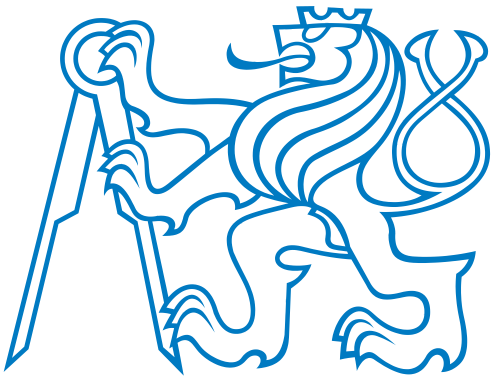
master's thesis

Passive NAT detection using HTTP logs

Tomáš Komárek



May 2015

Supervisor: Ing. Martin Grill

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Control Engineering

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Tomáš Komárek**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Diploma Thesis: **Passive NAT detection using HTTP access logs**

Guidelines:

1. Analyse the problem of detection NAT hosts in the computer networks.
2. Review the state of the art of the NAT host detection.
3. Design an algorithm that addresses the problem of NAT host detection using information stored in HTTP access log only.
4. Evaluate performance of the proposed algorithm on real network data.

Bibliography/Sources:

[1] Richard O. Duda, Peter E. Hart, David G. Stork: "Pattern Classification"
[2] Christopher M. Bishop: "Pattern Recognition and Machine Learning"
[3] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, Hsuan-Tien Lin: "Learning From Data"
[4] L. Rui, Z. Hongliang, X. Yang, L. Shoushan, Y. Yixian, and W. Cong: "Remote NAT detect Algorithm Based on Support Vector Machine"

Diploma Thesis Supervisor: Ing. Martin Grill

Valid until summer semester 2015/2016

L.S.

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 20, 2015

Acknowledgment

First and foremost, I would like to acknowledge my supervisor Ing. Martin Grill for his interest, help and guidance throughout my diploma thesis. In addition, I would like to thank Ing. Tomáš Pevný, Ph.D. for his helpful suggestions and comments during this work. Special thanks go to all the people at CISCO, for providing a nice and friendly working environment. Last but not least, I would like to thank my girlfriend Teraza and my brother Lukáš who inspire me all the time.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

Abstract

Network devices performing NAT prove to be a double edge sword. They can easily overcome the problem with the deficit of IPv4 addresses as well as introduce a vulnerability to the network. Therefore detecting NAT devices is an important task in the network security domain. In this thesis, a novel passive NAT detection algorithm is proposed. It infers NAT devices in the networks using statistical behavior analysis of HTTP logs. These network traffic data are often already collected and available at proxy servers, which enables the wide applicability of the solution. On the basis of our experimental evaluations, proposed algorithm detection capabilities are better than the state-of-the art NAT detection approaches.

Abstrakt

Síťová zařízení umožňující nativní překlad adres (NAT zařízení) se ukazují býti dvojsečná. Mohou obejít problém s nedostatkem veřejných IP adres stejně snadno jako způsobit zranitelnost sítě. Z toho důvodu je detekce NAT zařízení důležitou úlohou síťové bezpečnosti. V této diplomové práci je navržen detekční algoritmus odhalující NAT zařízení v počítačových sítích na základě statistické analýzy chování uživatelů sítě z HTTP proxy záznamů. Skutečnost, že tyto síťové záznamy jsou již sbírané a běžně dostupné na proxy serverech, umožňuje širokou aplikovatelnost tohoto řešení. Na základě provedených experimentů, detekční schopnosti navrženého algoritmu překonávají všechna současná řešení.

Contents

Chapter 1

Introduction

The number of Internet connected devices is continuously growing. It is estimated that over one hundred new devices is connected to the Internet every second. Each such device needs to be associated with an IP address to be able to communicate with others on the Internet. Looking to the future especially with respect to the upcoming phenomena Internet of Things, it is evident that even more and more devices like watches, fridges and cars will need IP addresses. However, transition to the modern Internet Protocol version 6 (IPv6) with sufficiently large address space seems to be struggling due to lack of backward compatibility with existing Internet Protocol, known as IPv4. As result, IPv4 still carries the vast majority of Internet traffic and the problem with almost depleted IPv4 address pool is curred with Network Address Translation (NAT) devices.

By using NAT, it is possible to hide a complete local network behind a single IP address. The local network then gives the impression of being just one network device from the public network perspective. This is used not only to solve the problem with running out of public IP addresses, but also to hide inner network topology, provide anonymity, filter content, monitor network performance, etc. Nevertheless this technique does not just provide benefits. For example, employees of companies might setup their own unauthorized

NAT devices to share Internet connection among workstations and mobile devices. Setting up such devices in that environments is considered as a major security threat, because these devices are out of the scope of any security policies and might provide an easy exploitable point for conducting malicious activity. It is therefore necessary for network administrators to have an overview of devices performing NAT in the networks to be able to successfully prevent from potential industrial espionage or any other cyber-attacks against the corporate networks. Taking this into account, there is a need for a solution that is capable of detecting NAT devices.

This thesis addresses the problem of NAT detection using information contained within HTTP logs. The structure is as follows. Chapter 2 describes the problem together with the herein used HTTP logs. Chapter 3 provides a brief survey of existing solutions and highlights their main drawbacks. In Chapter 4, a NAT detector based on statistical behavior analysis is proposed. Chapter 5 reports the performance of the proposed solution using real network data. Final Chapter 6 concludes the thesis after summarizing the results.

Chapter 2

Problem statement

In this chapter we describe the problem of NAT detection in more detail. Afterward, we present available data resources and specify requirements of the detection algorithm.

2.1 Network Address Translation

Network Address Translation (NAT) is a technique that allows a single network device (e.g., a router) to act as an agent between public network (e.g., the Internet) and private (local) network [rfc, 1994]. It assigns a public Internet Protocol (IP) address to a host or a group of hosts inside a private network. Consequently, only a single public IP address is needed to represent an entire group of hosts. This allows private networks that use unregistered IP address to connect to the Internet. In doing so, a device running NAT translates source addresses in IPv4 packet headers [rfc, 1981] of hosts inside the private network to a globally unique IP address before the packets are forwarded to public network. The returning IP packets go through similar address translation process to reach the corresponding recipient.

To be able to match responses from public network with individuals in the private network, outgoing packet headers from the private network have also

replaced source ports in addition to source IP addresses. In the NAT device memory, there is a stored address translation table, which maps host's private IP address and source port to the assigned public address and port. This table is used for translation process during packet forwarding. If no match is found in the table, the corresponding packet is dropped.

The main reason to use NAT is to overcome a deficit of public IP addresses. However, there are other justifiable reasons such as hiding inner network topology, providing anonymity, content filtering, monitoring network performance, etc. to install a NAT device[1] (or more generally a NAT software) at exit points between the private and public network. On the other hand, unauthorized NAT devices in the network present a significant network security threat because they may provide unrestricted access to the network. Wireless NATs (e.g., Wi-Fi routers) pose a particular security risk because they may allow unauthorized access to the network from considerable distances without wired connections. As such, network administrators should be informed about devices which potentially perform NAT in the network. Moreover, it is also important for purposes of advanced network behavior analysis to be able to identify NAT devices. In order to detect modern network security threats, such systems build statistical models of hosts behavior in the network based on network traffic and report deviations from the models. In these systems the behavior of NAT devices has to be, however, modeled separately. As the traffic generated by a NAT is actually mixture of a behaviors of inner individual hosts connected to the NAT.

[1] As used herein, the term "NAT device" means any entity or instance that performs NAT.

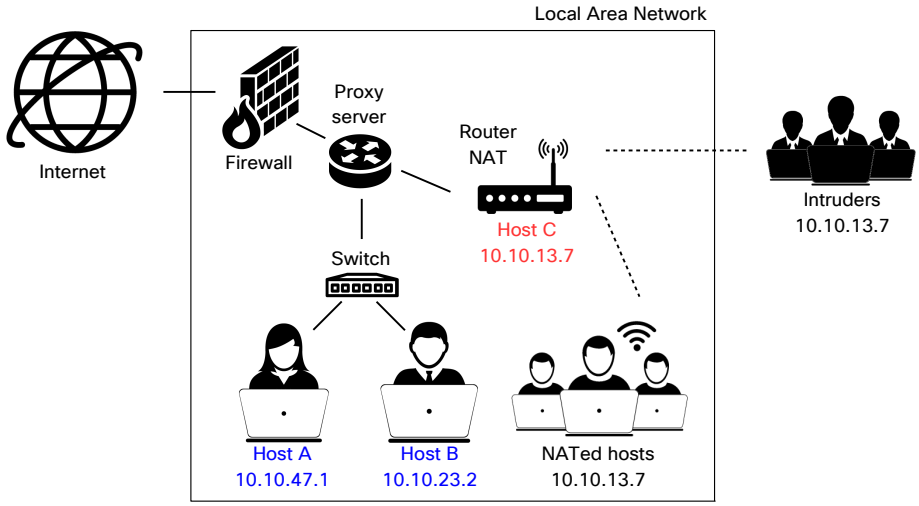


Figure 2.1: An example of network environment with a NAT device.

Figure 2.1 shows a simplified network topology. The network environment consists of a local area network (LAN) communicating with the Internet through a firewall. The LAN includes a proxy server, a network switch, a router performing NAT and several hosts in addition to the firewall. The proxy server acts as an intermediary or focal point for network traffic to/from the LAN. Thus, the proxy server has a comprehensive view of network traffic, which is utilized to monitor and capture logs of that network traffic. The captured logs are called proxy logs and their structure is described in the following Section 2.2. On the basis of proxy logs, the detection algorithm proposed in this thesis classifies each host identified in the logs as either a NAT device or an end host (i.e., non-NAT device). Other used names for the NAT device in this thesis are: gateway or positive sample/instance. Conversely, regular host/user or negative sample/instance are alternative terms for the end host. Returning again to Figure 2.1, Hosts A and B are examples of end hosts as each of them represents one device (e.g., user's computer, network printer, etc.) with its own IP address, unique within the LAN. In contrast to Host A and B, Host C can not be considered as an end host, because it actually includes more (wirelessly) connected devices (e.g., laptops of several employees). These NATed hosts are indistinguishable in proxy logs as they

are represented by a single IP address of Host C. Consequently, Host C is considered to be a NAT device. The illustrated intruders attempt to gain an unauthorized access into the LAN through the NAT device. In case of success, their malicious activity will blend in with the traffic of Host C (i.e., with all NATed hosts). Such type of malicious activity is hard to discover due to the presence of background traffic.

2.2 Proxy logs

As mentioned before, proxy servers such as [Squ, 2015] or [HAP, 2015] are capable of capturing network traffic logs. More specifically, Hypertext Transfer Protocol (HTTP) [rfc, 1999] proxy logs (also referred as HTTP logs) are collected by these servers. In Table 2.1, there is an example of HTTP proxy logs. For illustrative convenience, the table is split into two table portions. Each line represents a HTTP request made by a host to a web server. It contains the host IP address, the IP address and domain of the server, Uniform Resource Locator (URL) of the request (only for non-HTTPS requests), the User-Agent information (it identifies host's software originating the request), the sum of uploaded/downloaded bytes, the starting time of communication and its duration. Then, there is also Referrer field which contains the information about who referred the host to make the request to that particular server with that particular URL. Furthermore, there can be additional fields, like the host/server source port, HTTP status or HTTP method, depending on the configuration of the proxy server. All these fields are extracted from HTTP headers. Therefore, proxy logs only provide information about communication and not other types of network protocols. To avoid any misunderstanding, the records only store meta-data about the transferred packets. They do not include any payload data sent by the host or returned by the server apart from the transferred volumes. Besides that these records are often already collected and available at proxy servers.

For development purposes of this thesis, we are provided with annonymized

Timestamp	Duration	Method	Server IP	Server port	Host IP	Host port	URL
1424851268	20	GET	198.35.26.112	80	10.148.144.137	40003	http://en.wikipedia.org/wiki/
1424851253	120	GET	204.79.197.200	80	10.148.144.169	40475	ts4.mm.bing.net//th?q=proxy+server
1424851223	200	GET	173.194.40.121	80	10.148.144.211	40005	http://www.nytimes.com/
1424851423	50	GET	23.14.92.64	80	10.148.144.236	40477	https://www.facebook.com/
1424851899	10	GET	46.228.47.114	80	10.148.144.143	56497	https://www.yahoo.com/

User-Agent	Referrer	Uploaded bytes	Downloaded bytes
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)	-	10	130
Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0	http://bing.com/img/src?q=proxy+server	15	826
Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0	-	17	180
Opera/9.80 (Windows NT 6.0) Presto/2.12.388 Version/12.14	-	24	290
Mozilla/5.0 (X11; Linux i686; rv:10.0.7) Gecko/20100101 Iceweasel/10.0.7	-	32	410

Table 2.1: An example of five HTTP proxy logs.

HTTP proxy logs from three working days of four different networks. These networks are corporate networks (also referred to simply as companies) of various sizes operating on distinct fields of industry. Approximate numbers of hosts inside the networks are: 6 000, 12 000, 25 000 and 61 000.

2.3 Design requirements

In summary, each host in the network may be either a NAT device, or an end host device. As this is not inherently clear and simultaneously crucial from the reasons mentioned above, NAT detection is important and challenging task in the network security domain. The goal of this thesis is to develop a NAT detector that is able to detect NAT devices using the information contained in the above described proxy logs. Herein is a list of the main requirements, which the NAT detector should fulfill:

1. classify each host identified in proxy logs as end host, or NAT device,
2. real time processing of input proxy logs,
3. results available within 24 hours of the starting time,
4. focus on high classification precision.

Additionally, the NAT detector should cope with high speed networks with more than three thousand hosts. Hence the computational demands should be as low as possible. It should be noted that from the nature of data, we are not able to detect NAT devices in the network, that do not produce any

HTTP requests. However, with reference to the fact that HTTP is one of the most used network protocols among users, existence of such NAT devices is improbable. Note also that there is no attempt to identify individual NATed hosts connected to the NAT device. We are interested solely in detecting whether a particular IP address occurring in proxy logs is actually end host or not.

Chapter 3

State of the Art

An overview of existing solutions is presented in this section. Since network security still continues to be a hot topic in the market place, many scientific studies in the area of NAT detection have been done in recent years. Although we do not go into the details of all works, we would like to show main ideas as well as drawbacks of majority of them.

In general, there are two main approaches in order to infer concealing NAT devices in the network: *active* and *passive*. We will focus primarily on *passive detection techniques* which unlike the active ones do not generate any network traffic by sending data packets. They try to identify NAT devices just by observing normal data flow in the network. Moreover these techniques are undetectable, non-intrusive and capable to analyze stored historical records of traffic retrospectively. On the other hand, they require convenient monitoring point in the network to be able to capture communication of every host. Meanwhile active tools such as Nmap [Lyon, 2006] [1] can be run anywhere in the network. The passive techniques can be divided into the three following categories according to the used source of data for detection: TCP/IP packet headers (Section 3.1), TCP/IP packet payloads (Section 3.2) and NetFlow records (Section 3.3).

[1] Probably the most used open-source software for this purpose.

3.1 TCP/IP packet headers

Methods of this class are based solely on analyzing fields from Transmission Control Protocol and Internet Protocol (TCP/IP) packet headers. They do not use any information contained in a packet payload. This property can be important from privacy point of view. The below listed paragraphs briefly describe some TCP/IP header fields and their combinations which can serve for the NAT detection purpose.

IP TTL is an eight bit length field in IPv4 packet headers which limits packet's lifetime to prevent it from persisting (e.g., going in circles) on Internet. It works as a hop count. Each time the packet arrives at a router, its TTL value is decremented by one. When the value reaches zero, the packet is discarded. One method for detecting NAT devices, well described for example by [Phaal, 2009], is based on observation that operating systems have characteristic initial TTL values. The set of the unique initial values is relatively small. According to [Miller, 2008], almost all operating systems from Windows family have initial TLL value set to 128 by default. Once it is known that NAT devices (e.g., routers or gateways) decrement the value during packet forwarding, we can infer a host behind a NAT device upon seeing its packet with TTL value 127. There are two major disadvantages of this method. The initial values can be easily changed in computer settings and routers reconfigured to not decrement TTL. Hence it does not make any problem for intruders to hide a malicious NAT device in the network when this method is deployed.

