FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Classification of the traffic content within Tor connection
Student:	Lukáš Jančička
Supervisor:	Ing. Tomáš Čejka, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	until the end of summer semester 2021/2022

Instructions

Investigate Tor network protocol and its traffic, and also study flow-based network traffic monitoring and analysis.
Create a Tor traffic dataset either from publicly available samples or using some testing environment.
Design a feature set and a classification algorithm that can identify Tor communication and estimate content of the Tor connections.
Create a prototype of the classification algorithm that can distinguish several traffic categories (chosen in cooperation with the supervisor).
Evaluate the created software prototype for Tor traffic classification using the testing data.

Electronically approved by doc. Ing. Jan Janoušek, Ph.D. on 14 January 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Classification of the traffic content within Tor connection

Lukáš Jančička

Department of Theoretical Computer Science
Supervisor: Ing. Tomáš Čejka, Ph.D.

May 13, 2021

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Tomáš Čejka, Ph.D., for his valuable guidance. I would also like to express my appreciation to the Network Traffic Monitoring Lab members for their assistance. Finally, my thanks go to my friends and family for all their support and encouragement during the creation of this thesis.

Declaration

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

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 13, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Jančička, Lukáš. *Classification of the traffic content within Tor connection.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstract

This thesis deals with the detection of the Tor anonymity network and the classification of its traffic using machine learning techniques. Statistical properties of network traffic extracted from the network flow data are used for training a variety of supervised learning models. AdaBoost model was the best performing for both the Tor detection and Tor traffic category classification. Machine learning offers a viable approach to detecting Tor traffic, as the final classifier detected 94 % of Tor samples and was 99 % precise in those decisions, with the F-score being 96 %. The second classifier distinguishes between eight traffic categories and does that with an accuracy of 65 %. The results demonstrate that even though Tor encrypts the traffic, some information about the user's activity can still be revealed.

Keywords anonymity networks, network traffic analysis, Tor, Tor traffic detection, Tor traffic classification, machine learning, network flow

Abstrakt

Tato bakalářská práce se zabývá detekcí anonymizační sítě Tor a klasifikací jejího provozu pomocí metod strojového učení. Statistické vlastnosti síťového provozu získané z dat ve formě síťových toků jsou použity k trénování různých modelů supervizovaného učení. Model AdaBoost podával nejlepší výsledky jak v detekci Toru, tak v klasifikaci kategorie provozu sítě Tor. Strojové učení se ukazuje být vhodným přístupem pro detekci sítě Tor, neboť finální klasifikátor dokázal detekovat 94 % vzorků provozu sítě Tor a v těchto rozhodnutích byl přesný na 99 %, s F-skóre 96 %. Druhý klasifikátor rozlišuje mezi osmi kategoriemi provozu a vykazuje klasifikační přesnost 65 %. Výsledky ukazují, že některé informace o aktivitě uživatele lze zjistit i přes fakt, že síť Tor šifruje svůj síťový provoz.

Klíčová slova anonymizační sítě, analýza síťového provozu, Tor, detekce provozu sítě Tor, klasifikace provozu sítě Tor, strojové učení, síťový tok

Contents

List of Figures

List of Tables

Introduction

More and more Internet users are becoming aware of how their Internet activity can be tracked and surveilled, and the demand for a tool enhancing privacy and anonymity rises. One of the most popular solutions for that is using the Tor network. It allows its users to browse the Internet while protecting the user's identity, with relatively low latency and high ease of use.

Tor is a popular anonymisation tool for a variety of users. It can protect the identity of people sharing sensitive information in dangerous places, such as whistle-blowers, dissidents and journalists. It enables its users to access websites blocked by their Internet provider or government in countries where the Internet is censored. On the other hand, it can also give anonymity to various illegal and illicit activities, such as the sales of drugs and black market items.

This motivates researchers to make Tor a widely explored research topic. There have been various efforts of finding attacks that can deanonymise Tor. Ways of detecting and classifying Tor by the type of application have also been researched and will make the primary goal of this thesis. Even though the traffic between the user and Tor is encrypted, it is possible to discover some presumably hidden information, such as the type of application generating that traffic, just by analysing various statistical properties of that traffic. This can be achieved by utilising machine learning, which is getting more and more popular and can solve problems in many fields. It also proves to be useful in the area of network analysis.

The goal of the theoretical part is to research the Tor network protocol and its traffic. Flow-based network traffic monitoring and analysis should also be studied. The practical parts of the thesis consist of these goals: Firstly, a dataset of Tor traffic should be created from publicly available samples or using some testing environment. Based on that data, a feature set will be designed and a classification algorithm that can identify the Tor traffic. Another goal is the creation of an algorithm that can distinguish traffic categories. The final goal is to evaluate the quality of created software prototypes.

Structure of the Thesis

The theoretical part of the work is divided into three chapters based on the different researched topics. The first chapter describes various ways of analysing network traffic, with the emphasis on flow-based analysis as the classification will be done based on data extracted from network flows.

Research into the area of machine learning is done in the next chapter. The core paradigms of machine learning are described, and the principles behind common machine learning models considered to be used in the practical part are examined. In the last part, the best practices and various solutions for evaluating the quality of trained machine learning models is studied.

Chapter 3 describes various details behind the Tor network. The core principles behind the design and traffic of Tor are studied. Related works in the area of Tor detection and classification are outlined in the last part.

The fourth chapter researches the ways of creating a dataset of Tor traffic and the available solutions. The chosen dataset is then analysed to determine if it suits this work and to gather additional knowledge about Tor traffic.

Chapter 5 describes the creation of the classification models. It shows how the features for the machine learning models are extracted and the experiments with the various models in search of the most accurate model. In the end, the best performing models for each task are chosen to be used in the software prototype.

The final chapter provides a deeper evaluation of the best performing classifiers for both designated problems. On top of that, the software prototype is shown, and the results are discussed and compared with other works.

Chapter 1

Traffic analysis

Network traffic monitoring and analysis present a crucial step in the network administrators' work of keeping the network behaving as expected. Network monitoring offers the administrators a better understanding of how the network performs, which can help solve possible issues in the network needing to be resolved. It helps to discover malicious traffic and attacks on the network and the network misuse by its users.

1.1 Individual packet inspection

Packets represent the basic unit of network data as of the network layer of the OSI model. Packets consist of the control information inside of the header and the user data inside of the payload. One approach of network traffic monitoring focuses on analysing the contents of the individual packets. It has some positives over other methods, but also some negatives, mainly the concerns about the privacy of the network's users and the issues with speed, both explained in the following sections.

1.1.1 Packet inspection methods

The analysis of the packet contents can be done at several levels. The packet header contents can be analysed, but some methods inspect the payloads, which can be seen as an intrusion of the network's users' privacy. The examples of the less intrusive methods would be the analysis of the source and destination IP addresses and ports. The IP addresses can be used for blocking unwanted connections from the blacklisted IPs. The numbers of ports can be used for estimating the service that generated the traffic. The global standards organisation IANA (Internet Assigned Numbers Authority) assigns ports to network services. It can be assumed that the services are open for communication on those designated ports, but in the end, this depends on how the devices are configured. [1]

Deep packet inspection (DPI) introduces more complex analysis of the packet content. DPI presents a accurate and widely used method in the area of network monitoring and security. For example, the methods of DPI are highly effective for classification of the network traffic and detection of malicious connections or network attacks. Usually, DPI techniques work by pattern matching the packet contents to a database of known samples. Example of DPI using the Wireshark application can be seen in figure 1.1, where the service is identified as DNS and the DNS query can be seen. [1]

The ethical side of deep packet inspection is a topic complex enough for its own research, such as [2]. There are some concerns about how DPI affects the privacy of the network's users and how it can be misused by Internet providers, governments, and advertisements. The other inconvenience of DPI is its computational efficiency for real-time network monitoring. Inspecting each packet can be incredibly computationally intensive and can negatively impact the performance of the network.

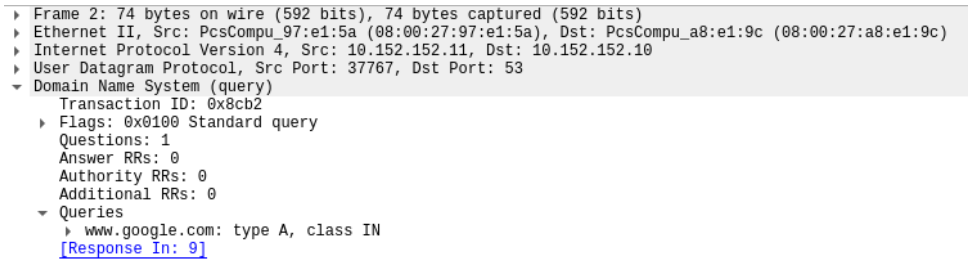

Figure 1.1: Example of deep packet inspection — analysis of DNS packet using the Wireshark application

1.1.2 Packet capturing

There are various tools for the capture and analysis of the network traffic on the packet level, sometimes known as packet sniffers. Packet sniffers can exist as a command-line applications or can have graphical interface, and there exist both commercial and open-source solutions. The examples of commonly used packet analysers would be tcpdump, OmniPeek, and Wireshark.

Packet sniffers work by first collecting the raw binary data from the network. This is usually done by using the selected network interface in a promiscuous mode. Promiscuous mode allows the network interface to listen to all traffic on its network segment, not only the traffic that got addressed to it. The captured raw binary data then gets converted into a human-readable form. This can be displayed in the command line or can be further analysed by the tool. The protocol or the service can be identified and based on that knowledge, sniffers can analyse and present information about the captured traffic specific to the protocol. [3]

1.2 Flow-based analysis

Network flows offer a layer of abstraction of captured network traffic data on top of the raw captured packets. Network flow is usually defined as a sequence of packets sharing the same quintuple of key features — source and destination IP address, source and destination port, and the protocol number. Various additional statistics, such as the count of transmitted packets, flow beginning and ending timestamps, and TCP flags, can also be measured. The captured flow data can be then analysed to examine the state of the network – to discover network incidents or to show network load, for example.

As described in [4], the idea of flow-based monitoring is based on observing the behaviour of the network and not the data itself. Even though packet-based monitoring is more powerful in some cases, monitoring flows is the more acceptable approach from the ethical point of view. Flow-based monitoring is also more robust and efficient in the use of computational resources. Flow-based analysis proves useful when working with large amounts of captured traffic. Flows sharing some common feature can be aggregated, creating flows with their statistics combined. This can be useful for long-term monitoring of the network and identification of trends.

1.2.1 Network flow standards

The idea of network flows was proposed by Cisco in the late 1990s with their NetFlow technology. To this day, there have been several iterations of the NetFlow standard and other network flow standards, as described in [5].

NetFlow v5 represents the original Cisco standard that is widely used and supported. However, it has been surpassed by other standards because of limitations such as not having the support for IPv6 traffic and not being extensible.

NetFlow v9 is the standard that dealt with the limitations of NetFlow v5. It allows the monitoring of IPv6 traffic and the customisation and extension of the flow fields.

IPFIX — Internet Protocol Flow Information Export — is the flow exportation format standardised by IETF (Internet Engineering Task Force) community of engineers. IPFIX was inspired by Cisco's NetFlow v9, but is meant to provide an open standard and be the modern and extensible alternative to NetFlow.

1.2.2 Capturing flows

The usual network flow real-time capturing architecture is designed as follows. The flow capturing is done using two different components. Flow exporter

monitors the packets and aggregates those packets into flows while calculating the flow statistics. The captured data gets exported to the flow collector, which stores the received data. It can then present the data to the administrator or send it to some flow analysis applications. The flow exportation can be done by software inside the network devices themselves or by a specially designed hardware probe. The examples of used flow exporters are:

Cisco is the original creator of the NetFlow format and offers the exportation of flows directly from their routers and switches, usually as a commercial or enterprise-level solution.

Flowmon Probe represents one of the most advanced flow exportation solutions, available both as a virtual machine or as dedicated hardware probes. It represents an example of a commercial and enterprise-level (for some of the models) solution. Flowmon offers the exportation of the application layer data, and their advanced models handle monitoring 100 Gb/s networks.

ipfixprobe flow exporter is a part of the NEMEA open-source network traffic analysis framework. It is available as software for exporting network flows from a network interface or from captured traffic in a pcap format. The captured flow traffic can be then logged to a human-readable CSV format or sent to various other analysis modules of the NEMEA framework using an internal binary format called UniRec. [6]

The different flow exports measure and store various statistics about the captured flow, other than the flow-defining quintuple. For example, in the case of ipfixprobe, the fields it exports in the basic configuration can be seen in table 1.1. On top of these primary fields, plugins can be used for exporting various additional statistic, such as those relating to a specific protocol.

