseline model input. It is mapped to an artificial IP to protect privacy.

AS autonomous system number (ASN), an administrative entity providing routing policy to the Internet, mapped to artificial ASN to protect privacy.

sp server port, a TCP/UDP port on the server to which the request was sent.

country abbreviation of country name of the server IP.

Feature name	Example	Description
cidh	ab9098f98e0	ID of network customer
cuih	09832cd098e09a8	user ID
ts	173984023	UNIX time stamp
sip	192.168.1.1	server IP
as	2093	autonomous system number
country	GB	country code
sp	443	server port
cb	20397	client bytes (sent)
sb	2097	server bytes (received)
ctaflowid	923706382	flow id
host	www.2039bc098ef09	hostname

Table 3.1: Feature description

hostname target variable, fully qualified domain name corresponding to given server IP. Top-level and second-level domains are hashed to preserve anonymity of customer data.

3.2 Feature extraction and preprocessing

Extraction and correct representation of features is an important and sometimes a vital part of any machine learning task. However, it is also one of the the most time-consuming parts in learning process because it is difficult and sometimes impossible to automate this process (e.g., in comparison to hyper-parameters tuning). Feature extraction allows obtaining additional, implicit knowledge from the dataset, which can subsequently significantly boost the model performance. In the case when most of the features are categorical (and this is just our case), it is also necessary to think about how this data should be represented to be the model input.

Since the dataset is quite large, feature extraction part was implemented in PySpark and executed in parallel on Amazon EMR cluster. Extraction was done for train and test set independently. The extracted features can be contingently divided into user-specific and company-specific ones. New features are in particular simple statistics and more detailed explanation can be seen in Table 3.2.

Next preprocessing technics were applied to the newly obtained set which included steps:

Feature	Type	Description
most_freq_sip_cuih	US	n most frequent SIP per user
most_freq_country_cuih	US	n most frequent country per user
most_freq_as_cuih	US	n most frequent as per user
most_freq_sp_ciuh	US	n most frequent port per user
nunique_sip_cuih	US	# unique SIP per user
num_sip_cuih_h	US	# unique SIP/hour per user
avg_sb_cuih	US	average server bytes per user
avg_cb_cuih	US	average client bytes per user
most_freq_sip_cidh	CS	n most frequent SIP per company
most_freq_country_cidh	CS	n most frequent country per company
most_freq_as_cidh	CS	n most frequent as per company
nunique_sip_cidh	CS	# unique SIP per company
num_cuih_cidh_h	CS	# unique users/hour per company
num_sip_cidh_h	CS	# unique SIP/hour per company
avg_sb_cidh	CS	average server bytes per company
avg_cb_cidh	CS	average client bytes per company

Table 3.2: Extracted features for $n = 3$. US – user-specific features, CS – company-specific features

Scaling: scaling is a useful step when we need to eliminate the influence of large values. The reason why to do so is that many statistical models assume that data is normally distributed, therefore if a feature has a variance that is orders of magnitude larger than others, it might influence the objective function more and make the estimator work badly and unable to learn from other features properly. Thus, all numeric features were normalized to zero mean and unit variance with sklearn Standard Scaler [40], i.e., each feature is rescaled by subtracting mean and dividing by standard deviation.

Adding new feature is_in_top: since the dataset is very large and has millions of records, it is no point in applying one-hot encoding on every categorical column since it will explode dimensionality. Nevertheless, information on users and server IP is still useful, and to add it to estimator input, we suggest introducing a new binary feature called *is_in_top* that indicates that a given entity (country, server IP, etc.) is in top $n = 3$ respective entities

SIP	host	probability
94.181.12.99	amazon.co.uk	0.33
	fit.cvut.cz	0.66
97.237.10.23	mail.google.com	0.5
	calendar.google.com	0.5
97.82.109.224	seznam.cz	1

Table 3.3: Example of hostname lookup tables

for given user or company. We performed this step for every *the most_freq_...* feature and obtained seven more binary columns.

One-hot encoding was made on features for which the number of their unique values is adequate. One-hot encoding was applied to ports, countries, and companies resulting in 160 new columns.

After some tests, it was obvious that after introduction *is_in_top* feature and one-hot encoding, the resulting metrics have improved without a noticeable increase in time and memory consumption. Such improvements can be explained by introduction of new information and data normalization, which is useful operation since, in general, most of the modern machine learning methods require normalized data for correct predictions.

3.3 Experiment methodology

The idea is to understand if there is a point to use contextual features in the prediction of the most probable SLD or hostname. To estimate the advantages of contextual data, two models were implemented. For convenience, these models will be described separately for hostname case and SLD case.

3.3.1 Hostname prediction

Baseline hostname model This model takes as input only server IP (sip) and returns the most probable hostname for a given server IP. To do so, it performs calculation of probabilities on the whole training set: $P(\text{host}|\text{sip})$. The examples of such probability tables are shown in Table 3.3.

For the test set, the model f takes as input server IP (sip) and returns the most probable hostname from the train set $\text{host} = \text{argmax}_{\text{host}} P(\text{host}|\text{sip})$. If there is no record with given server IP in train set, the model returns "NA" prediction as it is shown in Table 3.4. Note that the always returns only the most probable hostname.

Workflow of the baseline model is schematically shown on Figure 3.1.‘

SIP	host_1
94.181.12.99	fit.cvut.cz
97.237.10.23	mail.google.com
97.82.109.224	seznam.cz

Table 3.4: Example of hostname predictions in baseline model.

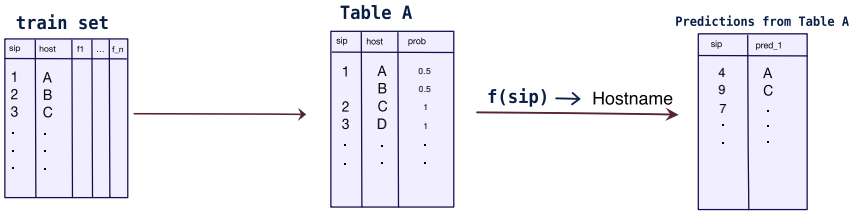


Figure 3.1: Baseline model workflow (hostnames)

Extended hostname model Our proposed extended model f_e, besides server IP, takes as input contextual data containing extracted user-specific and company-specific features, as well as the other standard features listed in Table 3.1. Instead of returning only one the most probable hostname, the extended model uses lookup table T which again contains calculated probabilities same as in the baseline model 3.3, but now we return k most probable hostnames.

When talking about prediction of the the most probable hostname in extended model, for arbitrary number of columns k, we use classifier C to determine **in which column of the probability lookup table should we look for a correct hostname**.

By target variable we understand a class $h \in \{h_1, \ldots, h_{k+1}\}$ which indicate:

1. h_k or k-th hostname corresponds to taking hostname from the k-th column in the lookup table. The first column contains the most probable hostname, thus, the k-th column contains k-th the most probable hostname.

2. h_{k+1} or N-class indicates that there is **different traffic type and there is no correct hostname in probability table** and we should not look for it in this table.

For parameter tuning we were using $k = 3$ most probable hostnames.

So first of all, it is necessary to modify the train dataset for this task in order to obtain target variables. This is done by performing the next steps:

1. We derive two tables of approximately same size from initial dataset. For the first look-up table A, we calculate hostnames and their respective

SIP	host_1	host_2	host_3	N_class
94.181.12.99	True	False	False	False
97.237.10.23	False	False	True	False
97.82.109.224	False	False	False	True

Table 3.5: Example of hostname evaluations.

probabilities for each server IP similar to the baseline model. E.g.:

$$A = \{(\text{sip}\,((\text{host}_1,\, p_1), (\text{host}_2,\, p_2), \ldots (\text{host}_k,\, p_k)))\}$$

$$p_1 \geq p_2 \geq \ldots \geq p_k$$

$$\sum p_i \leq 1$$

.

2. Table B is constructed in such way that we predict k most probable hostnames (host_1, $\text{host}_2, \ldots,$ host_k) according to probabilities values from the table A.

3. Now we can evaluate predictions for each flow by comparing the predicted i-th $i \in (1, \ldots, k)$ hostname in the flow with true hostname values. As a result we get a table with binary values where True means that prediction was correct and correct value can be found in corresponding column as it is shown in Table 3.5.

4. Last column N-class represents the situation when there is a different type of traffic and there is no record for it in the train set and, consequently, no prediction for it. N-class value is True only if all other classes are False.

5. Our target variable for classifier C is an index of column with **the first correctly predicted** hostname (in case when there are more than one correct columns). More precisely, having a set of possible IP addresses S, we perform classification $g : (S, \text{features}) \mapsto \{h_1^1, \ldots, h_k, N\}$, where $1, \ldots, k$ represents index of column, while N marks absence of IP address in the train set, or the situation where classifier g refuses to make the prediction out of the k available hostnames. In other words, classifier C takes IP address with user and company features as input and returns a column index.

SIP	SLD	probability
94.181.12.99	cuni.cz	0.33
	cvut.cz	0.66
97.237.10.23	google.com	1
97.82.109.224	seznam.cz	1

Table 3.6: Example of SLD lookup tables

SIP	sld_1	sld_host1	...	sld_3	sld_host3
94.181.12.99	cvut.cz	cuni.cz	...	NA	NA
97.237.10.23	amazon.com	google.com	...	weather.com	office365.com
97.82.109.224	seznam.cz	NA	...	NA	NA

Table 3.7: Example of SLD predictions. There are two sources for SLD prediction: pure SLD and SLD from predicted host.

3.3.2 SLD prediction

Second-level domain is also interesting. The reason is that SLD's are registered at domain registrars and **paid**, any higher-order domain names are the up-to-the decision of SLD's owner and free. Additionally, SLD's can be predicted more successfully since, for example, multiple hostnames can have the same second-level domain. A potential extension of this work can be a system that decides whether it is a point in predicting SLD instead of host.

Baseline SLD model Baseline model for SLD prediction takes as input server IP (sip) and returns the most probable SLD for given server IP SLD = $\text{argmax}_{\text{SLD}} P(\text{SLD}|\text{sip})$. It computes probabilities $P(\text{SLD}|\text{sip})$ in the same manner as for hostnames prediction as it is shown in Table 3.6.

For test set model f takes as input server IP (sip) and returns the most probable SLD from the train set. If there is no record with a given server IP in train set, model returns "NA" as it is shown in Table 3.7.

Workflow of SLD baseline model is schematically shown in Figure 3.2.

Extended SLD model Extended model f_e for SLD, as well as in case of hostname prediction uses contextual data as well as extracted user-specific and company-specific features. Instead of the most probable SLD, it now has multiple most probable values from two sources: SLD calculated from train set and SLD obtained from hostnames predictions. In the next paragraph approach for SLD extended model will be described.

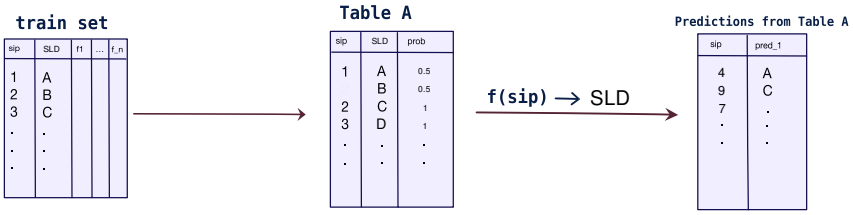


Figure 3.2: Baseline model workflow (SLD)

As it was mentioned, there are two possibilities to obtain SLD: to predict it from SLD probability table, or to extract SLD from predicted hostnames. Thus, for the SLD prediction we now define $2k + 1$ classes $d \in \left\{d_1, d_1^h, d_2, d_2^h \ldots, d_k, d_k^h, N\right\}$ and the model again will answer the question **in which column we should look for the correct SLD.** Following columns contain the next information:

1. d_k or sld_k corresponds to taking SLD from the k-th column in the SLD probability table. The first column contains the most probable SLD, thus, the k-th column contains k-th most probable SLD.

2. d_k^h or sld_from_host_k corresponds to taking SLD extracted from the k-th column in hostname probability table.

