Website Fingerprinting Analysis of Encrypted Network Traffic by Machine Learning Techniques Using Proxy Server

Authors:

Shaik Fathima,

Amatov Batyi,

Siddique reza khan.

Under the supervision of:

A.Panchenko,Prof.

T.Ziemann,PhD

E.Strehle,B.Sc.

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Cottbus, April 26, 2020

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

IQR	Inter Quartile Range.
KNN	K Nearest Neighbor.
NN	Neural Network.
RF	Random Forest.
SVM	Support Vector Machine.
URL	Uniform Resource Locator.
VM	Virtual Machine.

Abstract

In era of computer technology anonymity and confidentiality play a prominent role. In a simple client-server model a confidential connection should be provided between the client and web server, such that attacker will not be able to read the data from payload of the packets during connection. This confidentiality is achieved by network protocols which encrypt the payload of packets on a different network levels. But in this model anonymity of browsing is not provided. In order to browse anonymously we are using a proxy server as one of the solution for anonymity. In this case, proxy server plays a connection point between client and web server, such that client's provider will see only a connection to proxy server, not to a real web server.

A website fingerprinting is a special type of attack on anonymity browsing where adversary attempts to recognize the site which client has visited based on analyzing various of patterns in the traffic flow. The useful patterns which help out to analyze encrypted transmission between client and web server can be: packet size, directions, timestamps, loading time etc. The raw data for the attack is collected by capturing network data while crawling different web-pages of websites and its subpages. After the crawling, useful features are extracted from the capture traffic in a feature extraction part. In the training and testing part machine learning algorithms are used in order to predict a possible website which the client has visited via proxy server based on the training data [35].

Keywords: Website fingerprinting, proxy server, crawling, capturing traffic, interpolation, normalization, outlier detection, IQR, Z-score, isolation forest, Support Vector Machine (SVM), k-nearest neighbor, random forest, neural network, cross validation, confusion matrix.

1 Introduction

Machine learning is used for website fingerprinting of encrypted traffic between proxy server and victim. For website fingerprinting purposes it is necessary to capture the network traffic at the client side. From the captured traffic we need to extract useful features by counting the sizes of each packet, storing it's direction and meta-data for future analysis. Proxy server uses an encrypted connection and that is why it is not possible to view TCP payload in the packet as a plain text. So, the attacker can analyze only limited characteristics of network connection up to transport layer. In the further step, data splits in two parts for the training and testing purposes which are using for implementing different machine learning techniques in order to classify a network trace to a proper website. At the end we will get a trained classifier which potentially can reveal a client's active communication even with browsing via proxy server, there by breaking anonymity in the network using proxy.

Project is processed by following steps shown in Figure 1:

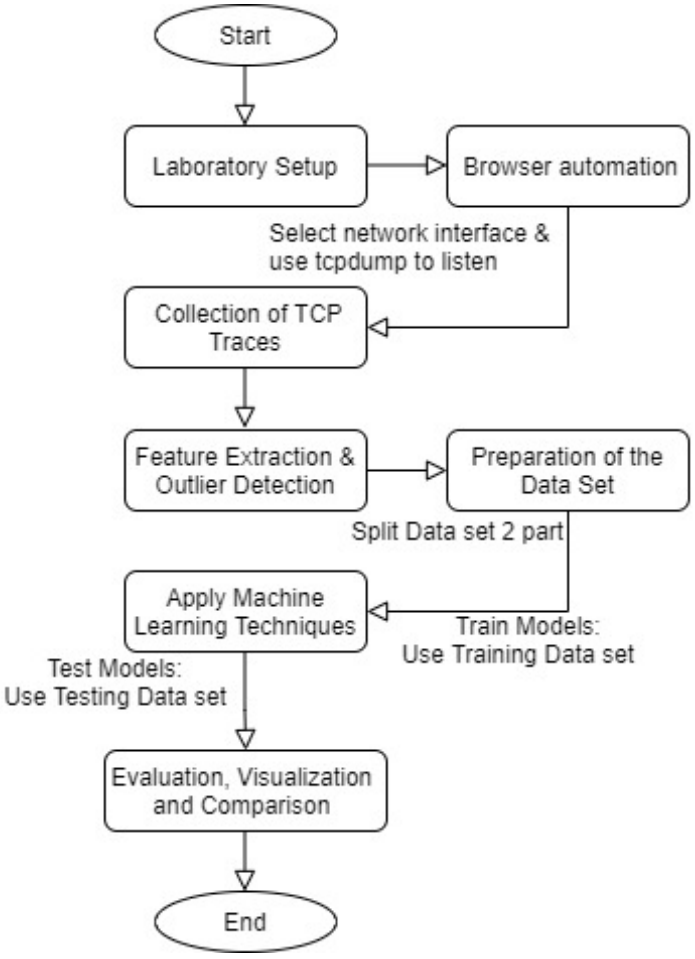


Figure 1: Step by step flow diagram for project implementation

In the initial step we are preparing environment which includes configuration of proxy server and connecting to the client. Next, we are writing a python script which allows browser work automatically. Using browser automation we are capturing network traffic during crawling of 100 websites.

Browser automatically crawls websites one by one, records traffic in dumps, collects the log information during automated actions, such as timestamps, IP addresses, server or client side errors, etc. The collection of such data allows us to held a granular control for loading of each page, to define and process all possible errors.

In the feature extraction and visualization part we extended the script such that it reads packet headers and can parse up to network layer. Using Scapy [2] python library we can extract the useful feature out of TCP records from captured network traffic during browsing of each website. Scapy parses the packet structure on different network layers. Later on we define features which can be useful for attack. The assumed useful features are: the number of outgoing, incoming packets, sum of TCP payloads lengths and cumulative chronicle of TCP record sequences.

After the feature extraction we are trying to prepare our data set for machine learning algorithms. The preparation includes detecting and removing of outliers which can be created during different network issues on the web server or client sides. These improper behaviour should be removed from the data set.

In the next step, we need to train different machine learning algorithms, such as: SVM, random forest, k-nearest neighbors, neural networks. They are used to classify the unknown network traffic to proper website in order to break anonymous browsing. Classifier algorithms may require different representation of data.

At the final part, evaluation and comparison of machine learning algorithms are done. Representation of training and testing of data will be shown in the view of confusion matrix and confidence intervals. The connection of website patterns and its detection accuracy by algorithms will be detected. At the end a brief conclusion of results will be held about traffic analysis attack for website fingerprinting.

1.1 Environment setup

For the gathering data we need to setup the environment. The proper setup is a vital role in the project, the data should be gathered in between of client and proxy server. The network topology is shown on the Figure 2. Proxy server is build with the help of Tinyproxy[25] software. Also, in our topology it is necessary to have an encrypted connection to proxy server. The encryption is done using Stunnel[25] open-source application, which provides a universal TLS and SSL tunneling service.

Tinyproxy is an HTTP and HTTPS proxy server. Using this we can set up Virtual Machine (VM) as a proxy server. On the client side we can route all HTTP and HTTPS traffic through this remote proxy server. The main key idea to route the traffic is to keep the network restrictions, which means adding an encrypted extra layer as remediation service to prevent data leakage. So, simply rerouting HTTP/HTTPS traffic to proxy server is not sufficient. Because, the HTTP traffic from client VM to Tinyproxy[25] remains unencrypted, the packet capturing and inspection on it is easy. To avoid all this issues and encrypt the traffic between client and proxy server we used Stunnel software.