IP ID is an identification field of IP packet headers allowing packet fragmentation and reassembling. Fragmentation is used when one network wants to transmit packets to another with a smaller maximum transmission unit. Each packet is splitted into smaller chunks called fragments. The IP ID uniquely identifies a group of these fragments to make reassembling possible on the receiver side. As the field ensures uniqueness of each packet, its value is changed every time a packet is sent from source to destination. An

algorithm for counting NATed hosts proposed by [Bellovin, 2002] is based on observing patterns in produced IP ID sequences. It turns out that some Operating Systems (OSs) such as Windows and FreeBSD implement it as a linear congruential counter (mod 2^{16}), which generates sequences of consecutive numbers. Thus, the total count of observed sequences from one IP address should correspond to the number of users behind the address. However, the algorithm does not work for OS where the IP ID is generated in a random manner (MacOS, OpenBSD) or as a per flow counter (current Linux distributions). In addition, if Don't Fragment bit (DF) is set to one, reassembly is not necessary and some devices reset IP ID to zero. Both issues render the technique as inappropriate for network security purposes.

IP TTL + DF + TCP window size + SYN packet size. Values of these fields are characteristic among different operating systems or even its versions and thus can be used to create an OS signature as suggested by [Miller, 2008]. *OS fingerprinting* is the name of a passive technique inferring host's OS from its collected TCP/IP communication. Unlike the popular freeware p0f tool [Zalewski, 2012] using rule-based matching, [Beverly, 2004] trained a Bayesian classifier to determine host's OS based on these packet header fields. The main advantage of the statistical model over the rule-based matching is ability to select an OS with the highest posterior probability, although there is no an exact signature match. Generally speaking, the best guess is always determined with respect to available training data. In application of NAT detection, a host with more detected OSs might indicate a NAT device. Nonetheless, this technique fails to detect NATed hosts with the same OS. Unfortunately, network hosts with one identical OS are commonly observed in corporate environments. Therefore, the method can be not recommended as a stand-alone solution for network administrators.

TCP timestamps in IPv4 headers help to determine the order of received packets. [Kohno et al., 2005] proposed a powerful way how to remotely fingerprint user's device with the timestamps. It exploits the fact that modern computer chips have small yet remotely detectable clock skews. To estimate

device's clock skew, the TCP timestamp option from outgoing packets is utilized. This technique is capable of identifying user's device not only masquerading behind a NAT device, but also changing IP addresses over time. Unfortunately, it suffers from the fact that TCP timestamp option can be disabled. Moreover, the option is disabled in Windows 2000 and Windows XP by default. As such, the technique is impractical in real-world applications dealing with NAT detection.

IP ID + TCP sequence number + TCP source port. The 32-bit TCP sequence number is generated at the beginning of each TCP connection establishment. The 16-bit TCP source port is assigned by a client computer when it tries to establish a TCP connection. Together with the already presented IP ID, the behavior of these three IP header fields is characteristic among popular OSs such as Windows, Linux, FreeBSD and MacOS. In the work of [Mongkolluksamee et al., 2012], behaviors of these fields were studied through popular OSs and the previously mentioned [Bellovin, 2002]'s method was extended by the newly discovered patters. The introduced algorithm can serve for identifying individuals as well as their OS. Nevertheless, it still fails to detect OpenBSD hosts because they implement all fields in a random manner.

3.2 TCP/IP packet payloads

In contrast to methods from previous category, the next two methods take advantage of packet payloads. Both methods inspect HTTP communication in order to uncover a NAT device in the network.

HTTP cookie is a small piece of data delivered by a web page and stored in user's browser. HTTP cookies typically contain an unique identifier allowing the web page to determine an user visiting the same page again. In some sense, it helps to overcome the stateless of HTTP protocol. [Bai Xue, 2009] developed an algorithm which tries to detect a host accessing the same web page with different cookie identifiers, because in reality it might be more users hidden behind a NAT device. Disadvantage of this technique is poor

performance and the fact that cookies can be disabled.

HTTP User-Agent is a string carrying information about user's client (e.g., browser, mail client, media player, etc.) connecting to the server via HTTP protocol. User-Agent string of a regular browser usually contains its name, version, OS family including version and potentially plugged extensions. This knowledge about a host in addition to its initial IP TTL value were invoked in a NAT analysis tool created by [Maier et al., 2011]. They assumed that it is rather unlikely to see different versions of the same browser or OS family on the same host, whereas having more different browsers running on one OS is quite common. They also showed that User-Agent feature is more valuable than IP TTL. However, the tool cannot distinguish hosts with the same browser and OS. This situation is usual in networks with homogeneous OS/software installations (e.g., banking and financial industry, public administration and some corporate environments).

3.3 NetFlows records

The last category includes techniques based on analyzing NetFlow records. A NetFlow is an aggregate of packets which have source/destination IP address, source/destination port and type of protocol in common. The aggregate contains information about the source/destination IP address and port, the number of transferred packets, the sum of bytes transferred by all packets, duration of the communication and other information. As many routers and switches are capable of generating NetFlows, it is convenient to use the data for purposes of the network analysis. Moreover, these techniques do not require any deep packet inspection. As such, privacy of monitored users is respected. In [Krmicek et al., 2009], a NetFlow based system for NAT detection is introduced. However, the system is analogous to methods from Section 3.1. It takes advantages of the similar observations about TCP/IP header fields and combines them together in order to get a more robust detector.

In contrast, the following two techniques collect statistics (derivable from NetFlow records) about host's behavior in the network. The statistics are gathered within a time window of a predefined length. Afterwards, an optimal decision is made according to a learned knowledge on previously analyzed data. Strictly speaking, it is a behavior oriented pursuit and not signature oriented. We consider these works to be the most relevant to our one as we also utilize machine learning approach. However, the NAT detector proposed in this thesis collects statistics from HTTP proxy logs. Note that the inceptions of the below paragraphs correspond to names of the used machine learning algorithms.

SVM denotes Support Vector Machine learning algorithm. [Rui et al., 2009] proposed to use SVM and Directed Acyclic Graph SVM to detect and estimate the number of hosts connected behind a NAT device. Eight features consisting of statistics of transferred IP packets as well as of a subset of flags of the TCP headers are collected for each host in the network every two minutes. Activity of a particular host is then represented by a series of these feature vectors. To train and evaluate algorithms, 1 637 550 packets of five hosts were collected in a lab. One of the hosts under observation was not placed behind a NAT device (436 320 packets), while the remaining four hosts generated NAT traffic (1 201 230 packets). Thus, approximately 75% of the traffic used for SVM training and testing was NAT traffic. By filtering low-volume entries and applying SVM with a Radial Basis Function (RBF) kernel, [Rui et al., 2009] achieved a maximum detection accuracy around 85% in the binary NAT/no-NAT detection task.

However, it can be objected that the used dataset is relatively small compared to sizes of real networks. Additionally, it is not entirely clear from the article what kind of network activity hosts produced. Regardless the fact that a lab is a very artificial environment not fully capable to simulate behavior of thousands of real network hosts.

C4.5 is an learning algorithm used to generate a decision tree models. It

yielded the best results in terms of NAT detection accuracy in the research made by [Abt et al., 2013]. Their approach relies on nine distinct features extracted from NetFlow records. Namely, the number of TCP, UDP, DNS, SMTP records, the number of records belonging to email protocols, the number of records with SYN and RST set flag, total number of packets and transmitted bytes are collected for each host separately within 120 seconds period window. These selected features are expected to be highly dependent on user specific behavior and thus able to distinguish end hosts from NAT devices. For training, validation and testing purposes, they used 6 631 383 anonymized NetFlow records from real-world environment (with majority of business customers) provided by German Internet Service Provider (ISP). NetFlow records were labeled according to expert knowledge of the sponsoring ISP. As the ISP had been providing managed services, it was able to label the NetFlow records based on the IP addresses of their customers. The achieved lower-bound accuracy on a balanced data set (the same amount of both classes) is 89.39% which outperforms the previous approach of [Rui et al., 2009].

To conclude, the last work represents our most relevant competitor. The only reproach is that the sizes of inner networks were not specified at all. Detection of huge gateways with many hosts behind is easier task than revelation of small NAT devices. From security point of view, these small devices performing NAT might, however, be more important. As they can allow unauthorized access to the network which is hard to discover.

Chapter 4

NAT detector

The aim of this chapter is to design an algorithm which would meet the requirements specified in Chapter 2.3. In the previous Chapter 3, where the related work was discussed, we have seen that the only method using data available in HTTP headers is based on the single static rule: more OS/browsers[1] indicate a NAT device. This technique suffers from the incapability to detect a NAT device with NATed hosts having the same OS and/or web browser, which is characteristic for the majority of networks in the banking and financial industry, public administration and corporate environments where the default setup of computers is enforced. In contrast, we employ a machine learning approach based on statistical behavior analysis. Briefly speaking, we let the machine reveal a hidden pattern in data instead of trying to find the rule by ourselves. Hence the first section Supervised learning 4.1 gives a short introduction to the learning problem. The section Problem formulation 4.2 uses the introduced framework to formulate the NAT detection task. However, even without an explanation, it is intuitive that the output performance depends heavily on quality of provided data. This exposes a classical problem of insufficient amount of labeled data in the network security domain as manual labeling is expensive and in some cases even impossible. Unfortunately our

[1] as parsed from User-Agent strings

situation is no exception, because even network administrators are not sure whether a host is end host or NAT device in every time. Moreover, the data are imbalanced as the amount of positive instances (NAT devices) is small compared to the total number of negative ones (end hosts). Nevertheless, in the section Artificial NATs 4.3 we overcome both mentioned issues in an elegant way. Pre-processing section 4.4 dedicates to preparing convenient data sets ready for training. The following Data analysis section 4.5 visualizes inner properties of the data. The section Classification 4.6 aims at training and selecting the most appropriate model. Finally, the section Structure of the detector 4.7 presents the proposed solution.

4.1 Supervised learning

Lets suppose that we have collected examples from which we would like to learn a general rule. More formally, we are given *training data* $\mathcal{D} = \{d_i : d_i = (\mathbf{x}_i, y_i)\}_{i=1}^{N}$ which consists of N *training examples* d_i. We assume that the *feature vectors* $\mathbf{x}_i$ are drawn independently from an unknown *input probability distribution* $\mathrm{P}(\mathbf{x})$ on $\mathcal{X} \subset \mathbb{R}^n$ and their *labels* y_i are computed from $y_i = \mathrm{f}(\mathbf{x}_i)$. Here $\mathrm{f} : \mathcal{X} \to \mathcal{Y}$ is called the *target function*, and it is also assumed to be unknown. With the given training data $\mathcal{D}$, we want to gain a function $\mathrm{h}^* : \mathcal{X} \to \mathcal{Y}$ as our inference of the target function f. The function h^* is usually chosen from a collection $\mathcal{H}$ of candidate functions, called *hypothesis* or *learning models*. Briefly speaking, the task of the learning from labeled examples, also known as *supervised learning*, is to use the information in the training data $\mathcal{D}$ to find some $\mathrm{h}^* \in \mathcal{H}$ that approximates f well.

If we admit noise in labeling (i.e. two identical feature vectors can have distinct labels), the *target distribution* $\mathrm{P}(y|\mathbf{x})$ is used instead of $y = \mathrm{f}(\mathbf{x})$. This modification is done without loss of generality, because the target function can be seen as a special case of the target distribution, where $\mathrm{P}(y|\mathbf{x})$ is zero except for $y = \mathrm{f}(\mathbf{x})$. Now, the training examples $d_i = (\mathbf{x}_i, y_i)$ are generated by the joint probability distribution $\mathrm{P}(y, \mathbf{x}) = \mathrm{P}(y|\mathbf{x})\mathrm{P}(\mathbf{x})$.

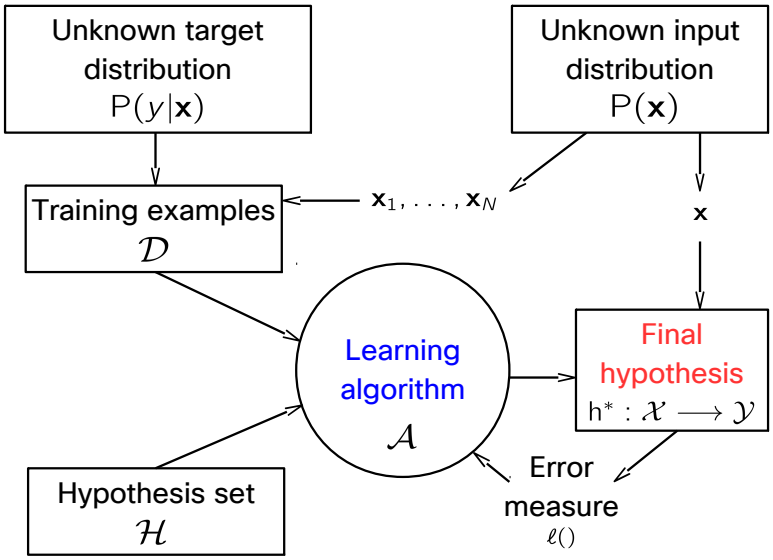


Figure 4.1: Learning diagram.

The above Figure 4.1 illustrates the supervised learning process. A *learning algorithm* $\mathcal{A}$ searches for the right hypothesis h^* from the hypothesis set $\mathcal{H}$ minimizing the expected out-of-sample error[2] $E_{\text{out}} = \mathsf{E}_{\mathcal{Y},\mathcal{X}}\left[\ell(y, \mathsf{h}^*(x))\right]$, where E is the expected value and ℓ is a *loss function*. Selection of the loss function depends on the particular application domain. It expresses how much we are paying for predicting $\mathsf{h}(\mathbf{x})$ in place of y. In a *regression problem* $\mathcal{Y} \subset \mathbb{R}$ (i.e., labels constitute a subset of real numbers), a common choice is the *square loss* $\ell_{\text{square}} \equiv (y - \mathsf{h}(\mathbf{x}))^2$. For a *classification problem* $\mathcal{Y} \subset \mathbb{Z}$ (i.e., labels are typically small integers representing classes) a natural loss function is the *zero-one loss* $\ell_{0-1} \equiv \mathbb{1}[y \neq \mathsf{h}(\mathbf{x})]$,[3] which penalizes all types of misclassification equally.

It is worth mentioning that although the zero-loss is convenient for evaluation purposes, it makes the optimization problem of minimizing the out-of-sample error intractable from a computational perspective. Later in Section 4.6, we will introduce learning algorithms using convex functions instead, such as the

[2] E_{out} is also known as the Bayes risk.
[3] $\mathbb{1}[\cdot]$ denotes the indicator function.

logistic loss $\ell_{log} \equiv \log(1 + e^{-z})$ and the *hinge loss* $\ell_{hinge} \equiv \max(0, 1 - z)$. The convexity property is computationally appealing and guarantees that, if we find a minimum, it is global. Here the z variable indicates a measure of accordance between a hypothesis and a true label.

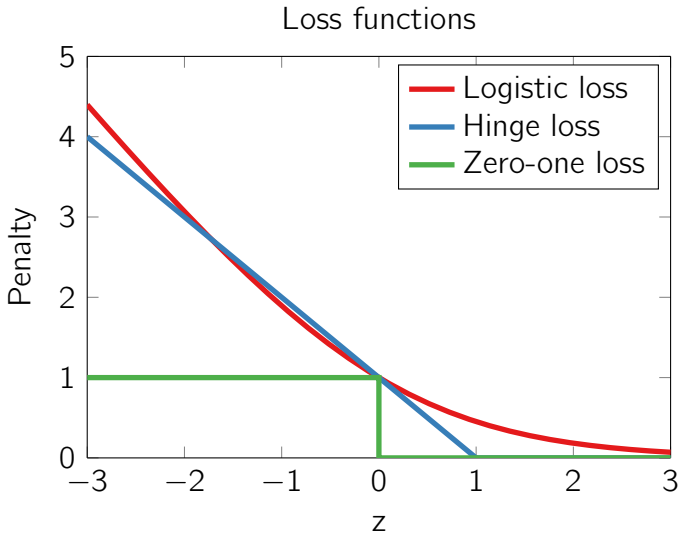


Figure 4.2: A graphical visualization of various loss functions. The farther values from the zero the more confident predictions are. Positive numbers correspond to correct predictions and negative numbers to mismatch predictions.

As Figure 4.2 shows, penalty goes to zero for correct predictions, otherwise it increases as discrepancies between predictions and right outputs diverge.

4.2 Problem formulation

In this section, we formulate the problem of NAT detection using the defined notation. NAT detection is a case of *binary classification problem*, since the resulting labels are from the set $\mathcal{Y} = \{-1, 1\}$. Minus one (negative sample) represents an end host and positive one (positive sample) corresponds to a NAT device. A target function having form $\mathcal{X} \to \{-1, 1\}$ is called a *classifier*.

In general, the feature vector $\mathbf{x}$ is a higher representation of an object. In our case, the object is a host which is uniquely identified by its IP address. The

feature vector should contain information about the host so that the classifier is able to determine its right label based on this information. Remind that our unique source of information are HTTP logs[4]. We assume that a NAT device is more active and exhibits mixture of user's behaviors in comparison with a single user. To capture this trait, for each host in the network we extract the following collection of eight features from HTTP logs:

1. the number of unique contacted IP addresses,
2. the number of unique User-Agent strings occurred in HTTP headers,
3. the number of unique Operating systems including their versions,
4. the number of unique Internet browsers including their versions,
5. the number of persistent connections[5],
6. the number of uploaded bytes,
7. the number of downloaded bytes,
8. the number of sent HTTP requests.