3. d_{k+1} or N_class indicates that there is **no correct SLD neither in SLD lookup probability table, nor in hostnames lookup table (from which we derived SLD)** and we should not look for it in these tables.

For parameter tuning we were using $k = 3$ most probable SLD and $k = 3$ SLD's from hostnames, respectively.

We use the same approach for dataset modification to obtain target variables as it has been done for hostname prediction. The approach includes the next steps:

1. We again derive two tables of approximately same size from initial dataset. For the first table A we calculate each hostname probabilities for each server IP same as it has been done in the baseline model. E.g.:

$$A = \left\{\left(sip\left((sld_1,\, p_1), (sld_1^h,\, p_1), \ldots, (sld_2,\, p_k), (sld_2^h,\, p_k)\right)\right)\right\},$$

$$p_1 \geq p_2 \geq \ldots \geq p_k,$$

$$\sum p_i \leq 1$$

.

SIP	sld_1	sld_host1	...	sld_3	sld_host3	N_class
94.181.12.99	True	False	...	False	False	False
97.237.10.23	False	False	...	True	False	False
97.82.109.224	False	False	...	False	False	True

Table 3.8: Example of SLD evaluations.

2. Table B is constructed in such way that we predict k most probable SLD's and SLD's derived from hostnames $\left(sld_1, sld_1^h, \ldots, sld_k, sld_k^h\right)$ according to probabilities values from the table A.

3. Now we can evaluate predictions by comparing them with true SLD values from table B. As a result we get a table with binary values where True means that prediction was correct and correct value can be found in corresponding column as it is shown in Table 3.8.

4. Last column N-class represents the situation when there is a different type of traffic and there is unknown server IP and there is no record for it in the train set and, consequently, no prediction for it. N-class value is True only if all other classes are False.

5. Our target variable for classifier C is an index of column with **first correctly predicted** SLD (in case when there are more than one correct columns). More precisely, having a set of possible IP addresses S, we perform classification $g : (S, features) \mapsto \{0, 1, \ldots, 2k, N\}$, where $1, \ldots, k$ represents index of column, while N marks absence of IP address in the train set or refuse for classification. In other words, classifier C takes IP address with user and company features as input and returns a column index.

3.3.3 Experiment summary

1. Experiment goal is to increase accuracy of prediction of baseline model.

2. The key idea is to find the best classifier that will predict column in which we should look for correct SLD/hostname.

3. Since dataset is large, multiple classifier candidates are compared on small fraction of data (0.1%).

4. Final evaluation is performed with a classifier C trained on 6 hours of NetFlow records dataset and tested on 3 hours from the following day.

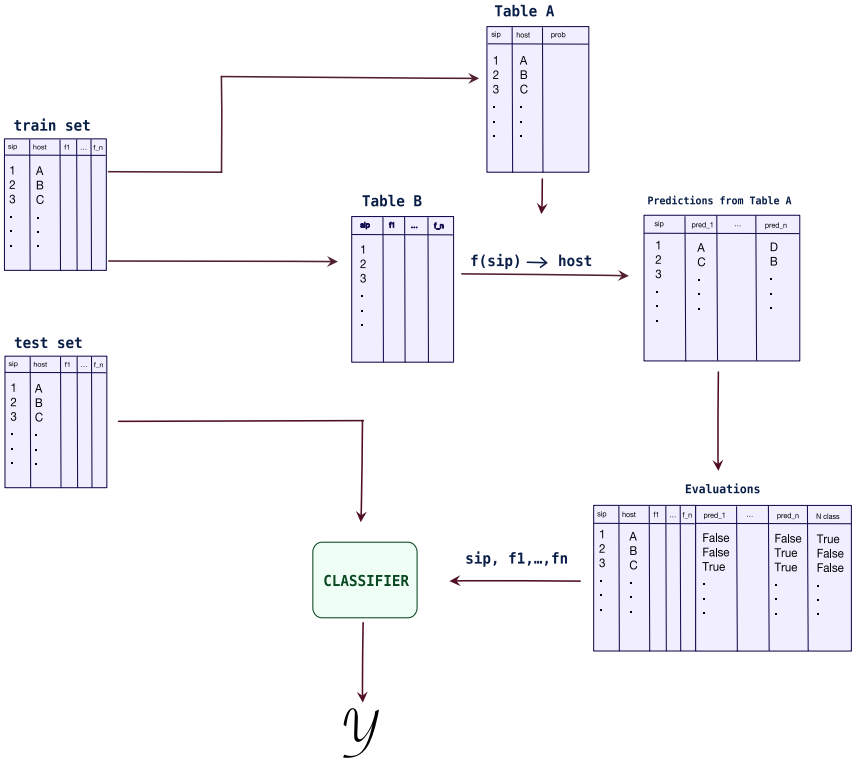


Figure 3.3: Workflow diagram for extended model.

In Figure 3.3 a hostname prediction process is shown schematically for the better understanding of the workflow. Baseline model accuracy is considered as a reference result and the purpose of the extended model is to improve accuracy value 0.81 for SLD prediction and 0.65 accuracy for the hostname prediction.

3.4 Implementation details

The code is structured as follows:

- feature extraction module (pdnsfeatex package) implemented in PySpark. Module contains functions that calculate statistical features
- hostname/SLD prediction module (pdnspred package), containing PDNS Base predictor class defining common functions for both SLD and hostname prediction and inherited classes SLDPredictor and HostPredictor defining task-specific functions. Code style follows the sklearn package style and its structure can be seen at Figure 3.4.

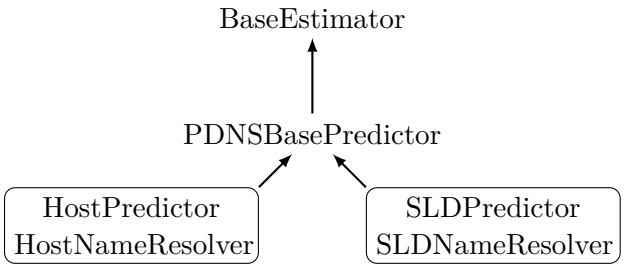


Figure 3.4: Code structure

	Original dataset	Subsampled dataset	Fraction
Number of cuih(s)	696186	674	0.00097
Number of host(s)	377233	2921	0.00774
Number of cidh(s)	47	46	0.97872
Number of sip(s)	199256	5296	0.02658

Table 3.9: Basic statistics of original dataset and subsampled dataset

3.5 Experimental part

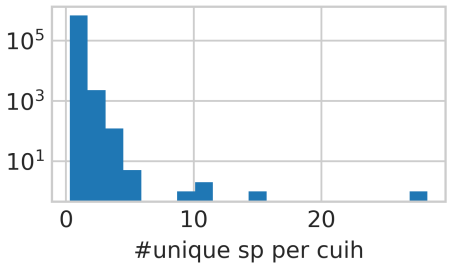
3.5.1 SLD and hostnames prediction

The main task of experimental part is to select the best classifier, which predicts the output class, corresponding to the column number with the correct value of the hostname or SLD. For this purpose, the grid search on different classifiers and hyperparameters was performed. Further, to get statistically meaningful results, a 5-fold cross-validation was used.

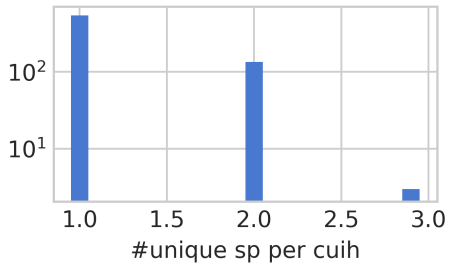
3.5.1.1 Subsampling

Due to the large amount of data, cross-validation was conducted on 0.1% of the train data, which was obtained by random subsampling of cuih feature (which corresponds to a user ID). Number of rows in original dataset: 43581438, in subsampled dataset: 40303.

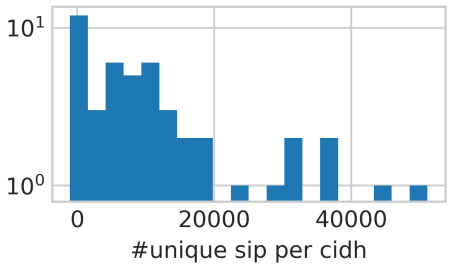
Before proceeding to the process of cross-validation itself, it was shown that the distributions of key features of the train dataset and the subsampled one approximately correspond. Visualizations of initial and subsampled feature distributions are provided in the Figure 3.5.



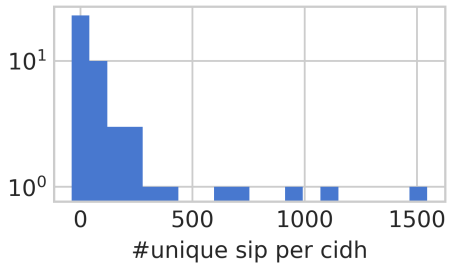
(a) Number of unique server ports per user ID (original dataset)



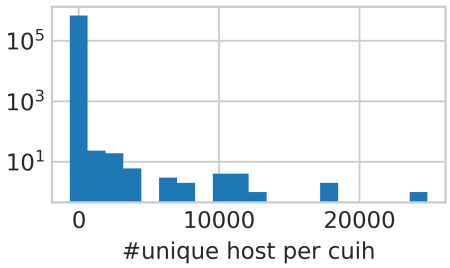
(b) Number of unique server ports per user ID (subsampled dataset)



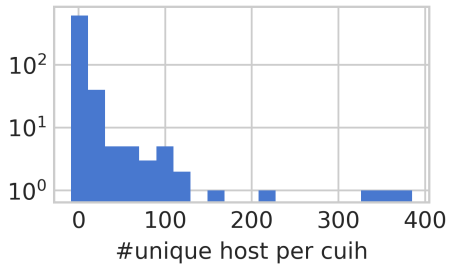
(c) Number of unique server IPs per customer (original dataset)



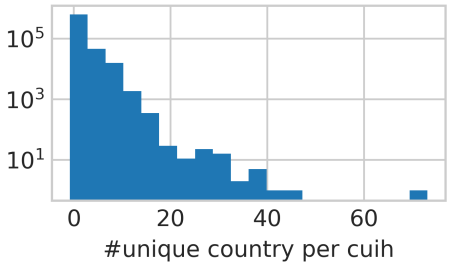
(d) Number of unique server IPs per customer (subsampled dataset)



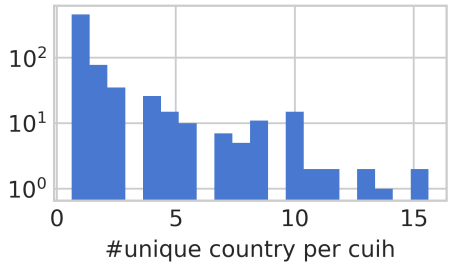
(e) Number of unique hostnames per user ID (original dataset)



(f) Number of unique hostnames per user ID (subsampled dataset)

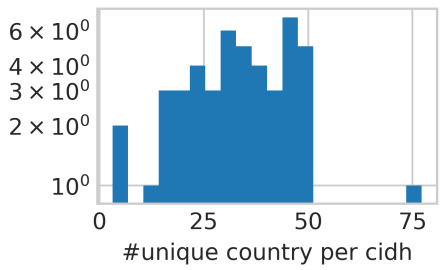


(g) Number of unique countries per user ID (original dataset)

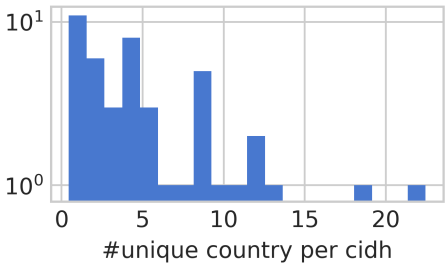


(h) Number of unique countries per user ID (subsampled dataset)

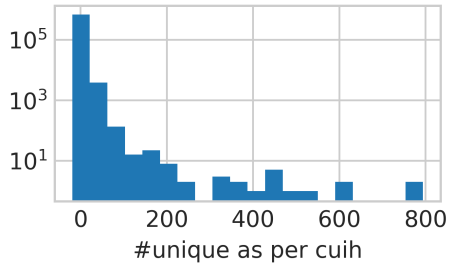
Figure 3.5: Distribution comparison. Original train set vs. subsampled train set