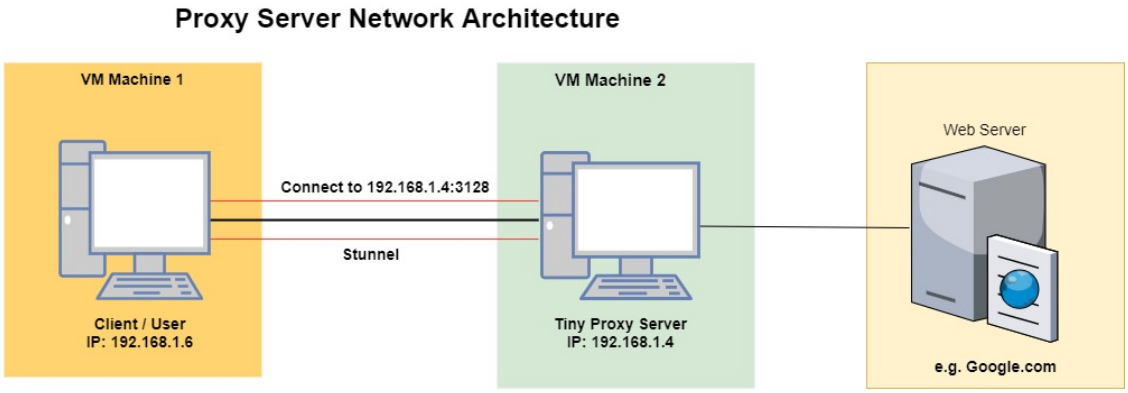


Figure 2: Environment setup for proxy server using Stunnel[4]

As it shown on the Figure 2, client VM connected with proxy server VM via encrypted channel of Stunnel[25] software. All the traffic from client goes through proxy server to the target web-server. Such that a possible attacker or client's provider will not know which website browses the client. For the attacker only IP address of proxy server will be visible in the packet header. All other information is stored in an encrypted format inside the TCP payload of the packet.

So, we have two virtual machines connected with network address translation protocol. The first machine is a client VM, the second machine is for proxy server. Stunnel software was installed on both machines and properly configured. On the proxy VM the Tinyproxy was installed and configured to accept traffic from the client IP address.

This type of network topology allows to create own proxy server with encrypted traffic between itself and client VM which gives a user some level of anonymity as secure browsing. Step by step setup process is shown in Figure 37, Appendix A.

1.2 Programming languages, tools, drivers, libraries

For our project two virtual machines were used on the host Windows 10 machine. These two virtual machines act as client VM and proxy server VM. Client's browser is crawling different websites through proxy VM, that is, active communication between them an attacker can use as captured network traffic. Below are mentioned tools, libraries and supporting software which were used for successful completion of the tasks:

- **Host Machine**: Windows 10
- **Virtual Machine**: Ubuntu Linux[18] 18.04 LTS
- **Programming Language**: Python 3.6.8[39]
- **IDE**: Sublime Text[36]
- **Browser**: Google Chrome[17].
- **Browser Automation Tool**: Selenium[3]
- **Collecting TCP/IP traces**: TCPdump tool[8] used for collecting and saving TCP traces into files with pcap extension.
- **Wireshark**: The tool wireshark[20] is used to inspect the packet structure on the different network layers.

2 Traffic Capturing

After preparation of environment the next part of the task is a capturing of network traffic by using TCPdump[8] tool which saves captured traffic in a identifier_runnumber_website files. Automated browser crawled 100 different Uniform Resource Locator (URL) in the depth of four subpages, it means that browser opens a main page for single URL and goes on until the depth of 4. The depth of 4 mean traversing 4 subpages consecutively. The term "subpage" is URL link gathered for crawling, it may be a main URL link or it's child page. In our case subpage is chosen randomly from one of visited URL of the main website with several valid links. So, browser crawls website's subpages and collects 40 traces for each URL, which means 10 crawlings for each subpage.

2.1 Automation and browser settings configuration

Browser automation is a process of running various tasks with strictly specified functionality without human interaction[1]. In our case for browser automation we wrote a python script. For this purpose we used Selenium[3] software which is a framework generally used for testing of web applications. In our case we are using it for Google Chrome for browser automation[19]. Browser automation was implemented to configure browser settings for proper crawlings. The following browser settings was configured: deleting cookies after each crawling, opening a webpage in a full screen size and turning on logs functions.

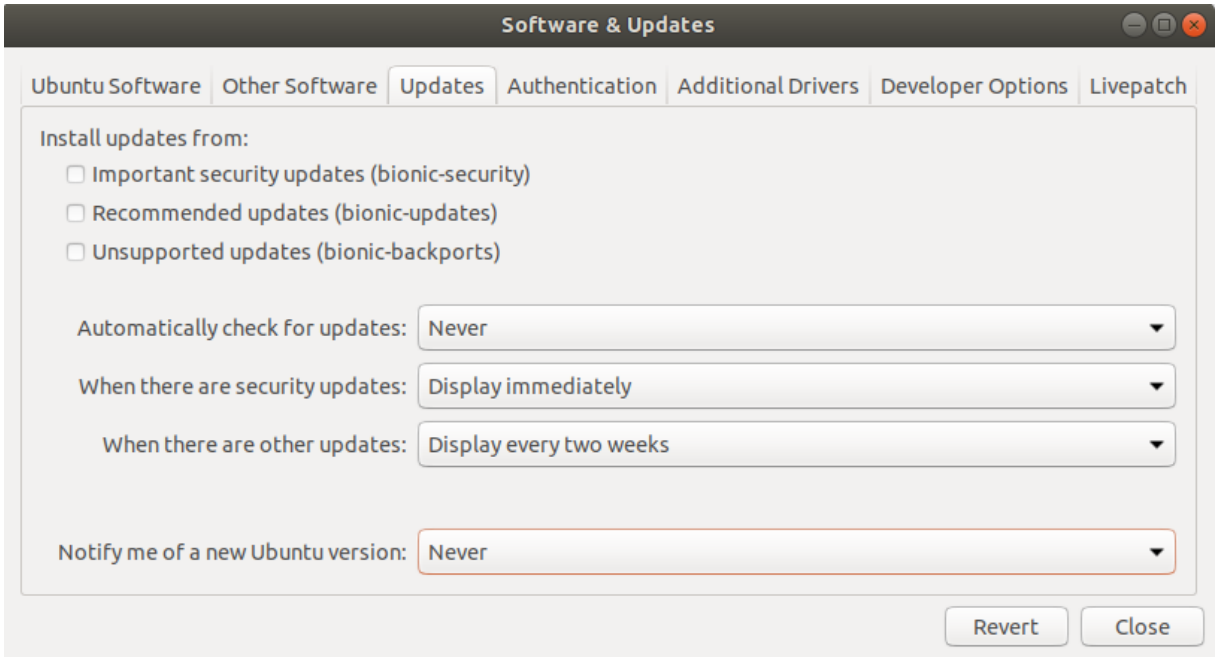


Figure 3: Turning off OS updates

For a proper network capturing, a python script is implemented to manipulate browser settings where we can disable caching and storing cookies after each crawling. This means that each crawling would have the same conditions and chronology of packets exchange should be the same. One of the main condition for equal and proper crawlings is to turn off all possible side effects, such as software updates, OS updates as on Figure 3, browser updates, background traffic. It could influence on the data set and should be avoided in order not to gather deviated and untruthful results.