Furthermore, we suppose that the activity of both types of hosts differs in time. The NAT device can be active for longer time due to joining traffic of more users with different working habits. In order to capture the behavior in time, the features are collected in consecutive non-overlapping time windows of predefined length. We experimented with the length set to five minutes, 30 minutes, one hour and 24 hours. Thus, a specific host i can be represented by its feature vector $\mathbf{x}_i$ with numeric components $x_{i,f}^{w}$ where the index $f = 1, 2, \ldots, 8$ corresponds to a particular feature from the above list of features and w denotes a window. In the case of five minutes length windows, the index w goes from 1 to 288 as there is $W = 288$ windows of this length in 24 hours. Later in Section 4.4, we will see that the time windows also enable

[4] A detailed description of HTTP proxy logs can be found in Section 2.2.

[5] A connection to an IP address is considered to be persistent if it is active in at least five time windows from ten last consecutive time windows. The windows are set to have the size of one, two and four hours.

to filter out undesirable abnormal host's activity yet preserve the stable one in a convenient way.

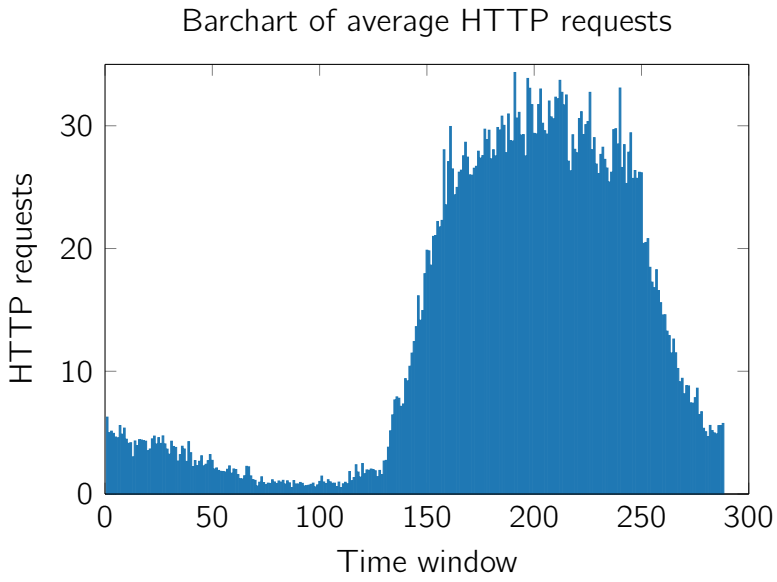


Figure 4.3: Average number of sent HTTP requests (the eight feature) during a day. The window period was set to five minutes in the data collection phase. One can clearly distinguish working hours from night time.

For instance, in a dataset gathered with the windows period set to 24 hours, a randomly selected host $i = 24$ has the following feature vector[6]:

$$\mathbf{x}_{24} = (74, 7, 1, 4, 1, 2 \text{ MB}, 11 \text{ MB}, 527)^{\mathsf{T}}. \tag{4.1}$$

As can be seen, the host contacted 74 distinct IPs, produced 7 different User-Agents strings, used one OS with one browser, etc. Whereas the next deliberately picked up host $i = 305$ is represented with the vector:

$$\mathbf{x}_{305} = (992, 9, 2, 18, 2, 18 \text{ MB}, 437 \text{ MB}, 8535)^{\mathsf{T}}. \tag{4.2}$$

Apparently, this host was more active than the previous one and used two different operating systems and Internet browsers. Is this evidence strong

[6]For illustrative convenience, uploaded/downloaded bytes are converted to megabytes and rounded.

enough to allow us to claim that the second host acts as a NAT device? We should take into account that some users can utilize a virtual machine to run an application incompatible with their native OS. Other users can have more Internet browsers installed for different reasons, etc. Is there any boundary defining what can be still considered as a behavior of one user and what yet not? This is exactly what we would like to learn from the data, a *decision boundary* enabling us to make these decisions and simultaneously minimizing the average overall error.

In the above example, we did not capture host's behavior in time, because the window duration was set to the whole 24 hours. In the case of five minutes windows, the feature vector of the first host can be expressed in the following manner:

$$\mathbf{x}_{24} = (x^1_{24,1}, x^1_{24,2}, \ldots, x^1_{24,8}, x^2_{24,1}, x^2_{24,2}, \ldots, x^2_{24,8}, x^3_{24,1}, \ldots, x^{288}_{24,8})^\mathsf{T}. \quad (4.3)$$

Features from the first window are placed firstly and followed by features from the second window, and the like. In total, there are $8 \times 288 = 2304$ numeric components. Vectors of other hosts can be unrolled in the analogous way.

4.3 Artificial NATs

The key requirement for the supervised learning is to have labeled data at disposal. Nonetheless, we are given the data without labels. As discussed at the beginning of this chapter, manual labeling is hard, especially in the security domain because of ambiguity and heavily imbalanced ratio of positive versus negative samples. Deficit of labeled data would force us to deploy other machine learning paradigms such as semi-supervised or unsupervised learning which can deal with the (partially)missing labels. These approaches are, however, considered to be less effective in comparison with the supervised learning when labels are available.

To generate labels for the data, we can leverage the fact that NAT devices

join traffic of multiple users into one, and create artificial NATs by manual merging logs of multiple hosts together. In other words, we mark all hosts with the negative label and then join HTTP logs of several negative instances to create one positive instance. In this way, we are able to generate an arbitrary number of positive instances. Notice that the mislabeling error, introduced by marking all hosts as end hosts (possibly true NAT devices are wrongly labeled as end hosts), is negligible due to the heavily imbalanced property of the original data.

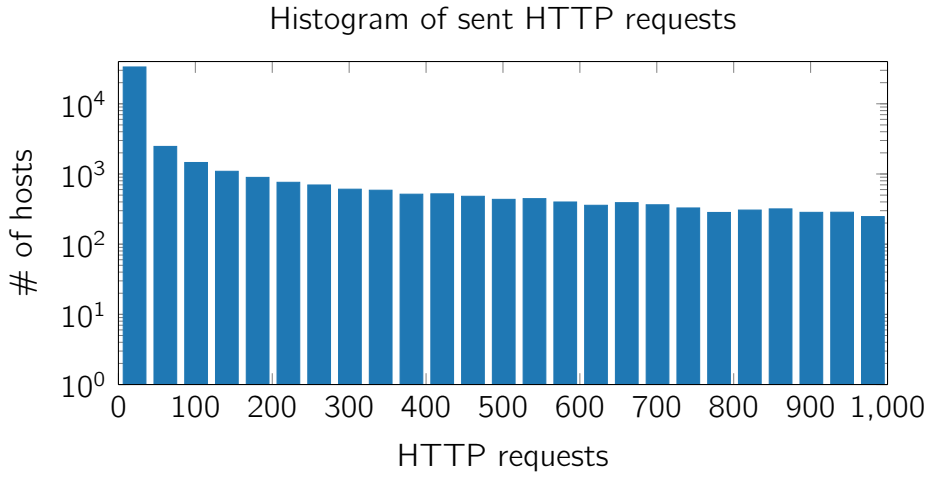


Figure 4.4: Histogram of sent HTTP requests per day. About 30 000 out of 61 000 hosts made less than ten requests. This is the case of the network of size large. Looking for candidates for NATed hosts, such hosts are inappropriate.

Figure 4.4 shows that the majority of hosts in the network does not use the HTTP protocol actively. They produce only a few HTTP requests. This is simply because we are dealing with corporate networks, where a significant amount of network devices is constituted by VoIP phones, network printers, storages, employee's personal mobile phones, tablets, etc. Another factor contributes to this observation when a network utilizes DHCP protocol for distributing IPs. In that case some users may renew their IPs with potentially endless ones by restarting their devices. Regardless the reason, in the merging procedure described above, we would like to create artificial NATs by merging

active hosts only. An IP is considered to be an *active host* if it produces at least six HTTP logs in the fifth most active window. The windows are set to have the length of 30 minutes and the window activity is also measured in terms of sent HTTP requests. To put it differently, a host is active when it sends at least six HTTP requests in five arbitrary selected 30 minutes lasting windows from the whole day.

At this point, it is natural to wonder how the distribution of artificially generated NATs should look like. How many differently large NATs would have to be prepared, in terms of included hosts? The best case scenario avails when there is a match between distribution in the training set and real world scenario case[7]. However, without any prior knowledge, we decided to make an assumption of uniformity. The decision is in compliance with the principle of maximum entropy [Jaynes and Rosenkrantz, 1983]: "If nothing is known about a distribution, then the distribution with the largest entropy should be chosen as the default". The motivation is that maximizing entropy minimizes the amount of prior information built into the distribution. Under the unique constrain that the sum of the probabilities is one, the maximum entropy discrete probability distribution is the uniform distribution. Consequently, each size is equally likely to be observed. The discrete uniform distribution $\mathcal{U}[a, b]$ has only two parameters: the starting point and the ending point. We assume that the smallest detectable NAT device consists of three users and in networks there is no single user producing more HTTP requests than twelve randomly selected active users together. Therefore, the final distribution is $\mathcal{U}[3, 12]$. To ensure an equally balanced training set, the total number of generated NATs is equal to the number of active hosts in the network. In doing so, a random sampling with replacement is used in the merging process, but no NAT device can contain more identical hosts.

Notice that generating artificial NATs breaks the assumption of mutually independent instances. As a host can appear in multiple NATs simultaneously

[7] Re-sampling techniques are used to match the target distribution when its shape is known. It helps to avoid bias outcome [Abu-Mostafa et al., 2012].

and positive instances are made from the negative ones. In general, this violation can potentially cause high variance (over-fitting phenomena) of a learning model. As the model tends to adapt to some specific properties of training set which are not typical for an independent testing set. This may lead to poor generalization ability. However, both parts of the i.i.d. assumption are commonly violated in many real-life problems a priori[8]. According to [Dundar et al., 2007]: "Even thought the machine learning community frequently ignores the non-i.i.d. nature of data, it seems that algorithms work well in practice". The aforementioned article also covers situations in which this does not hold, and proposes methods relaxing the i.i.d. assumption which enhanced accuracy of the learning model in the particular case. Nevertheless in the first trial, we believe that ignoring the independence will not have significant impact on the later classification.

A reader might ask, referring to Section 4.2, why source ports were not involved in the list of features. NAT's distribution of assigned source ports is more manifold as the ports are used for distinguishing among its actual end hosts. Although, a couple of related works take advantage of this fact successfully [Bellovin, 2002], [Mongkolluksamee et al., 2012], we can not do it, simply because NAT devices change the original source ports according to for us an unknown rule and the described merging process does not mimic it. Hence simulated NAT devices do not have the correct source ports. Nonetheless, it is an interesting area for future research. Upon detecting real NATs, their source ports can be studied and the gained knowledge can be incorporated into the existing NAT detector.

[8] Partially because many natural phenomena can be described by dynamic systems which consist of a set of fixed rules where the future state depends explicitly on the preceding one.

4.4 Pre-processing

At this stage, we are equipped with the labeled data from one day and four different networks of various sizes:

$$|\mathbb{D}_{\text{tiny}}| \doteq 9\ 000 \qquad |\mathbb{D}_{\text{small}}| \doteq 17\ 000$$
$$|\mathbb{D}_{\text{medium}}| \doteq 38\ 000 \qquad |\mathbb{D}_{\text{large}}| \doteq 77\ 000.$$

On average, 40% of all instances form inactive hosts, 30% active ones and the left 30% are artificially generated NATs. Every dataset exists in four versions (five minutes, 30 minutes, one hour and 24 hours) according to the used time window during the data collection phase. The objective of this section is to prepare the dataset $\mathcal{D}$ ready for training.

From Figure 4.3, showing average HTTP activity during a day, we could easily distinguish labour hours $w = (154, \ldots, 250)$ from night time for a particular company. It is important to note that companies might differ in these active windows as they can operate in various time-zones. On top of that, a network spread over more time-zones might exist[9]. Thus, a preprocessing step discarding time dependency is needed. Two widely known techniques are: a fast Fourier transform (FFT) algorithm computing discrete Fourier transformation $\mathcal{F}$, which converts time space to frequency domain, and an arbitrary effective sorting algorithm $\mathcal{S}$ (e.g., Quicksort, Merge sort, Heapsort, etc.). The magnitude of a Fourier transform of the discrete signal $s_w = x^w_{i,f}$ of a host and its feature (indexes i, f are fixed) is identical to the magnitude of a Fourier transform of the shifted signal by δ windows in the time domain[10]

$$|\mathcal{F}\{s_w\}| = |\mathcal{F}\{s_{w-\delta}\}|, \quad \delta \in \mathbb{Z}. \tag{4.4}$$

It means that magnitude of FFT is time-shift invariant [Smith, 2002]. In

[9] For instance, a local branch connecting via VPN to the head office network on the opposite side of the world.

[10] It yields from: $|\mathcal{F}\{s_{w-\delta}\}| = \left|\exp\left(-ik\frac{2\pi}{W}\delta\right)\right| \cdot |\mathcal{F}\{s_w\}| = |\mathcal{F}\{s_w\}|$, where the time shift property of a Fourier transform is utilized and then the fact that $|e^{i\varphi}| = 1$ for $\varphi \in \mathbb{R}$.

contrast, sorting is permutation invariant:

$$\mathcal{S}\{(s_1, s_2, \ldots, s_W)\} = \mathcal{S}\{(s_2, s_1, \ldots, s_W)\} = \ldots = \mathcal{S}\{(s_W, \ldots, s_2, s_1)\}. \tag{4.5}$$

In other words, sorting removes relations between time windows. This property is convenient as the classifier should not depend on the order of host's activity during a day. In addition to that, sorting also provides an easy way to filter out host's abnormal activity. For example, an user can download a movie of size 1GB in 30 minutes period which roughly equals to an amount of data downloaded by three working users during a day. This short-time abnormal behavior can be suppressed by discarding several windows with the largest values. From another point of view, we can select a range of interesting *quantiles* for each particular feature respectively as depicted in Figure 4.5. Consequently, we prefer sorting to FFT and all features of each host are sorted separately along window dimension in the preprocessing step. Note that asymptotic complexity is $\mathcal{O}(n \log n)$[11] in both cases. If we were interested in only one particular quantile (a representative window), a selection algorithm (e.g., Quickselect or Median of medians) [Knuth, 1998], seeking for the kth largest value in an array, with the linear complexity $\mathcal{O}(n)$ could be used. This would bring appreciable time savings in a real deployment, where for instance 500 000 arrays ($hosts \times features$) of length 288 need to be analyzed repetitively. We will describe the selection of representative quantiles in more detail in Section 4.6.6. At this point, the sorting is used just as the first preliminary step to get rid of time dependency.

In the previous paragraph, we met another issue. There are substantial differences in downloaded/uploaded bytes among hosts in the network. The range of transfered bytes is wide, from kilobytes up to gigabytes, and the gaps are in orders of magnitude. The scale of differences does not match the scale of the added information. To keep it in similar orders, both features are *normalized* with a logarithm function $\ln(x_{i,f}^w + 1)$, $f \in \{6, 7\}$. The added one serves as a

[11] Here n denotes a number of input elements.

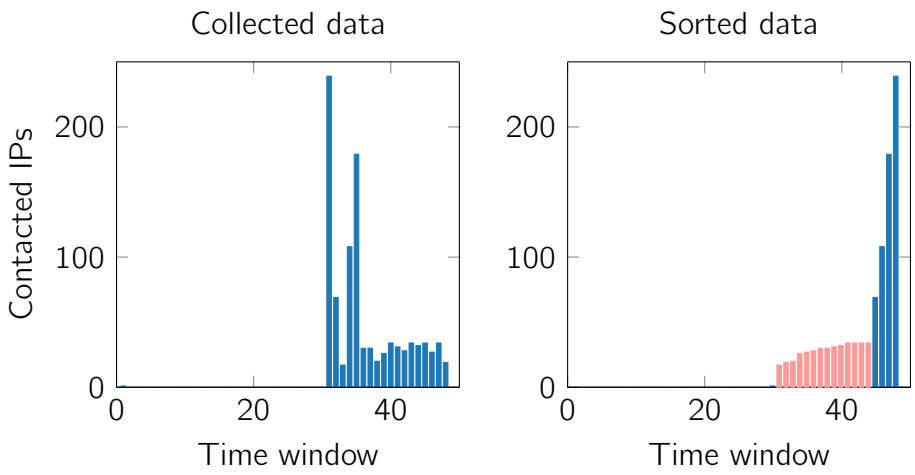


Figure 4.5: Sorting $\mathcal{S}\{x^w_{58,1}\}$ for a specific host $i = 58$ and its first feature $f = 1$ (the number of unique contacted IPs) followed by quantile selection (vertical bars marked by red color). The rest of the windows corresponds to uninteresting night time and abnormal host's activity. The demonstration is done using dataset with 30 minutes period windows.

protection when there are no transferred bytes.