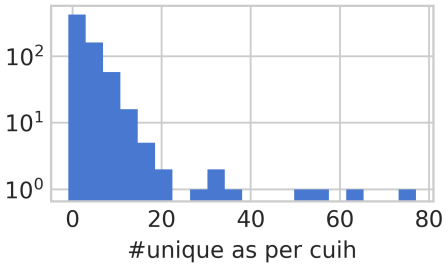
(i) Number of unique countries per customer (original dataset)



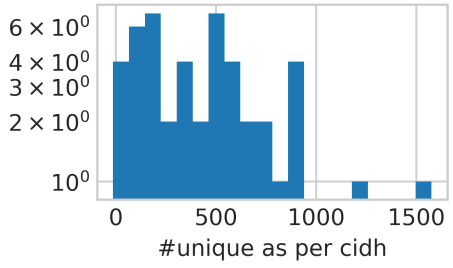
(j) Number of unique countries per customer (subsampled dataset)



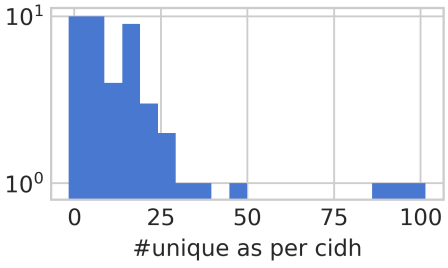
(k) Number of unique AS numbers per user ID (original dataset)



(l) Number of unique AS numbers per user ID (subsampled dataset)



(m) Number of unique AS numbers per customer (original dataset)



(n) Number of unique AS numbers per customer (subsampled dataset)

Figure 3.5: Distribution comparison. Original train set vs. subsampled train set

Classifier	Parameters list
Logistic regression	C=[0.5, 1, 2] penalty=['l1', 'l2', 'elasticnet'] solver=['lbfgs', 'sag', 'newton-cg']
K-Nearest Neighbors	n_neighbors=[1,5,10] algorithm=['ball_tree', 'kd_tree', 'brute'] metric=['manhattan', 'chebyshev', 'canberra']
Multi-layer perceptron	hidden_layer_sizes=[128, [64, 64], [32, 32, 32]] activation=['relu', 'tanh', 'logistic'] learning_rate=['adaptive', 'constant', 'invscaling']
Bagging	n_estimators=[50, 100, 200] max_samples=[0.1, 0.2, 0.5] max_features=[0.3, 0.5, 0.7]
Gradient Boosting	n_estimators=[50,100,200] min_samples_split=[50, 200, 500] max_depth=[5, 8, 12]

Table 3.10: Parameter lists for grid search

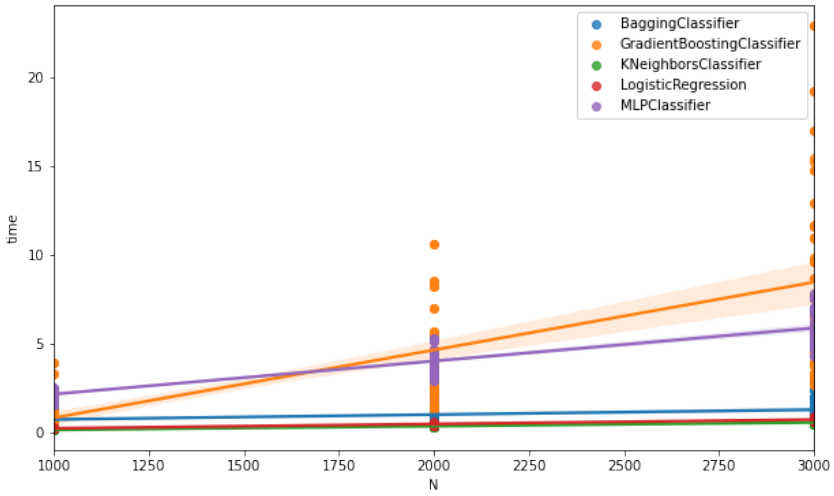
3.5.1.2 Grid search

Configurations First of all, let us specify chosen classifiers and hyperparameters. We have selected 5 classifiers, which are commonly used in different machine learning applications. This set consists of representatives of different approaches:

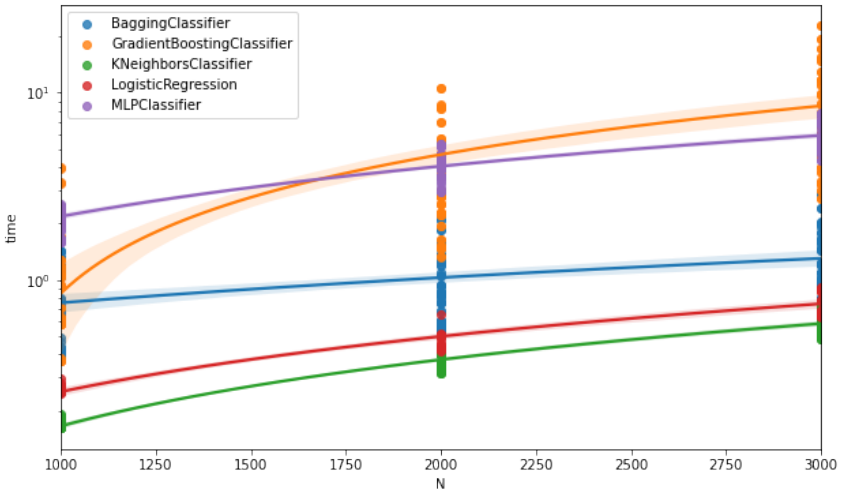
1. Logistic regression classifier
2. K-Nearest Neighbors classifier – a neighbor-based approach
3. Multi-layer perceptron – a representative of classifiers looking for the separating hyperplane
4. Random forest with bagging and Gradient boosting – representatives of ensemble methods

For each classifier three hyperparameters and three values of each hyperparameter were analyzed on 5 folds of subsampled train data. This has been done for both hostname and SLD prediction tasks. The detailed configurations are provided in the Table 3.10.

Time estimation Also, with subsampled data it has become possible to estimate overall time spent by each classifier to process train set of different sizes (Figure 3.6).



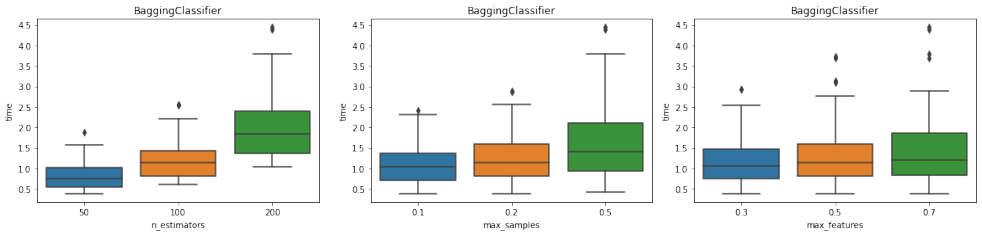
(a) Time, [s]



(b) Time, $\log_{10}$[s]

Figure 3.6: Overall time spent by each classifier depending on number of rows in train set

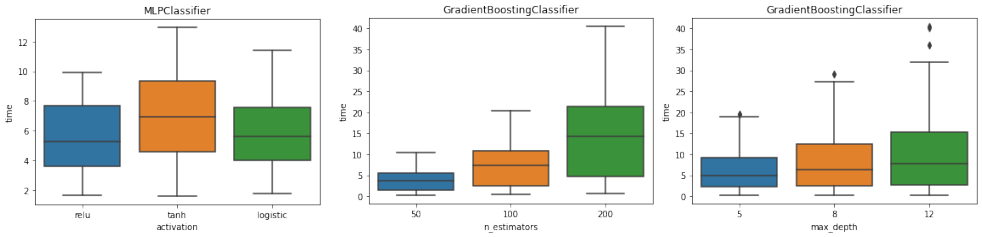
In addition to estimation depending on train set size, we analyzed how each hyperparameter influence total computational time for each classifier. The most interesting (those, which impact on total time is obvious from visual representations) are provided in Figure 3.7.



(a) Bagging, number of estimators

(b) Bagging, maximum samples

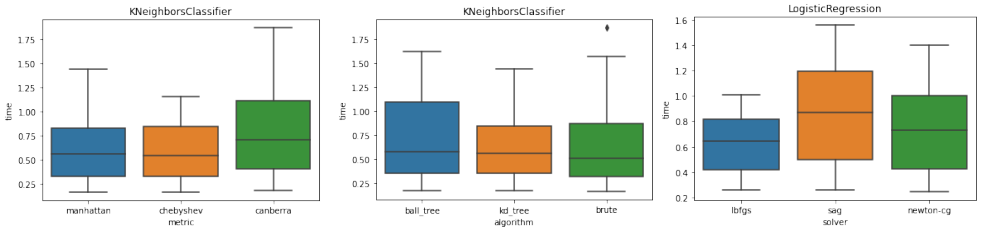
(c) Bagging, maximum features



(d) MLP, activation

(e) Boosting, number of estimators

(f) Boosting, maximal depth



(g) KNN, metric

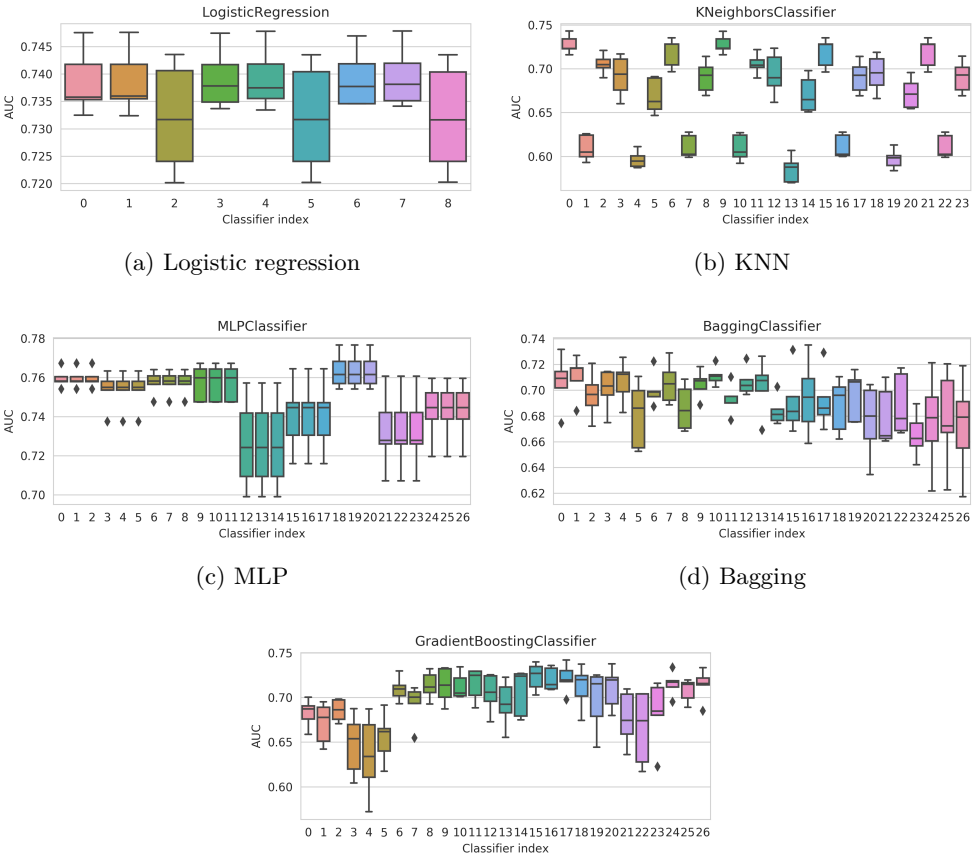
(h) KNN, algorithm

(i) Logistic regression classifier, solver

Figure 3.7: Hyperparameters influencing computational time

Metrics For this task we analyzed following metrics: accuracy, area under the curve (AUC), and binary metrics such as precision, recall and F1-score and so-called binary accuracy. Let us explain the nature of these binary metrics. We reduced our multiclass task to binary problem in the following way: instead of considering $2n + 1$ (SLD) or $n + 1$ (hostnames) we now have:

- class 0 which stands for taking value from probability lookup table ($1 \dots 2n$ and $1 \dots n$ classes from multiclass problem form this class),
- class 1 which stands for N-class, which indicates that there is different type of traffic, so we do not need to use Passive DNS probability lookup tables.