1.2.3 Flow analysis examples

Flow-based monitoring has many practical use cases in the area of network management and security, as described in [4]. For example, it can be used to observe how the users comply with the network usage policy. Users and services unnecessarily straining the network can be identified. From the point of view of network administrators, one of the potentially unwanted services would be BitTorrent and other Peer-to-peer (P2P) traffic. Not only is it a potential source of illegal data sharing, but it can also put an unnecessary burden on other legitimate traffic of the network. P2P traffic can be discovered using flow monitoring.

Flow-based monitoring can also help discover attacks on the network. Port scanning can be detected by an increased number of flows. Flows can be then filtered and aggregated in a way that helps to discover the attacker.

Table 1.1: Fields exported by the ipfixprobe flow exporter in its basic configuration, taken from [6]

Field	Type	Description
DST_MAC	macaddr	destination MAC address
SRC_MAC	macaddr	source MAC address
DST_IP	ipaddr	destination IP address
SRC_IP	ipaddr	source IP address
BYTES	uint64	number of bytes (src to dst)
BYTES_REV	uint64	number of bytes (dst to src)
LINK_BIT_FIELD	uint64	exporter identification
TIME_FIRST	time	first time stamp
TIME_LAST	time	last time stamp
PACKETS	uint32	number of packets (src to dst)
PACKETS_REV	uint32	number of packets (dst to src)
DST_PORT	uint16	transport layer destination port
SRC_PORT	uint16	transport layer source port
DIR_BIT_FIELD	uint8	determines outgoing/incoming traffic
PROTOCOL	uint8	transport protocol
TCP_FLAGS	uint8	TCP protocol flags (src to dst)
TCP_FLAGS_REV	uint8	TCP protocol flags (dst to src)

Another example would be the detection of Denial-of-Service attacks, where the attacker tries to deny access to the legitimate users by overwhelming the resources of the server. Those attacks can be detected by discovering a large number of flows containing only a single packet or finding an increased number of RST flags in the opposite direction of communication.

1.3 Traffic analysis by machine learning

A variety of traditional approaches to traffic analysis were described in the preceding sections. Some tasks of the traffic analysis require a human analyst, and some are automated. Network traffic was usually classified using pattern matching of packet payloads or by relying on known port numbers. Machine learning offers an alternative approach of classification based on various statistical traffic characteristics, as described in [7].

There exist various ways how machine learning aids the area of network analysis. In the mentioned case of traffic classification and service detection, machine learning models work by training on captured traffic examples. The captured traffic is labelled by the desired class. The models then detect and generalise the differences in traffic statistics between the classes and classify unknown examples based on that trained knowledge.

Another example would be the detection of anomalous traffic. There are several ways of anomaly and outlier detection that can detect traffic anomalies based on how much they share the traffic statistics with other legitimate flows. The field of machine learning is further described in the following chapter.

Chapter 2

Machine learning

2.1 Introduction

Machine learning (ML) is a field of study combining artificial intelligence, statistics, and computer science, resulting in an alternative approach to the creation of algorithms. These algorithms improve themselves by extracting knowledge from data. This approach differs from the classic way of explicit programming, where the programmer has to exactly describe the algorithm. Machine learning can be applied to problems in various fields and can often provide a simpler solution than creating a human-made algorithm. For example, there are applications such as anomaly detection, customer segmentation of an e-shop, image recognition and stock price estimation. [8]

The ideas of machine learning are similar to the way humans learn. A child seeing some object for the first time gets told by the parents what the object is. For the child to understand how to identify other instances of the same object, it has to select relevant features of that object, such as its shape, size, colour etc. These features are often called independent variables. After learning from examples, the child can make a decision when presented an example it hasn't seen before. [9]

2.2 Paradigms

Machine learning is usually divided [9] into two main paradigms, derived from the way the algorithm learns — supervised and unsupervised learning. These two approaches can be combined, and there exist other categories of machine learning, but only these two paradigms will be further described for simplicity reasons.

2.2.1 Supervised learning

In the case of supervised learning, the ML algorithm is presented with data labelled with the desired output. The training data consists of the target variable Y and a set of independent variables X. The goal of the training is to find a mapping from X to Y, which is accurate for most examples, thus can be used to predict labels for unseen data. When the values of Y are taken from a set of a few discrete labels, we are talking about a classification problem. The other case being the regression problem, where the values of Y are taken from a continuum of real values.

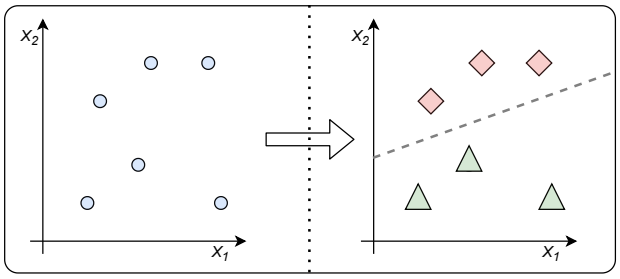

Figure 2.1: Diagram of classification

2.2.2 Unsupervised learning

In the process of unsupervised learning, no label or class is given to the algorithm. The goal of these problems is not to predict a class but to find some intrinsic structure in the data. The usual output of unsupervised learning is the segmentation of the data into clusters, where the clusters consist of data with a similar structure. Because there isn't a defined desired output in our data, the assessing of the quality of unsupervised learning models gets more complicated.

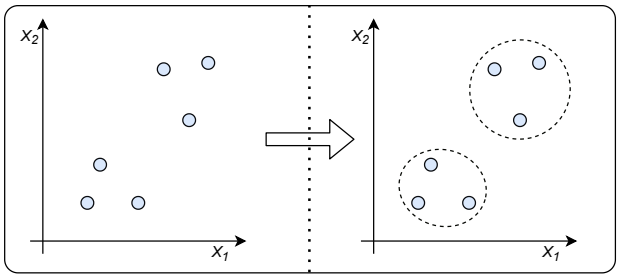

Figure 2.2: Diagram of clustering

2.3 Classification models

As the goal of this thesis is to create a classification algorithm, machine learning models used for classification, considered to be helpful with the task, will be further discussed. A wide variety of commonly used classification algorithms will be used for the experiments in this work.

2.3.1 Decision tree

Decision trees present a simple and easily understandable tree-structured model and are one of the oldest and most used techniques. The way the model makes its decision can be read and understood by humans by following the visual representation of the decision tree. The nodes represent conditional moves based on the values of the features, and the leaves represent the predicted value of the target variable. The classification of unknown data is made by following the path based on the unknown data from the root to the leaf, representing the predicted value. In the case of decision trees, a greedy approach is often used for the learning process consisting of repeatedly splitting the dataset while minimising some quantifier of disorder, such as entropy. [10]

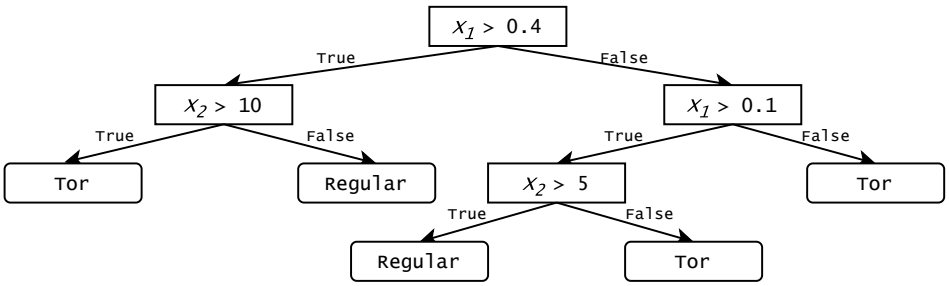

Figure 2.3: Example of a decision tree

2.3.2 Random forests

Random forests represent a model based on the technique of ensemble learning. Ensemble models are based on the idea of training multiple simpler models and combining their predictions into the final decision. This decision is usually more accurate than any of the decisions of the individual models. There exist two common approaches to combining the models in ensemble techniques — bagging (bootstrap aggregating) and boosting.

Random forests implement the ideas of bagging. Various subsets are created from the training dataset using the bootstrapping technique — selection

of samples with repetition. A decision tree, which doesn't need to be very deep, is constructed for every subset. While classifying unknown data, each tree provides a decision, with the final result being determined by the majority of those decisions. [11]

2.3.3 AdaBoost

AdaBoost (Adaptive Boosting) represents the latter of mentioned ensemble techniques — boosting. Boosting works by creating a set of models, and the decision of the individual models gets averaged into a final decision, the same as with bagging. The main difference between these two ensemble approaches is that in boosting, the models aren't independent. The models are trained sequentially, new model each round. At the end of each round, misclassified examples are found, and their weights are increased, thus making the training focus more on these misclassified instances. This means that the training of a new model in the sequence depends on previous models.

AdaBoost can use various classification methods in its ensemble of models. The only requirement is the usage of weights in the training of the model. Shallow decision trees are often used as the base estimators. [12]

2.3.4 K-nearest neighbours

K-nearest neighbours (KNN) takes a different approach than the previously discussed models. It requires no training; the computation is done during the prediction. The prediction of unknown data is made by finding k neighbouring points (meaning points having the shortest distance) to our unknown point we want to classify. The decision of the classifier is then made as the most common class of the neighbours.

The distance metric can be defined, depending on the nature of the problem, with the Euclidean or Manhattan distance being popular examples. The number of neighbours k also has to be defined by the user and changes how the model behaves. If the value is set too low, it can lead to the insufficient generalisation of the problem and the fixation on the training data. This effect is known as overfitting and results in a model that behaves worse on new, unseen data.

Because KNN makes its decision by finding the nearest points, it is very sensitive to having different types or scales in its features. Imagine having a feature representing a boolean (being either zero or one) and then having a second feature that ranges from zero to millions. The differently scaled features would not contribute to the final distance in the same way. Because of that, the data should be preprocessed; re-scaling the data to the interval $< 0, 1 >$ is usually done. [13]

2.3.5 Naive Bayes

The Naive Bayes algorithm works by estimating the conditional probabilities of the data being correctly classified based on the feature vector and selecting the most probable class. The calculations based on the Bayes rule dictate the strong assumption that the features are independent (in the sense of probability), given the class. The naming of Naive Bayes is based on this assumption, which is often false, and it can yield satisfactory results anyway.

There are several advantages to using Naive Bayes. It is very computationally efficient, robust in the face of missing values and noise. It is also less affected by the issues with high-dimensional feature vectors than many other models. However, in the case of classifying captured traffic data, the dependencies of the features might result in worse quality of the model. [14]

2.3.6 Logistic regression

Logistic regression is a classification algorithm based on linear regression, which is used for problems of regression, not classification. The linear regression model can be described by the following formula

$$Y \approx w_0 + w_1 x_1 + \ldots + w_p x_p$$

where x denotes the features and w the weights.

Using this regression model in the area of classification can be achieved by it predicting the probability of Y having a value of 1, being classified as true (considering the case of binary classification). This means the results of the regression formula have to be transformed to stay within the range of $< 0, 1 >$. Commonly used function for that is the sigmoid function, defined by:

$$f(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

The final performance of the classifier highly depends on the nature of the data. The model assumes that the relation between the target variable and the independent variables can be effectively captured with the linear regression formula. However, how well it estimates the relation varies case by case. [15]

2.3.7 Support vector machines

Support vector machines (SVMs) work by finding a way of linearly separating the training data, assuming that the data is linearly separable. The training set with examples containing n features can be understood as points in n-dimensional space. In the least complicated case of two-class classification, SVMs find a hyperplane separating the two classes with the largest possible margin. Margin is defined as the distance between the hyperplane and the closest point. Finding the largest margin leads to a better generalisation of the model and better accuracy on new, unseen data. [16]

2.4 Evaluation

At the start of the training process for supervised learning models, we have a labelled dataset, preprocessed to have no missing or non-numerical values. Some models require additional preprocessing for a better quality model, such as the normalisation of values to the interval $< 0, 1 >$.

The training is not done on the whole dataset, because of the effects of overfitting — a bad generalisation of the problem and accuracy on new data. The data has to be split in some ways; the simplest solution is to divide the data into a training and testing set. Better solution would be cross-validation, described in one of the following section 2.4.3.