For further application purposes, it is often desirable to *standardize* data $\mathbf{x}_i \rightarrow \mathbf{z}_i$ over all observations such that features are centered to have zero mean and scaled to have standard deviation equal to one. As all feature vector components should have the same scale for a fair comparison between them. This can be done by computing $\mathbf{z}_i = (\mathbf{x}_i - \bar{\mathbf{x}})/\mathbf{s}$, where $\bar{\mathbf{x}}$ is the sampled mean, $\mathbf{s}$ is the sampled standard deviation and the symbol / denotes element-vise division. An alternative to mean and standard deviation are median and interquartile range IQR (difference between the 75th and the 25th percentiles of the sample data). Like mean and standard deviation, median and IQR measure the central tendency and spread, but are robust against outliers.

The final training set $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ consists of a *matrix of observations* $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\top$ and an associated matrix of labels $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]^\top$. A vector $\mathbf{y}_i$ tracks to which network host $\mathbf{x}_i$ belongs and the number of involved users in addition to positive and negative labels. As all networks are equally important, it is desirable that each network contributes with the same

amount of active end hosts and NAT devices to the final set. The amount is determined by the smallest network. Herein is the complete list of pre-processing steps:

Pre-processing

1. [12] Every feature f of each host i is sorted along the window dimension w in the ascending order: $\mathcal{S}\{(x_{i,f}^1, x_{i,f}^2, \ldots, x_{i,f}^W)\}$.
2. [12] Sorted features are reshaped (refer to Example 4.3) into features vectors $\mathbf{x}_i$.
3. [12] The first half of components in the feature vectors is discarded as it captures 12 hours of uninteresting night time.
4. Features representing downloaded and uploaded bytes are transformed with the logarithm function $\ln(x+1)$.
5. Equal amount of active regular hosts and artificial NATs is randomly drown from each network to create the matrix of observations $\mathbf{X}$ and the matrix of associated labels $\mathbf{Y}$.
6. The matrix $\mathbf{X}$ is standardized using vectors $\bar{\mathbf{x}}$ and $\mathbf{s}$.
7. The procedure terminates with the set $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ ready for training.

After applying these steps, the final training set $\mathcal{D}$ counts approximately 20 000 instances. It exists in four versions according to used time window (five minutes, 30 minutes, one hour and 24 hours). Thus, dimensions of feature vectors (i.e., columns of the matrix $\mathbf{X}$) are 1152, 192, 96 and 8, respectively.

[12] Applied only on data sets with more time windows than one.

4.5 Data analysis

This section focuses on data inspection in order to unfold inner properties. One of the most common phrases in data science is: "Always look at the data and try to understand them" [Moore, 2010]. The visualization is essential as human brain process information faster when points are displayed rather than given in numerical matrices. Potential patterns, discrepancies or outliers can be spotted easily even when data volumes are large.

There are several ways how to visualize data. We have already used some of them such as histograms and bar charts. Histogram shows frequencies of occurrences of individual values of one particular feature over all observations. On the basis of the estimated distribution of downloaded and uploaded bytes, we were able to discover a discrepancy in scales and reduce it by logarithmic transformation. On the other hand, bar chart 4.3 helped us realize the time dependency, which we removed by sorting the time windows. To see relation between two variables without information loss, a scatter plot is often used. This approach is, however, unsuitable when there are many dimensions. In these situations, dimensionality reduction techniques like e.g., Principal Component Analysis (PCA), take place. PCA projects high dimensional data onto lower dimensional subspace (e.g., 2D plane) while preserving as much information as possible. The information loss is measured in terms of sum of square projection errors. Orthonormal coordinates of the new subspace are called principal components and can be calculated as eigenvectors of the covariance matrix. Corresponding eigenvalues then provide estimation of retained variance. Commonly, a few principal components (e.g., two in the case of planar visualization) with the largest eigenvalues are used and the rest is dropped. Finally, all high dimensional observations are projected onto these principal components using the inner product.

Figure 4.6 shows PCA projection of feature vectors from one particular network onto a plane. The points visualize captured behavior of hosts in the network. Blue ones represent regular users (up to potentially mislabeled NATs),

whereas colors from yellow to red belong to artificially created NATs. Moreover, color scale indicates the number of involved regular users in a NAT device.

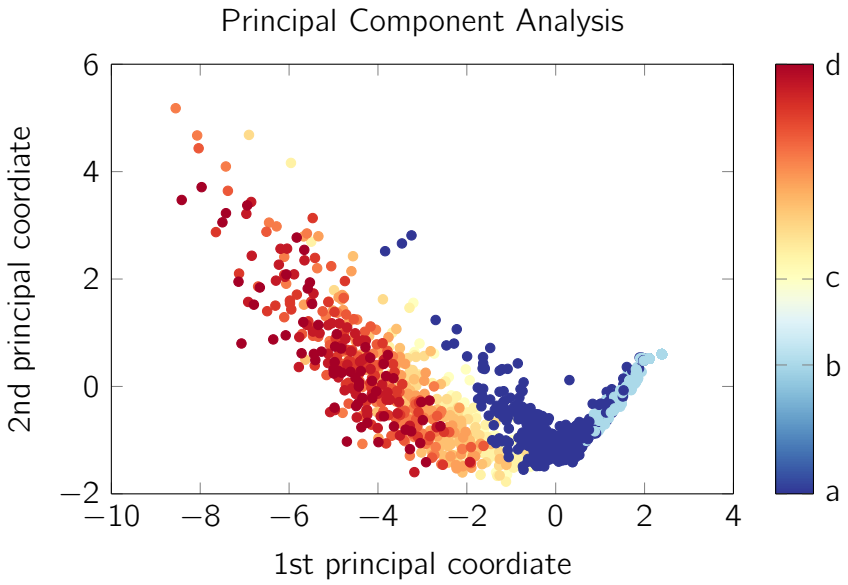


Figure 4.6: Visualization of one particular network of size small using PCA. The color intensity (a) denotes end hosts. A special case of end hosts are inactive ones, which are highlighted with the intensity (b). The color range from (c) to (d) is reserved for NATs following the uniform distribution $\mathcal{U}[3, 12]$.

As can be seen, the classes are well linearly separable from each other. This can serve as an evidence that the selected features provide a good representation. According to [Bengio et al., 2012], a right representation is usually more important than a complexity of learning models or bulk data[13]. Furthermore, the inactive hosts are displayed with the light blue color. As one could expected, they lie far away from the decision boundary and thus do not represent "hard examples" from the classification perspective.

[13]As it plays such an important role, unsupervised feature learning has become a field itself in the machine learning community recently. It aims at finding good representation of data that can support effective machine learning without any prior specific domain knowledge. Unlike the herein used approach where we manually selected features (List 4.2) according to our knowledge, this one is automated. The mentioned paper reviews recent work in this area.

The next collection of Figures 4.7 depicts PCA projections of equal amounts of active hosts from all available company networks. In other words, the whole matrix of observations is projected onto 2D plane. Colors separate companies and brightness classes. The lighter dots are regular hosts, whereas the dark ones correspond to devices performing NAT.

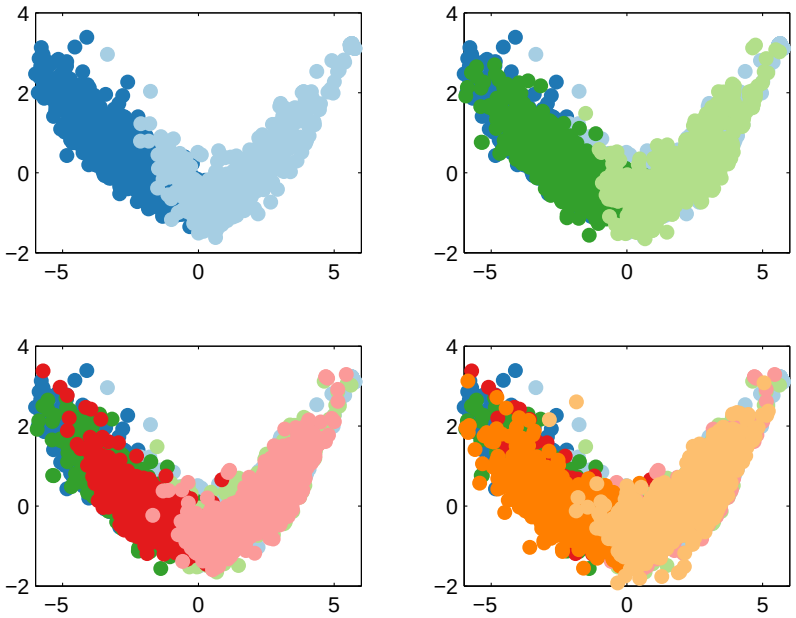


Figure 4.7: The overlaps among color layers on the similar positions support applicability of one pre-trained classifier on all companies without significant loss of accuracy. The lighter points in the dark areas typically represent mislabeled NATs that were already presented in the networks.

It seems that decision boundaries are almost identical in these companies. This pictorial hypothesis states that the conditional probability $P(y|\mathbf{x})$ is the same over all networks. It does not have to be generally true[14], but it makes the situation easier. If the hypothesis holds, we can use one pre-trained classifier for an arbitrary network, otherwise we have to include a learning phase into the NAT detector and classifiers would be made to measure according to

[14]Probably, a company providing accounting services will use HTTP communication less frequently than a company dedicating to Internet help desk support.

hosts in a specific network[15]. Although we are not quite sure, we will assume its truth, based on our limited observations. This issue is also related to the second part of the i.i.d. assumption, which states that instances should follow the identical distribution $P(\mathbf{x})$. If there were significant differences among companies, a low prediction error on yet unseen data could not be guaranteed.

By plotting the entire network of large size, considerable outliers emerge immediately. In general, outliers are anomaly observations lying suspiciously far from the rest of observations. Regarding our data, the outliers correspond to huge gateways in real networks, which were incorrectly labeled with negative label as end hosts during the process of generating artificial NATs. We justified the procedure by the assertion that the introduced mislabeling error will be small, which indeed is in terms of the number of occurrences. In the case of the huge gateways it may, however, be substantial. From a classifier perspective these hosts act like regular users with behavior of NATs. Referring to Figure 4.2, the convex loss functions (with missing a natural bound) will penalize actually correct hypotheses by enormous costs. Consequently, a final learning model might be biased due to the present outliers. To avoid it, these extraordinary observations are usually removed from training sets. There are several methods designed for this purpose. A survey made by [Hodge and Austin, 2004] describes the majority of them.

We decided to apply a distance-based approach which works as follows. The centroid $\boldsymbol{\mu}$ of active regular users (i.e., negative samples) in the feature space is calculated including distances MD_i from all hosts to the centroid. Then, distances of negative hosts only are compared against a certain threshold. If a particular distance is greater than the threshold, the associated host is considered to be outlier. The threshold is determined by 99% quantile of distances which belong to NAT devices. In other words, if a distance of a regular user to the centroid of regular users is larger than the distance of the waste majority of NAT devices, it is a wrongly labeled NAT device with high

[15] In fact, it wont be as simple, because networks with one or more huge gateways might exist.

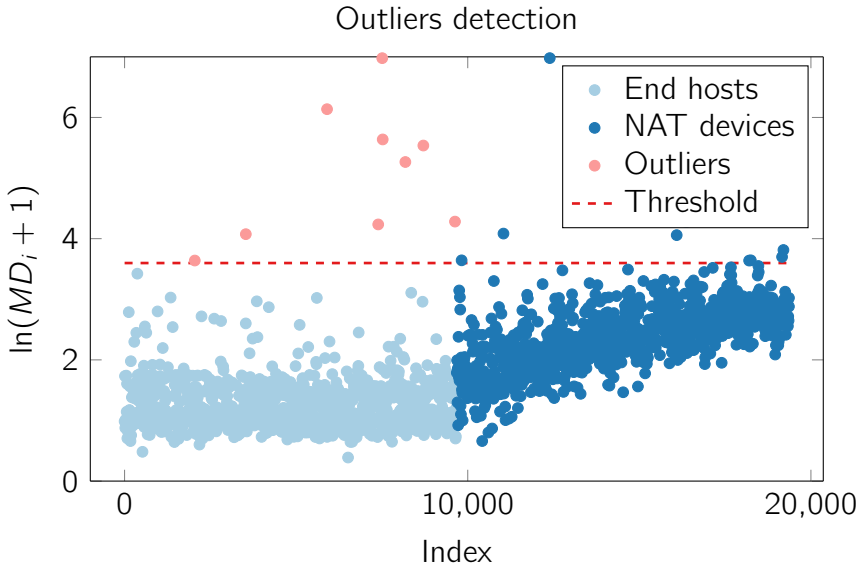


Figure 4.8: Outliers detection method based on Mahalanobis distance. The end hosts occurring above the calculated threshold are considered as potentially wrongly labeled gateways. These hosts are preliminary removed from the training set.

confidence. As the metric, we use the Mahalanobis distance:

$$MD_i = \sqrt{(\mathbf{x}_i - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}_i - \boldsymbol{\mu})}. \tag{4.6}$$

It takes into consideration the inter-feature dependencies so that features are compared on the same scale[16]. Since we know that the data are contaminated with outliers, we should utilize robust methods for estimating statistics. Robust estimates of the centroid $\boldsymbol{\mu}$ and the sampled covariance matrix Σ can be obtained by the Minimum Covariance Determinant (MCD) estimator [Verboven and Hubert, 2004]. The MCD method looks for the h out of H observations whose classical sample covariance matrix has the lowest possible determinant[17]. The raw MCD estimate of the centroid is then the average of these h points, whereas the raw MCD estimate of covariance matrix is their

[16] This is completely true only when observations follow a normal distribution.

[17] The fast algorithm (FAST-MDC) using re-sampling method to avoid an exhaustive investigation of all h-subsets is implemented in a MATLAB Library for Robust Analysis (LIBRA).

sample covariance matrix, multiplied with a consistency factor depending on size of h. This technique can resist $h - H$ outliers, hence the proportion $\frac{h}{H}$ determines the robustness of the estimator. Notice that the covariance matrix for PCA purposes should be also estimated in the analogous manner. Figure 4.8 shows the outliers detection procedure. The detected host with the lowest distance has the following feature vector[18]:

$$\mathbf{x}_{2346} = (4261, 453, 3, 667, 5, 217 \text{ MB}, 1.6 \text{ GB})^{\mathsf{T}}. \tag{4.7}$$

The higher values when compared to the regular user (Example 4.1) indicate that we are indeed dealing with the real but incorrectly labeled gateway. Remind that the aim of this procedure is to detect extraordinary hosts with the negative label which behave like NAT devices from a classifier point of view. In fact, there is a chance that the behavior is caused by a malware infection. As situations when a malware attempts to contact hundreds of various IPs are well known. This short abnormal activity should be filtered out by the choice of features from non-maximum windows. Therefore, the whole analysis in this section is done using the dataset with the 30 minutes lasting windows and features corresponding to 90% quantile. Nevertheless due to the ambiguity, we decided to remove the detected hosts from the training set only. The final performance of a classifier will be evaluated on the original set.

For the sake of completeness, note that besides herein used PCA there are other dimensionality reduction methods using transformations such as Multi-dimensional Scaling (MSD), Linear Discriminant Analysis (LDA), non-linear kernel PCA and so on, with different information loss criteria. An exhausted survey of these techniques together with their comparison can be found in [C.O.S. Sorzano, 2014].

[18] The feature vector is drawn from the dataset with the 24 hours windows according to its index. The dataset has not been pre-processed.