(a) Logistic regression

(b) KNN

(c) MLP

(d) Bagging

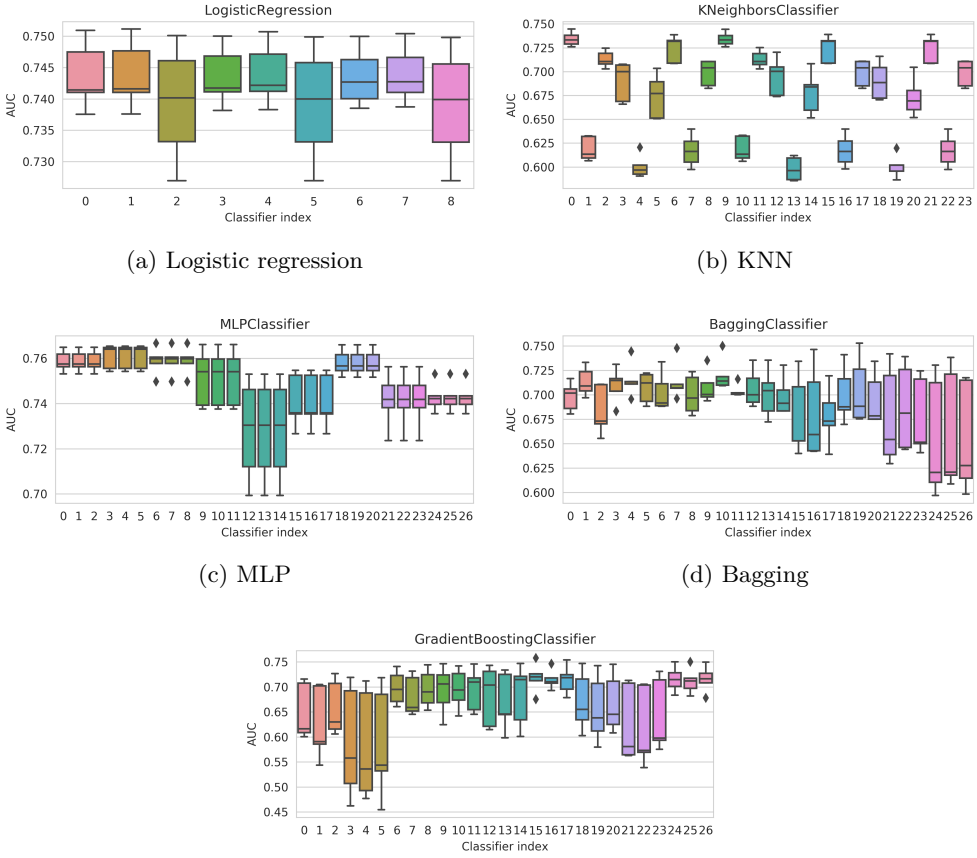
(e) Gradient Boosting

Figure 3.8: SLD cross-validation AUC

Grid search results The cross-validation AUC scores of each configuration are represented via boxplots for both SLD (Figure 3.8) and hostnames (Figure 3.9) predictions. Similar boxplots for metrics as accuracy, binary accuracy, precision, recall and F1-score are provided in the Appendix.

Finally after gathering cross-validation statistics, we were able to determine the most successful classifiers (by mean value obtained with 5-fold cross-validation) in terms of each metric, which is represented on Figure 3.10 (SLD) and Figure 3.11 (hostnames). The most successful instances of these classifiers are shown in Table 3.11 (SLD) and Table 3.12 (hostnames).

Finding one classifier for final evaluation is challenging: each classifier is good for the one particular metric. For example, in Figure 3.11, it is obvious that for F1-score the best option is MLP classifier, while for recall the Gradient boosting classifier mean score is the highest, however, Logistic regression classifier gave the best accuracy. All these metrics are important for the task,



(a) Logistic regression

(b) KNN

(c) MLP

(d) Bagging

(e) Gradient Boosting

Figure 3.9: Hostnames cross-validation AUC

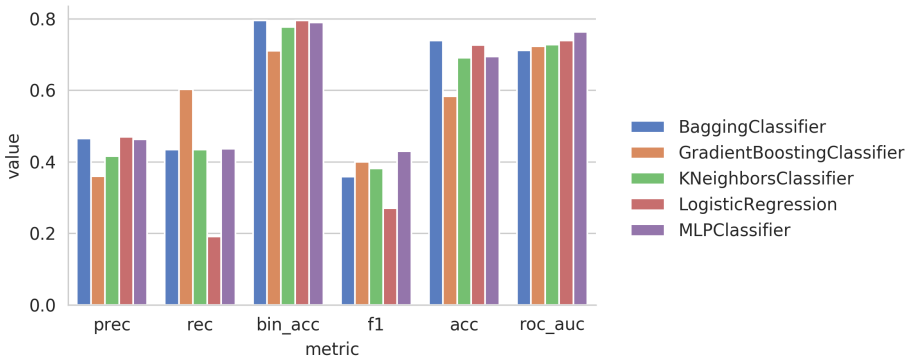


Figure 3.10: Most successful SLD classifiers by metric

Classifier	Precision	Recall	Bin acc	F1	Acc	AUC
Bagging	7	25	1	19	1	10
GradientBoosting	17	19	8	16	8	15
KNeighbors	0	1	0	11	12	9
LogisticRegression	1	0	5	0	2	7
MLPClassifier	3	9	3	9	3	18

Table 3.11: Most successful SLD instances

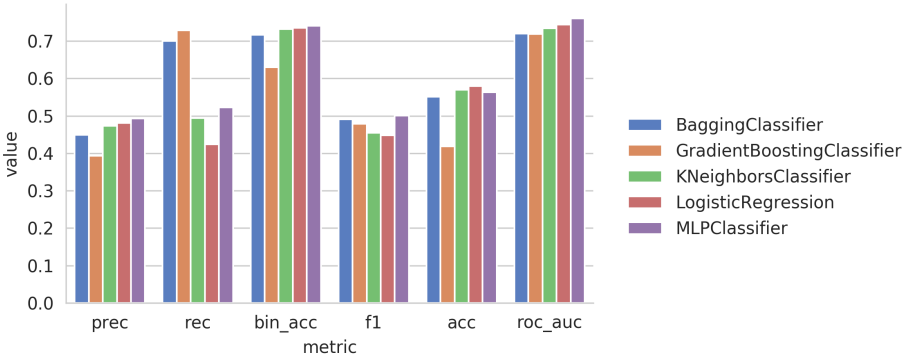
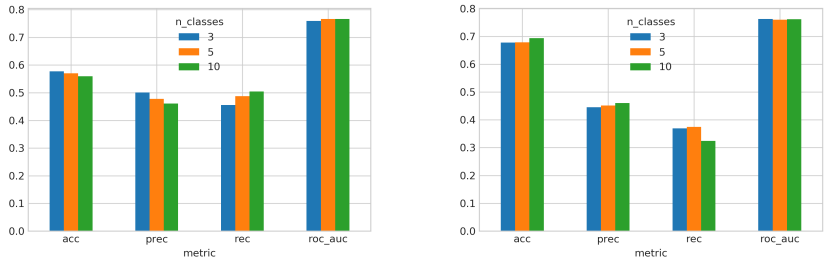


Figure 3.11: Most successful hostname classifiers by metric

Classifier	Precision	Recall	Bin acc	F1	Acc	AUC
Bagging	5	19	1	19	5	10
GradientBoosting	17	2	26	16	17	15
KNeighbors	0	1	0	9	0	0
LogisticRegression	1	7	1	7	2	4
MLPClassifier	18	3	18	0	18	3

Table 3.12: Most successful instances (host)



(a) Hostnames metrics

(b) SLD metrics

Figure 3.12: Experiments with different number of classes

so suggested solution is as follows:

1. give each classifier a rank according to next metrics: accuracy, precision, recall and AUC,
2. for each classifier calculate mean rank,
3. choose classifier with the highest mean rank.

The best classifier according to the mean rank was MLP classifier with hyperbolic tangent activation function and one layer with 128 neurons.

3.5.1.3 Experiments with different number of classes

In addition to classifier selection, we carried out the experiments with different number of classes. Hence, the question is: how many most probable values should we consider - 3, 5 or 10? Again, the experiments were performed on a fraction of data (0.1%) and cross-validated metrics (from 5 folds) (Figure 3.12) were compared using the rank approach described earlier.

Results of experiments with different number of classes showed that:

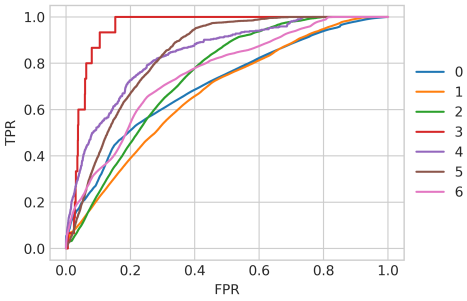
- The optimal number of classes for SLD prediction is 10 classes.
- For hostname prediction all options were equivalent

3.5.1.4 Results

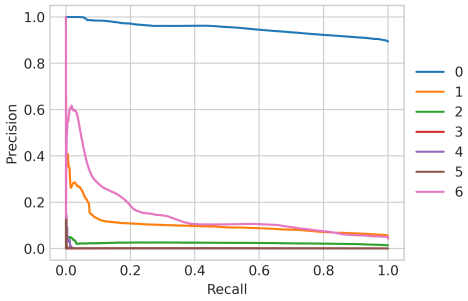
After classifier selection, we applied chosen one to perform calculations on the whole train set containing around 64 million of records. Due to the large size of the dataset, the calculation ran only once.

The extended model's accuracy increased:

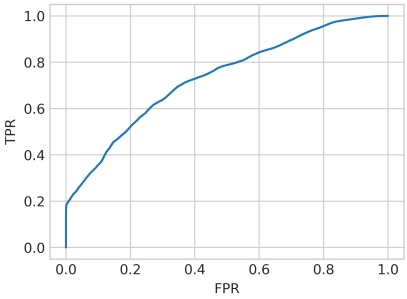
- from 65% to 77.4% (std=1.8%) for hostname prediction



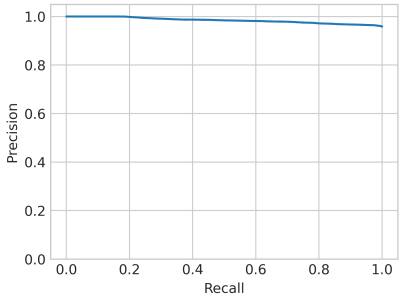
(a) ROC curves for each class



(b) PR curves for each class



(c) ROC curve for binary case



(d) PR curve for binary case

Figure 3.13: SLD. Visualizations of the final metrics

- from 81% to 89.3% (std=2%) for SLD prediction task[1].

As we can see, the extended model has demonstrated superiority over the baseline model, which is especially noticeable for predicting hostnames. In the case of predicting second-level domains, the difference is slightly smaller, which can be explained by the fact that this task was initially simpler, since it is, in a sense, a generalization of the problem of predicting hostnames. Visualizations of the results can be seen in Figures 3.13 and 3.14.