For crawling websites and capturing of traffic without human interaction the following steps are performed:

1. By manipulating the Chrome options we run delete_all_cookies()function, which explicitly deletes all the saved cookies.
2. Set up TCPdump and make it listen to the proper interface “enp0s9”. Exchanged packets between the client and server will be recorded and saved.
3. Browser automatically visits URL and randomly chooses subpages for crawling.
4. Browser waits until the web-page is loaded, after loading the script retrieves the timestamp before and after visiting, remote IP, local IP, any actions and unexpected errors in a special logs.
5. TCPdump saves traffic in pcap files.
6. After each crawling script terminates TCPdump’s recording.
7. Repeat from step 3 until browser will reach a certain depth of crawling subpages.
8. Collect all possible meta data during crawlings.
9. Write log files and keep granular control for error handling.

3 Feature Extraction

After the collecting data we have 4000 pcap files which contains captured network traffic for 100 different websites. For each website we have 40 instances of 4 unique subpages, where each 10 belongs for one subpage. Files with pcap extension were created by TCPdump library tool for capturing network traffic. These files can be opened by Wireshark[20] software where all requests and responses in a traffic will be visible. Although TCP packets' payloads are encrypted and it is not possible to see the content, but the headers of the packets contain useful metadata which can be used for the traffic analysis. Generally, we need length of the TCP record and their direction. Our goal is to extract all useful features from pcap files for machine learning algorithms in order to classify unknown page to a proper website based on the feature trace.

3.1 Extracting features

For the extraction purposes the Scapy[21] python library is used. Scapy allows to parse pcap and extract attributes from packet headers. For the machine learning purposes, we need the following features from traffic which was captured during crawling of each subpage:

- Packet length of TCP payload
- Packet direction (outgoing -, incoming +)
- Total number of outgoing packets
- Total number of incoming packets
- Total TCP payloads length of outgoing packets
- Total TCP payloads length of incoming packets
- Cumulative sum of TCP packet

Cumulative sum of TCP packet length is a sequence of the difference between outgoing and incoming packets TCP payload lengths. By our notation we are calling "cumulative sums" as "cumulative sequence" time to time. The first two features was extracted directly using Scapy[2] library, the all others features was calculated after parsing. For the granular control for possible error handling purposes, we extracted and stored in a file additional features: time stamp, source and destination IP addresses, TCP sequence and acknowledgment.

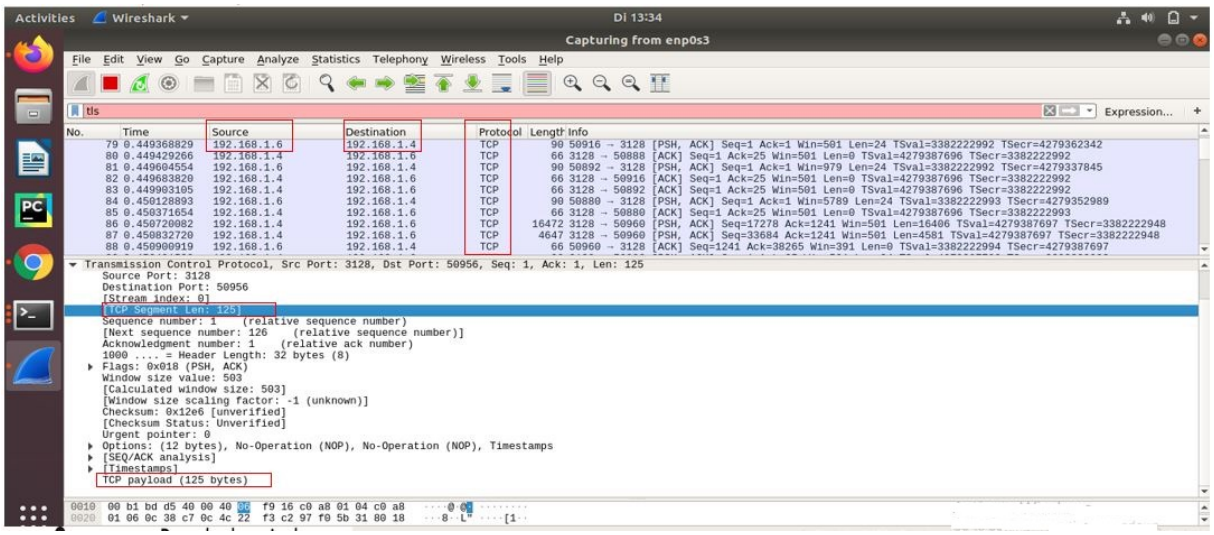


Figure 4: Capturing traffic with Wireshark

On the Figure 4 we can see the encrypted packets by TLS protocol on the application layer. It means that we can only see TCP payload in transport layer. So, that the payload is encrypted in our case and not visible to the attacker.

Figure 5 shows the parsed TCP packet's features with the help of Scapy library. In this picture the TCP payload of the packet equals to 24 bytes.

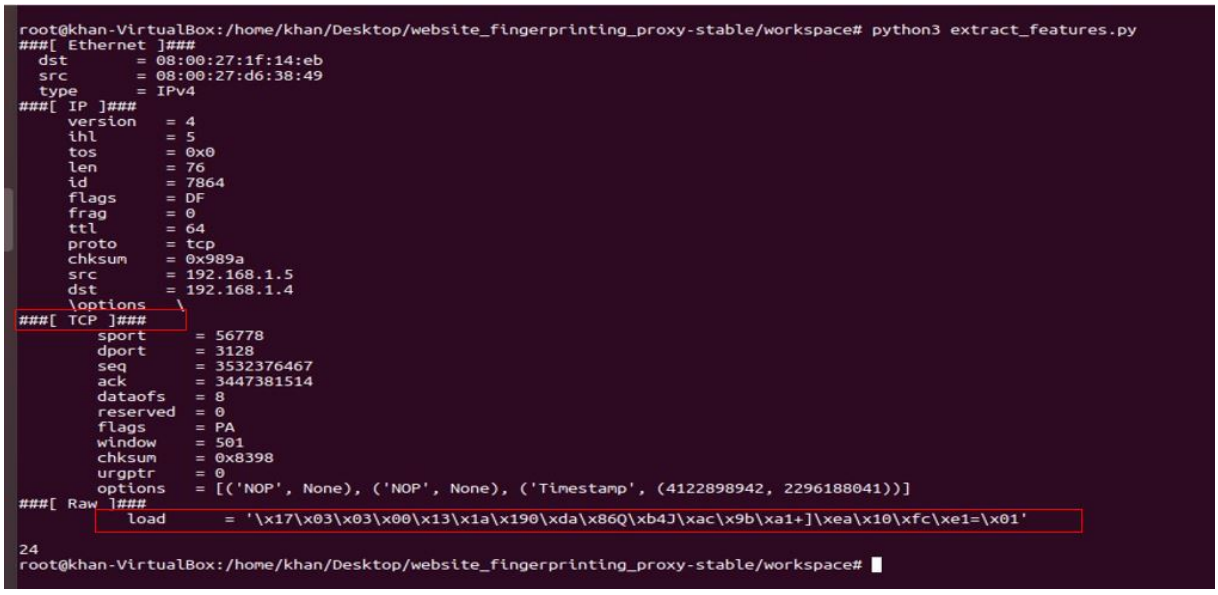


Figure 5: Analysis traffic with Scapy

3.2 Interpolation

Interpolation is a process of creating new data points on the curve based on the known points. On the Figure 6 we can see blue colored data points which are connected by orange linear and by cubic green interpolated lines.