Usually, the models have various parameters, which change how the individual model learns and behaves. They are called hyperparameters, and their examples could be the depth of a decision tree or a number of individual models in ensemble methods. The combination of hyperparameters that yields the best result is usually not known, so the model is trained for a variety of combinations, and the one with the best quality metric is chosen. After having a trained model, its performance can be evaluated on the testing dataset, which consists of unknown data to the model and thus should represent performance on completely new data.

2.4.1 Classification quality metrics

There exist various metrics used for assessing the quality of trained classifiers. The final quality assessment should be made by observing multiple of those metrics and can be based on the understating of the problem. Commonly used metrics, chosen from [17], considered to be used in this work are:

Accuracy denotes the fraction of correct predictions and is calculated as:

$$\text{Accuracy} = \frac{\text{count of correctly classified examples}}{\text{count of all examples}}$$

This metric gives us the idea of overall performance and is easy to understand and imagine what it represents. However, there are some caveats to be aware of when evaluating the classifier performance using this metric. For example, take the case of a highly imbalanced dataset, where 99 % of data consists of class A and only one per cent of class B. A classifier that would predict everything to be of class A would have 0.99 accuracy, which seems like a perfect result. In spite of that, it would be unusable in real life as it doesn't detect any samples of class B.

Precision is calculated as:

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

The terms positive and negative refer to the prediction of the classifier, and the terms true and false indicate how the prediction corresponds to reality. For example, true positive can be understood as a count of examples where the classifier predicted a true label and was correct in that prediction when compared to the actual label.

Recall is defined as:

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

F-score combines the information from precision and recall. The traditional F-score, known as F_1 score is defined as their harmonic mean:

$$F_1 \text{ score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

2.4.2 Confusion matrix

The confusion matrix offers a useful solution for evaluating the quality of the classifiers. It visualises the performance into a table, so it can be easily read and understood. The rows of the matrix represent the true class of the examples, while the columns represent the predicted classes. This means that the cell with coordinates i, j stores the number of examples with true class of i, which were classified as j.

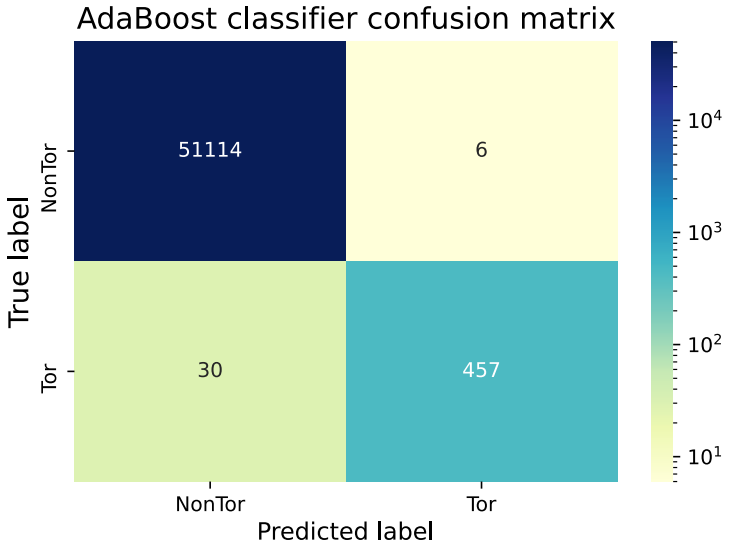

Figure 2.4: Example of a confusion matrix

2.4.3 Cross-validation

Cross-validation offers a better solution than simply dividing the dataset into training and testing sets, as described in [13]. It helps eliminating some effects of randomness as the performance of a classifier on the same data with the same hyperparameters can slightly differ every time. Using a mean of the metrics from several samples helps better estimating the general performance of the classifier on unseen data.

A common approach is a technique called the k-fold cross-validation:

1. The dataset is divided into k equally sized sets.
2. The classifier gets trained on $k - 1$ sets.
3. The set that got left out consists of unknown data, so is then used for testing and measuring of the metrics.
4. This process gets repeated k times, so every set is used for testing in one run of the algorithm.
5. The final cross-validation metrics are calculated as the mean of those metrics from individual runs.

A variation of k-fold called Stratified k-fold cross-validation proves useful on datasets consisting of imbalanced classes. It is designed for the individual folds to have approximately the same ratio of samples of each target class as the whole dataset.

Chapter 3

Tor

3.1 Introduction

Tor is a privacy-enhancing tool offering protection against common ways of network surveillance and traffic analysis. By tunnelling the traffic through a worldwide, volunteer-run network, it provides the anonymisation of its user's Internet activity and identity. Tor stands for "The Onion Router" as the service is built on the technique of onion routing[1]. The Tor network is currently maintained by The Tor Project — a research non-profit organisation.

Tor is an overlay network on top of the Internet and offers relatively low latency and ease of use. This means it is targeted at a wide variety of users. It offers a solution for Internet users aware of their digital footprint by protecting them from third-party web trackers and their Internet activity from their ISP (Internet service provider). On top of extending privacy, it provides unrestricted access to websites and services restricted by the user's ISP or in the country of their origin. This means Tor can be used to bypass censorship in countries with restricted access to the Internet. It allows reporters to protect their source, for example, when communicating with whistle-blowers or dissidents. Tor has been used by a branch of the U.S. Navy while deployed abroad and by law enforcement agencies. [18]

Although Tor was built as a tool for preventing censorship, there are ways it is being misused for illegal activities. Researchers [19] discovered that BitTorrent accounted for the majority of their captured Tor traffic. Tor is used to distribute copyrighted content anonymously, hidden from anti-piracy groups and ISPs. Tor offers a way for servers to stay anonymous called the onion services[2]. This way of running Internet services while protecting their location creates a popular platform for a variety of illegal activities such as sales of drugs and black market items, child abuse, and pornography [20]. The majority of onion services content was found to be illicit [21]. Onion services

[1] described in section 3.3
[2] described in section 3.4

were also used to protect the identity of botnet command and control servers when they transmit the instructions to the infected devices [20].

The idea of onion routing originated in 1995 when military scientists at the Naval Research Laboratory were developing ways of protecting the United States' intelligence communications over the Internet. A proof of concept prototype consisting of five nodes simulated on a single machine was shown in 1996. A year later, the U.S. military research agency DARPA (Defense Advanced Research Projects Agency) became a major investor in the project of onion routing. After the release of the research paper called *Anonymous Connections and Onion Routing* [22] describing the first generation of onion routing, its development was suspended for some time because of missing funding. The generation 0 prototype network was shut down in 2000, and its operation was further analysed for possible changes in a future generation of onion routing. This single machine setup was active for circa two years and processed over twenty million requests from more than sixty countries.

The second-generation onion routing became the implementation known as Tor and used to this date. The original Tor network was deployed in the October of 2003, and the Tor source code was released under an open-source MIT licence. A paper presenting the original design and goals of Tor called *Tor: The Second-Generation Onion Router* [23] was published in 2004. Tor became ready for the use by the general public, and after that, funding from the Naval Research Laboratory and DARPA was cut. Electronic Frontier Foundation, a non-profit advocating digital rights, became the new major investor in Tor. At that time, there were over a hundred Tor nodes on three continents. The non-profit organisation The Tor Project, Inc. was founded in 2006 to maintain and develop Tor further, and it keeps maintaining it to this day. [24, 25, 26]

Throughout the years, Tor has gained its user base and became one of the most used privacy enhancement tools. At this time, the Tor network consists of over 6,500 relays [27] with the advertised total bandwidth of nearly 600 Gbits/s of which circa 250 Gbit/s gets usually consumed daily [28]. It is estimated, that on average, more than two million people use Tor every day [29] and most users come from the United States, Russia and Germany [30].

3.2 Design goals

The main design of Tor is to provide extended anonymity and privacy on top of the TCP protocol. The latency should be low enough that it is possible for interactive applications such as web browsing, instant messaging and file transfer to be used. Tor's fundamental goal is to complicate connecting which user is communicating with which server to the attackers. The core design principles were described by the developers in their original design paper [23] as follows:

Deployability: Tor has to be designed in a way that allows its deployment and usage in the real world. This means it shouldn't be difficult and costly for the volunteers to set up and run the relays. Neither can it place a heavy liability burden on its volunteer operators.

Usability: A large enough user base is a basic requirement for the anonymity of the network. User-friendly system will be adopted by a larger amount of people. Users should be able to run Tor without complex configurations and do so on a variety of common operating systems.

Flexibility: The protocol should be designed in a flexible and well-defined way, so its design can prove itself useful for future research of low-latency anonymity networks.

Simple design: Tor should be deployed as a simple and stable system, based on proven and secure privacy enhancement techniques. The protocol has to be designed without using complex and experimental, untested principles.

The creators also defined [23] which goals aren't prioritised in their design, because they are solved in other systems or would make the design more complex:

Not peer-to-peer: There are other solutions based on decentralised peer-to-peer networks. The creators found these solutions appealing, but with too many unsolved issues.

Not secure against end-to-end attacks: The attacks where an adversary has control or can observe both the traffic incoming from client to Tor and the traffic from Tor to the sever are a possible weakness. Tor creators do not claim it protects their users against these types of attack.

No protocol normalisation: When using complex and variable protocols, other services, such as Privoxy[3], should be used to protect the identity of the client.

Not steganographic: The fact client is accessing Tor isn't hidden.

3.3 Onion routing

Onion routing is the underlying principle behind the design of Tor. The idea of onion routing is that instead of the client directly communicating with the server, the connection passes through several Onion routers (ORs). A path through various routers to the destination is created, and the client may begin communicating with the first OR. The communication gets encrypted several

[3] `https://www.privoxy.org/`

times, once for every OR in the path (This represents the onion analogy as the network consists of encryption layers that get "peeled off"). When Onion routers receive data, they remove their layer of encryption and pass the data to the next OR in the path. The final data sent to the destination isn't encrypted by the routers. This method ensures each router knows the identity of the previous router (or the client in the case of the first router) and of the next router (or the destination server in case of the last router), but the information of which client communicated with which server gets protected. [31]

Tor implements the ideas of onion routing in its specific way. Let's suppose user A wishes to communicate with a server B via Tor. Usually, a three-hop circuit to the destination is created, consisting of the ORs called entry (or guard) node, middle node and exit node. For the client to understand which routers to communicate with, which act as entry nodes, which have been compromised etc. it fetches this information from directory servers. Directory servers are trusted ORs defined by the creators, storing the state of the network. At the time of writing, there are nine running directory servers [32]. When creating the circuit, a set of symmetric keys gets negotiated, one for every OR in the circuit.

The construction of the circuit and key negotiation is done incrementally, with one router at a time. The key negotiation is based on the Diffie-Hellman key exchange method. After the circuit is constructed, client has a negotiated key with every OR and may begin communicating. Client encrypts the message with every one of the negotiated symmetric keys. The encrypted message gets sent to the first OR — the guard node. Guard node decrypts the message with its key and relays it to the middle node. At that time, only one layer of encryption was "peeled off", so the middle node decrypts it once more with its key and sends it to the exit node. After the final decryption by the exit node, the message isn't encrypted by Tor and can be sent to the server. This means that using more secure protocols such as HTTPS is needed if the user wants the data to be encrypted on the way leaving Tor. These principles are visualised in figure 3.1. In the first prototypes of onion routing, a new circuit was built for every TCP stream. This would be too time-consuming, so circuits in Tor are shared with multiple streams. After some time, they expire and are periodically rebuilt.