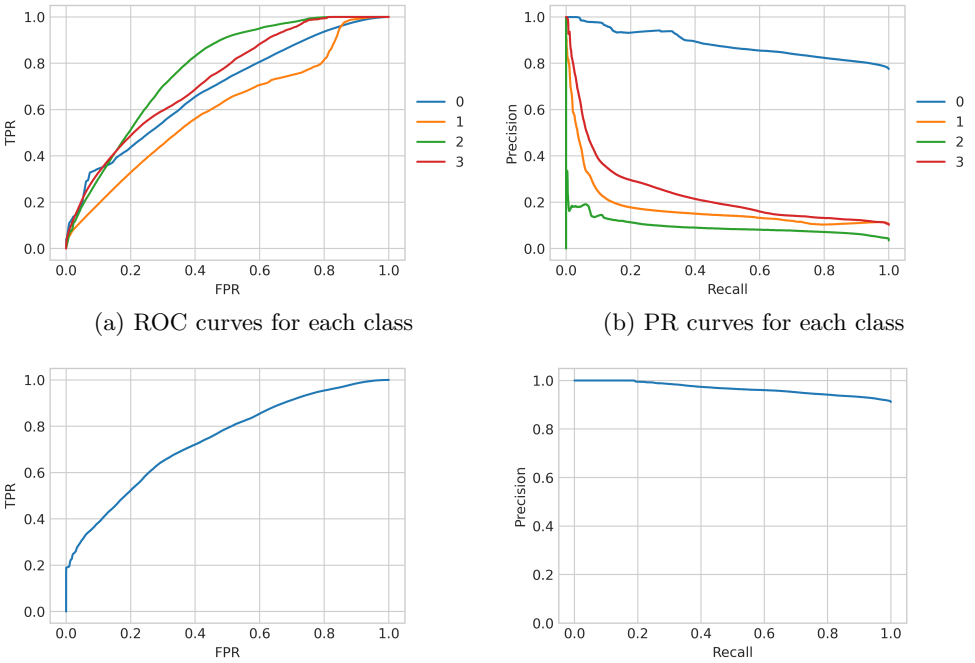
It is clear from these results that it makes sense to take into account more than one most probable value.

Experiments have shown that the best number of classes for SLD prediction task is 10. Thus, we trained the model on the whole train set with specified number of classes $n = 10$. The results are provided in Table 3.13 which contains metrics obtained after averaging 5-fold cross-validation scores.

3.5.2 Risk prediction

Since we have data augmented by feature extraction, the essential decision is to see what other knowledge about the observed system we can obtain. An

[1]Standard deviation for both cases is obtained during grid search phase



(a) ROC curves for each class

(b) PR curves for each class

(c) ROC curve for binary case

(d) PR curve for binary case

Figure 3.14: Hostnames. Visualizations of the final metrics

	n_classes	type	acc	bin_acc	f1	prec	rec	roc_auc
0	3	sld	0.8935	0.9642	0.0212	0.5270	0.0108	0.7389
0	10	sld	0.8933	0.9586	0.0544	0.6008	0.0285	0.7443

Table 3.13: Comparison of mean SLD metrics for 3 classes and 10 classes

interesting variable in this regard is the variable **risk**, which indicates how safe a given IP address in the Cisco gradation is. This variable can take values from 0 to 9, where 0 indicates the lowest risk and 9, respectively, the highest (Figure 3.14). To determine the value of risk, we utilized data augmented by the extracted features.

In the cybersecurity field the very common problem is highly imbalanced datasets. Thus, a vast majority of records are marked as safe and zero-risk, while number of cases of real interest (representatives with risk > 0) may be fractions of a percent. Accordingly, in this case, the accuracy value, say 0.99, will rather indicate that 99% of the dataset are zero-risk representatives and the classifier marks all input data as low-risk representatives. To overcome the issue, a class balancing strategy, Random Under Sampling from *imblern* [32] library was applied. The majority class (zero-risk) was undersampled without replacement. After applying this strategy, a 29 GB dataset was reduced to 40

risk	count	risk	count
0	62593266	5	12
1	0	6	2007
2	0	7	72
3	114	8	3
4	594	9	66

Table 3.14: Number of representatives of each class in the train set

KB (3125 rows). We looked at the problem from three points of view:

Binary:

- $risk \in \{0, \ldots, 3\}$ is Legit risk
- $risk \in \{4, \ldots, 9\}$ is High risk

Ternary:

- $risk \in \{0, \ldots, 3\}$ is Legit risk
- $risk \in \{4, \ldots, 6\}$ is Low risk
- $risk \in \{7, \ldots, 9\}$ is High risk

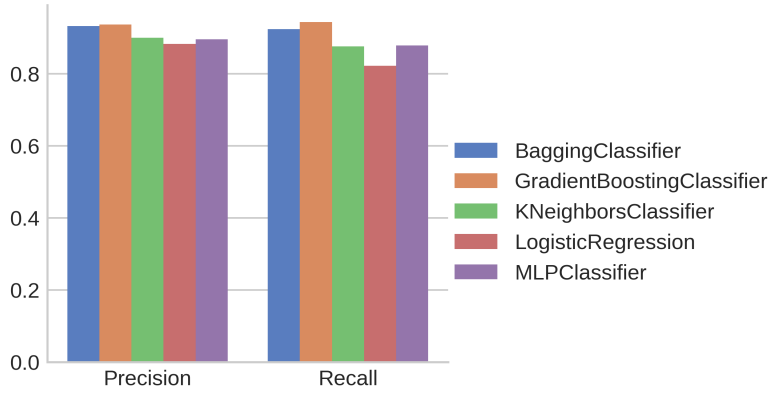
Multiclass classification, where we predict the values that are used in the Cisco dataset:

- $risk \in \{0, \ldots, 9\}$ where each risk value is considered as a single class

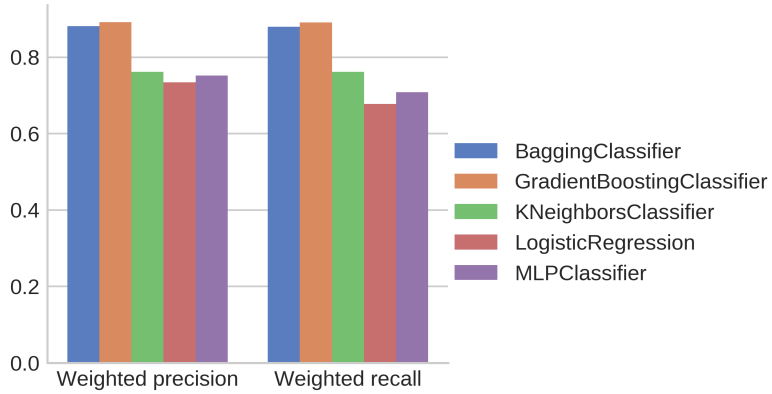
The grid search was carried out by analogy with the predictions of the hostnames and SLD in the previous section with similar classifiers and their instances. Again, for each number of classes we went through several sets of hyperparameters and calculated the metrics to find the best model for each task (Figure 3.15).

Results Results obtained with the best model for binary task, a Gradient Boosting classifier, are provided in Table 3.15 and Figure 3.16. Hyperparameters are:

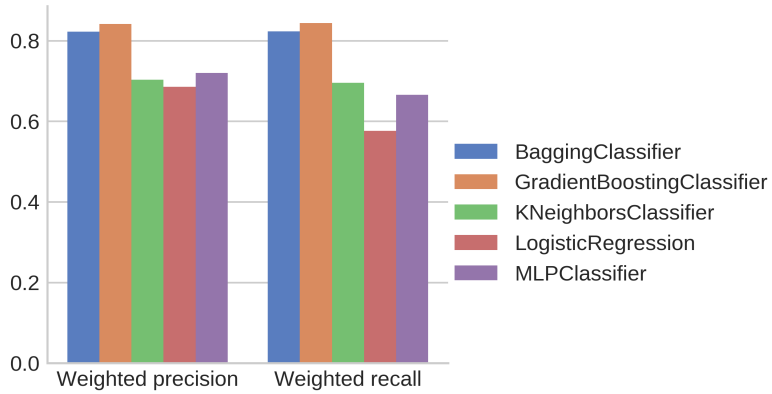
- max_depth: 8
- min_samples_split: 50
- n_estimators: 200



(a) Binary case model selection



(b) Ternary case model selection, weighted metrics



(c) Multiclass case model selection, weighted metrics

Figure 3.15: Metrics comparison

Ternary classification Results obtained with the best model for ternary task, a Gradient Boosting classifier, are provided in Table 3.16, 3.17 and Figure 3.17. Hyperparameters are:

- max_depth: 12
- min_samples_split: 50
- n_estimators: 200

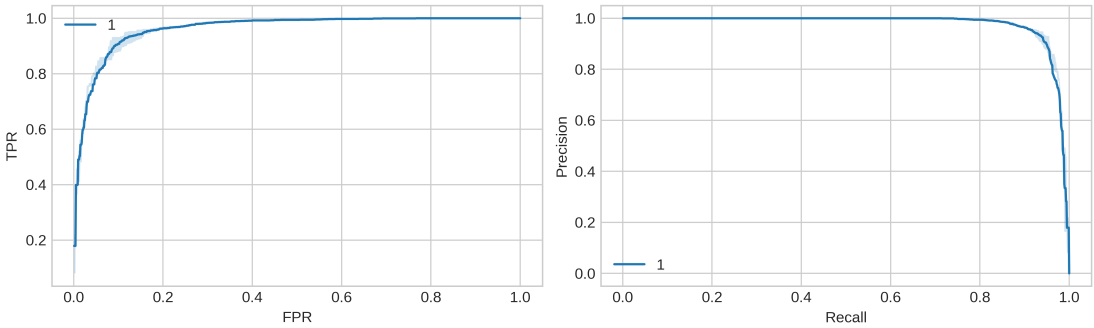
Multiclass

Results obtained with the best model for multiclass task, a Gradient Boosting classifier, are provided in Table 3.18, 3.19 and Figure 3.18. Hyperparameters are:

- max_depth: 12
- min_samples_split: 200
- n_estimators: 200

	Precision	Recall
1	0.9366	0.9350

Table 3.15: Binary task. Precision and recall. Best model results



(a) ROC curve

(b) PR curve

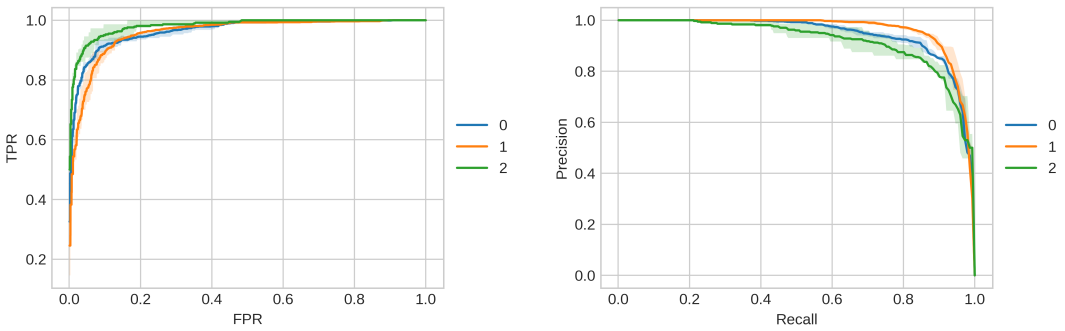
Figure 3.16: Binary task. ROC and PR curves for High risk class. Best model results

	Precision per label	Recall per label
0	0.8832	0.8758
1	0.9002	0.9170
2	0.8762	0.8230

Table 3.16: Ternary task. Precision and recall per labels. Best model results.

	Metric value
Weighted precision	0.8917
Weighted recall	0.8912

Table 3.17: Ternary task. Weighted precision and recall. Best model results.



(a) ROC curve

(b) PR curve

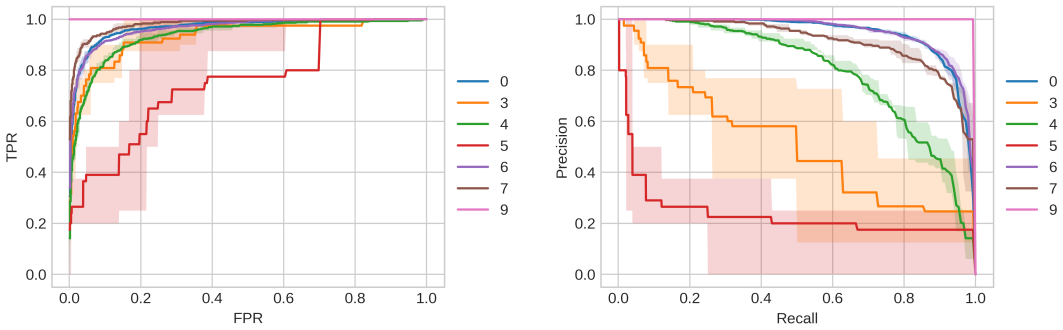
Figure 3.17: Ternary task. ROC and PR curves for all classes. The best selected model

	Precision per label	Recall per label
0	0.8730	0.8831
3	0.5933	0.2999
4	0.7158	0.7123
5	0.6000	0.1750
6	0.8692	0.9000
7	0.8528	0.8253
9	0.9200	0.9267

Table 3.18: Multiclass task. Precision and recall per labels. Best model results.