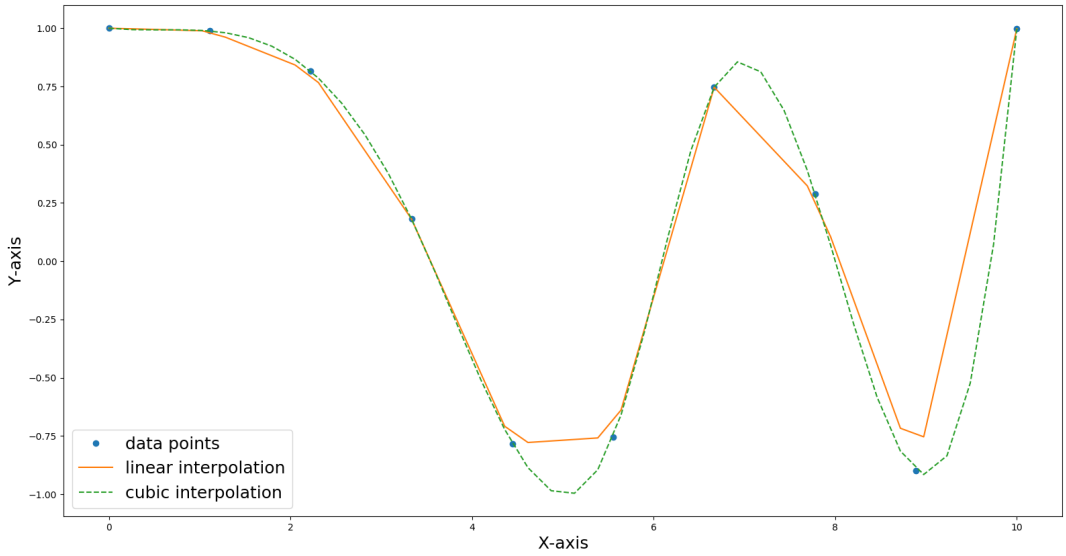


Figure 6: Interpolation explanation

Since the length of network traffic which was captured during each crawling is different, we need to interpolate it to make the comparison of cumulative sequence possible.

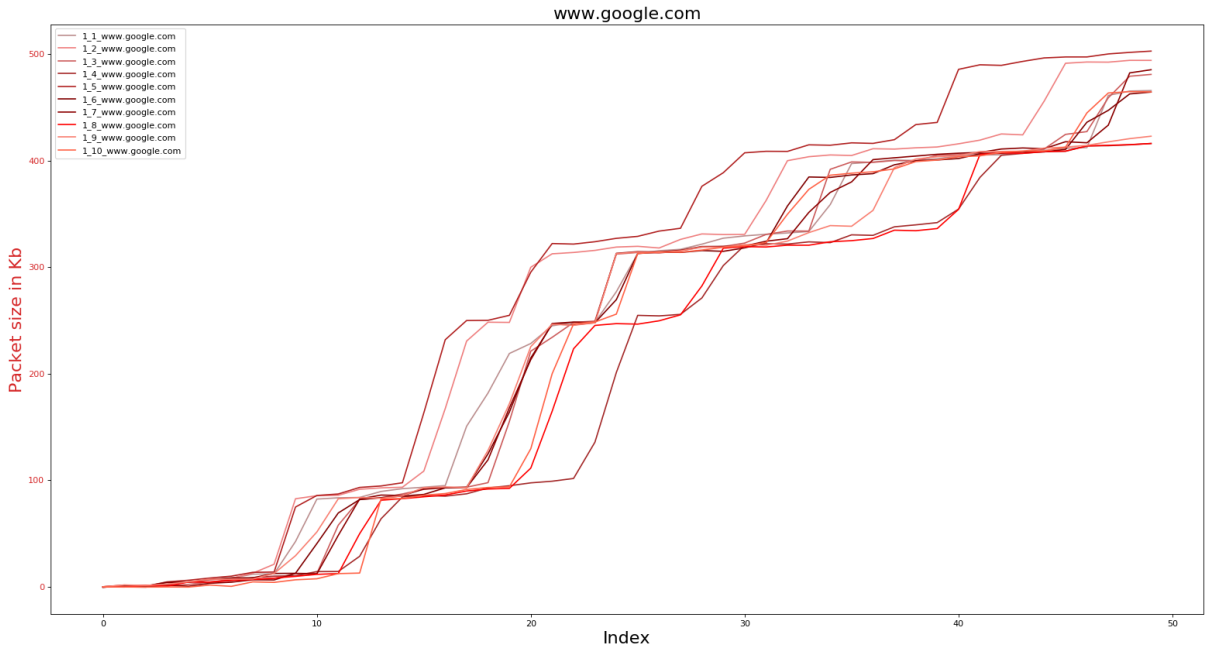


Figure 7: 10 curves of the first subpage for www.google.com after interpolation

For example, the first two crawling of the same subpage have different number of exchanged packets. Since, the number of exchanged packets is different, the length of cumulative sequences will also be different. This means, that curves on our graphs will be not comparable. In order to avoid it the interpolation is necessary. For the interpolation we are using scipy[21] python library and invoking interpolate function. This means that we can artificially compress our curve until 50 points and make curves comparable as it shown on the Figure 7.

4 Outlier Detection

The term *"outlier"* can be defined as: ”an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism”[28]. In general words an outlier is an observation which differs from set of patterns in a given sample data, the procedure to find this unusual observation is called *"outlier detection"*.

Outliers are detected based on the features after their extraction from the raw data. Outliers can reduce the accuracy of classification algorithms because their does not represent a trustful situation. In our case comparative study of outliers is based on unusual curves. They are represented by cumulative sequences with unexpectedly high or low amplitude which does not follow variance boundaries.