4.6 Classification

To be able to start with learning, we have to choose between *generative* and *discriminative* type of learning algorithm $\mathcal{A}$. Generative algorithms try to model the joint probability distribution $\mathrm{P}(y, \mathbf{x})$. Assuming the NAT detection assignment, the generative approach firstly estimates $\mathrm{P}(\mathbf{x}|y = 0)$ and $\mathrm{P}(\mathbf{x}|y = 1)$ separately using available data set $\mathcal{D}$ to learn how both classes look like. Then, one way[19] to classify (i.e., determine a label $\hat{y}$) is to calculate the probability whether a new host $\mathbf{x}$ looks more like a regular user or a NAT device[20]: $\hat{y} = \arg\max_y \mathrm{P}(y|\mathbf{x})$. The formula can be rewritten to more practical one where all terms are known: $\hat{y} = \arg\max_y \mathrm{P}(\mathbf{x}|y)\mathrm{P}(y)$ by taking advantage of the Bayes rule and the fact that $\mathrm{P}(\mathbf{x})$ is identical for both classes. Here, $\mathrm{P}(y)$ denotes a priori probability of the given class. Conversely, discriminative algorithms try to model $\mathrm{P}(y|\mathbf{x})$ directly or learn a direct mapping $\mathrm{h}^* : \mathcal{X} \to \mathcal{Y}$ by seeking for a decision boundary between the classes in the feature space $\mathcal{X}$. Then, the classification is done by checking on which side of the decision boundary the new host falls. The common belief is that discriminative algorithms outperform generative ones as far as the classification is concerned. This is supported by [Vapnik, 1998] statement: "One should solve the [classification] problem directly and never solve a more general problem as an intermediate step [such as modeling $\mathrm{P}(\mathbf{x}|y)$]". Additionally, [Ng and Jordan, 2002] show that while discriminative learning has lower asymptotic error, a generative classifier may also approach its higher asymptotic error much faster. The inferior performance can be explained by differences between the model and true distribution of the data. On the other hand, the generative approach can exploit unlabeled data in addition to labeled ones and is capable to generate synthetic examples of the feature vector. In an attempt to gain the benefit of both approaches an interpolating

[19] Supposing the zero-one loss function $\ell_{0-1} \equiv \mathbb{1}[y \neq \mathrm{h}(\mathbf{x})]$ and a priori probability $\mathrm{P}(y)$ to be known.

[20] The maximum a posteriori strategy is the special case of the Bayesian strategy (a strategy minimizing the Bayesian risk) with zero-one loss function.

procedure has been also proposed by [Lasserre and Bishop, 2007].

Regarding the NAT detection problem, we decided to apply the discriminative learning as the classification is of interest only. Partially to avoid estimating $P(\mathbf{x}|y)$[21] and partially because a priori probabilities $\mathrm{P}(y)$ are unknown and even can not be known in principle, as there might be networks with one huge gateway and also with none at all. The second mentioned limitation would force us to solve a non-Bayesian task such as the Neyman-Pearson task [Schlesinger and Hlavac, 2002], which is generally considered to be non-trivial. As such, discriminative approach seems to be more straightforward in this case.

VC analysis. It is important to realize that without the joint probability distribution $\mathrm{P}(\mathbf{x}, y)$, we are not able to calculate the expected out-of-sample error E_{out} directly yet. The metric which we care about as we are primarily interested in the performance on yet unseen samples. [Vapnik, 1998] proposed a method[22], called *VC analysis*, decomposing E_{out} into the in-sample error E_{in} and the generalization error Ω[23]. The in-sample error represents the error on the training set and the generalization error is the difference $E_{\text{out}} - E_{\text{in}}$. In words, a learning model generalizes well when there is a small difference in the performance on the testing and the training set. The VC analysis states that:

$$\mathrm{P}\left(E_{\text{out}} \leq E_{\text{IN}} + \Omega(\mathcal{H}, \delta, N)\right) \leq 1 - \delta, \tag{4.8}$$

where Ω goes up with higher complexity of hypothesis set $\mathcal{H}$ or the confidence term δ and is pulled down by larger training set N. The complexity is measured with VC dimension[24] expressing a learning capacity of the functions class $\mathcal{H}$. Intuitively, the more complex hypothesis set, the higher variability to fit the target function f. The precise derivation and explanation can be found in

[21] We do not observe that the data obey any standard probability distribution. Therefore, non-parametric or semi-parametric models should be used, which are not easy to handle.

[22] Another commonly used method is the Bias-variance decomposition.

[23] E_{in}, Ω are also known as the empirical and the structural risk

[24] VC dimension measures the maximum number of training examples where the function class $\mathcal{H}$ can still be used to learn perfectly.

[Abu-Mostafa et al., 2012]. However, the key message is that one has to balance the trade-off between E_{in} and Ω to achieve a small E_{out}. It is not sufficient to reach zero error on the training set $E_{in} = 0$ with a complex learning model and little data as it would result in high Ω, and hence high final E_{out}. From another point of view, one should match the model complexity to the available data resources, not the complexity of the target function.

There are lots of learning algorithms as well as attempts to compare them. However, it appears that selection of the most appropriate algorithm depends heavily on structure of provided data. "Even the best models sometimes perform poorly, and models with poor average performance occasionally perform exceptionally well." [Caruana and Niculescu-Mizil, 2005]. The situation with no single model that would work best for every problem is often referred to the "No Free Lunch" theorem [Wolpert, 1996]. Moreover, it shows that better data often beats better algorithms. It means that performance of different types of learning algorithms can be similar on large training sets [Banko and Brill, 2001]. Another famous quote is from Google's Research Director Peter Norvig: "We don't have better algorithms. We just have more data." [Halevy et al., 2009]. Hence the choice of an algorithm might not really matter so much in terms of performance[25], but rather algorithm's properties such as speed of training and classification, number of model's parameters, ability to interpret results, etc. As one of our priorities is the speed of classification, we decided to try linear models, namely: linear Support Vector Machine (linear SVM) and Logistic Regression (LR). Since the data seem to be well linearly separable (Figure 4.6), there is no need for a more complex decision boundary. According to [Bruzzone and Persello, 2009], linear SVM is a good candidate as it showed to be quite resistant towards mislabeled instances in training sets when compared to others. Remind that, in our case, the wrongly labeled instances are actually the real NAT devices[26]. For comparison purposes, we will also try more complex models represented by SVM

[25] Assuming enough data and a good representation

[26] It is a case of a specific class mislabeling, whose impact on accuracy is unfortunately more unpleasant, because it deflects a model in one preferred direction.

with RBF kernel. Just to see if there is any significant improvement which would be worth reconsidering the classification speed requirement.

4.6.1 Support Vector Machine

SVM belongs among the most frequently implemented binary discriminative learning algorithms in machine learning community. Especially because it requires only a small set of parameters [27] to be set up and simultaneously provides a high accuracy of predictions. Its special property is ensuring low Ω in addition to minimizing E_{in}. In doing so, it seeks for a separating hyperplane in the feature space with the maximum margin. The maximality constrain implies low complexity of $\mathcal{H}$, because there is only one separating hyperplane with the maximal (hard) margin for linearly separable data. In the case of non-linearly separable data (noisy data), so called (soft) margin is parametrized by an additional parameter C, expressing a cost of misclassification, which need to be tuned. The geometric motivation is that the further from the decision boundary the more confident decision is made. The task of searching such a decision boundary can be formulated as a quadratic optimization problem (QP) [Abu-Mostafa et al., 2012]:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{Q}\boldsymbol{\alpha} - \mathbf{e}^{\mathsf{T}}\boldsymbol{\alpha} \\ \text{subject to} \quad & \mathbf{y}^{\mathsf{T}}\boldsymbol{\alpha} = 0, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, N. \end{aligned} \tag{4.9}$$

Herein, the labels are $y_i \in \{-1, 1\}$, $\mathbf{e} = \begin{bmatrix} 1, \ldots, 1 \end{bmatrix}^{\mathsf{T}}$ denotes a column of ones and the symmetric matrix $\mathbf{Q}$ is constructed in the following manner: $Q_{ij} = y_i y_j \mathrm{K}(\mathbf{x}_i, \mathbf{x}_j)$, where the indexes i, j go from 1 to N. As before, N stands for the number of observations in the training set $\mathcal{D}$. $\mathrm{K}(\mathbf{x}_i, \mathbf{x}_j)$ is a kernel function, which should satisfy Mercer's conditions. Two common[28]

[27] Typically two (C, γ) corresponding to the cost of misclassification and the kernel's parameter.

[28] Nevertheless, the variability in the Kernel functions substantially extends the applicability of SVM in itself, [Schölkopf and Smola, 2002].

choices are: the inner product $\mathrm{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\mathsf{T}\mathbf{x}_j$ and the Radial Basis Function (RBF) $\mathrm{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ for $\gamma > 0$. The first choice yields to *linear SVM* while the second option is known as the *kernel method*. The "kernel trick" generalizes the dot product and implicitly maps the feature vectors into a higher dimensional space (in the case of RBF kernel, even up to infinite dimensional) believing that in the new space the classes will be better separable. By solving the constrained optimization Problem 4.9 with any QP solver, we obtain the vector $\boldsymbol{\alpha}$. Interestingly, only a few α_i will be greater than zero. The corresponding feature vectors $\alpha_i > 0 \Rightarrow \mathbf{x}_i$ are called Support Vectors (SV) as they lie on the margin (support the decision boundary). The resulting model $\mathrm{h}(\mathbf{x})$ classifies a new sample $\mathbf{x}$ by evaluating into which half-space the sample falls:

$$\mathrm{h}(\mathbf{x}) = \mathrm{sgn}\left(\sum_{\alpha_i>0} y_i\alpha_i\mathrm{K}(\mathbf{x}_i, \mathbf{x}) + b\right), \quad b = y_m - \sum_{\alpha_i>0} y_i\alpha_i\mathrm{K}(\mathbf{x}_i, \mathbf{x}_m). \quad (4.10)$$

The bias coefficient b can be determined using an arbitrary support vector and its label $(\mathbf{x}_m, y_m)$. As only SVs define the decision boundary, the rest of vectors (i.e., $\mathbf{x}_i$ where $\alpha_i = 0$) can be removed without affecting the solution. Conversely, all SVs $\mathbf{x}_i$ including α_i, y_i and b need to be stored for purposes of further classification. In the case of linear SVM, it is, however, favorable to pre-calculate $\boldsymbol{\omega} = \left[\sum_{\alpha_i>0} y_i\alpha_i\mathbf{x}_i; b\right]$, extend feature vectors by additional one $[\mathbf{x}; 1]$, and thus simplified the final hypothesis:

$$\mathrm{h}(\mathbf{x}) = \mathrm{sgn}\left(\boldsymbol{\omega}^\mathsf{T}\mathbf{x}\right). \quad (4.11)$$

Now, $\boldsymbol{\omega}$ needs to be stored only. Besides that, the classification of a new sample is as fast as the calculation of the inner product. Just as a matter of interest, it can be shown [Abu-Mostafa et al., 2012] that solution of the optimization problem:

$$\min_{\boldsymbol{\omega}} \quad \frac{1}{2}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{\omega} + C\sum_{i=1}^{N}\max\left(0, 1 - y_i\boldsymbol{\omega}^\mathsf{T}\mathbf{x}_i\right), \quad (4.12)$$

leads to the identical $\boldsymbol{\omega}$. Note that the second term corresponds to the hinge loss function $\ell_{hinge}(z)$ (Figure 4.2), where $z = y\boldsymbol{\omega}^\top \mathbf{x}$. As an aside, several libraries with the open source LIBSVM in head, implement SVM learning algorithm as well as its other modifications.

4.6.2 Logistic Regression

LR is another discriminative learning algorithm which tries to model a target distribution $\mathrm{P}(y|\mathbf{x})$ directly. Hence it turns the binary predictions $\mathcal{Y} = \{-1, 1\}$ into posterior probabilities $(0, 1)$ that samples belong to the positive class. An assumption is that the target distribution has a form of the sigmoid function: $\mathrm{P}(y = 1|\mathbf{x}) = (1 + \exp(\boldsymbol{\omega}^\top \mathbf{x}))^{-1}$. To find the model parameters $\boldsymbol{\omega}$, a method Maximum-Likelihood Estimation (MLE) is used, or more specifically, its logarithm. The method seeks for values maximizing logarithm of the likelihood function over all samples. It corresponds to the second term of the following optimization problem:

$$\min_{\boldsymbol{\omega}} \quad \frac{1}{2}\,\boldsymbol{\omega}^\top \boldsymbol{\omega} + C \sum_{i=1}^{N} \log\left(1 + \exp\left(-y_i \boldsymbol{\omega}^\top \mathbf{x}_i\right)\right). \tag{4.13}$$

As can be seen, it yields to the logistic loss $\ell_{log}(z)$ (Figure 4.2), where $z = y\boldsymbol{\omega}^\top \mathbf{x}$. For simplicity, $\mathbf{x}$ is again extended by the additional one $[\mathbf{x}; 1]$ to avoid a separate bias term. As before, C has the meaning of cost misclassification and needs to be adjust manually. The first term in Formula 4.13 is called L2-regularization[29] and its purpose is to penalize too complex hypothesis. As such, the whole optimization problem can be related to VC analysis (Inequality 4.8), where the regularization term stands for Ω and the logistic loss over all samples for E_{in}. The weight between these two terms is determined by the parameter C. Solving Problem 4.13 leads to minimizing the out-of-sample error E_{out}. Unfortunately, the optimization problem does not have the close-form solution and the iterative one, using e.g. (Stochastic) Gradient Descent or Newton's optimization method, has to be used. To ob-

[29] Therefore, this version of LR is also called L2-Regularized Logistic Regression.

tain binary labels, the probability distribution is usually thresholded by a value from the interval $(0, 1)$:

$$\mathrm{h}(\mathbf{x}) = \mathbb{1}[\mathrm{P}(y = 1|\mathbf{x}) \geq 0.5]. \tag{4.14}$$

The threshold allows to balance a trade-off between precision and recall metric, which are presented in the next paragraph. Further details about LR can be found in [Abu-Mostafa et al., 2012]. Note that the open source library LIBLINEAR was deployed to train LR and linear SVM in this work.

4.6.3 Evaluation metrics

So far, we have been using the term "error" qualitatively. To be able to assess performance of an classifier and compare it with another one, a quantitative definition is needed. In binary classification problem, there are four possible outcomes. If the instance is positive and it is classified as positive, it is counted as a *True Positive* (TP); if it is classified as negative, it is counted as a *False Negative* (FN). If the instance is negative and it is classified as negative, it is counted as a *True Negative* (TN); if it is classified as positive, it is counted as a *False Positive* (FP)[30]. A natural way how to evaluate classifier's performance is to count its true hits against all. This metric is named *accuracy*. Unfortunately, it is not applicable with a skewed class distribution. Consider an example with 90% positive and 10% negative instances. Then, a dummy classifier predicting only positive class and nothing else gains 90% accuracy for free. Furthermore, in some cases FPs might be more important than FNs and vise-versa. For instance, a false fire alarm (FP) can be annoying as one has to leave a building, however, a missed alarm (FN) can put someone in danger of her/his life. From these reasons, *precision* and *recall* metrics are used. Assuming NAT detection, a perfect precision score of 1.0 means that every host classified as NAT device is indeed a NAT device (but says nothing about whether all NAT devices are detected) whereas a perfect

[30] FPs and FNs are often called Type I and Type II errors.

recall score of 1.0 means that all NAT devices are detected (but says nothing about how many regular hosts are misclassified). Additionally, recall is also called *True Positive Rate* (TPR) as it counts TP hits against all positive instances. In contrast, *False Positive Rate* (FPR) counts FP hits against all negative instances.

$$\text{accuracy} = \frac{\text{TPs} + \text{TNs}}{\#\text{ of instances}}, \quad \text{precision} = \frac{\text{TPs}}{\text{TPs} + \text{FPs}}, \quad \text{recall} = \frac{\text{TPs}}{\text{TPs} + \text{FNs}},$$

$$\text{FPR} = \frac{\text{FPs}}{\text{TNs} + \text{FPs}}, \quad F_\beta = \left(1 + \beta^2\right) \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}. \tag{4.15}$$

Nonetheless, to be able to decide which classifier is better, it is desirable to represent each of them by one value. For this purpose, F_β *measure* is often applied. The traditional F_1 measure is the harmonic mean of precision and recall. Two other usually used F measures are the F_2 measure, which weights recall higher than precision, and the $F_{0.5}$ measure, which puts more emphasis on precision than recall. Figure 4.9 illustrates the metrics from two-variable function point of view.

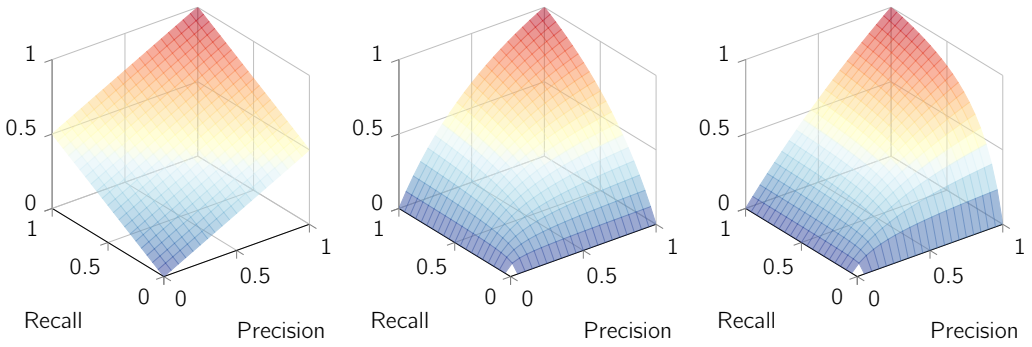


Figure 4.9: Arithmetic mean, harmonic mean (F_1) and $F_{0.5}$ measure, respectively from the left to the right.