There are several positions the adversary can be in and ways Tor protects its users. When the adversary observes the communication from the user to the guard node (This would be the position of the user's ISP), it can't know its destination and contents because of the encryption. However, the fact Tor is being used isn't concealed in the original design and the identities of the routers are publicly known. This creates the possibility of governments restricting access to these known Onion routers, thus restricting access to Tor. Solution for that exists in a way of Tor bridges, which are Tor relays, that aren't listed publicly [33]. Another way Tor tries to prevent the censorship and blocking of Tor access is called Pluggable Transports. This process obfuscates

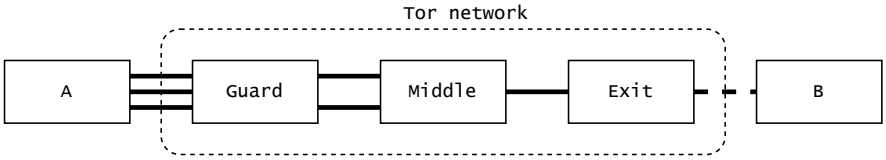

Figure 3.1: The diagram represents a communication through the Tor circuit. User A anonymously communicates with a server B via Tor. Solid lines represent the individual layers of encryption of the transmission between two points in the path, while the dashed line represents the data that isn't encrypted by the Tor itself.

the Tor traffic in a way that should confuse the analysis of said traffic and prevent the detection of Tor [34]. This process is not used on all Tor traffic, but users in countries where Tor is restricted use this solution. The Onion routers themselves have knowledge only about the adjacent ORs in the circuit. The observers of the traffic leaving the exit node can read the message, but cannot easily connect the traffic to a specific user. Tor stays susceptible to end-to-end attacks in situations where the adversary controls both the traffic leaving the user and the exit node, such as those based on correlation [35]. The communication between the ORs is comprised of 512 bytes long cells. The fixed size hides the information of how many times has the message been decrypted, which would show in which part of the circuit the message currently is. There is a validity checking mechanism which destroys the circuit if the cells were tampered with. [23]

3.4 Onion services

Onion services (formerly known as hidden services) offer a way of connecting to a server through Tor without knowing its IP address, thus protecting its identity and location. Onion addresses are used for accessing these services, which usually consist of pseudo-random automatically generated 16- or 56-character strings followed by the .onion pseudo-top-level domain. Connection to onion services can be described in six parts:

1. In order for the onion service to be contacted, it needs to advertise its existence to the Tor network. To do that, the service picks a couple of relays at random, shares them the service's public key and makes them act as introduction points. This communication between a service and introduction points goes through full Tor circuits, so the IP address of the server can stay protected.

2. Onion service generates an onion service descriptor storing its public key and the addresses of the introduction points, signed with its private key. These descriptors are then uploaded to a distributed hash table, spread across ORs designated as "hidden service directories".

3. To establish transmission to the service, a circuit to a randomly selected OR is created. The client asks it to act as a "rendezvous point" by sharing a one-time secret with it. If needed, the client also downloads the service descriptor from the distributed hash table.

4. Client creates a Tor circuit to one of the service's introduction points and instructs the point to forward an introduce message to the service. The message consists of the address of the rendezvous point and the one-time secret and is encrypted with the onion service's public key.

5. After decrypting the client's introduction request, the onion service may establish a Tor connection to the designated rendezvous point and send it the one-time secret to it in a rendezvous message.

6. The rendezvous point verifies the one-time secret and announces to the client that the connection to the onion service has been established. After that, the rendezvous point relays the traffic, connecting the two circuits, enabling the client to communicate with the onion service.

   [23, 36]

3.5 Ways of accessing Tor

Tor has always stated it is meant to be simple to use in order to attract a wide variety of users. Because of that, it offers other solutions than manually configuring a client on user's machine or router, which requires some technical knowledge. Tor tries to be accessible from as many platforms and operating systems as possible. The creators offer various applications based on Tor and there are solutions created by third parties, such as Tor-based operating systems heavily focused on security. [37]

Tor browser [4] is the flagship project aimed at the general public. It is a modified version of the Mozilla Firefox ESR (Extended Support Release) browser that automatically routes the traffic through the Tor network and requires little configuration.

Mobile phones have their own ways of accessing Tor. There is an official version of the Tor browser for Android[5]. A solution for routing other

[4] `https://www.torproject.org/download/`

[5] `https://play.google.com/store/apps/details?id=org.torproject.torbrowser`

Android applications also exists in the way of the Orbot[6] proxy application. Users of iOS can use the Onion Browser[7] application. However, because of the limitations of the system, some privacy features could not be implemented. [38]

Tor-based security-oriented operating systems are complete solutions, which tunnel every connection through Tor. Tails (The Amnesic Incognito Live System)[8] offers booting off a live USB/CD into a preconfigured modified version of Debian. Tails leaves no trace on the local system and the user data gets erased after system shutdown. Whonix[9] is based on running two virtual machines, a workstation and a gateway. The workstation is protected from the network, and its data is stored persistently.

3.6 Works detecting and classifying Tor

Tor remains a popular anonymisation tool helping people to access the Internet freely. However, there are several ways it is being misused for various illegal activities. This makes Tor a widely researched topic, both by the network research communities and by law enforcement agencies. There have been numerous efforts of detecting and blocking Tor, with complete deanonymisation of Tor being the final goal, which can be achieved in some scenarios. Traffic correlation attacks have been found to offer a viable solution for deanonymising Tor in the case where adversary observes the guard and the exit node [39, 40]. However, this work focuses on detecting the Tor traffic and then the classification of Tor into various categories based on the type of application.

3.6.1 Tor detection

Tor stated in its original design paper that the fact user is accessing Tor is not hidden in the original design of Tor [23]. The identity of the Tor relays is publicly known; Tor Project itself offers tools for Tor relay lookup[10] and has a bulk list[11] of all exit nodes. This list can be used by administrators of services that wish not to be accessible from Tor.

Research of detecting Tor using known addresses of Tor relays has been done [41]. They created a working solution that can be incorporated into real-time network monitoring tools. However, there is one caveat of techniques that detect Tor based on the known Tor servers. Tor bridges are not publicly listed, so connecting to Tor using them prevents this type of detection.

[6] `https://play.google.com/store/apps/details?id=org.torproject.android`
[7] `https://apps.apple.com/us/app/onion-browser/id519296448`
[8] `https://tails.boum.org/`
[9] `https://www.whonix.org/`
[10] `https://metrics.torproject.org/rs.html`
[11] `https://check.torproject.org/torbulkexitlist`

Another approach is detecting Tor by understanding its statistical features, which can be done using machine learning. Cuzzocrea et al. [42] researched detecting Tor using machine learning models trained at statistical time-based features extracted from network flow data. They proved this can be an effective approach to detecting Tor as many of the models had the accuracy and F-score better than 0.99, some approaching flawless classification. They used data from a publicly available dataset from the Canadian Institute for Cybersecurity[12]. The research [43] of the creators of the dataset is one of the most influential works in the field of Tor detection and classification and will be further described in the following section.

3.6.2 Tor classification

There are several approaches to classifying Tor into categories based on the application used. They are based on various machine learning techniques, but the main difference is the type of data used for training. One research [44] was based on burst volumes, with bursts being defined as a set of consecutive packets sent in one direction before another is sent from the opposite direction. They chose four categories — P2P (Peer-to-peer), web, file transfer and instant messaging. They were fairly successful in their approach, resulting in accuracy and F-score exceeding 0.8 in some instances. Their experiments represented an attack where the adversary observes the traffic incoming to the entry node.

Another two possible approaches are based on extracting statistical features from either circuits or flows. Shahbar and Zincir-Heywood compared these two techniques in their research [45]. Obtaining the statistical data from circuits requires the adversary to have a compromised OR. This approach differs from the goal of this thesis, which focuses on analysing traffic between the user and the guard node, but can be solved by their second approach — extracting traffic flow features. They classified the Tor traffic into three categories — browsing, video streaming and BitTorrent.

The researchers from Canadian Institute for Cybersecurity [43] experimented with both the detection and classification of Tor while making their dataset publicly available. They decided to classify Tor into eight categories — Browsing, Audio streaming, Chat, E-mail, P2P, File transfer, VoIP (Voice over Internet Protocol), and Video streaming. For generating and capturing their Tor traffic, they used the Whonix security-oriented system, which routes its connection through Tor. Whonix is based on running two virtual machines, a workstation, which is for the user, and a gateway, which handles the routing. This enabled them the simultaneous capturing of both the regular traffic, coming from workstation to gateway, and Tor traffic, which leaves the gateway to the entry node of Tor. Their focus was purely on time-based statistical data extracted from flows, such as the inter-arrival times between the packets.

[12]dataset available from: `https://www.unb.ca/cic/datasets/tor.html`

They experimented with the effect the length of timeout has on the quality of the result, splitting the flows with shorter timeouts. They ran all the experiments on data exported with timeout of 10, 15, 30, 60, 120 seconds and compared the results. Their Tor detection model had the best results when trained on the data with the longest timeouts. In the case of the Tor application type classifier, shorter flows helped the results by having more data samples. They observed the best classification results when the timeout was set at 15 seconds. The results of their best Tor detection model was the recall of the *NonTor* class of 0.994 and the precision of 0.992. In the case of the classifier of application types, they achieved a recall of 0.841 and a precision of 0.836.

Chapter 4

Dataset creation and analysis

4.1 Dataset requirements

There are several approaches to creating the dataset required for training the machine learning experiments. The examples of the Tor traffic can be manually generated and captured in some controlled network. The alternative would be discovering a publicly available dataset to base the experiments on. Either way, the first step is to analyse the goals of the work and understand the requirements for the data. These requirements can help design the data capture procedure or determine whether some publicly available dataset offers a viable solution.

The first question is the position of the observed point in the Tor network. There exist attacks on Tor that require having compromised ORs, capturing both the traffic incoming to and outgoing from the Tor network etc. This work's approach is simpler as it replicates the point of view of a security analyst monitoring some network or the user's Internet service provider. The traffic between the client and the Guard node should be captured.

The first classifier distinguishes between Tor traffic and regular non-Tor traffic. This means that on top of the Tor traffic data, some examples of regular traffic have to be captured as well. The variety of the data is important, so traffic from multiple types of applications should be captured. Additionally, the traffic should originate from the same applications in both classes in order to prevent some systematic error unknowingly being brought into the dataset. Imagine the case where the Tor data would capture only web traffic while the non-Tor class would be comprised of data of peer-to-peer file transfer, resulting in a systematic error in the data. The ideal solution would be capturing the regular traffic and its Tor-encrypted equivalent simultaneously, making the effects of Tor tunnelling the only distinguishing factor between the classes.

The second model classifies Tor by the type of application that generated the traffic. The chosen Tor dataset should consist of several classes of traffic, which can well represent the usual traffic categories of common Internet usage.

The perfect case would be obtaining a dataset that can be used for both classifiers. A logically labelled dataset of simultaneously captured Tor and non-Tor traffic from a mixture of application types, and represents the real-word traffic well, would be well suited for the machine learning experiments.

4.2 Available sources

The possibility of using a training dataset based on publicly available data should be researched first. Finding a suitable dataset would speed up this research, and the final results could then be compared.

4.2.1 Anon 17

The researchers [46] of multiple anonymity networks made their dataset[13] called Anon 17 publicly available. The dataset consists not only of Tor traffic but of other anonymity networks — JonDonym and I2P. The majority of the Tor part of the dataset was carried using Tor pluggable transports, a way of obfuscating Tor's characteristics in order to confuse the Tor detection. This thesis aims to recognise the traditional traffic statistics of Tor, so researching pluggable transports is better suited for some following work. A minor part of the dataset is labelled by the type of application carried over Tor, divided into three classes — Browsing, Video and BitTorrent.

Anon 17 is available in the ARFF format for the Weka machine learning suite, with the features already extracted from the captured network traffic. For the experiments in this thesis, we require unprocessed traffic samples, ideally in the pcap format. All these characteristics of the dataset make its use in this thesis unfeasible.

4.2.2 ISCXTor2016

Research [43] into Tor detection and classification resulted in a publicly available dataset[14]. Their dataset consists of the regular non-Tor and Tor traffic; both were captured simultaneously. The captured traffic was carried over Tor using the Whonix security-oriented operating system. The dataset offers a large variety of types of traffic as they captured eight traffic categories.

The Whonix distribution is provided as a set of two virtual machines — the gateway and the workstation. The workstation is meant to be the system used while its connection to the Internet via the Tor network is handled by the gateway. This means the regular traffic can be captured leaving the workstation before the Tor client encrypts it when leaving the gateway, which can be also captured. This way, a pair of regular traffic and Tor traffic can

[13] dataset can be accessed via: `https://web.cs.dal.ca/~shahbar/data.html`

[14] dataset can be accessed via: `https://www.unb.ca/cic/datasets/tor.html`

be captured at the same time. They captured a total of 22 gigabytes of data in the pcap format. Additionally, they offer the data exported to the network flow format using their flow exporter called ISCXFlowMeter.

ISCXTor2016 seems to offer viable data to base the training dataset on. It is also available in the unprocessed form before it was exported with a flow exporter. It has both the samples of regular non-Tor traffic and traffic tunnelled over Tor. The Tor traffic was labelled into eight different categories that are described in the following section. They captured a wide range of use cases, and the data should be representative of real-world traffic.

Traffic categories

ISCXTor2016 [47] captured traffic from the following categories, as they represent the real-word Internet usage well:

Browsing label denotes HTTP and HTTPS traffic generated while browsing using Chrome and Firefox browsers.

Email traffic was generated using a Thunderbird client. Mail was delivered using the SMTP/S, and received using POP3/SSL in one client and IMAP/SSL in the other.