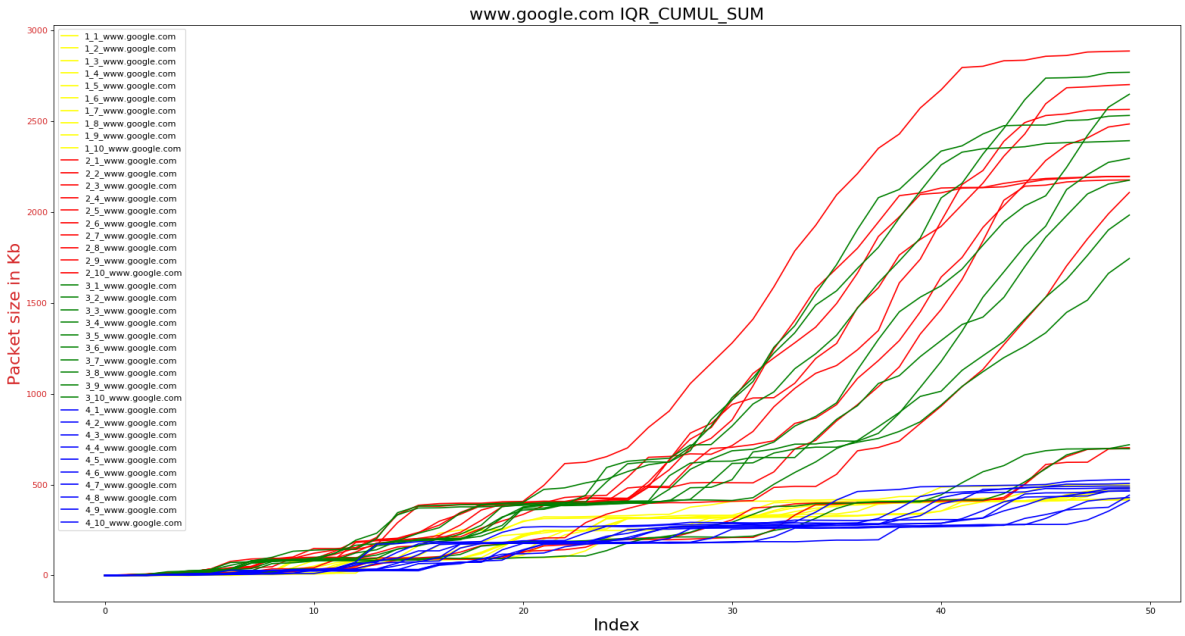


Figure 8: Visualisation for www.google.com before removing outliers

The above Figure 8 is the visualization of the www.google.com and it‘s subpages. Visualization of data is necessary to easily identify outliers based on the graphical representation. To represent it in the graphical format we used python library matplotlib[22]. The plotted graph is based on the cumulative packet sequences. Lines are differentiated by the color, but curves which belongs to same subpage has the same group of colors. In this study project we evaluated 3 types of outlier detection methods:

- Z-score
- Isolation Forest
- IQR - Interquantile Range

4.1 Z-score

Z-score [16] is one of the statistical method for outlier detection. It represents the number of standard deviations of data points which are deviated from the sample mean. The Z-score value can be calculated by using the following equation:

$$Z = \frac{x - \mu}{\sigma} \tag{1}$$

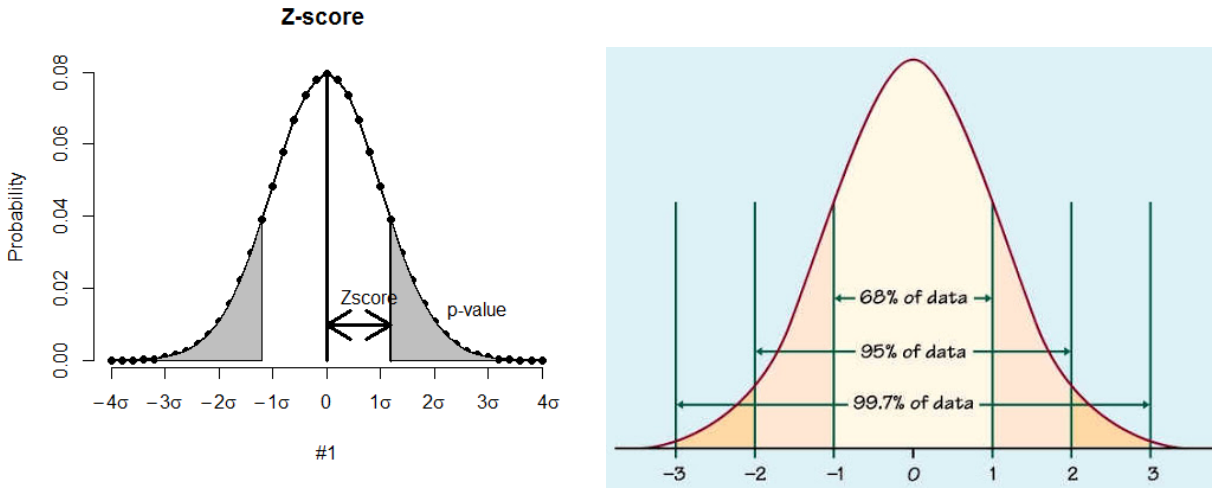
Here x is a value from our population. In our case x is a final cumulative sum which represents sum of total sizes of incoming and outgoing packets for each trace. σ is a statistical standard deviation and μ represents a population mean. We can interpret Z-score values based on the following ranges:

- If Z-score < 0, then value < mean
- If Z-score > 0, then value > mean
- If Z-score = 0, then value = mean

STANDARD NORMAL DISTRIBUTION: Table Values Represent AREA to the LEFT of the Z score.