An interesting property can be seen from the above illustration. Unlike arithmetic mean, the harmonic mean is very conservative as the return value is close to the minimum value of the input arguments. Referring to Section 2.3, we decided to utilize $F_{0.5}$ measure, because precision is the high priority in our

NAT detection assignment. It worth mentioning that although F measure is quite popular, it suffers from several drawbacks. As an example, the metric does not take into account TNs at all. This is not crucial in herein usage as long as we know that TPs are of interest. Further details as well as other alternative metrics, can be found in [Powers, 2007].

4.6.4 Validation

Once a proper metric is established, we are ready to measure classifier's performance. In general, the assessment can serve for two main purposes: to decide which learning model or which combinations of model parameters such as (C, γ) is the best one (*model selection*) and to test the overall performance of the final hypothesis h^* (*final evaluation*). However, care should be taken when we do this. According to Inequality 4.8, measuring score on the training set E_{in} does not give a good estimate of the true out-of-sample error E_{out} as there is an extra Ω term. It would result in an optimistic estimate, which probably would not correspond with the reality on unseen data. [Abu-Mostafa et al., 2012]: "Learning the parameters of h and testing it on the same data is a methodological mistake". To avoid it, a common practice is to split the dataset $\mathcal{D}$ into three mutually exclusive sets[31]: a training set $\mathcal{D}_{\text{train}}$, a validation set $\mathcal{D}_{\text{val}}$ and a testing set $\mathcal{D}_{\text{test}}$. As the names suggest, the first set is used for training, the second one for the model selection and the last one for the final evaluation. Note that the test set $\mathcal{D}_{\text{test}}$ must stay untouched till the last moment to be able to provide an unbiased estimate. In our case, we will not split the dataset, but rather we use three entire datasets from three distinct days. Also, rather than optimizing on the average score over all instances in the validation dataset, we will optimize on the worst score over all networks in the set. It means that each network in the validation set is

[31] This approach is known as the *hold out* technique. Nevertheless, splitting data into three independent sets can be an expensive act, as we lose a significant amount of training data, which can lead to poorer performance. So called *cross-validation* techniques are able to overcome it and use the same data for the training and the validation phase of learning, see [Abu-Mostafa et al., 2012] for more details.

evaluated separately and a decision is made with respect to the network with a minimum score. In this manner, we want to ensure reasonable behavior in all networks instead of excelling in one particular network and doing poorly in the rest of them. This step can be related with the previous hypothesis claiming that the conditional probability $P(y|\mathbf{x})$ is identical through all networks. It can be viewed as a precaution against the situation when the hypothesis, assuming to be true, eventually breaks.

4.6.5 Training

After all, we are ready to train the classifiers. To recap, our final training set $\mathcal{D}$ contains approximately 20 000 instances representing one day and four distinct networks. It is balanced in terms of positive vs. negative classes and the portions of individual networks. The negative instances involve only active end hosts as just theses hosts pose hard examples from classification perspective (Figure 4.6). Additionally, the extreme outliers with negative labels are removed, because they most probably correspond to mislabeled real gateways and could negatively influence the learning process. The sizes of artificially generated NATs follow uniform distribution $\mathcal{U}[3, 12]$. As far as the training is concerned, we deploy three learning algorithms: linear SVM, LR and SVM with RBF kernel. The hyperparameters (i.e., C, γ) are selected using *grid search* and the validation. Grid search is a straight forward parameter optimization method, which exhaustively tries and validates all candidate values for parameters. The set of values is generated by cross-product of C values ranging in $\{2^{-10}, 2^{-8}, \ldots, 2^{8}\}$ and γ values in $\{2^{-15}, 2^{-13}, \ldots, 2^{3}\}$. Once all values are tried, the pair (C, γ) with the best validation score is picked. In fact, there are more advanced methods [Bergstra et al., 2011] than this naive approach, which can save computational time and even find better combination of parameters using adaptive range scaling. However, grid search is widely used for its simplicity. Note that the validation is done according to the scenario described in the previous paragraph. The validation set contains all hosts including inactive ones. Moreover, it is not modified in

	linear SVM		LR		RBF SVM	
	min F_{05}	max F_{05}	min F_{05}	max F_{05}	min F_{05}	max F_{05}
5 min	0.9575	0.9785	0.9564	0.9795	–	–
30 min	**0.9665**	0.9864	**0.9652**	0.9859	**0.9696**	0.9885
1 hour	0.9635	0.9835	0.9649	0.9860	0.9692	0.9890
24 hour	0.9559	0.9798	0.9529	0.9807	0.9625	0.9831

Table 4.1: Comparison of learning algorithms and window periods.

sense of removed outliers.

As can be seen from Table 4.1, the results are very narrow. It seems that the best window period is 30 minutes. On the other hand, the winner among classifiers is the SVM with RBF kernel. This is perhaps not surprising as the model has the highest complexity compared to others. Nevertheless, it takes almost thousand times more computational resources to classify a new host. As there are around 1 000 support vectors out of 20 000 feature vectors which need to be stored and used during the classification (Equation 4.10). The training time is even worse and thus we did not train the classifier on training set with five minute windows as they would not be used anyway. The purpose of the SVM with RBF kernel was just to show that the efficacy will not significantly drop when choosing model with lower complexity. Considering linear models, the highest score (i.e., the highest worst case F_{05} measure over all networks) is achieved by the linear SVM on the training set with 30 minutes window periods. The cost of misclassification (i.e., the model parameter) is $C = 2^{-8} \doteq 0.004$. Table 4.2 shows particular results of the linear SVM on each network separately including various metrics. Remind that the portions of active hosts, NATs and inactive hosts are approximately: 30%, 30% and 40% of all hosts, respectively.

On the basis of the data analysis in Section 4.5, we know that the largest network contains several gateways. Therefore, it is not surprising that the classification performance is the poorest on this network. As these gateways are evaluated as FPs due to the mislabeling introduced in the NAT generating

network size	accuracy	F_1	F_{05}	precision	recall	FPs
tiny	0.9819	0.9677	0.9676	0.9675	0.9679	80
small	0.9836	0.9725	0.9813	0.9873	0.9582	66
medium	0.9927	0.9827	0.9864	0.9888	0.9767	180
large	0.9780	0.9702	**0.9665**	0.9640	0.9765	518

Table 4.2: Achieved results with the linear SVM and 30 minutes windows.

process. This phenomena explains the lowest precision. Consequently, all results should be considered as lower-bound results and FPs as detected NAT devices.

4.6.6 Dimensionality reduction

Referring back to Table 4.1, an interesting phenomena can be observed. The data with the highest dimensionality (i.e., training set with five minutes windows) have not yielded to the best results on the validation set even thought they provide the highest variability. As already discussed, learning algorithm with a higher possible variability to fit the data is also more prone to adapt noise in the data. To put it into another perspective, if a learning algorithm is less likely to fit the data, it is more significant when it happens. This can be referred to Occam's razor principle: "The simplest model that fits the data is also the most plausible.", [Abu-Mostafa et al., 2012]. An interesting question that arises at this time is whether we could reduce dimensionality of our data without serious performance degradation, and thus increase classifier's generalization ability. The data set, which exhibited the best results (30 minutes period windows), is 192 dimensional one and consists of 20 000 samples. For training linear SVM, a rule of thumb is that the number of training samples should be at least ten times greater than the number of features. According to the rule, we are on the safe side. However, the data can be considered as redundant in the sense that there are only eight distinct features, but collected in several consecutive time windows. Figure 4.10 shows importance of individual time windows from the linear SVM classifier point of view. To do this, we plot absolute values of corresponding weights $\boldsymbol{\omega}$. Assuming that

features are on the same scale, the higher the values are, the more important role they play in the final hypothesis 4.10. Weights which pertain to one feature (they form vertical bars in the figure) are divided by the value of the largest one. The importance of individual windows of one particular feature is expressed on the scale from zero to one (the color intensity). Quantiles indicate window's position index (Figure 4.5).

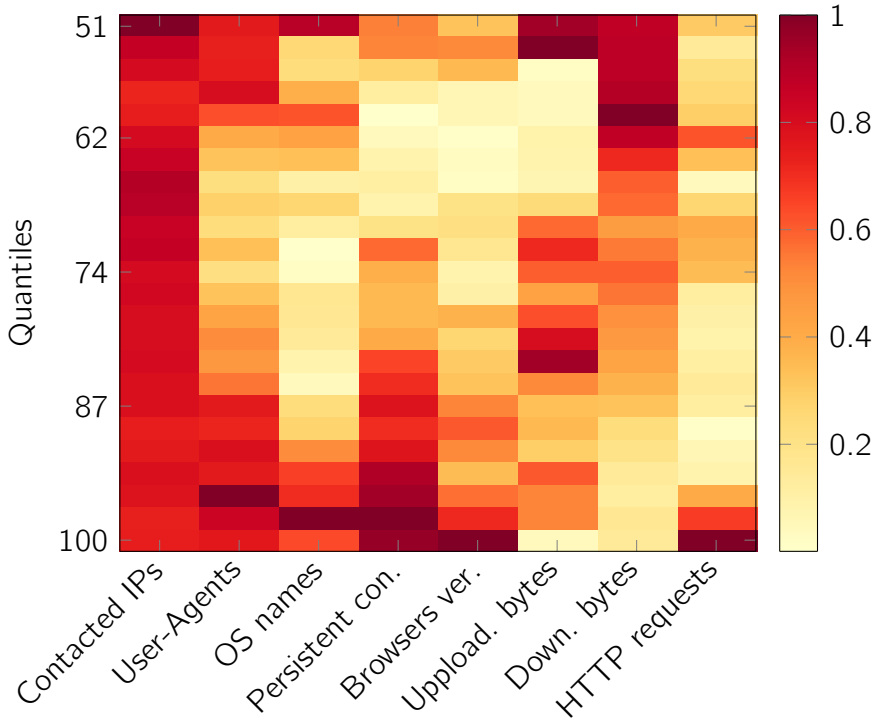


Figure 4.10: Importance of individual time windows according to the linear SVM classifier. The darker color the more important the window is. The most important quantiles (i.e., window positions after sorting) are 51%, 95%, 98%, 98%, 100%, 53%, 60%, 100%, respectively for each particular feature.

Looking at Figure 4.10, we can see that the classifier naturally prefers windows associated with non-maximum quantiles, especially in the case of contacted IP addresses and downloaded/uploaded bytes. We interpret the preference as follows. Values of these windows better characterize host's stable behavior in the network, whereas windows associated with the maximum values (i.e., 100% quantile) can be caused by host's short-time abnormal activity

network size	accuracy	F_1	F_{05}	precision	recall	FPs
tiny	0.9762	0.9574	0.9592	0.9603	0.9545	97
small	0.9749	0.9575	0.9717	0.9813	0.9348	95
medium	0.9840	0.9613	0.9758	0.9857	0.9382	223
large	0.9658	0.9533	**0.9538**	0.9541	0.9525	650

Table 4.3: Results when only eight features were used to train linear SVM.

(e.g., by downloading or uploading few large files in a short-time period). As mentioned before, the abnormal behavior can also be a signature of a malware infection. Nevertheless in the NAT detection task, we are interested in host's stable behavior only. Therefore, it suggests itself to select just a few most representative time windows for each particular feature and discard the rest. In the extreme case, only one specific quantile might be chosen to represent one feature. Table 4.3 shows classification results following the same scenario as before, but when only eight features, each represented with one the most important window, are used to train the linear SVM classifier.

Considering the fact that we reduced the dimensionality from 192 to eight dimensions, the results are not as bad as might seem to be at the first sight, when compared to the previous Table 4.2. Clearly, the overall performance is a bit worse, especially aiming at recall, but the generalization property should be substantially better. Moreover, we also reduce the computing complexity as a selection algorithm (e.g., Median of medians, [Knuth, 1998]) with the linear complexity $\mathcal{O}(n)$ can be utilized to select the specific quantile instead of a sorting one (e.g., Merge sort) with the logarithmic complexity $\mathcal{O}(n \log n)$.

Figure 4.11 illustrates the importance of individual features using the same approach with absolute values of weights as before. It appears that the most important are downloaded bytes followed by the number of unique User-Agent strings. The rest of features is more or less equally important apart from the number of persistent connections. The good news is that the classifier is not solely dependent on the number of host's operating systems and Internet browsers. This means that HTTP proxy logs contain richer information, than

just these two particular features, which can be leveraged in order to infer NAT devices. This source of information has not been considered in works of [Maier et al., 2011] nor [Bai Xue, 2009], even thought they used HTTP meta-data as well.

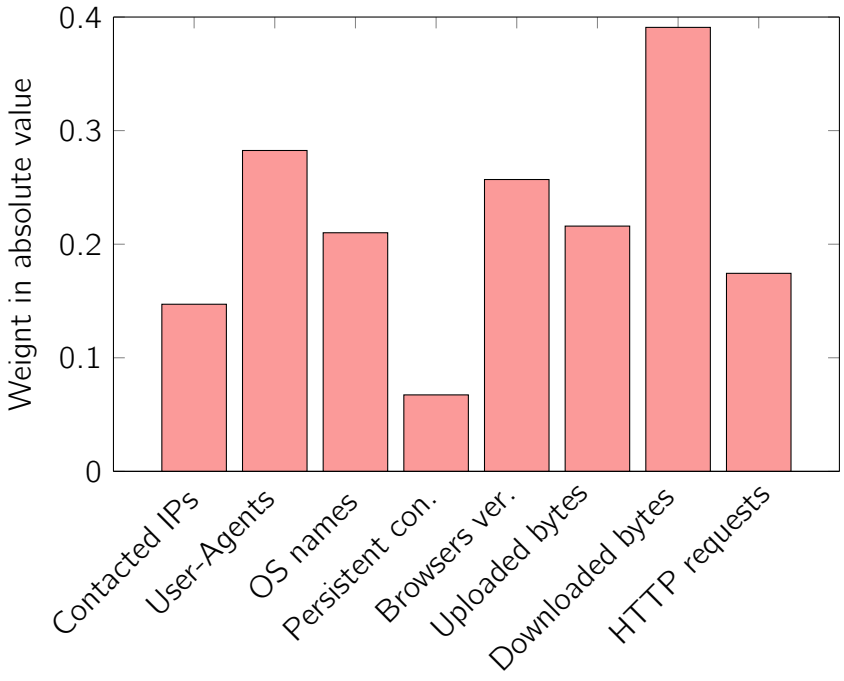


Figure 4.11: Importance of individual features.

The process of selecting a small subset of the most informative[32] features, in order to reduce the risk of over-fitting, is known as *feature selection* (FS). By using a proper FS method such as the Sequential Feature Selection (SFS), we could achieve better results than obtained by following the weights of the classifier. This may be reconsidered in the potential future work. For the sake of completeness, we also applied the same scenario on LR. The results were similar to the ones in Table 4.3, but not significantly better according to our primary F_{05} worst case metric.

[32]Meaning, the most informative with respect to the given task.

4.6.7 Detection trade-off

One of the requirements for the detector from Chapter 2 is to have high precision. This means that we should be assured of classifier's positive predictions with a certain level of confidence. Once a host is identified as NAT device, it should be indeed a real NAT device. We have already included our preference of precision into the model selection phase, since we used F_{05} measure as the selection criteria. In this subsection, we present another way to increase the precision. However, the purpose of this part is only illustrative as it will not be applied in the rest of the thesis.

Intuitively, a degree of prediction belief is related to distance from the decision boundary in the feature space. The further the host is, the more confident about the prediction we are. Figure 4.12 depicts a histogram of distances from all hosts to the decision boundary of the linear SVM classifier. To do this, we projected feature vectors of individual hosts onto SVM's hyperplane using the inner product. For this purpose, we prepared a validation set which is balanced in terms of portions of individual networks and positive vs. negative instances. Likewise the training set, the new validation set contains about 20 000 instances of active hosts. Moreover, only the representative quantiles, one specific window for each of eight features, are selected from the both sets.

As can be seen, there is a typical trade-off between the number of FPs (type I error) and FNs (type II error). The default zero threshold (the dashed line) given by the signum function (Equation 4.10) is not always optimal in all applications. For instance here, in the NAT detection task, we could be willing to increase FNs in order to decrease FPs.

A new threshold[33], optimal for a particular application, can be found by solving Neyman-Pearson task [Schlesinger and Hlavac, 2002]. A solution (decision strategy) minimizes probability of false negative predictions, for a given certain level of false positive predictions. The task can be formulated and solved using

[33] In general, the solution can be a set of thresholds (strategy) defining prediction intervals with positive/negative class.