	Metric value
Weighted precision	0.8418
Weighted recall	0.8442

Table 3.19: Multiclass task. Weighted precision and recall. Best model results.



(a) ROC curve

(b) PR curve

Figure 3.18: Multiclass task. ROC and PR curves. The best selected model

Chapter 4

Conclusion

This work was dedicated to the upgrade of the basic passive DNS approach used in hostname/SLD prediction system in Cisco.

Following steps were taken:

1. Implementation and evaluation of purely probabilistic baseline model to serve as a reference.
2. Extraction of advanced statistical features from NetFlow data.
3. Grid search on subsampled data over different models and hyperparameters with aim to select the best one for given task.
4. Implementation and evaluation of extended model which combines probabilistic approach with the advanced machine learning technics and utilizes additional knowledge from extracted features.

The implementation was separated into following stages:

- Feature extraction from raw large-scale data implemented in Apache Spark run on Amazon cluster.
- Baseline and extended model were implemented in Python in scikit-learn style and packaged to facilitate their use.

The main results of this work are:

- increase in accuracy **from 65% to 77.4%** for hostname prediction
- increase in accuracy **from 81% to 89.3%** for SLD prediction task in comparison to reference model.
- Experiments with different number of classes showed that the optimal number of SLD values to keep is 10. For hostnames prediction there were no difference in average metrics for different number of classes.

Risk prediction Another bunch of experiments were carried out for prediction of the risk variable, which lead to the following results:

- **93.6%** precision and **93.5%** recall for the binary classification task
- **89.2%** weighted precision and **89.1%** weighted recall for the ternary task
- **84.2%** weighted precision and **84.5%** weighted recall for the multiclass task.

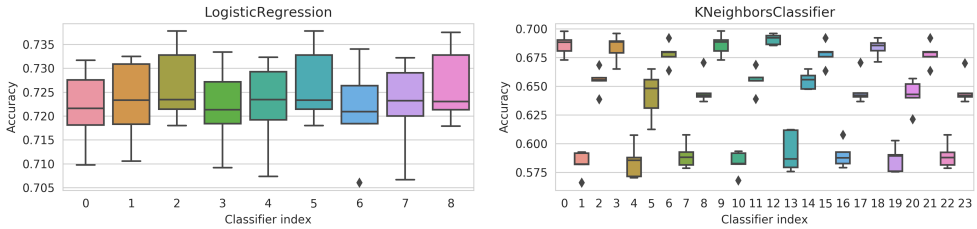
Possible directions of future work:

- Investigate influence of different splitting strategies of train tables, i.e., which part of training dataset to utilize for probability tables building and which one for the classification.
- To extend model with additional module able to choose, whether to predict SLD or hostname depending on the input data.
- To apply ordinal regression to predict risk value.

During working on this thesis a valuable experience was gained in sense of working with large-scale data. The amounts of data made this task unexpectedly non-trivial, as it was necessary to keep in mind not only structure of data and theoretical concepts of machine learning, but also consider the time and memory complexity of every single operation involved in the process. In general, work at that scale is qualitatively differs from typical educational tasks. The outcome of the proposed approach suggests that this concept is worth further investigation and extension, and potentially can be used in real-life problem solving in Cisco. Overall, the completion of this thesis gave an overview and experience with such tools as AWS S3, AWS Elastic Map Reduce, Apache Spark and in general with business-oriented environment of the enterprise.

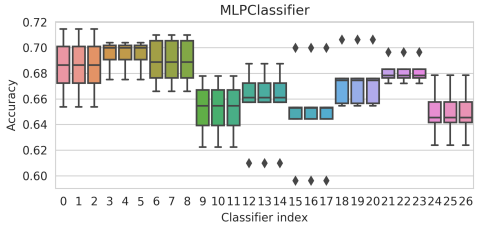
Appendix A

Grid search results

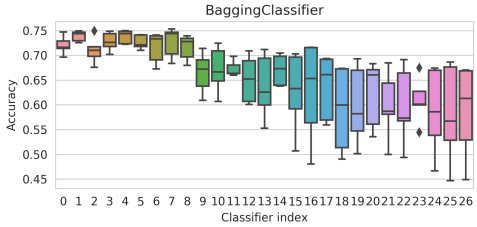


(a) Logistic regression

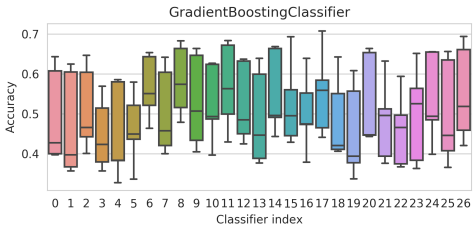
(b) KNN



(c) MLP

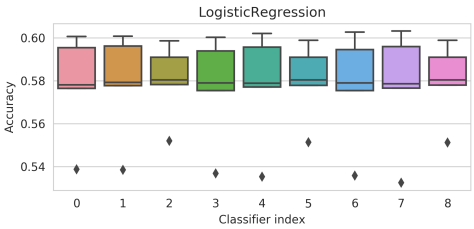


(d) Bagging

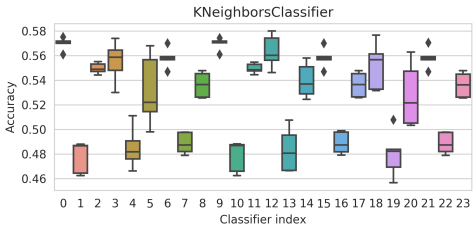


(e) Gradient Boosting

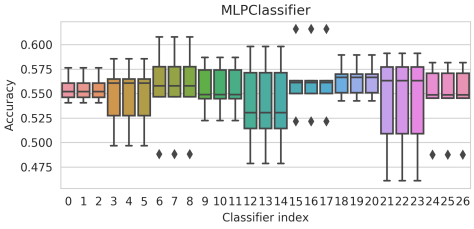
Figure A.1: SLD cross-validation accuracy



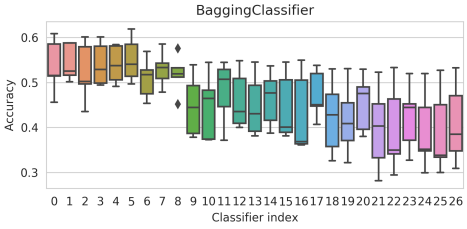
(a) Logistic regression



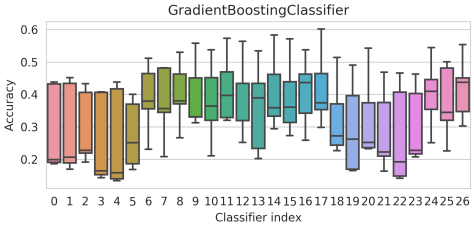
(b) KNN



(c) MLP

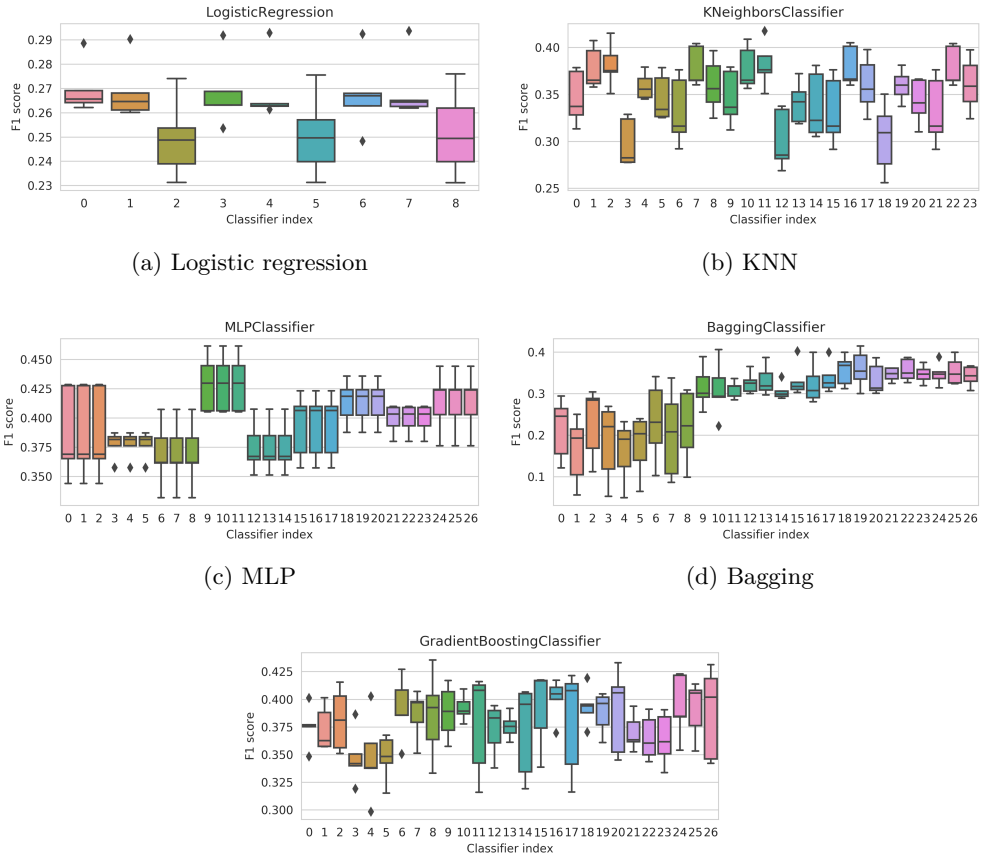


(d) Bagging



(e) Gradient Boosting

Figure A.2: Hostnames cross-validation accuracy



(a) Logistic regression

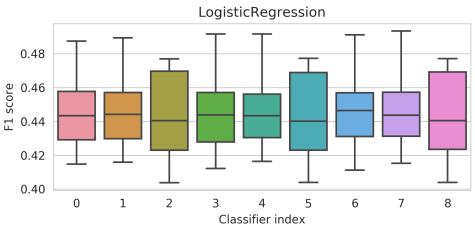
(b) KNN

(c) MLP

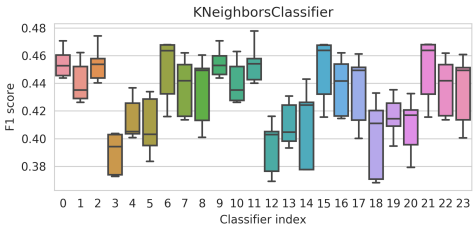
(d) Bagging

(e) Gradient Boosting

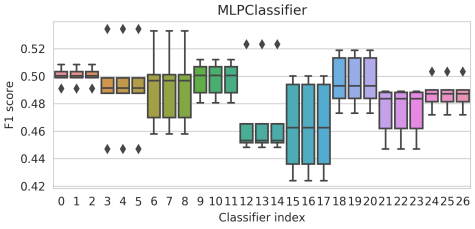
Figure A.3: SLD cross-validation F1-score