Z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.50000	.50399	.50798	.51197	.51595	.51994	.52392	.52790	.53188	.53586
0.1	.53983	.54380	.54776	.55172	.55567	.55962	.56356	.56749	.57142	.57535
0.2	.57926	.58317	.58706	.59095	.59483	.59871	.60257	.60642	.61026	.61409
0.3	.61791	.62172	.62552	.62930	.63307	.63683	.64058	.64431	.64803	.65173
0.4	.65542	.65910	.66276	.66640	.67003	.67364	.67724	.68082	.68439	.68793
0.5	.69146	.69497	.69847	.70194	.70540	.70884	.71226	.71566	.71904	.72240
0.6	.72575	.72907	.73237	.73565	.73891	.74215	.74537	.74857	.75175	.75490
0.7	.75804	.76115	.76424	.76730	.77035	.77337	.77637	.77935	.78230	.78524
0.8	.78814	.79103	.79389	.79673	.79955	.80234	.80511	.80785	.81057	.81327
0.9	.81594	.81859	.82121	.82381	.82639	.82894	.83147	.83398	.83646	.83891
1.0	.84134	.84375	.84614	.84849	.85083	.85314	.85543	.85769	.85993	.86214
1.1	.86433	.86650	.86864	.87076	.87286	.87493	.87698	.87900	.88100	.88298
1.2	.88493	.88686	.88877	.89065	.89251	.89435	.89617	.89796	.89973	.90147
1.3	.90320	.90490	.90658	.90824	.90988	.91149	.91309	.91466	.91621	.91774
1.4	.91924	.92073	.92220	.92364	.92507	.92647	.92785	.92922	.93056	.93189
1.5	.93319	.93448	.93574	.93699	.93822	.93943	.94062	.94179	.94295	.94408
1.6	.94520	.94630	.94738	.94845	.94950	.95053	.95154	.95254	.95352	.95449
1.7	.95543	.95637	.95728	.95818	.95907	.95994	.96080	.96164	.96246	.96327
1.8	.96407	.96485	.96562	.96638	.96712	.96784	.96856	.96926	.96995	.97062
1.9	.97128	.97193	.97257	.97320	.97381	.97441	.97500	.97558	.97615	.97670
2.0	.97725	.97778	.97831	.97882	.97932	.97982	.98030	.98077	.98124	.98169
2.1	.98214	.98257	.98300	.98341	.98382	.98422	.98461	.98500	.98537	.98574
2.2	.98610	.98645	.98679	.98713	.98745	.98778	.98809	.98840	.98870	.98899
2.3	.98928	.98956	.98983	.99010	.99036	.99061	.99086	.99111	.99134	.99158
2.4	.99180	.99202	.99224	.99245	.99266	.99286	.99305	.99324	.99343	.99361
2.5	.99379	.99396	.99413	.99430	.99446	.99461	.99477	.99492	.99506	.99520
2.6	.99534	.99547	.99560	.99573	.99585	.99598	.99609	.99621	.99632	.99643
2.7	.99653	.99664	.99674	.99683	.99693	.99702	.99711	.99720	.99728	.99736
2.8	.99744	.99752	.99760	.99767	.99774	.99781	.99788	.99795	.99801	.99807
2.9	.99813	.99819	.99825	.99831	.99836	.99841	.99846	.99851	.99856	.99861
3.0	.99865	.99869	.99874	.99878	.99882	.99886	.99889	.99893	.99896	.99900
3.1	.99903	.99906	.99910	.99913	.99916	.99918	.99921	.99924	.99926	.99929
3.2	.99931	.99934	.99936	.99938	.99940	.99942	.99944	.99946	.99948	.99950
3.3	.99952	.99953	.99955	.99957	.99958	.99960	.99961	.99962	.99964	.99965
3.4	.99966	.99968	.99969	.99970	.99971	.99972	.99973	.99974	.99975	.99976
3.5	.99977	.99978	.99978	.99979	.99980	.99981	.99981	.99982	.99983	.99983
3.6	.99984	.99985	.99985	.99986	.99986	.99987	.99987	.99988	.99988	.99989
3.7	.99989	.99990	.99990	.99990	.99991	.99991	.99992	.99992	.99992	.99992
3.8	.99993	.99993	.99993	.99994	.99994	.99994	.99994	.99995	.99995	.99995
3.9	.99995	.99995	.99996	.99996	.99996	.99996	.99996	.99996	.99997	.99997

Figure 9: Standard normal distribution table



(a) Z-score representation (b) Data percentage on the graph

Figure 10: Z-score graph

To discard outliers we took as threshold Z-score value 1.8; in Figure 9[9] and Figure 10(a)[10] and 10(b)[14] value approximately represents 96 percent of probability of getting a trace into the interval. This means, remaining 4 percent of traces will be considered as outliers and will be removed.

4.2 Isolation Forest

Isolation forest [33] is an ensemble regressor, it uses a concept of isolation to separate outliers. Algorithm builds an ensemble of random trees for a given data set. Outliers are points with the shortest average path length. An outlier score can be calculated by following equation:

$$S(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{2}$$

h(x) is a number of edges in the tree for a point x
n is number of nodes
c(n) is a normalisation constant for a data set of size n

So, if E(h(x)) $\rightarrow$ 0, outlier score $\rightarrow$ 1, means anomalous data points are easier to isolate from the normal data points.

On the Figure 11[13] we can see red point which was detected as outlier. On the most right side the tree depiction with red outlier point is depicted.

For the isolation forest we passed all points from traces after interpolation from each sub-page. For example, after the crawling of 4 "google.com" sub-pages 10 times, we got 40 traces. We passed all points from traces which belongs to the same sub-page.

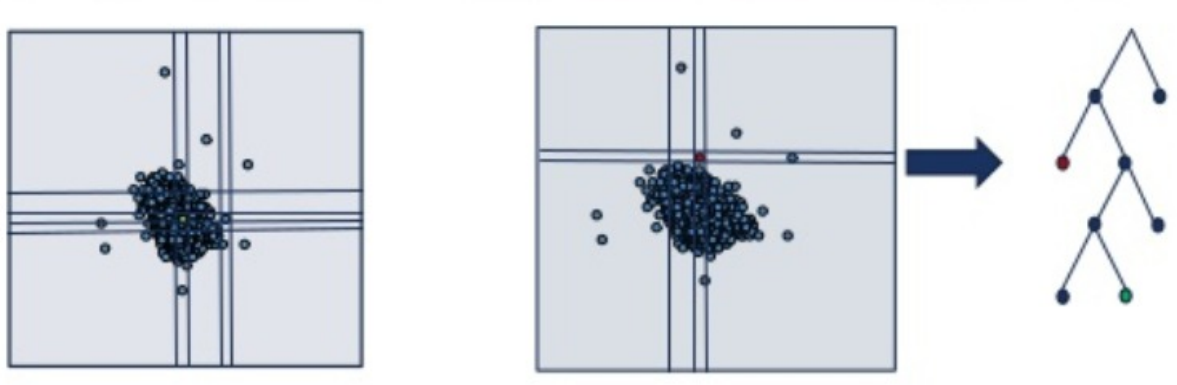


Figure 11: Outlier detection with Isolation Forest

Since, each trace consist of 54 values and for each sub-page we have 10 related traces. So, we get 540 values which builds a data set. Algorithms detects outlier points inside this data set and based on these outlier points we can find outlier traces.

4.3 Interquartile Range (IQR)

The interquartile[5] range is also one of the methods to implement outlier detection. Our actual dimension of the data set is 10 by 50 for each sub page, where 10 is the number of rows and 50 is the number of columns. Each row represents one crawling, each column represent one feature of the whole feature vector.

As shown on the Figure 12, for the outlier detection algorithm we are taking the last column of the data set for each sub page, which represent the final cumulative sums. Since, each sub page has 10 crawlings we have 10 cumulative sums. These values we are passing to interquartile range algorithm.

	A	B	C	D	E	F	G	H	AX	AY
1		0	1	2	3	4	5	6	48	49
2	1_1_www.google.com	-442	1330.48	533.76	4478.44	4500.68	6710.4	8200.52	465451.16	465936.32
3	1_2_www.google.com	67	1704.68	844.52	2823.72	4637.52	6675.4	7992.28	494288.36	494256.68
4	1_3_www.google.com	-48	109.72	1543.8	851.38	2308.48	4485.1	7153.52	479376.56	481091.38
5	1_4_www.google.com	67	893.92	1274.76	2290.76	4687.48	4289.8	6036.88	415001.72	416264.52
6	1_5_www.google.com	67	1553.6	708.72	4681.56	6206.08	8291.8	10054.68	501761.84	502975.68
7	1_6_www.google.com	-442	1330.8	537.44	4178.08	4444.36	6924.6	8385.36	462775.92	464512.16
8	1_7_www.google.com	0	-46.08	-236.16	1509.96	577.76	3339.4	4380.28	482512.8	485485.72
9	1_8_www.google.com	-48	-176.52	1518.16	680.44	4571.32	4540.4	6109.56	415146.32	416216.28
10	1_9_www.google.com	67	1755	1350	2534	4728	7212	7207	420790	423069
11	1_10_www.google.com	0	-146.4	-235.2	-88.8	-212.4	1692.5	470.8	464912.4	464696.9

Figure 12: Data set for 1-st sub page of google.com

As shown on the Figure 13[12], we can see the representation of interquartile range and detected outliers.