Chat label represents traffic from instant-messaging applications, generated using Facebook and Hangouts via web browser, Skype, and IAM and ICQ using an application called pidgin.

Audio-Streaming identifies audio applications with a continuous stream of data, represented with traffic generated using Spotify.

Video-Streaming identifies video applications with a continuous stream of data, represented with traffic captured from YouTube (HTML5 and flash versions) and Vimeo services using Chrome and Firefox.

FTP class represents applications whose main purpose is to send or receive files. The captured traffic consists of Skype file transfers, FTP over SSH and FTP over SSL traffic sessions.

VoIP consists of voice calls captured from Facebook, Hangouts and Skype.

P2P class represents file-sharing protocols, such as BitTorrent. Creators of the dataset captured traffic sessions using the Vuze application, downloading various .torrent files of the Kali Linux distribution, using various combinations of upload and download speeds.

4.3 Dataset analysis

ISCXTor2016 seems to be a viable basis for the training data. The quality of the datasets will be analysed in this section. This analysis helps to decide if the goal can be achieved using this data. The analysis can also discover some facts that have to be considered when creating the classifiers and can help us understand the Tor traffic better.

4.3.1 Flow export

The dataset offers both the unprocessed pcap data and the exported flows in the CSV format using their flow exporter, formerly called ISCXFlowMeter, known as CICFlowMeter today. For the final classifier to be easily incorporated in the already existing flow capturing infrastructure used by CESNET, only the pcap data was used from the dataset, and the same flow exportation process was used. The trained prototype of a classifier will then accept the same network flow data CESNET's system for traffic analysis already uses.

The pcap data from ISCXTor2016 was exported using the ipfixprobe flow exporter [6], resulting in network flow data based on the IPFIX standard. Apart from the primary exported fields (storing information such as the source and destination IP addresses and ports, number of bytes and timestamps), the PSTATS plugin was used. PSTATS gathers statistics about the first n (30 by default) packets of the flow, the statistic being the packet lengths, timestamps, packet directions and TCP flags. As ipfixprobe operates with an internal binary format called UniRec, the binary data was then logged and converted to a human-readable CSV format, which can be then analysed.

4.3.2 Tor detection dataset analysis

The first classifier is intended to distinguish between the regular traffic and the traffic that was tunnelled over the Tor network. The whole dataset was used and divided into two classes - *Tor* and *NonTor*.

The whole dataset consists of 633 flow records of the *Tor* class and 96,568 records of the *NonTor* class. Duplicate flow records were then thrown away, totalling 112 *Tor* records and 33,827 cases of the *NonTor* class. After removing duplicate flows, there are 521 records of the *Tor* class and 62,741 records of the *NonTor* class. This would make this a highly imbalanced dataset, with an imbalance ratio (proportion of samples of the majority class to the number of samples of minority class) of $1 : 120$. This is an expected behaviour resulting from how network flows are defined and how tunnelling works. It means that imbalance should be considered in the training and evaluation of the classifier.

The empty flows were then studied. There aren't any records without outgoing traffic; having flows without outgoing traffic could indicate some errors in the flow exportation or missing data. There aren't any *Tor* flows

without incoming traffic. The *NonTor* class differs in this case; there are 7,149 *NonTor* flows without incoming traffic (circa 11 % of the *NonTor* data).

The distribution of flow lengths was then analysed and displayed as histograms based on the number of outgoing packets. The flow length distribution of the *NonTor* class, shown in figure 4.1, can be compared with the flow length distribution of the *Tor* class, shown in figure 4.2.

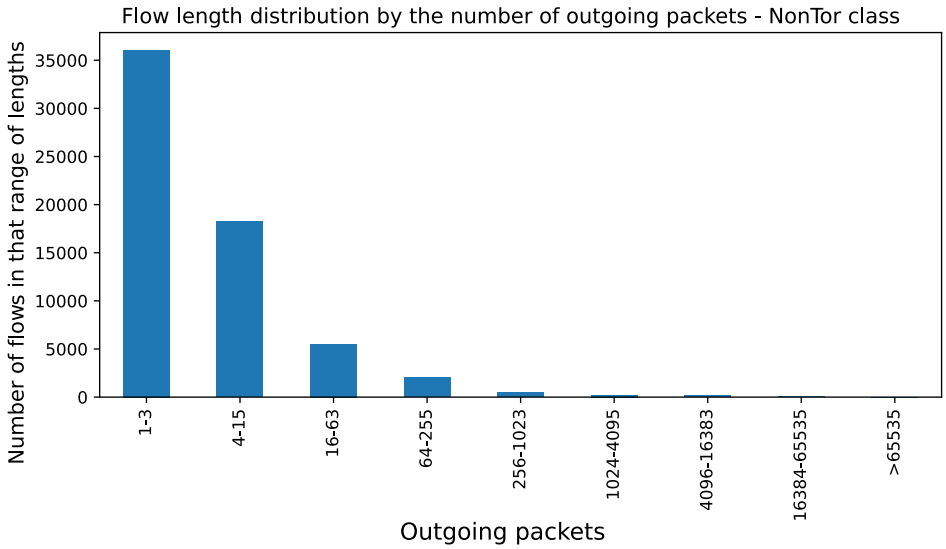

Figure 4.1: Flow length distribution of the *NonTor* class

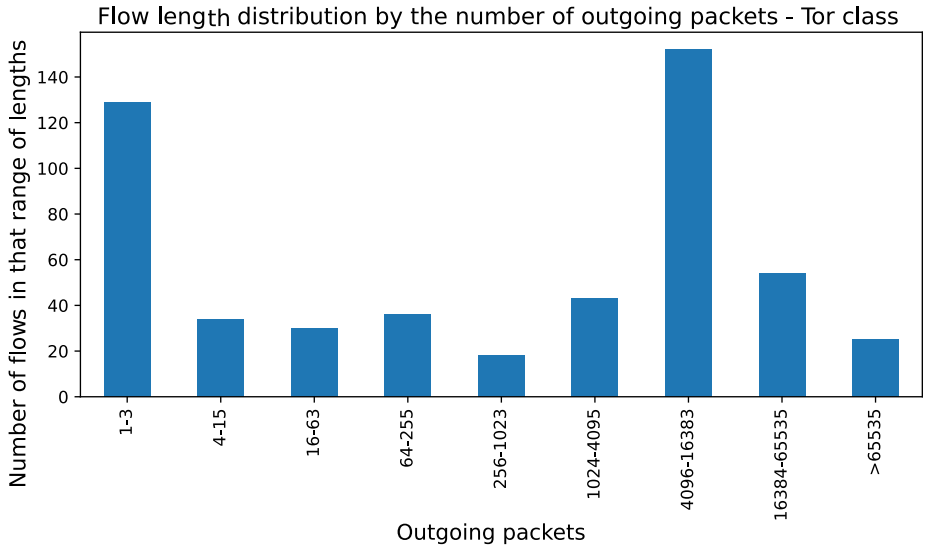

Figure 4.2: Flow length distribution of the *Tor* class

The most commonly used destination ports were then analysed and the differences between the classes compared. As seen in the table 4.1, the majority of the *NonTor* data represents communication with destination port assigned by IANA (Internet Assigned Numbers Authority) to the DNS, HTTP and HTTPS protocols.

Table 4.1: Most common ports of the *NonTor* class

Port	**Count of samples**	**Assigned to by IANA**
53	11,997	DNS
443	11,846	HTTPS
80	11,078	HTTP
51413	2,734	Port from dynamic range
9100	2,179	PDL Data Streaming

The destination ports of the *Tor* class can be seen in the table 4.2. Circa 40 % of the samples communicated with the port 443, assigned to HTTPS. Majority of the remaining samples used various ports from the dynamic range or ports unassigned by IANA. Port 80 assigned to HTTP traffic was not present in the *Tor* data at all.

Table 4.2: Most common ports of the *Tor* class

Port	**Count of samples**
443	218
9010	25
49580	24
9001	22
110	20

How the protocols used differ between the classes was studied in the next part of the analysis. As seen in the table 4.3, *Tor* traffic uses exclusively the TCP protocol. On top of TCP, a circa of a third of *NonTor* traffic was carried using the UDP protocol.

Table 4.3: Comparison of the protocols between the *Tor* and *NonTor* class

(a) *NonTor* class

Protocol	**Count of samples**
TCP	43640
UDP	19056
ICMP	42
IGMP	3

(b) *Tor* class

Protocol	**Count of samples**
TCP	521

In the end, possible issues with the dataset were searched for. There were not any flows with missing or illogical values, which would indicate errors in the flow exportation process. There was not any indication of mis-labelled data or of systematic errors being brought in because of different data structure between the classes.

4.3.3 Tor classification dataset analysis

The goal of the second classification model is to classify the traffic carried over Tor by the type of application that generated the traffic. Only the part of the dataset capturing Tor traffic will be considered in this section; data is labelled with eight different traffic classes. The detailed description of the classes was described in the section 4.2.2. In this stage, the duplicate samples were already deleted from the dataset.

The imbalance of the dataset was studied first. The numbers of samples corresponding to the classes can be seen in the table 4.4. The table shows potential issues the classifier will face as there are not many samples for some of the classes.

Table 4.4: Counts of records corresponding to Tor traffic classes

Class	Count of samples	Percentage of samples
Audio-Streaming	68	13 %
Browsing	137	26 %
Chat	28	5 %
FTP	47	9 %
Mail	19	4 %
P2P	56	11 %
Video-Streaming	54	10 %
VoIP	112	21 %

Analysis of destination ports did not discover any useful information, which is the expected result of Tor tunnelling. All Tor data used TCP protocol, so the study of protocols was not needed. There were no flows with missing or illogical values, and there were not any samples without any outgoing traffic and without any incoming traffic.

Distributions of lengths of the various classes were compared in the next step. The results can be seen in the figure 4.3 and look like as expected based on the type of traffic captured.

4.3.4 Analysis results

The dataset was analysed to determine if it is well suited for the classifiers to be based on. New knowledge about the data was also gained, and some differences between the classes were discovered.

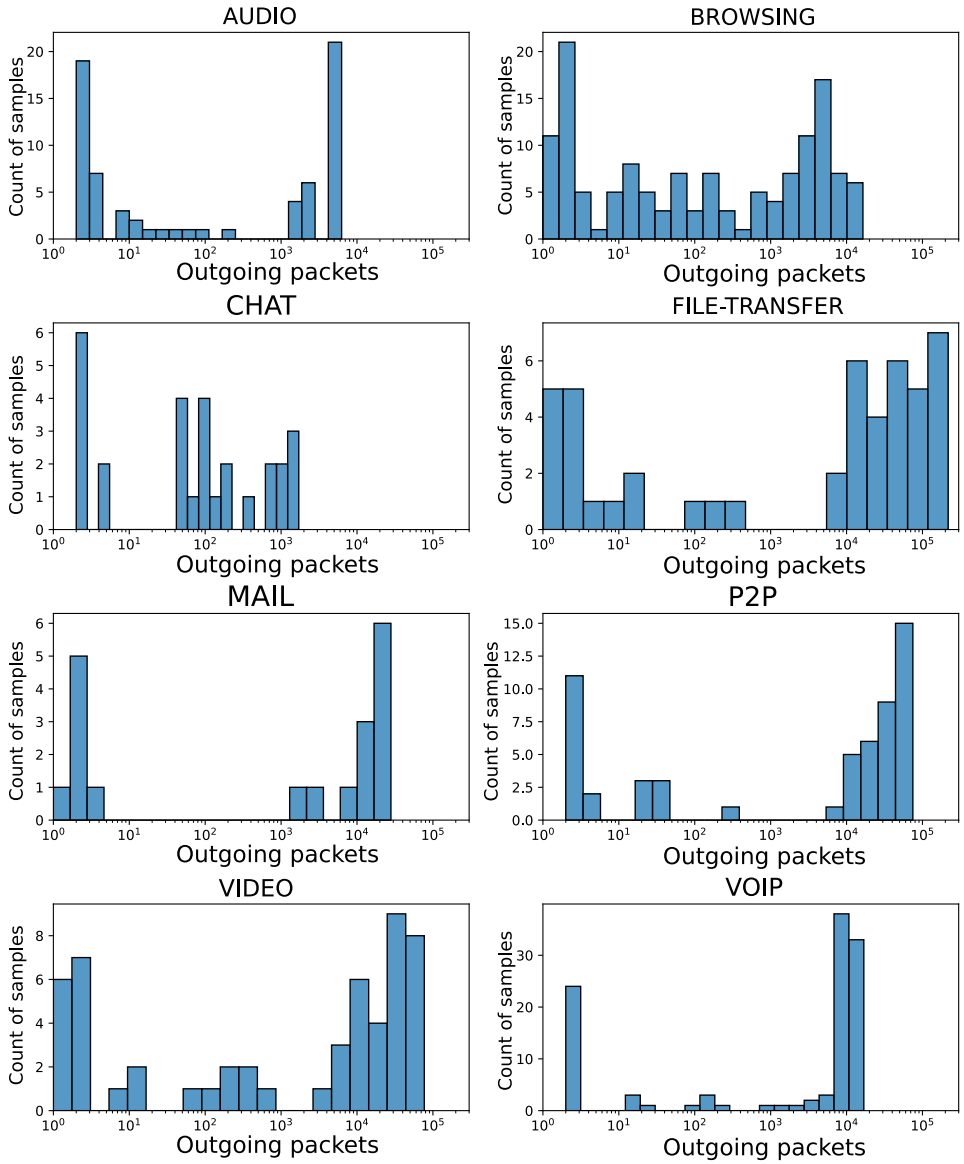

Figure 4.3: Flow length distribution comparison (Tor classification data)