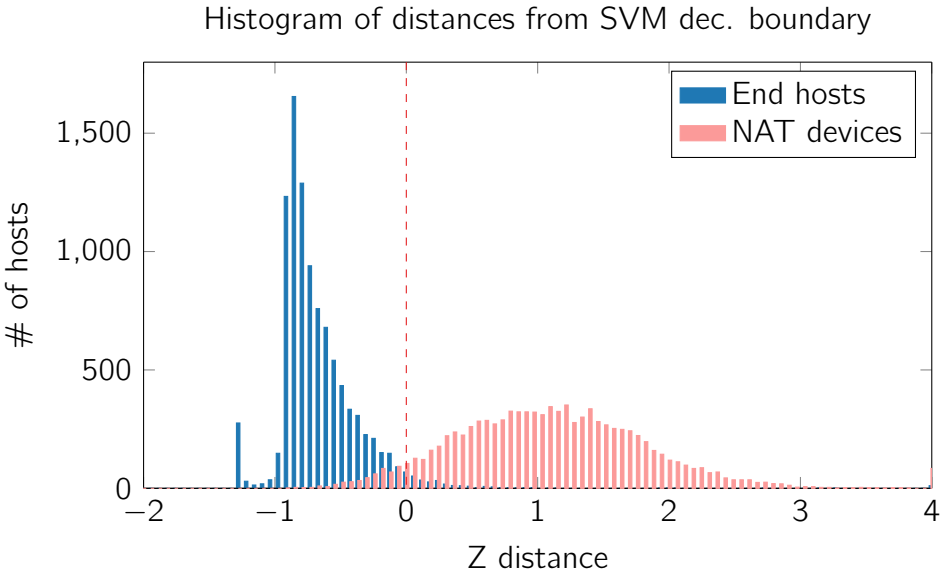


Figure 4.12: Histogram of projections onto SVM's decision boundary.

linear programming (LP). An alternative approach is to turn binary predictions into probabilities:

$$\mathrm{P}(y = 1|\mathbf{x}) = (1 + \exp(Az + B))^{-1}, \quad z = \boldsymbol{\omega}^{\mathrm{T}}\mathbf{x}, \tag{4.16}$$

similarly as LR does it (Equation 4.14). The predictions then have probabilistic interpretation of belonging to the positive class and the classification threshold can be easily adjust according to Receiver Operating Characteristic (ROC) curve or Precision-Recall curve, for example. ROC curve shows True Positive Rate (TPR) vs. False Positive Rate (FPR) depending on the threshold value. To this end, we utilized [Platt, 1999]'s algorithm implemented in LIBSVM library. This technique is based on the observation that distribution of distances is usually exponential when hosts are on the wrong side of the decision boundary. The algorithm estimates A and B by minimizing the negative log likelihood of training instances. Figure 4.13 compares LR and linear SVM augmented by the probability estimates using ROC curve and Precision-Recall curve.

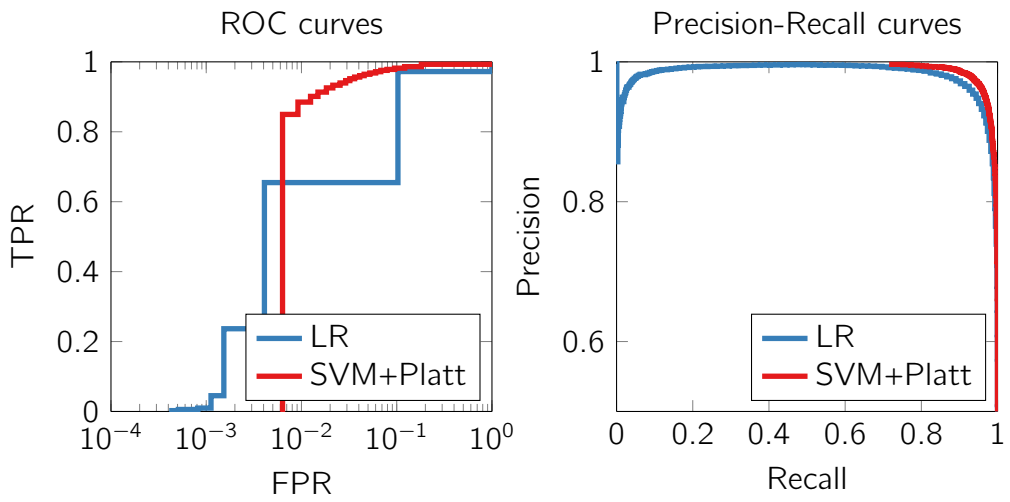


Figure 4.13: Comparison of LR and linear SVM with probability estimates.

The results can be characterized by calculating areas under the ROC curves. This typical metric is known as Area Under Curve (AUC) and counts 0.9862 and 0.9915 for LR and linear SVM, respectively. Again, liner SVM offers slightly better overall performance. A new (probabilistic) threshold can be easily picked using one of these two characteristic curves.

There is also a room for improvement. A higher confidence in predicting the positives, and therefore a lower chance of FPs, could be obtained with Asymmetric Support Vector Machine (ASVM) learning algorithm, [Wu et al., 2008]. It allows to train the classifier at a specific false positive rate. Furthermore, we could shift the distribution of generated NATs in the training set, e.g., from the original configuration $\mathcal{U}[3, 12]$ to $\mathcal{U}[5, 12]$, which should result in a movement with the decision boundary in the desired direction as well. On top of that, other metric weighting more precision than F_{05} measure is doing, could be used during the model selection phase.

4.7 Structure of the detector

The overall detection system can be viewed as two stage procedure. Figure 4.14 depicts the individual steps. At the input, the NAT detector is fed

by HTTP proxy logs. In the first stage called Host behavior analysis, the collection of eight features is extracted for each host identified in the logs. To capture behavior in time, statistics of these features are collected in non-overlapping time windows with a predefined length. The sequence of windows for one particular feature forms first-in fist-out (FIFO) queue covering last 24 hours. From this sequence, one representative window is selected according to the given quantile using a selection algorithm (e.g., Quickselect or Median of medians). Hence at the end of this stage, every host is represented by the feature vector consisting of eight components. On the basis of this feature vector, the next stage of the procedure classifies the corresponding host as either NAT devices or end host. The classification process involves: logarithm transformation $\ln(x+1)$ of features associated with downloaded and uploaded bytes, standardization by vectors $\bar{\mathbf{x}}$ and $\mathbf{s}$ (List of pre-processing steps 4.4), and evaluation of linear SVM (Equation 4.11). The output of the procedure can be viewed as a table of hosts with assigned labels.

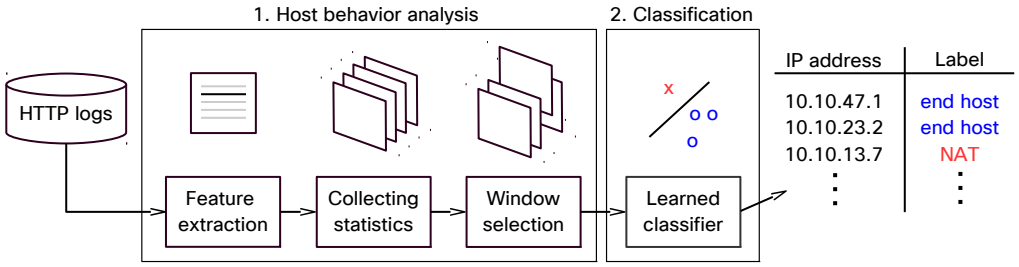


Figure 4.14: Two stage detection procedure.

On the whole, the proposed system is specified by the sequence of described steps and the following set of numbers $\bar{\mathbf{x}}, \mathbf{s}, \boldsymbol{\omega}$ including quantiles (i.e., indices of representative windows, Figure 4.10). These parameters are derived from the provided data for the training phase.

Chapter 5

Experimental evaluation

So far, the training and the validation set have been used to train and select the final classifier. In this chapter, we evaluate the proposed NAT detector according to various scenarios on the yet unseen testing set in order to obtain unbiased estimate of its real performance.

5.1 Concept drift

The phenomena known as *concept drift* is related to i.i.d. assumption made on data. In many real world applications, the data are drawn from non-stationary distributions instead of fixed, yet unknown, distributions. This might not be obvious at the first sight and it may result in poor classifier's generalization ability, [Hoens et al., 2012]. The first two experiments can be viewed as a test for the presence of concept drift in time-changing data. Generally speaking, we are wondering whether the detector trained on samples from one day is able to reliably operate on future days.

We evaluate the classifier on the training and the testing set. The sets are created using available data from the first and the third day of all four networks. Both sets are pre-processed in the standard way as described in Section 4.4. The only difference is that outliers have not been removed from the

data set	accuracy	F_1	F_{05}	precision	recall
training	0.9581	0.9574	0.9672	0.9739	0.9414
testing	0.9552	0.9546	**0.9622**	0.9674	0.9421

Table 5.1: Discrepancy in performance between the training set and the testing test. The scenario with balanced sets and different days is applied. As it is seen, the classifier generalizes well and concept-drift in time has not emerged.

testing set. As discussed in Section 4.5, the outliers might represent potentially wrongly labeled NAT devices. The mislabeling error is introduced during the process of generating artificial NATs (Section 4.3). As result, the real NAT devices (i.e., not artificially simulated ones) already present in the networks will actually appear among classifier's false positive predictions (FPs). With a perfect classifier, all classifier's false positives would be constituted by these real NAT devices. Unfortunately, because we are not sure about their true origin in general, we assess the performance including them. Therefore, the presented metrics can be considered as lower-bound estimates. We only try to remove mislabeled huge gateways (i.e., with many connected NATed hosts) from the training set to avoid learning biased models. As convex loss functions (Figure 4.2) dictate enormous penalizations for wrong predictions, which are actually correct, especially when training samples lie far from the decision boundary (the case of huge gateways). To cover all networks equally, hosts are randomly sampled from the prevalent networks. Moreover, we are interested in active hosts only, because they represent hard examples from the classification point of view as they are located in the vicinity of classifier's decision boundary (Figure 4.6). Active hosts were defined as those who make at least six HTTP requests in five arbitrary selected 30 minutes period windows from last 24 hours. Finally, the portion of selected end-hosts and NAT devices is equal. Table 5.1 shows detector's performance on the testing and the training set for the described scenario.

The test is considered to be negative, as we have not noticed any substantial deterioration when we trained the detector on one day and evaluated it on another one. This implies that our detector should work as time evolves, which

network	all hosts	real hosts	accuracy	F_1	F_{05}	precision	recall	FPs
tiny	8 632	6 182	0.9751	0.9563	0.9534	0.9515	0.9612	120 (1.94%)
small	17 241	11 901	0.9733	0.9557	0.9718	0.9828	0.9301	87 (0.73%)
medium	37 860	24 190	0.9786	0.9510	0.9722	0.9869	0.9176	213 (0.88%)
large	77 548	60 018	0.9652	0.9519	**0.9503**	0.9493	0.9546	697 (1.16%)

Table 5.2: Detailed results on particular networks applying the scenario with different days. The average percentage of discovered NAT devices is about 1.18% of all real hosts.

is of course desirable. Moreover, the results indicate that the NAT detector generalizes well as there is no significant change between performance on the training and the testing set. There are only small deteriorations in metrics apart from the precision. The above described sampling process might cause randomnesses in values. In the extreme case of the largest network, about 5 000 instances out of 23 000 are randomly selected. Thus to provide representative results, the values correspond to mean values from ten separate runs with ten different sampled data. The results seem to be stable as the maximum standard deviation among all metrics is 0.002. In comparison with works of others from Section 3, the results are remarkable as none of metrics falls below 94%.

The next Table 5.2 presents an output from the experiment, where the identical training set is used, but instead of evaluating on the subsample testing set, the detector is tested on each network individually. The metrics might be skewed as roughly 40% of all hosts are inactive, 30% are active and the rest is artificially generated NAT devices. Nevertheless, they reveal the numbers of discovered NAT devices in these networks. According to FPs the average percentage of NAT devices is about 1.18% of all real hosts (i.e., excluding artificial NATs). This finding is in accordance with the related work of [Mongkolluksamee et al., 2012]. They analyzed longitudinal traffic traces at a trans-Pacific link in 2001-2010 and found that the percentage of the NAT devices is stably less than 2% over years.

In Section 4.5 we were suspecting whether the conditional distribution is shared among all networks. In other words, we would like to be sure if the de-

aggregation	accuracy	F_1	F_{05}	precision	recall
average	0.9536	0.9529	0.9624	0.9691	0.9379
min	0.9475	0.9451	**0.9470**	0.9444	0.9036

Table 5.3: Cross-validation (leave one network out) supporting detector's applicability on first seen networks. Each network is represented by randomly drawn active hosts in total number of 5 000 samples.

tector trained on these four networks, can be used in other networks. To test our detector for this type of concept drift, we propose the following experiment based on cross-validation. The classifier is trained on three networks and then evaluated on the remaining one. This cross-validation process is repeated four times, so that each network is used exactly once as the testing set. The partial results from each iteration are aggregated by selecting the minimum or the average value. The aggregated results are shown in Table 5.3.

The minimum value represents the worst case combination of three training networks and one testing network. Again, the results indicate that the test against this type of concept drift is negative. Consequently, the detector trained on samples from a few networks can be deployed on unseen networks without significant loss of performance. The minimum achieved classification accuracy 94.75% still outperforms the accuracy 89.39% reported by our most relevant competitor [Abt et al., 2013].

5.2 Error rate

Considering classifier's FPs (discovered real NAT devices), we are able to verify their nature only by manual inspection of their behavior. Just a few of them disclose their origin via hostnames including sub-strings like "gateway" or "gw". Hence, mostly labels of large gateways were successfully checked. Unfortunately, we could not verify small NAT devices because of their ambiguous behavior.

On the contrary, classifier's false negative predictions (FNs) can be analyzed in more detail. Unlike works of others, reviewed in Chapter 3, we have the distribution of NATs under our control. This means that we are able to simulate and evaluate various scenarios. In Section 4.3 we decided to use the uniform distribution $\mathcal{U}[3, 12]$ for generation, referring to the principle of maximum entropy. Figure 5.1 depicts the distribution of the testing set from the first experiment. Note that NAT devices of size one represent end-hosts. Apart from 10 000 end-hosts, there are 10 000 NAT devices covering ten distinct sizes, each with 1 000 instances. On the right side of the figure, there are classifier's FNs corresponding to overlooked NAT devices from the testing set.

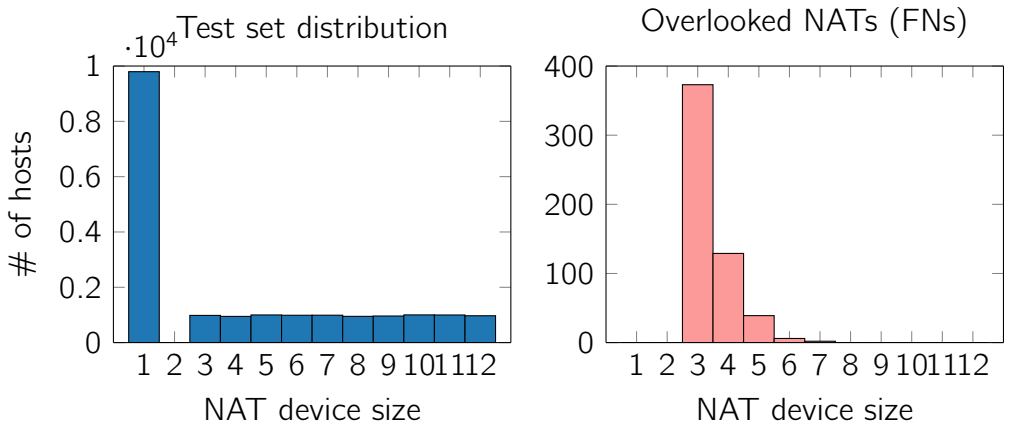


Figure 5.1: Distribution of the testing set and overlooked NATs with an exponential error rate decay.

As one can see, the detector missed 38% of all NAT devices of size three. Since the error rate follows nice exponential decay, NAT devices with five connected hosts are detected in more than 95% cases. From this illustration is clear that huge gateways with dozens of hosts are easily detectable when compared to small NAT devices with only a few (i.e., three or four) NATed hosts. This lead us to raise objections against a prior art, especially [Abt et al., 2013] using a similar behavior oriented approach on NetFlow data, that the NAT sizes are not specified at all. As we believe that in the real world scenario, the prevalence of bigger NATs is more common. This hypothesis

is also supported by [Mongkolluksamee et al., 2012], who reported that the average number of hosts connected to NATs was six in 2010. Taking into account their historical data, the tendency was gradually increasing. This observation can be correlated with the decreasing number of available public IPv4 addresses. Moreover, their estimate represents lower-bound as their solution was not capable to detect OpenBSD hosts. Hence our scenario with the uniform distribution and the consequent results can be considered as a more critical assessment in contrast to [Abt et al., 2013].