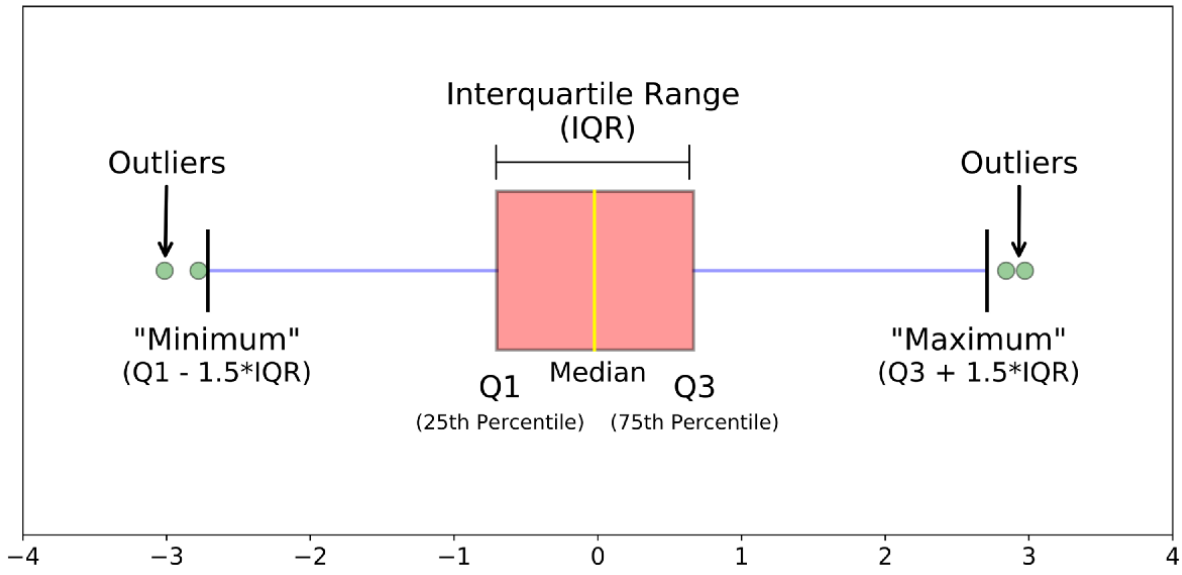


Figure 13: Interquartile Range

For detecting outliers through interquartile range we need to do the following preparations:

1. Find the lowest or minimum value of data set
2. Take Q1 (first quartile) which represent the first quarter or half of the data set
3. Find the median which shows the midpoint of the data set
4. Take Q3 which is the third quartile which represents three-quarters of data set

$$IQR = Q1 - Q3 \tag{3}$$

Outlier detection based IQR can be done in four step:

- Calculate the interquartile range based on the data set (i.e. 40 final cumulative sums from four sub-page)
- A constant 1.5 value is multiplied to the interquartile range
- Add 1.5 * IQR to third quartile. The value which is greater than the quartile, can be suspected as outlier.
- Subtract 1.5 *IQR from the first quartile. The value which is less than the first quartile can also be suspected as outlier.

4.4 Results of outlier detection

The scipy[21] and sci-kit[38] python libraries provides already implemented algorithms in functions for Z-score, IQR and isolation forest algorithms. In order to define the best algorithm we tried to compare them, for example, on "google.com" dataset which contains traces only for "google.com" website. Here we have 40 curves which represent a cumulative sequence of TCP payload length sum. Each curve consist from 50 points after interpolation. Out of this dataset we need to find outliers curves. For IQR and Z-score algorithms we pass a final cumulative sum which represents the sum of the differences of incoming and outgoing TCP packet's lengths. For the isolation forest we passed all points of each curve. The comparison table is shown on the Figure 14.

Based on google dataset (40 features):

	IQR	Z-score (threshold = 1.8)	Isolation Forest (contamination = 0.02, random_state = 0)
number of outliers:	7	6	9
number of features:	40	40	40
mean variance reduced by:	59%	61%	46%
variance average efficiency:	8%	10%	5%
outliers:	1_4 1_5 1_8 2_7 2_8 3_2 3_9	1_8 2_7 2_8 3_2 3_9 4_5	1_3 1_5 1_6 2_6 3_7 4_3 4_6 4_8 4_9

Figure 14: Comparison of outliers detection algorithms

According to the comparison, we can notice that Z-score and IQR showed better results then Isolation Forest. The comparison is calculated by the mean of reduced variance between dataset before removing outliers and after removing. The variance is calculated by the difference of points values of each curve to the appropriate points to other curves. After all, the mean variance is taken and percentage of reducing after outlier removal is calculated.

Since, each algorithm removes different number of outliers, we need to calculate the variance average efficiency. It shows that Z-score shows the best result by removing a smaller number of outliers with high average efficiency. But, the main disadvantage for Z-score that we need to find a proper value for the threshold. For the google dataset the best value was 1.8 which approximately 96 percent of confidence interval. So, from our point of view the best algorithm which does not require manual parameter selection is IQR. It shows comparable results to Z-score with manually chosen threshold parameter.

So, by using IQR detection algorithm we removed 5 percent of outliers for each website. In our case, it is 2 curves out of forty for each website, and in total 200 outliers out of whole dataset. We removed the same number of outliers in order to have equal sizes of datasets for each website. As it shown on the Figure 15, IQR detects outliers based on final cumulative sum. The lines which consist from points are outliers.

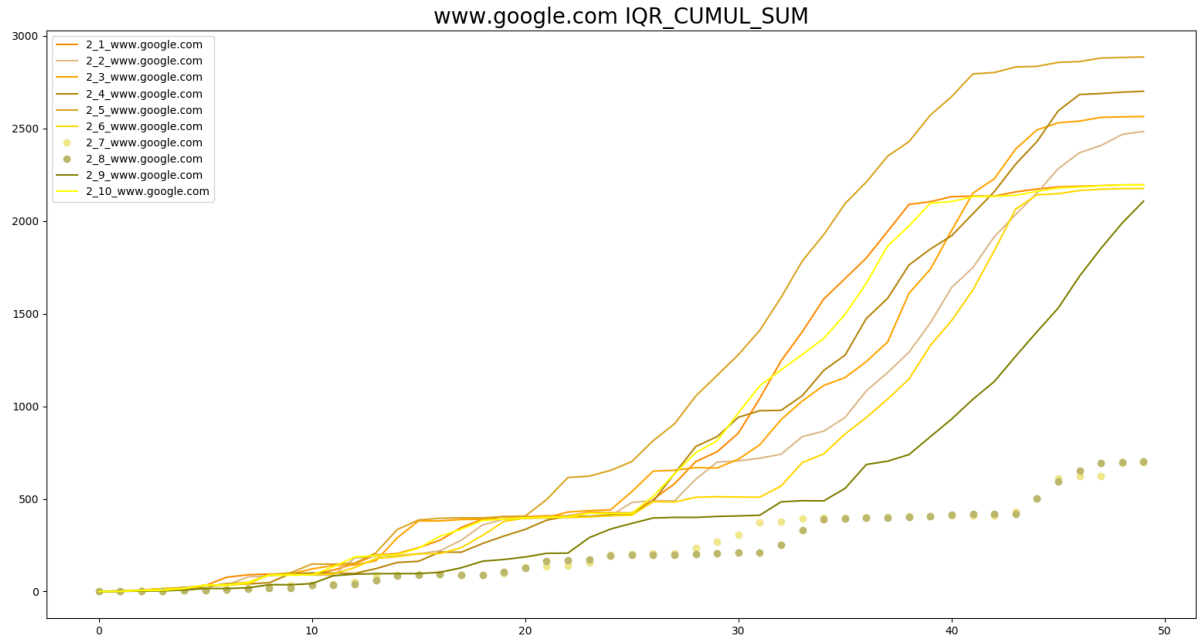


Figure 15: IQR outlier detection based on final cumulative sum on 10 crawlings of one subpage