There have not been found any issues preventing from successfully using the dataset in the following process. However, some minor caveats were discovered, and they have to be addressed in the training and evaluation of the classifiers. The data for the Tor detection classifier is highly imbalanced, with an imbalance ratio of 1 : 120. Because of that, it was experimented with class weighing in the training process, and stratified k-fold cross-validation was used. The hyperparameter selection and evaluation of the final classifier was done more thoroughly and based on more metrics than accuracy. The second

classifier will probably face issues because of an insufficient amount of data samples, which can be especially seen in some classes. To address that issue, the creators of the dataset experimented in their research [43] with exporting the flows with various timeouts, thus shortening the flows and gaining more data samples. This method can drastically improve the quality of the final classifier. However, this approach was not used in this research, as the resulting classifier could not be used for real deployment. In practice, the parameters have to be stable as they are shared across multiple analysis tools.

The studied flow data proved helpful for analysing the captured traffic but is not well suited for the classifiers to be trained on. Better suited time-based features were then extracted from said flow data. On top of that, some additional features that seem to divide the classes for the Tor detection classifiers were added to the feature vector. Those features sere mainly designed around the observable differences in the used destination ports and transport protocols between Tor and non-Tor traffic.

4.3.5 Flow-based dataset analysis tool

As the task of analysing unknown datasets or comparing data from multiple sources is often repeated in this type of research, I created a universal tool for that. The tool is available as two Jupyter notebooks. The first one requires two data sources, which it then compares; the second one uses a single data source, which has to be labelled. The required data source is a set of network flows exported with ipfixprobe with the PSTATS plugin enabled and logged to the CSV format.

The tool compares the classes based on the two source files or the class label column and then outputs various statistics. Some of the statistics and graphs it created can be seen in the previous sections. On top of that, it also exports a standard time-based feature vector and analyses created features in various graphs.

Chapter 5

Experiments with ML models

5.1 Feature extraction

The procedure of creating training data was done as follows. The pcap data from the ISCXTor2016 dataset represents the captured traffic used as a basis for the training data. Ipfixprobe flow exporter (with the PSTATS plugin enabled) was then used to export the captured packet data into network flows. The last step in the preparation of data for the machine learning models to be trained on was the extraction of features describing various traffic statistics from the network flow data.

Some features can be designed based on knowledge gained from analysing the dataset, but the majority of features were designed based on the commonly used statistics of network traffic data. Feature Exploration Toolkit[15] (FET) by Daniel Uhricek offers various ways of analysing and processing network flow data and also offers the extraction of features. It can be then used to further analyse and graph out those features, which can lead to the creation of a better performing machine learning model. Feature Exploration Toolkit was used for the extraction of commonly used statistical features (described in the following section) and additional analysis of those features. As most of the features are time-based, some samples with the duration of communication shorter than the sampling rate of timestamps had to be removed.

5.1.1 Feature vector

Feature Exploration Toolkit extracts these features from the flow data:

- duration of flow in seconds
- transmitted bytes and packets per second in the forward direction, backward direction and both directions

[15] can be accessed via: `https://gitlab.liberouter.org/uhricdan/fet`

- count of following TCP flags and ratio of packets with those flags:
  - FIN, SYN, RST, PSH, ACK, URG
- minimum, maximum, mean and standard deviation of these metrics:
  - length of packets in the forward direction, backward direction and both directions
  - packet inter-arrival time in the forward direction, backward direction and both directions
- mean and standard deviation of the normalised inter-arrival times in the forward direction, backward direction and both directions (Normalised inter-arrival times are calculated as either zero for inter-arrival times shorter than a specified margin and one in the other case)

5.1.2 Feature selection

As described in [48], feature selection is a process that can improve the performance and training speed of machine learning models by removing some of the features. Because of that, it was experimented with feature selection in the creation of the classifiers. The field of study surrounding feature selection is quite complex and there exists plenty of approaches; These two methods were chosen:

- Feature ranking was used to determine how much the features attribute to the quality of the classifiers. The worst performing features were not used in the final feature vector.
- Highly correlated features were removed as they provide little additional information and can potentially cause some issues in the final quality.

5.2 Models used

It was experimented with a large variety of common machine learning models to determine which ones are best suited for this specific task. Which models offered the highest quality results can also give us some useful information about the distribution of the data, how well is it linearly separable etc. For every model, it was searched for the best performing hyperparameters (decided using the F-score metric) and some models gained better performance after some data preprocessing specific to those models. For assessing the qualitative differences between the models, stratified 5-fold cross-validation was used. The performance of those models was then evaluated and the best performing one chosen for the software prototype. It was experimented with the following surpervised learning models; used data preprocessing approaches described for the specific models.

Decision tree, Random forest, AdaBoost

K-nearest neighbours — it was experimented with using PCA dimensionality reduction and data normalization — rescaling every feature to the interval $< 0, 1 >$.

Naive Bayes

Logistic regression — the data was preprocessed using standardisation — changing the distribution to have a mean of zero and a standard deviation of one.

Support vector machines — preprocessed using standardisation.

Scikit-learn is a popular open-source machine learning library for the Python language, and it offers a vast range of machine learning models and tools for data preprocessing, model evaluation etc. Because of that, it was chosen for the machine learning experiments in this work. The experiments were done inside the Jupyter Notebook environment, which offers the creation of interactive documents containing code (Python code in this work), text and visualisations.

5.3 Tor detection classifier

5.3.1 Feature vector

Feature Exploration Toolkit was used for extracting the majority of used featured from the flow data. These extracted features can be seen in the previous section. On top of that, it was experimented with several other features based on the discovered facts from the dataset analysis in section 4.3. The additional features are as follows:

- `Protocol` feature is set to one for the traffic using the TCP protocol and zero for the other protocols
- `Port DNS` feature is true when the destination port was 53, which is assigned to DNS
- `Port HTTP` feature is true when the destination port was 80, which is assigned to HTTP
- `Port HTTPS` feature is true when the destination port was 443, which is assigned to HTTPS
- `Port Dynamic` feature is true when the destination port was from the dynamic/private range

5.3.2 Results

The performance of the various models was analysed based on the mean of metrics from the 5-fold cross-validation. Note that the training dataset is heavily imbalanced, with the majority class — *NonTor* — being roughly a hundred times more represented than the minority one in the dataset. Because of that, the model selection focuses on the metrics of the minority class — *Tor*. The comparison of the validation F-scores of the *Tor* class of the various models can be seen in the figure 5.1.

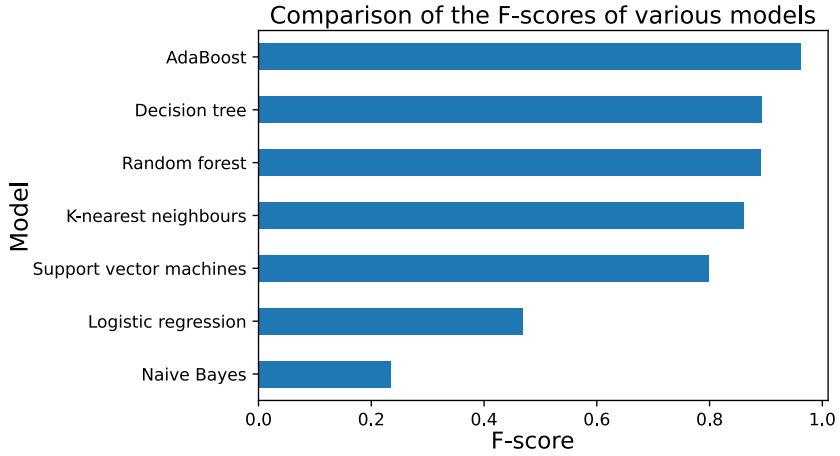

Figure 5.1: Tor detection classifier model ranking by F-score

The graph shows that ensemble models based on decision trees and decision trees themselves are among the best performing models based on this metric. Logistic regression and Naive Bayes, especially, do not seem to be suitable models for this specific task based on this metric. Additional assessments can be done using the table 5.1, showing the validation accuracy and the precision, recall and F-Score of the *Tor* class.

Table 5.1: Comparison of the Tor detection models; *The following metrics are shown: validation accuracy, and the precision, recall and F-Score of the Tor class.*

Model	Accuracy	Precision	Recall	F-Score
Decision tree	0.9980	0.9122	0.8747	0.8931
Random forest	0.9978	0.8469	0.9425	0.8921
AdaBoost	`0.9993`	`0.9870`	0.9384	`0.9621`
K-nearest neighbours	0.9975	0.9011	0.8234	0.8605
Naive Bayes	0.9551	0.1395	0.7269	0.2341
Logistic regression	0.9793	0.3089	`0.9670`	0.4682
Support vector machines	0.9958	0.7268	0.8850	0.7981

The table shows that accuracy does not capture the final quality of the models well in this case of a heavily imbalanced dataset. Because of that, additional metrics were used for assessing the performance of the various models. Naive Bayes is shown not to be a fitting model for this type of classification problem, showing the worst performance in every metric. Decision tree and random forest demonstrated decent results, having fairly similar performance, with the difference in focus on precision or recall. However, AdaBoost manifested the best results in the majority of metrics, being beaten only in the recall metric.

The logistic regression model was measured to have the best recall of the *Tor* class. However, it was also measured to have the second-worst precision and thus was not considered to be a quality model for this specific task. This is an example of how the precision and recall metrics are connected. Usually, improving precision leads to worsening recall and vice versa, and some compromise sometimes has to be made, for example, based on the knowledge of the classified problem. The same decision, usually based on some additional knowledge, has to be made when comparing models showing roughly the same F-score and precision but noticeably differing in their focus on recall or precision. However, the decision was fairly clear in this case. Even though logistic regression model has managed to discover a few per cent more *Tor* samples (indicated by the better recall) than the AdaBoost model, its classification of *Tor* is not trustworthy as most of the samples classified as *Tor* were false positives. AdaBoost was considered to be the best performing model for the Tor detection task and thus was chosen for the software prototype.

5.4 Tor traffic category classifier

5.4.1 Feature vector

Only the feature vector generated by the Feature Exploration Toolkit was used in the creation of this classifier. As with the Tor detection classifier, a feature selection was done, eliminating the correlated and low-ranking features.

5.4.2 Results

As with the previous classifier, a 5-fold cross-validation was used for assessing the performance of the classifiers. The comparison of the weighted average of the F-scores can be seen in the figure 5.2.

The graph shows that the order of the best performing models is fairly similar as in the case of Tor detection classifier, with the exception of the decision tree model. As in the case of the first classifier, additional evaluation was done to determine which model to use in the software prototype. The weighted validation precision, recall and F-score of the models can be seen

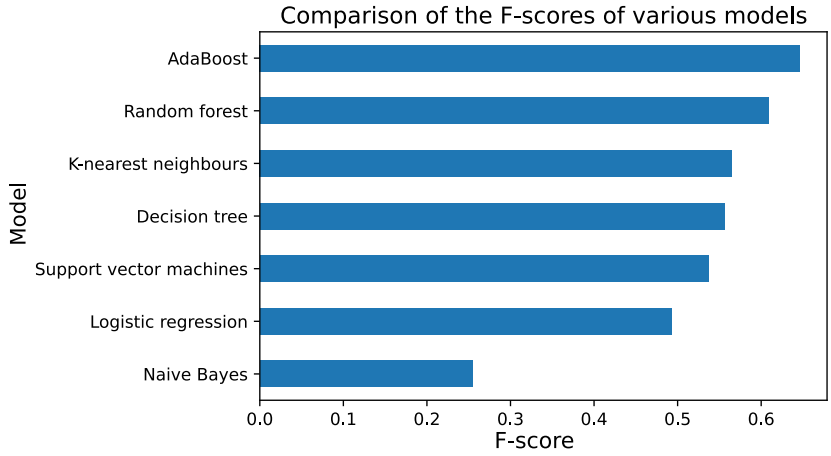

Figure 5.2: Tor traffic category classifier model ranking by F-score

in the table 5.2. The accuracy is not specifically shown in that table as the weighted recall provides virtually the same metric.