5.3 Degenerate networks

In the industry, popular technique for the NAT detection using HTTP metadata is OS/browser fingerprinting. In the given time period (e.g., 24 hours), the fingerprinting method counts the number of different OS and/or browsers used by a host. For example, a host running Windows 7 and Ubuntu, and using Firefox, Chrome and Internet Explorer will have fingerprint (3,2). That is, the counts of different Internet browsers and OSs. A host can be labeled as a NAT if both counts are above a certain threshold.

We implemented the fingerprinting method to compare its performance against our solution. On our data, the method achieved the best F_{05} measure when predicting as a NAT host if more than two OSs and two Internet browsers are identified. As can be seen in Figure 5.2, the fingerprinting method with this choice of threshold return a large number of FNs, as many NATs use two OSs and two Internet browsers. Comparing the fingerprinting method against our NAT detector (Figure 5.3) on the testing data, we can see that the fingerprinting method performs slightly worse.

The advantage of our solution, however, is getting on importance especially in degenerate networks with NAT devices consisting of NATed hosts having the identical OS and/or Internet browser. This scenario is common in the majority of networks in the banking and financial industry, public administration and corporate environments where the default setup of computers is enforced.

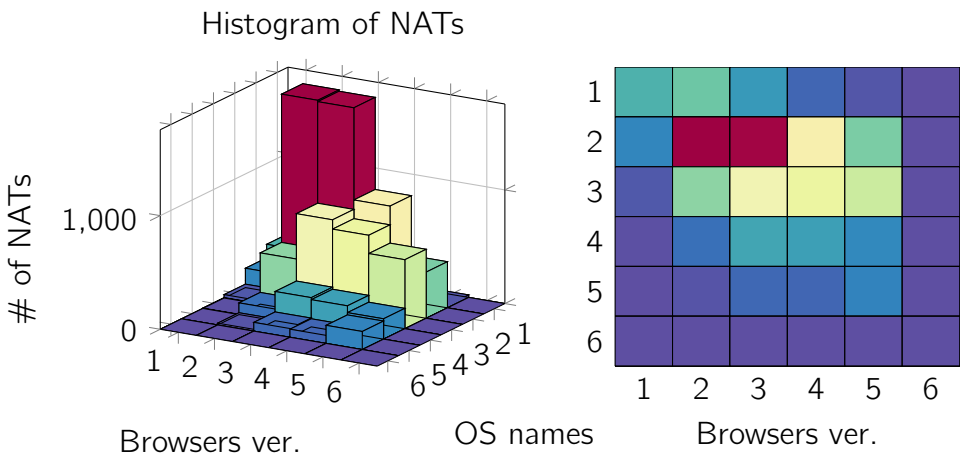


Figure 5.2: 2D histogram of Internet browsers and OS counts for NATs from the testing set (a regular network). Not negligible amount of NATs use less than three OS and three Internet browsers. These NAT devices can not be discovered by the fingerprinting method on principle. Relaxing thresholds would lead to higher FPs as there are lots of hosts having two OSs and two Internet browsers as parsed from User-Agent strings.

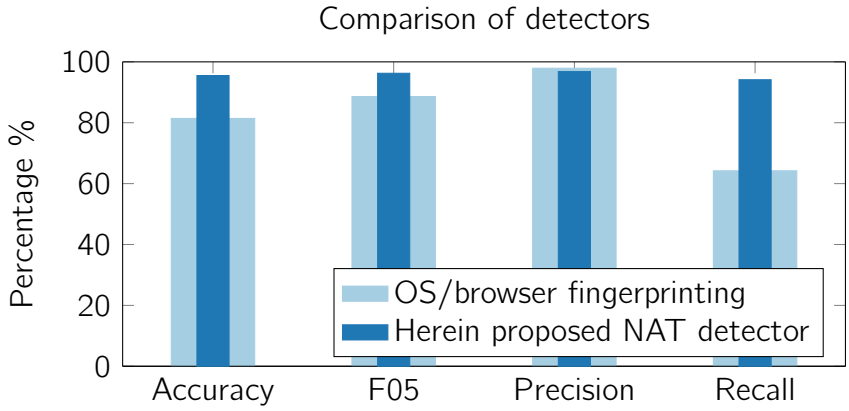


Figure 5.3: Comparison of the fingerprinting method and the proposed NAT detector on the testing set (a regular network). The fingerprinting method performs slightly worse.

In these degenerate networks, the fingerprinting method unavoidably fails. Nevertheless, our proposed solution is still capable of detecting NATs with high accuracy, as can be seen from Table 5.4.

detector	accuracy	F_1	F_{05}	precision	recall
	A regular network				
fingerprinting	0.8954	0.8937	0.9024	0.9083	0.8795
NAT detector	0.9551	0.9545	0.9624	0.9680	0.9413
	Hosts with the same OS				
fingerprinting	0.5000	N/A	N/A	N/A	0
NAT detector	0.9391	0.9362	0.9638	0.9831	0.8935
	Hosts with the same OS and Internet browser				
fingerprinting	0.5000	N/A	N/A	N/A	0
NAT detector	0.9431	0.8671	0.8819	0.8921	0.8435

Table 5.4: Comparison of the fingerprinting method and the proposed NAT detector on three networks. The first network represents a regular network with hosts running naturally manifold OSs and Internet browsers. The next two networks stand for degenerate networks constituted by hosts with the same OS and/or Internet browser.

5.4 NAT devices in networks

In our last experiment, we run an implementation of the proposed NAT detector on 60 corporate networks. The networks are of various sizes from 50 to 150 000 hosts. Figure 5.4 shows a histogram of percentages of detected NAT devices in these networks.

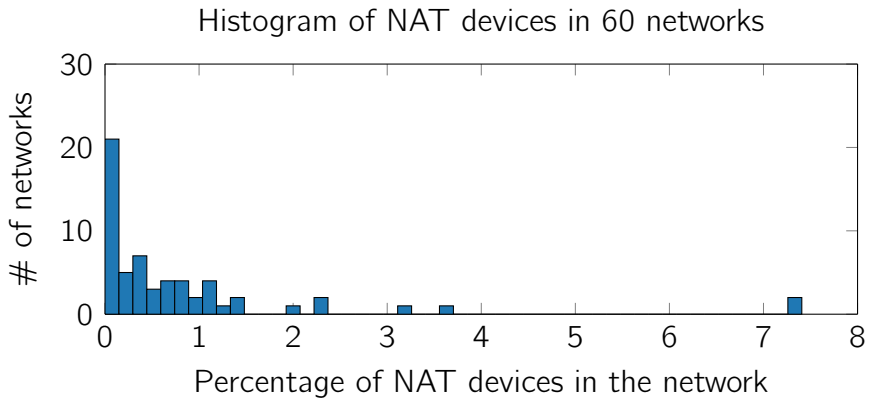


Figure 5.4: Percentages of detected NAT devices in 60 various networks.

The majority of networks does not exceed 2% level of NAT devices. Again, this observation is in accordance with the already mentioned analysis made by [Mongkolluksamee et al., 2012]. The networks with higher percentages are typically of size small in terms of visible hosts, but contain huge authorized gateways operating as central focal points for inner (i.e., invisible) hosts.

Chapter 6

Conclusion

This thesis addresses the problem of detecting NAT devices in the computer networks. In Chapter 2, we motivate our effort by realization that each NAT device is a potential vulnerability. As these rogue network devices open the possibility for conducting malicious activity. Moreover, it also turns out that an effective and accurate solution for inferring NAT devices is needed for purposes of further network behavior analysis.

Although the NAT detection is long lasting field of research as it arises from Chapter 3, we found only two methods based on behavior analysis. Unlike traditional approaches that search for signatures in packet headers, behavior oriented methods do not rely on any specific field in packets. This makes the behavioral approach more useful in the area of network security as it is not easy to trick such a system in order to avoid discovery. In contrast to the two existing methods, the herein proposed detector uses information contained within HTTP proxy logs instead of NetFlow data, and as far as we know this has not been examined before.

In Chapter 4, we developed a passive NAT detector using supervised learning algorithm. We model behavior of network hosts using eight features extracted from HTTP proxy logs. These features are collected within consecutive non-overlapping time windows. According to our empirical analysis, 30 minutes

period windows yielded to the best results. To be able to train classifiers with a sufficient amount of labeled data, we proposed a way to generate artificial NAT traffic by merging HTTP logs of multiple hosts. This is the essential idea, since labeled data is really hard to obtain because of their natural imbalanced ratio. Typically in the network, there is only a few NAT devices compared to the total number of hosts. Having the data, we experimented with three learning algorithms. Linear SVM showed to be the right choice due to its high accuracy and speed of classification. In order to increase classifier's generalization ability, for each feature only one representative time window from last 24 hours is selected. This feature selection is done according to classifier's preference. Results indicate that it makes do without significant loss of performance. The analysis of the preference showed that the classifier generally prefers windows associated with lower quantiles. We provided a possible explanation accompanied with the illustration that lower quantiles better characterize host's stable behavior, whereas higher quantiles might be caused by host's abnormal short-time activity. Finally, we prestented a way to balance the trade-off between the confident in predicting and overlooking NAT devices using probabilistic estimates.

In Chapter 5, we experimentally evaluated the proposed solution on real network data applying various scenarios. Tests for the presence of concept drift showed that the detector trained on samples from one day is able to reliable work on futures days. Likewise, we also demonstrated using cross-validation that the NAT detector is capable of operating on yet unseen networks. We achieved detection accuracy of 94.75%, which outperforms the state of the art represented by [Abt et al., 2013]. The detector was run on a balanced data set with the equal portion of negative and positive samples. The testing set was based on four different networks. It is important to note that only active hosts representing hard examples from classifier's perspective were included. On top of that, we achieved the result on the worst cross-validation combination with three training networks and one testing network. Unlike [Abt et al., 2013], we specified the detection rate with respect to the number

of connected hosts. For example, NAT devices with three connected hosts are correctly detected in 68% of the cases and NATs with five hosts in 96% of the cases. The error rate seems to have an exponential decay. Additionally, we simulated degenerate networks with hosts having the same OS and/or Internet browser. Even thought the common fingerprinting method failed under this scenario, our detector operated with still acceptable accuracy. This implies that HTTP meta-data contain richer information, than just host's OS name and Internet browser, which had not been leveraged by any of the prior works. Using the NAT detector, we explored 60 different networks. In average, the detected NAT devices constituted by no more than 2% percentages of all network hosts.

Through out this thesis we mentioned several open points for future work. Classifier's performance could be improved by using a proper feature selection method and/or direct training at a low false positive rate. To this end, it could be also useful to study discovered NAT devices in order to reveal other characteristic properties such as the distribution of assigned source ports. NAT size estimation as well as identifying individual NATed hosts are another challenging areas of future research.

Bibliography

[rfc, 1981] (1981). RFC 791 internet protocol. Internet resource, http://tools.ietf.org/html/rfc791.

[rfc, 1994] (1994). RFC 1631 the ip network address translator. Internet resource, http://tools.ietf.org/html/rfc1631.

[rfc, 1999] (1999). RFC 791 hypertext transfer protocol http/1.1. Internet resource, https://tools.ietf.org/html/rfc2616.

[HAP, 2015] (2015). HAProxy. Internet resource, http://www.haproxy.org/.

[Squ, 2015] (2015). Squid proxy server. Internet resource, http://www.squid-cache.org/.

[Abt et al., 2013] Abt, S., Dietz, C., Baier, H., and Petrović, S. (2013). Passive remote source NAT detection using behavior statistics derived from NetFlow. 7943:148–159.

[Abu-Mostafa et al., 2012] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012). *Learning From Data*. AMLBook.

[Bai Xue, 2009] Bai Xue, Qian Bu-ren, L. H.-q. (2009). A scheme for counting NATted hosts. page 46.

[Banko and Brill, 2001] Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation.

[Bellovin, 2002] Bellovin, S. M. (2002). A technique for counting NATted hosts. pages 267–272.

[Bengio et al., 2012] Bengio, Y., Courville, A. C., and Vincent, P. (2012). Representation learning: A review and new perspectives. *CoRR*, abs/1206.5538.

[Bergstra et al., 2011] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc.

[Beverly, 2004] Beverly, R. (2004). A robust classifier for passive TCP/IP fingerprinting. 3015:158–167.

[Blum et al., 1973] Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L., and Tarjan, R. E. (1973). Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448 – 461.

[Bruzzone and Persello, 2009] Bruzzone, L. and Persello, C. (2009). A novel context-sensitive semisupervised SVM classifier robust to mislabeled training samples. *IEEE T. Geoscience and Remote Sensing*, 47(7-2):2142–2154.

[Caruana and Niculescu-Mizil, 2005] Caruana, R. and Niculescu-Mizil, A. (2005). An empirical comparison of supervised learning algorithms using different performance metrics. In *In Proc. 23 rd Intl. Conf. Machine learning ICML'06*, pages 161–168.

[C.O.S. Sorzano, 2014] C.O.S. Sorzano, J. Vargas, A. P. M. (2014). A survey of dimension reduction techniques.

[Dundar et al., 2007] Dundar, M., Krishnapuram, B., Bi, J., and Rao, R. B. (2007). Learning classifiers when the training data is not IID. In *IJCAI*, pages 756–761.

[Halevy et al., 2009] Halevy, A., Norvig, P., and Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12.

[Hodge and Austin, 2004] Hodge, V. J. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22.

[Hoens et al., 2012] Hoens, T., Polikar, R., and Chawla, N. (2012). Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 1(1):89–101.

[Jaynes and Rosenkrantz, 1983] Jaynes, E. T. E. T. and Rosenkrantz, R. D., editors (1983). *E.T. Jaynes : papers on probability, statistics, and statistical physics*. Synthese library. D. Reidel Hingham, MA, Dordrecht, Holland, Boston. Includes index.

[Knuth, 1998] Knuth, D. (1998). *The Art of Computer Programming: Sorting and searching*. The Art of Computer Programming. Addison-Wesley.

[Kohno et al., 2005] Kohno, T., Broido, A., and Claffy, K. (2005). Remote physical device fingerprinting. pages 211–225.

[Krmicek et al., 2009] Krmicek, V., Vykopal, J., and Krejci, R. (2009). Netflow based system for nat detection.

[Lasserre and Bishop, 2007] Lasserre, J. and Bishop, C. M. (2007). Generative or discriminative? getting the best of both worlds. *BAYESIAN STATISTICS*, 8:3–24.

[Lyon, 2006] Lyon, G. (2006). Remote OS detection via TCP/IP fingerprinting (2nd generation). Internet resource, http://nmap.org/.

[Maier et al., 2011] Maier, G., Schneider, F., and Feldmann, A. (2011). NAT usage in residential broadband networks. 6579:32–41.

[Miller, 2008] Miller, T. (2008). Passive OS fingerprinting: Details and techniques. Internet resource, http://www.ouah.org/incosfingerp.htm.

[Mongkolluksamee et al., 2012] Mongkolluksamee, S., Fukuda, K., and Pongpaibool, P. (2012). Counting NATted hosts by observing tcp/ip field behaviors. pages 1265–1270.

[Moore, 2010] Moore, D. (2010). *The Basic Practice of Statistics*. Freeman.

[Ng and Jordan, 2002] Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press.

[Phaal, 2009] Phaal, P. (2009). Detecting NAT devices using sFlow. Internet resource, http://www.sflow.org/detectNAT/.

[Platt, 1999] Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press.

[Powers, 2007] Powers, D. M. W. (2007). Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. Technical Report SIE-07-001, School of Informatics and Engineering, Flinders University, Adelaide, Australia.

[Rui et al., 2009] Rui, L., Hongliang, Z., Yang, X., Yang, X., and Cong, W. (2009). Remote NAT detect algorithm based on Support Vector Machine. pages 1–4.

[Schlesinger and Hlavac, 2002] Schlesinger, M. and Hlavac, V. (2002). *Ten Lectures on Statistical and Structural Pattern Recognition*. Computational imaging and vision. Kluwer Academic.

[Schölkopf and Smola, 2002] Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press.

[Smith, 2002] Smith, S. W. (2002). *Digital Signal Processing: A Practical Guide for Engineers and Scientists*.

[Vapnik, 1998] Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

[Verboven and Hubert, 2004] Verboven, S. and Hubert, M. (2004). LIBRA: a MATLAB Library for Robust Analysis.

[Wolpert, 1996] Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms.

[Wu et al., 2008] Wu, S.-H., Lin, K.-P., Chen, C.-M., and Chen, M.-S. (2008). Asymmetric support vector machines: Low false-positive learning under the user tolerance. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 749–757, New York, NY, USA. ACM.

[Zalewski, 2012] Zalewski, M. (2012). Passive OS fingerprinting tool. Internet resource, http://lcamtuf.coredump.cx/p0f3/.