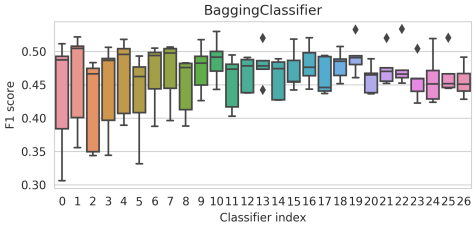
(a) Logistic regression



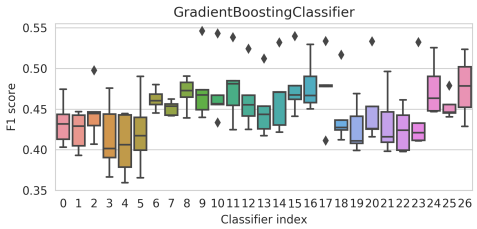
(b) KNN



(c) MLP

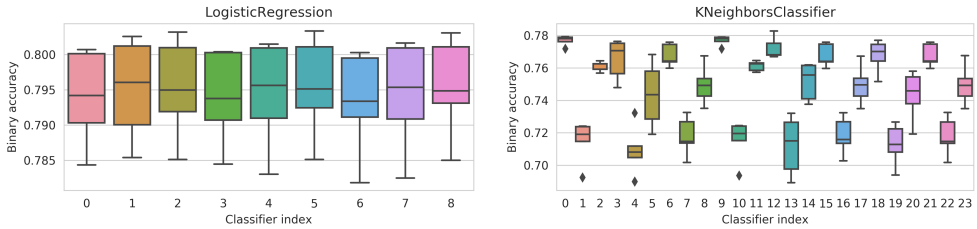


(d) Bagging



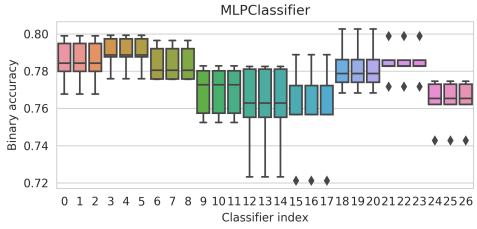
(e) Gradient Boosting

Figure A.4: Hostnames cross-validation F1-score

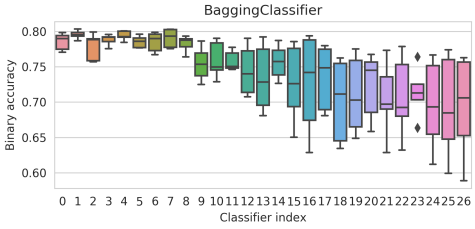


(a) Logistic regression

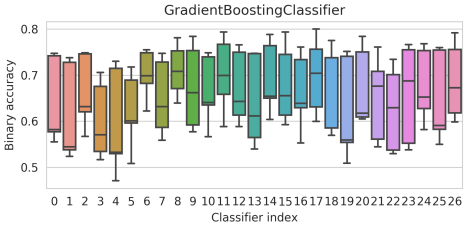
(b) KNN



(c) MLP

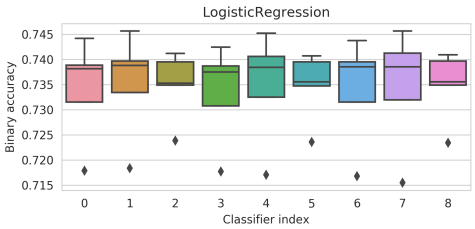


(d) Bagging

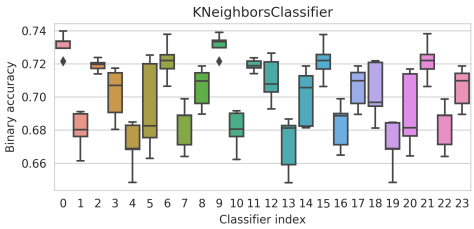


(e) Gradient Boosting

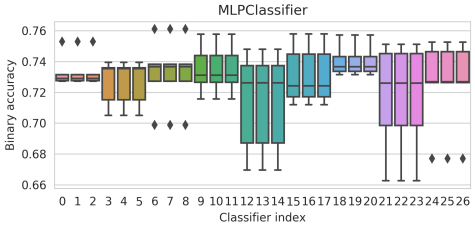
Figure A.5: SLD cross-validation binary accuracy



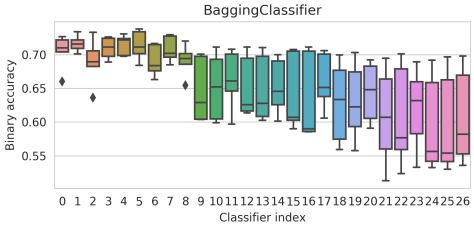
(a) Logistic regression



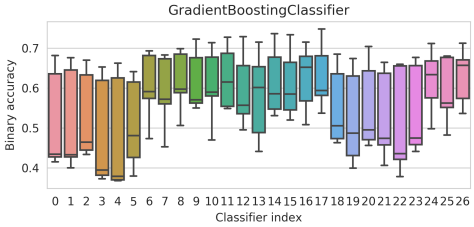
(b) KNN



(c) MLP

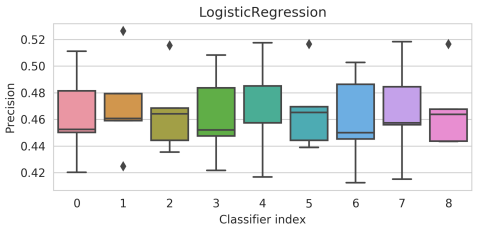


(d) Bagging

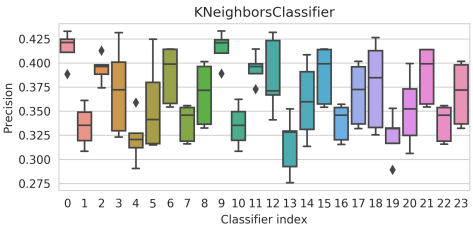


(e) Gradient Boosting

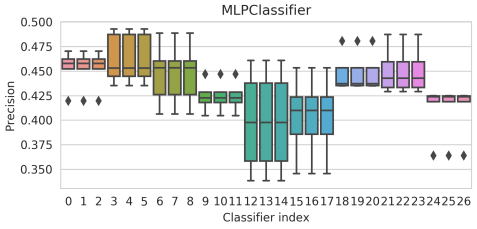
Figure A.6: Hostnames cross-validation binary accuracy



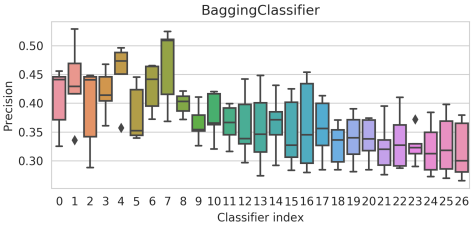
(a) Logistic regression



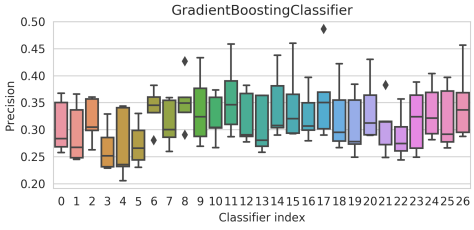
(b) KNN



(c) MLP

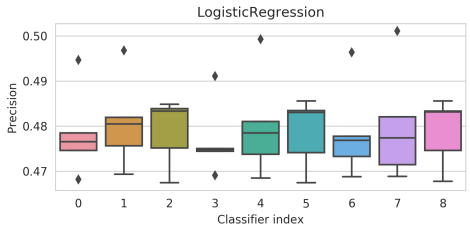


(d) Bagging

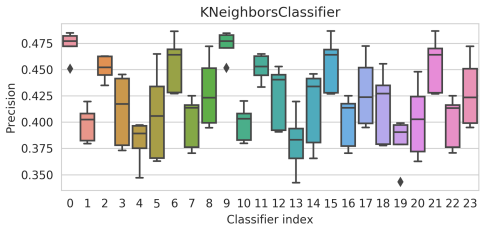


(e) Gradient Boosting

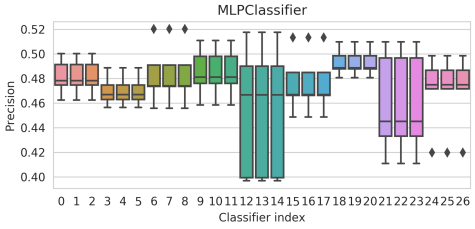
Figure A.7: SLD cross-validation precision