Table 5.2: Comparison of the Tor classification models; *The following metrics are shown: weighted validation precision, recall and F-score.*

Model	Precision	Recall	F-Score
Decision tree	0.5892	0.5646	0.5561
Random forest	0.6428	0.6112	0.6086
AdaBoost	0.6844	0.6485	0.6465
K-nearest neighbours	0.5923	0.5784	0.5650
Naive Bayes	0.5116	0.2944	0.2548
Logistic regression	0.5120	0.4947	0.4924
Support vector machines	0.5738	0.5255	0.5367

Ensemble models with the decision tree as their base classifier proved to be the best performing. As with the previous classifier, it was shown that Naive Bayes isn't a suitable model for this type of data. AdaBoost is once again the best performing model, showing the best results in all the measured metrics. The results clearly indicate that AdaBoost should be used for the software prototype.

Chapter 6

Outcomes of the thesis

6.1 Software prototype

In the task of creating two classification models — Tor detection classifier and Tor traffic category classifier — it was experimented with a large variety of supervised learning models. After assessing the performance of those models, the best performing model for each classifier was then chosen. The AdaBoost model proved to be the most suitable in both cases. The software prototype was created based on those highest performing models.

Software prototype is available in the form of a python command-line program. The chosen best performing models were saved in an already trained state, prepared to classify unknown data. As is described in [49], there are several ways of persistently storing trained machine learning models, such as using Python's built-in persistence model from the module called pickle or using the joblib library. Joblib was chosen as it is described to be more efficient in saving machine learning models from scikit-learn. At this time, the result is a simple prototype designed for showcasing the created classifier. A more complex application or incorporation into an existing network flow analysis solution can be made in future work.

The required data input for the prototype is a set of network flows exported by the ipfixprobe flow exporter. The flow exporter has enabled PSTATS plugin and the output is converted into the CSV format. The application can then use the Tor detection model, Tor category classification model or a mix of both. In a hybrid mode, it filters the Tor traffic using the detection model, and then uses the category classification model on the discovered Tor traffic. The prototype is operated using the following arguments:

--input (-i) input_file.csv — Mandatory argument for specifying the path to the network flows in the CSV format

--output (-o) output_file.csv — When the output argument is present, predicted labels will be added to the flow data and stored at the specified

location. The predicted labels are printed to the standard output when no output location is given.

--mode (-m) detect/classify/hybrid — Mandatory argument for selecting which classifier will be used

6.2 Evaluation

Machine learning experiments with various supervised learning models were designed and carried out, as described in the previous chapter. The performance of those various models was then assessed, and the most suitable model chosen; AdaBoost performed the best in both cases. This chapter further evaluates the performance of the final models (with the metrics measured using a 5-fold cross-validation) and compares it with the results from research of the creators of the ISCXTor2016 dataset.

6.2.1 Tor detection classifier

First impressions from the comparison of models for Tor detection in the preceding chapter are that machine learning offers an effective approach for detecting Tor based on its traffic characteristics. To support this claim, additional evaluation was done and described in this section. Table 6.1 shows the precision, recall and F-score of all the classes and their weighted average. Confusion matrix, shown in the figure 6.1, can be used for understanding what types of miss-classification were made by the model.

Table 6.1: Metrics of the AdaBoost model for Tor detection

Class	Precision	Recall	F-Score
NonTor	0.9994	0.9999	0.9996
Tor	0.9870	0.9384	0.9621
Weighted average	0.9993	0.9993	0.9993

It is shown that the detection of Tor is very precise and reliable as only circa 0.01 % of *NonTor* flows were misclassified to be of *Tor* class. Most of the model's misclassifications come from some *Tor* samples not being detected. If the classifier performs the same on the real-world data, some minority of Tor traffic would not be detected by the final tool, but the decision of the tool could be relied on.

The final Tor detection classifier was compared with the classifier from the creators of ISCXTor2016, described in their work in [43]. For that comparison, see the table 6.2. My classifier shows a slightly better recall and noticeably better precision. To conclude, the final classifier seems to represent a suitable approach to detecting Tor traffic as it performed well and even better than the classifier from the creators of the original dataset.

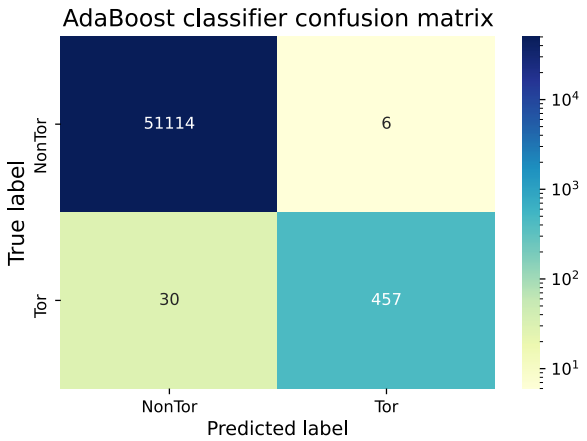

Figure 6.1: Tor detection classifier confusion matrix

Table 6.2: Comparison of my Tor detection classifier with existing solutions; *precision, recall and F-score of the Tor class are shown; my classifier is compared with the classifier from [43]*

Classifier	Precision	Recall	F-Score
My	0.9870	0.9384	0.9621
ISCXTor2016	0.9487	0.9343	0.9410

6.2.2 Tor traffic category classifier

A further assessment of the AdaBoost model for Tor traffic category classification is described in this section. One approach for that is studying the precision and recall of all the classes. It shows which classes are tricky to detect or are being falsely detected. Table 6.3 show said comparison of metrics. Studying the confusion matrix (see figure 6.2) also helps better understanding which classes were being mistaken for each other, among other possible findings.

The following findings were derived from the confusion matrix and the metrics of all the classes. The classifier was most successful in detecting the *VoIP* class, shown by the highest precision and recall. *Browsing* class presented the second-best recall. On the other hand, it also had the worst precision. It can be seen that all the classes had some noticeable percentage of samples mistakenly classified as the *Browsing* class. In the case of *Chat* and *Mail* classes, less than half of related samples was correctly classified. The majority of *Chat* data was misclassified as *Browsing*. Most of the remaining classes showed no noteworthy trend, with the metrics being close to the average.

The table 6.4 shows the comparison between my Tor traffic classifier and the classifier from the creators of ISCXTor2016. In their research [43], they used the same classes, so the results are directly comparable. It can be seen

Table 6.3: Metrics of the AdaBoost model for Tor traffic category classification

Class	Precision	Recall
Audio	0.6176	0.6364
Browsing	0.5535	0.7323
Chat	0.6428	0.3462
File transfer	0.6470	0.5238
Mail	0.7000	0.4118
P2P	0.6923	0.6923
Video	0.6136	0.5510
VoIP	0.9077	0.7763
Weighted average	0.6844	0.6485

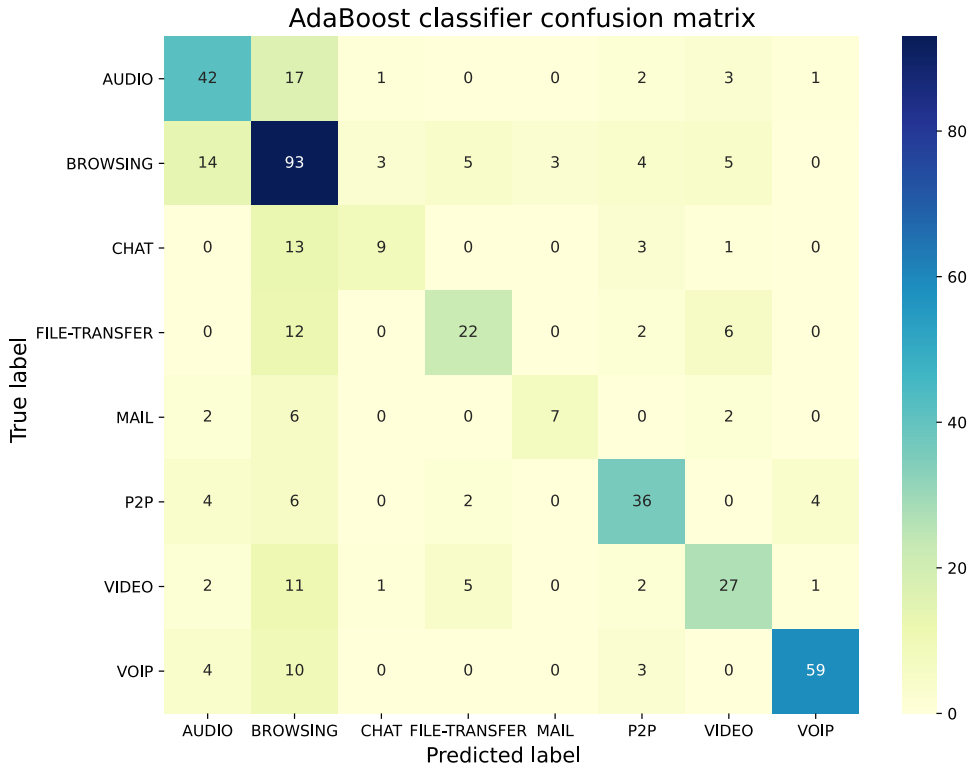

Figure 6.2: Tor traffic category classifier confusion matrix

that their performance is substantially higher in all the metrics. This finding confirms the suspicion that the Tor dataset based on ISCXTor2016 was not large enough for creating a classifier that manages to generalise the problem well. The approach of the creators of ISCXTor2016 of shortening the flows was not feasible in this work as the created classifier could not be used in practice, where the exportation parameters are shared across multiple analysis modules.

Table 6.4: Comparison of my Tor traffic category classifier with existing solutions; *weighted precision, recall and F-score are shown; my classifier is compared with the classifier from [43]*

Classifier	Precision	Recall	F-Score
My	0.6844	0.6485	0.6465
ISCXTor2016	0.8430	0.8380	0.8404

To sum up, this work showed that even-though Tor uses encryption, its traffic can be classified by analysing various statistical properties. The work demonstrated how Tor users should be aware that Tor's encryption and tunneling process does not completely prevent all ways of analysing their activity. However, the final performance of the classifier is hampered by the dataset not being large enough and it did not reach its full potential. In future work, capturing a more extensive dataset would yield a more reliable Tor traffic category classification tool.

Conclusion

This bachelor's thesis dealt with ways of using supervised learning methods for detecting the traffic of the Tor network and classifying it into multiple traffic categories. The first part of the research described the different approaches for network traffic analysis — either based on analysing packets or network flows. A modern approach for network traffic analysis is using various machine learning techniques. An overview of the field of machine learning was also done in this thesis. The functionality and traffic of Tor were studied in the next part, and existing works into Tor detection and classification were investigated.

For the creation of the machine learning models, a dataset of labelled Tor traffic had to be created. Options of basing the dataset on some publicly available data were researched. One of the researched works into Tor classification made their dataset called ISCXTor2016 [43] publicly available. The raw packet capture data from ISCXTor2016 was exported into the format of network flows using the ipfixprobe flow exporter. An analysis of this existing data was done to decide if this data sufficed the needs of this work and gain more knowledge about the workings of Tor. It was found out that the dataset is more or less sufficient for the training data to be based on, with the only issue being the lack of samples for the Tor traffic category classification.

A feature vector based on extracting various statistics from the network flows was then designed. Part of the features was using a standard set of various time-based features and features based on the TCP flags. Additional features were based on the knowledge gained from the dataset analysis. The work dealt with two different classification problems – distinguishing between Tor traffic and regular traffic outside the Tor network and classifying the Tor traffic into several categories by the type of traffic. For both of the problems, it was then experimented with a variety of supervised learning models. The performance of those models was then assessed to determine which model to use as the final classifier. AdaBoost ensemble model with decision trees as its base learners proved to be the best performing in both cases, so they were then stored and used in a software prototype.

The final classifiers were then further analysed, and the findings discussed. In the case of Tor traffic detection, machine learning offers a viable approach. The classifier managed to detect 94 % of Tor samples and was 99 % precise in those decisions, with the F-score being 96 %. The misclassifications of regular traffic samples as Tor traffic happened only for roughly one in every 10,000 non-Tor samples. My classifier showed a better performance than the classifier from the creators of ISCXTor2016. Throughout the existence of the Tor network, there have been censorship attempts of restricting access to it by some governments. This makes a compelling incentive for additional research in some future work of how these techniques prevent my method of Tor detection as Tor can be made to obfuscate its traffic characteristics using so-called pluggable transports.

The second classifier was designed to distinguish between several Tor traffic categories, such as web browsing, file transfer and video streaming. As Tor encrypts its traffic and passes it around multiple routers, it could seem that knowledge of what the user does is protected. This work demonstrates that machine learning algorithms can analyse various statistical properties of the traffic and use them to detect the type of Tor traffic somewhat effectively. The performance of the final model which classifies Tor traffic into eight trafic categories was measured to have the weighed precision of 68 %, recall of 65 % and F-score of 65 %. Its performance was hindered by dataset not being large enough for the classifier to properly generalise the problem. Creating a larger dataset of Tor traffic in the future can lead to a significantly better performance of the classifier.

For now, the created classifiers can be used in a software prototype. The next logical step is the real deployment of the classifier and its incorporation into other analysis tools. It was shown how well the classifiers behaves in some presumably controlled environment. For the creation of more complex software tool in the future, additional capturing of more sources of data representing real life traffic should be done. This will result in a more reliable tool, which would better generalise the Tor detection and classification problems and would be tested on multiple independent data sources.

Bibliography

1. DERI, L.; MARTINELLI, M.; BUJLOW, T.; CARDIGLIANO, A. nDPI: Open-source high-speed deep packet inspection. In: *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2014, pp. 617–622. Available from DOI: 10.1109/IWCMC.2014.6906427.
2. FUCHS, Christian. SOCIETAL AND IDEOLOGICAL IMPACTS OF DEEP PACKET INSPECTION INTERNET SURVEILLANCE. *Information, Communication & Society*. 2013, vol. 16, no. 8, pp. 1328–1359. Available from DOI: 10.1080/1369118X.2013.770544.
3. SANDERS, Chris. Packet Analysis and Network Basics. In: *Practical packet analysis: Using Wireshark to solve real-world network problems*. No Starch Press, 2017, chap. 1. ISBN 978-1593278021.
4. ŽÁDNÍK, Martin. Network Monitoring Based on IP Data Flows. *Best Practice Document GN3-NA3-T4-CBPD131*. 2010.
5. FLOWMON. *NetFlow / IPFIX Monitoring* [online] [visited on 2021-05-01]. Available from: https://www.flowmon.com/en/solutions/network-and-cloud-operations/netflow-ipfix.
6. GITHUB – CESNET. *ipfixprobe - IPFIX flow exporter* [online] [visited on 2021-05-01]. Available from: https://github.com/CESNET/ipfixprobe.
7. NGUYEN, Thuy T.T.; ARMITAGE, Grenville. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*. 2008, vol. 10, no. 4, pp. 56–76. Available from DOI: 10.1109/SURV.2008.080406.
8. MÜLLER, Andreas; GUIDO, Sarah. *Introduction to machine learning with Python: a guide for data scientists*. O'Reilly Media, Inc., 2016. ISBN 978-1449369415.

9. FERNANDES DE MELLO, Rodrigo; ANTONELLI PONTI, Moacir. A Brief Review on Machine Learning. In: *Machine Learning: A Practical Approach on the Statistical Learning Theory*. Springer International Publishing, 2018, pp. 1–74. ISBN 978-3-319-94989-5. Available from DOI: 10.1007/978-3-319-94989-5_1.

10. FÜRNKRANZ, Johannes. Decision Tree. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Springer US, 2010, pp. 263–267. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_204.

11. BROWN, Gavin. Ensemble Learning. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Springer US, 2010, pp. 312–320. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_252.

12. SCIKIT LEARN. *Ensemble methods* [online] [visited on 2021-04-11]. Available from: https://scikit-learn.org/stable/modules/ensemble.html.

13. CAPEK, M.; DEDECIUS, K.; KLOUDA, K. kNN and Cross-validation technique. In: *Data Mining* [online]. FIT, CTU, 2020 [visited on 2021-04-11]. Available from: https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/05/BI-VZD-05-en-slides.pdf.

14. WEBB, Geoffrey I. Naïve Bayes. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Springer US, 2010, pp. 713–714. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_576.

15. CAPEK, M.; KLOUDA, K. Logistic regression. In: *Data Mining* [online]. FIT, CTU, 2020 [visited on 2021-04-11]. Available from: https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/09/BI-VZD-09-en-slides.pdf.

16. ZHANG, Xinhua. Support Vector Machines. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Springer US, 2010, pp. 941–946. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_804.

17. SCIKIT LEARN. *Metrics and scoring: quantifying the quality of predictions* [online] [visited on 2021-04-11]. Available from: https://scikit-learn.org/stable/modules/model_evaluation.html.

18. THE TOR PROJECT, INC. *Tor: Overview* [online] [visited on 2021-03-30]. Available from: https://2019.www.torproject.org/about/overview.html.en.

19. CHAABANE, A.; MANILS, P.; KAAFAR, M. A. Digging into Anonymous Traffic: A Deep Analysis of the Tor Anonymizing Network. In: *2010 Fourth International Conference on Network and System Security*. 2010, pp. 167–174. Available from DOI: 10.1109/NSS.2010.47.

20. MEDIA.CCC.DE. *Dr Gareth Owen: Tor: Hidden Services and Deanonymisation* [online]. 2015 [visited on 2021-03-30]. Available from: https://www.youtube.com/watch?v=-oTEoLB-ses.

21. MOORE, Daniel; RID, Thomas. Cryptopolitik and the Darknet. *Survival*. 2016, vol. 58, no. 1, pp. 7–38. Available from DOI: 10.1080/00396338.2016.1142085.

22. REED, M.G.; SYVERSON, P.F.; GOLDSCHLAG, D.M. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*. 1998, vol. 16, no. 4, pp. 482–494. Available from DOI: 10.1109/49.668972.

23. DINGLEDINE, Roger; MATHEWSON, Nick; SYVERSON, Paul. *Tor: The second-generation onion router*. 2004. Technical report. Naval Research Lab Washington DC.

24. THE ONION ROUTER. *A Brief History of Onion Routing* [online] [visited on 2021-03-30]. Available from: https://www.onion-router.net/History.html.

25. THE TOR PROJECT, INC. *Tor: History* [online] [visited on 2021-03-30]. Available from: https://www.torproject.org/about/history/.

26. LEVINE, Yasha. *Almost Everyone Involved in Developing Tor was (or is) Funded by the US Government* [online]. 2014 [visited on 2021-03-30]. Available from: https://pando.com/2014/07/16/tor-spooks/.

27. THE TOR PROJECT, INC. *Tor metrics: Servers* [online] [visited on 2021-04-03]. Available from: https://metrics.torproject.org/networksize.html.

28. THE TOR PROJECT, INC. *Tor metrics: Traffic* [online] [visited on 2021-04-03]. Available from: https://metrics.torproject.org/bandwidth.html.

29. THE TOR PROJECT, INC. *Tor metrics: Users* [online] [visited on 2021-04-03]. Available from: https://metrics.torproject.org/userstats-relay-country.html.

30. THE TOR PROJECT, INC. *Tor metrics: Users by country* [online] [visited on 2021-04-03]. Available from: https://metrics.torproject.org/userstats-relay-table.html.

31. THE ONION ROUTER. *The Onion Routing Solution* [online] [visited on 2021-03-30]. Available from: https://www.onion-router.net/Summary.html.

32. THE TOR PROJECT, INC. *Tor Metrics: Authority relays* [online] [visited on 2021-04-01]. Available from: https://metrics.torproject.org/rs.html#search/flag:authority.

33. THE TOR PROJECT, INC. *Tor Manual: Bridges* [online] [visited on 2021-04-01]. Available from: https://tb-manual.torproject.org/bridges/.

34. THE TOR PROJECT, INC. *Tor: Pluggable Transports* [online] [visited on 2021-04-01]. Available from: https://2019.www.torproject.org/docs/pluggable-transports.html.en.

35. CROMBÉ, Henri; DECLERCQ, Mallory; PEREIRA, Olivier; CANINI, Marco; ROCHET, Florentin. *Correlation attacks on the Tor network*. 2016. Master's thesis, École polytechnique de Louvain (EPL).

36. THE TOR PROJECT, INC. *Tor: Onion Service Protocol* [online] [visited on 2021-04-01]. Available from: https://2019.www.torproject.org/docs/onion-services.html.en.

37. THE TOR PROJECT, INC. *Tor: Software and Services* [online] [visited on 2021-04-01]. Available from: https://2019.www.torproject.org/projects/projects.html.en.

38. THE TOR PROJECT, INC. *Tor Mobile* [online] [visited on 2021-04-01]. Available from: https://support.torproject.org/tormobile/.

39. JOHNSON, Aaron; WACEK, Chris; JANSEN, Rob; SHERR, Micah; SYVERSON, Paul. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In: Association for Computing Machinery, 2013. ISBN 9781450324779. Available from DOI: 10.1145/2508859.2516651.

40. NASR, Milad; BAHRAMALI, Alireza; HOUMANSADR, Amir. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In: Association for Computing Machinery, 2018. ISBN 9781450356930. Available from DOI: 10.1145/3243734.3243824.

41. GHAFIR, I.; SVOBODA, J.; PRENOSIL, V. Tor-based malware and Tor connection detection. In: *International Conference on Frontiers of Communications, Networks and Applications (ICFCNA 2014 - Malaysia)*. 2014, pp. 1–6. Available from DOI: 10.1049/cp.2014.1411.

42. CUZZOCREA, A.; MARTINELLI, F.; MERCALDO, F.; VERCELLI, G. Tor traffic analysis and detection via machine learning techniques. In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 4474–4480. Available from DOI: 10.1109/BigData.2017.8258487.

43. HABIBI LASHKARI, Arash; DRAPER GIL, Gerard; MAMUN, Mohammad; GHORBANI, Ali. Characterization of Tor Traffic using Time based Features. In: 2017, pp. 253–262. Available from DOI: 10.5220/0006105602530262.

44. HE, Gaofeng; YANG, Ming; LUO, Junzhou; GU, Xiaodan. A novel application classification attack against Tor. *Concurrency and Computation: Practice and Experience*. 2015, vol. 27, no. 18, pp. 5640–5661. Available from DOI: https://doi.org/10.1002/cpe.3593.

45. SHAHBAR, K.; ZINCIR-HEYWOOD, A. N. Benchmarking two techniques for Tor classification: Flow level and circuit level classification. In: *2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. 2014, pp. 1–8. Available from DOI: 10.1109/CICYBS.2014.7013368.

46. SHAHBAR, Khalid; ZINCIR-HEYWOOD, A Nur. Packet momentum for identification of anonymity networks. *Journal of Cyber Security and Mobility*. 2017, pp. 27–56. Available from DOI: 10.13052/2245-1439.612.

47. CANADIAN INSTITUTE FOR CYBERSECURITY [online] [visited on 2021-04-17]. Available from: https://www.unb.ca/cic/datasets/tor.html.

48. LIU, Huan. Feature Selection. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Springer US, 2010, pp. 402–406. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_306.

49. SCIKIT LEARN. *Model persistence* [online] [visited on 2021-05-03]. Available from: https://scikit-learn.org/stable/modules/model_persistence.html.

Appendix A

Acronyms

CSV Comma-separated values

DARPA Defense Advanced Research Projects Agency

DNS Domain Name Server

DPI Deep packet inspection

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IANA Internet Assigned Numbers Authority

IETF Internet Engineering Task Force

IP Internet Protocol

IPFIX Internet Protocol Flow Information Export

ISP Internet service provider

KNN K-nearest neighbours

MAC Media Access Control

ML Machine learning

OR Onion router

OSI Open Systems Interconnection

P2P Peer-to-peer

PDL Page description language

SVMs Support vector machines

TCP Transmission Control Protocol

UDP User Datagram Protocol

VoIP Voice over Internet Protocol

Appendix B

Contents of the SD card

- `readme.txt` ..... description of the contents of the SD card
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `experiments` ..... Jupyter notebooks used for the experiments
- `software` ..... software prototype and the trained models
  - `Tor_classifier.py` ..... Python script of the software prototype
- `data` ..... the directory of the training datasets