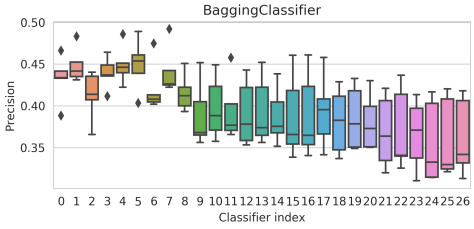
(a) Logistic regression



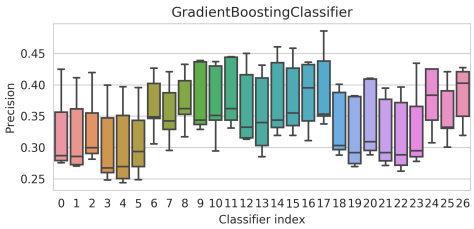
(b) KNN



(c) MLP

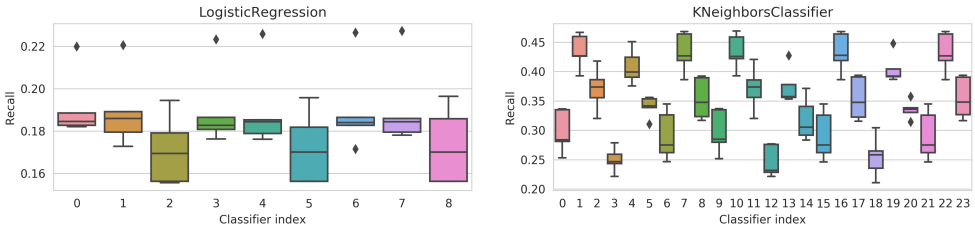


(d) Bagging



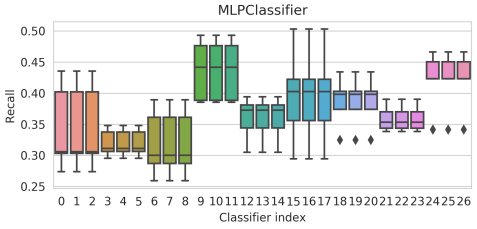
(e) Gradient Boosting

Figure A.8: Hostnames cross-validation precision

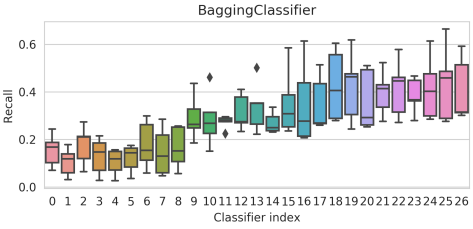


(a) Logistic regression

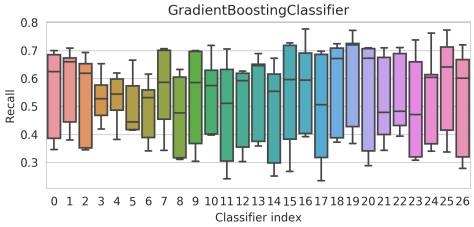
(b) KNN



(c) MLP

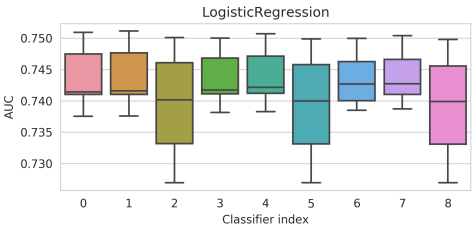


(d) Bagging

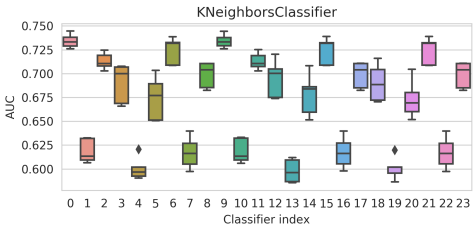


(e) Gradient Boosting

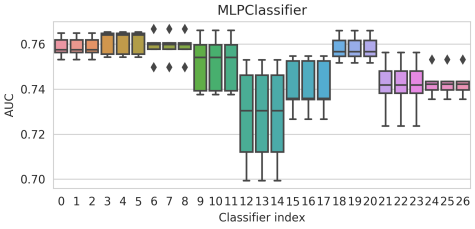
Figure A.9: SLD cross-validation recall



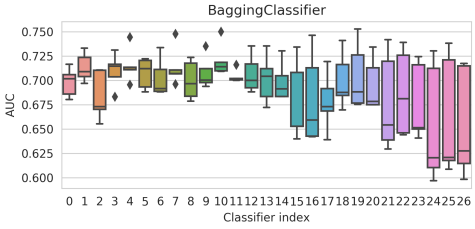
(a) Logistic regression



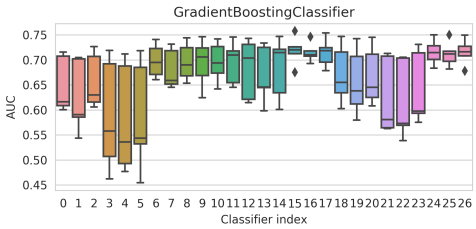
(b) KNN



(c) MLP



(d) Bagging



(e) Gradient Boosting

Figure A.10: Hostnames cross-validation recall

Appendix B

Files structure

- `thesis.pdf` ..... diploma thesis in PDF format
- `pdnsfeatex` ..... feature extraction module
  - `requirements.txt` ..... module dependencies
  - `setup.py` ..... installation script
  - `scripts` ..... folder with demo script
    - `feature_extractor.py` ..... demo script
  - `pdnsfeatex` ..... module source code
    - `spark_functions.py` ..... PySpark functions
- `pdnspred` ..... prediction module
  - `requirements.txt` ..... module dependencies
  - `setup.py` ..... installation script
  - `Pipfile` ..... config file for pipenv packaging
  - `bootstrap.sh` ..... script for environment initialization
  - `pdnspred` ..... module source code
    - `feature_transformer.py` ..... class for extracting new features
    - `hostname_resolver.py` ..... class containing methods specific for hostname prediction
    - `host_predictor.py` ..... main class for predicting hostnames
    - `pdns_base_predictor.py` base class for hostname/SLD predictors
    - `sld_predictor.py` ..... main class for SLD predictions
    - `sld_resolver.py` ..... class containing methods specific for SLD prediction
    - `tests/` ..... tests

Bibliography

[1] *Multinomial logistic regression.* https://en.wikipedia.org/wiki/Multinomial_logistic_regression. Accessed: 04.05.2021.

[2] 1CLOUD COMPANY BLOG. *What is DNS?* https://1cloud.ru/blog/chto-takoe-dns. Accessed: 04.05.2021.

[3] AFONJA, Tejumade. *Accuracy paradox.* https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b. Accessed: 04.05.2021.

[4] A.KORNIENKO, I.Slyusarenko. *Intrusion detection systems: state of the art and directions of improvement.* http://citforum.ru/security/internet/ids_overview/. Accessed: 04.05.2021.

[5] ALLEN, Julia, CHRISTIE, Alan, FITHEN, William, MCHUGH, John, and PICKEL, Jed. *State of the practice of intrusion detection technologies.* Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2000.

[6] ALOWAISHEQ, Eihal, TANG, Siyuan, WANG, Zhihao, ALHARBI, Fatemah, LIAO, Xiaojing, and WANG, XiaoFeng. *Zombie Awakening: Stealthy Hijacking of Active Domains through DNS Hosting Referral.* In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1307–1322. 2020.

[7] BHARDWAJ, Rupali and VATTA, Sonia. *Implementation of ID3 algorithm.* International Journal of Advanced Research in Computer Science and Software Engineering, 3(6), 2013.

[8] BILGE, Leyla, KIRDA, Engin, KRUEGEL, Christopher, and BALDUZZI, Marco. *EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis.* In Ndss, pp. 1–17. 2011.

[9] Black, Paul E. *Manhattan distance"" Dictionary of algorithms and data structures.* http://xlinux. nist. gov/dads//, 2006.

[10] Breiman, Leo. *Bagging predictors.* Machine learning, 24(2):123–140, 1996.

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

[12] Chomboon, Kittipong, Chujai, Pasapitch, Teerarassamee, Pongsakorn, Kerdprasop, Kittisak, and Kerdprasop, Nittaya. *An empirical study of distance metrics for k-nearest neighbor algorithm.* In Proceedings of the 3rd international conference on industrial application engineering, pp. 280–285. 2015.

[13] Cover, Thomas and Hart, Peter. *Nearest neighbor pattern classification.* IEEE transactions on information theory, 13(1):21–27, 1967.

[14] Cybenko, George. *Approximation by superpositions of a sigmoidal function.* Mathematics of Control, Signals and Systems, 5(4):455–455, 1992.

[15] Davis, Jesse and Goadrich, Mark. *The relationship between Precision-Recall and ROC curves.* In Proceedings of the 23rd international conference on Machine learning, pp. 233–240. 2006.

[16] Denning, Dorothy E. *An intrusion-detection model.* IEEE Transactions on software engineering, (2):222–232, 1987.

[17] Dietrich, Christian J, Rossow, Christian, Freiling, Felix C, Bos, Herbert, Van Steen, Maarten, and Pohlmann, Norbert. *On Botnets that use DNS for Command and Control.* In 2011 seventh european conference on computer network defense, pp. 9–16. IEEE, 2011.

[18] Espíndola, Rogério P and Ebecken, Nelson FF. *On extending f-measure and g-mean metrics to multi-class problems.* WIT Transactions on Information and Communication Technologies, 35, 2005.

[19] Fukuda, Kensuke and Heidemann, John. *Detecting malicious activity with DNS backscatter.* In Proceedings of the 2015 Internet Measurement Conference, pp. 197–210. 2015.

[20] Gallop, Robert J, Crits-Christoph, Paul, Muenz, Larry R, and Tu, Xin M. *Determination and interpretation of the optimal operating point for ROC curves derived through generalized linear models.* Understanding statistics, 2(4):219–242, 2003.

[21] Gasti, Paolo, Tsudik, Gene, Uzun, Ersin, and Zhang, Lixia. *DoS and DDoS in named data networking.* In 2013 22nd International Conference on Computer Communication and Networks (ICCCN), pp. 1–7. IEEE, 2013.

[22] Green, Ian. *DNS spoofing by the man in the middle.* 2005.

[23] Hlávka, Zdeněk. *CART Implementation Issues.* `https://www2.karlin.mff.cuni.cz/~hlavka/vyuka/past/CART_technical.pdf`. Accessed: 02.05.2021.

[24] Hornik, Kurt, Stinchcombe, Maxwell, and White, Halbert. *Multilayer feedforward networks are universal approximators.* Neural networks, 2(5):359–366, 1989.

[25] Impe, Koen Van. *How to Use Passive DNS to Inform Your Incident Response.* `https://securityintelligence.com/how-to-use-passive-dns-to-inform-your-incident-response/`. Accessed: 04.05.2021.

[26] Jurman, Giuseppe, Riccadonna, Samantha, Visintainer, Roberto, and Furlanello, Cesare. *Canberra distance on ranked lists.* In Proceedings of advances in ranking NIPS 09 workshop, pp. 22–27. Citeseer, 2009.

[27] Kambourakis, Georgios, Moschos, Tassos, Geneiatakis, Dimitris, and Gritzalis, Stefanos. *Detecting DNS amplification attacks.* In International workshop on critical information infrastructures security, pp. 185–196. Springer, 2007.

[28] Khalil, Issa, Yu, Ting, and Guan, Bei. *Discovering malicious domains through passive DNS data graph analysis.* In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 663–674. 2016.

[29] Kleinbaum, David G, Dietz, K, Gail, M, Klein, Mitchel, and Klein, Mitchell. Logistic regression. Springer, 2002.

[30] Konyukhov, Alexandr. *NetFlow, Cisco and Traffic Monitoring* . `https://habr.com/en/post/175359/`. Accessed: 04.05.2021.

[31] Korting, Thales Sehn. *C4. 5 algorithm and multivariate decision trees.* Image Processing Division, National Institute for Space Research–INPE Sao Jose dos Campos–SP, Brazil, 2006.

[32] Lemaître, Guillaume, Nogueira, Fernando, and Aridas, Christos K. *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning.* Journal of Machine Learning Research,

18(17):1–5, 2017.
URL http://jmlr.org/papers/v18/16-365.html

[33] Leung, Kelvin T and Parker, D Stott. *Empirical comparisons of various voting methods in bagging.* In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 595–600. 2003.

[34] Liberti, Leo, Lavor, Carlile, Maculan, Nelson, and Mucherino, Antonio. *Euclidean distance geometry and applications.* SIAM review, 56(1):3–69, 2014.

[35] Maszke, Szymon. *What is the Search/Prediction Time Complexity of Logistic Regression?* https://stackoverflow.com/questions/54238493/what-is-the-search-prediction-time-complexity-of-logistic-regression/54239814. Accessed: 02.05.2021.

[36] Moubayed, Abdallah, Injadat, MohammadNoor, Shami, Abdallah, and Lutfiyya, Hanan. *Dns typo-squatting domain detection: A data analytics & machine learning based approach.* In 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–7. IEEE, 2018.

[37] Natekin, Alexey and Knoll, Alois. *Gradient boosting machines, a tutorial.* Frontiers in neurorobotics, 7:21, 2013.

[38] Nazario, Jose and Holz, Thorsten. *As the net churns: Fast-flux botnet observations.* In 2008 3rd International Conference on Malicious and Unwanted Software (MALWARE), pp. 24–31. IEEE, 2008.

[39] Onan, Aytuğ, Korukoğlu, Serdar, and Bulut, Hasan. *A multiobjective weighted voting ensemble classifier based on differential evolution algorithm for text sentiment classification.* Expert Systems with Applications, 62:1–16, 2016.

[40] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12:2825–2830, 2011.

[41] Powers, David MW. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.* arXiv preprint arXiv:2010.16061, 2020.

[42] RADER, Ross. *One history of DNS.* Byte. org, 2006.

[43] Rosenblatt, Frank. *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review, 65(6):386, 1958.

[44] Russell, Stuart and Norvig, Peter. *Artificial intelligence: a modern approach.* 2002.

[45] Safavian, S Rasoul and Landgrebe, David. *A survey of decision tree classifier methodology.* IEEE transactions on systems, man, and cybernetics, 21(3):660–674, 1991.

[46] Sibi, P, Jones, S Allwyn, and Siddarth, P. *Analysis of different activation functions using back propagation neural networks.* Journal of theoretical and applied information technology, 47(3):1264–1268, 2013.

[47] Singh, Archana, Yadav, Avantika, and Rana, Ajay. *K-means with Three different Distance Metrics.* International Journal of Computer Applications, 67(10), 2013.

[48] Son, Sooel and Shmatikov, Vitaly. *The hitchhiker's guide to DNS cache poisoning.* In International Conference on Security and Privacy in Communication Systems, pp. 466–483. Springer, 2010.

[49] Sonoda, Sho and Murata, Noboru. *Neural network with unbounded activation functions is universal approximator.* Applied and Computational Harmonic Analysis, 43(2):233–268, 2017.

[50] Terzi, Duygu Sinanc, Terzi, Ramazan, and Sagiroglu, Seref. *Big data analytics for network anomaly detection from netflow data.* In 2017 International Conference on Computer Science and Engineering (UBMK), pp. 592–597. IEEE, 2017.

[51] Torabi, Sadegh, Boukhtouta, Amine, Assi, Chadi, and Debbabi, Mourad. *Detecting Internet abuse by analyzing passive DNS traffic: A survey of implemented systems.* IEEE Communications Surveys & Tutorials, 20(4):3389–3415, 2018.

[52] Vorontsov, Konstantin. *Classification.* `https://bit.ly/2Shut1h`. Accessed: 04.05.2021.

[53] Vorontsov, Konstantin. *Logistic Regression.* `https://bit.ly/3ulbMb3`. Accessed: 04.05.2021.

[54] Vorontsov, Konstantin. *Nearest neighbors method.* `https://bit.ly/3ekHZKd`. Accessed: 04.05.2021.

[55] Weimer, Florian. *Passive DNS replication.* In FIRST conference on computer security incident, vol. 98. 2005.

[56] Zhou, Kun, Hou, Qiming, Wang, Rui, and Guo, Baining. *Real-time kd-tree construction on graphics hardware.* ACM Transactions on Graphics (TOG), 27(5):1–11, 2008.