5 Machine Learning Techniques

Machine learning technique can be defined as *"The ability for computer to learn without explicitly being programmed" Arthur Samuel[34]* In simple words, algorithms which predicts data based on the learnt data set. In this study project, we evaluated our results for supervised machine learning technique.

Supervised Machine Learning can be classified based on input data. In this project we used classification algorithms and we got results using different classification techniques.

Classification in machine learning is used to identify to which subgroup a new observation belongs, and it can be further divided into certain categories or classes. Before implementing the classification algorithms we need to prepare our data set. Hence this data set should be divide on the trained and tested parts. For training and testing of data sets we implemented the following classification algorithms:

- Support Vector Machine (SVM)
- Random Forest (RF)
- K Nearest Neighbor (KNN)
- Neural Network (NN)

5.1 K-Fold Cross Validation

For splitting for testing and training data sets we used "k-fold cross validation" technique. Cross validation is a statistical method which is used to prepare data for implementing the machine learning techniques. In this technique we have single parameter "k" which can be referred as number of groups that given data sample can be spitted into. So, this procedure can be called as k-fold cross-validation[24].

The k-value should be chosen accurately in order to give the highest accuracy for the algorithm and make the result objective. The smaller k the higher variance appears in the model results, the bigger k the higher is the bias which overestimates the skill of the model.

As shown on the Figure 16, we took the k-value 10 which gave the best result with low bias and small variance using try and error approach. The procedure is simple to understand and works as follows:

- Shuffle the data randomly
- Split the data set into k=10 different groups
- Consider one group as a test dataset
- Consider remaining groups as training dataset
- Fit the model on training dataset and evaluate it on testing dataset
- Preserve evaluation score and end the procedure after each fold of data was tested. This means each sample can be 1 time for testing and 9 times for training[24].

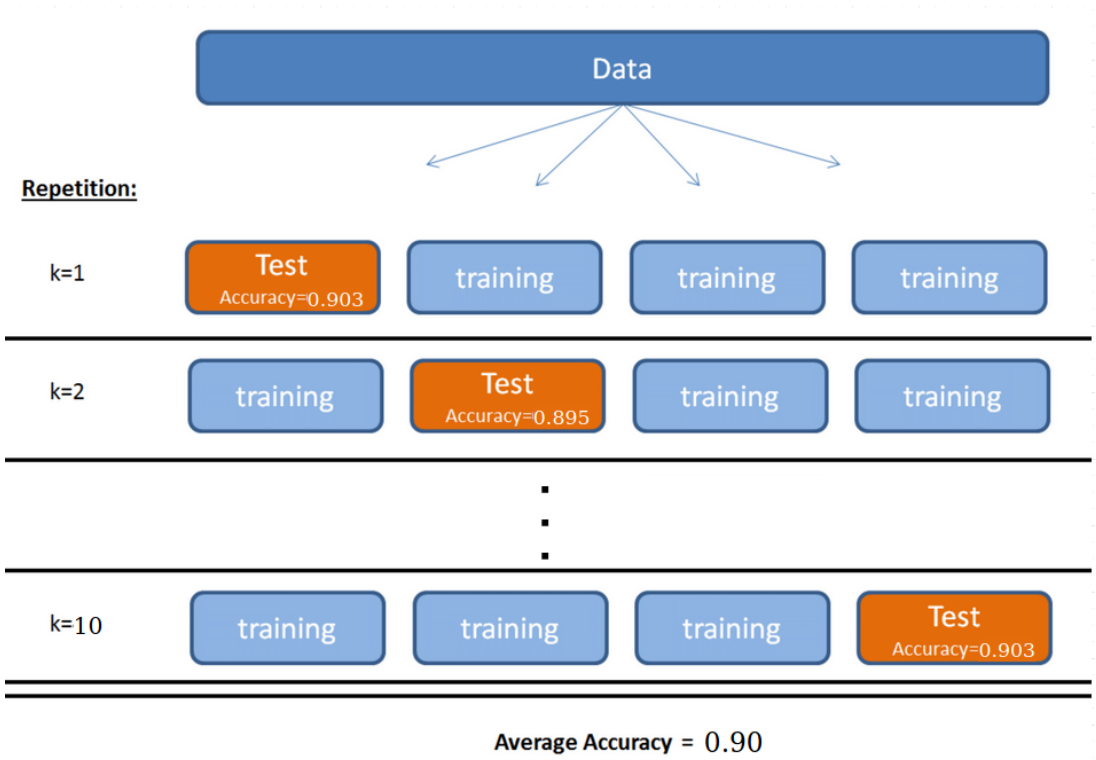


Figure 16: Example of 10-fold cross validation on Random Forest classifier

In this study project we trained and evaluated classifier using 10-fold cross validation in the multi-label learning setting, where each monitored trace was labeled with its corresponding website[11]. We used the simplest approach for verifying and comparing the all accuracy of the aforementioned classification algorithm by using cross-validation function from library, it called "$cross\_val\_score$" from $sklearn.model\_selection$[38]. It calculated the accuracy for all classification models.

The partition during the cross validation is done with all traces among folds. This means for example, each website we may split it's traces by the following procedure: 9 out of 10 crawlings for each sub-page belongs to training sets and 1 left for testing set. However, in this study project, we have total randomly shuffled 3420 traces after

removing outliers for training and 380 for testing set. This partitioning answers for a scientific question how website can be defined the main page though the browsing is based on combination of several sub-pages.

As shown in Figure 17, the following code demonstrates how to assess the accuracy of the classification algorithm. The function *cross_val_score* uses 4 parameters, fist parameter is classifier model, second parameter is “X” which is feature vector, third parameter is “y” which contains URL results for feature vectors, fourth is the k-fold number. Cross validation computes the score for each iteration. So it splits the data, fits a model and computes 10 accuracy scores consecutively.

```
50 def cross_val(clf, X, y):
51     scores1 = cross_val_score(clf, X, y, cv=10)
52     print(scores1)
53     print("Accuracy of K-fold: %0.2f (+/- %0.2f)" % (scores1.mean(), scores1.std() * 2))
```

Figure 17: The python code for cross validation

As shown on the Figure 18, k-folds cross-validator splits the data into train and testing sets without default shuffling. However, we need a shuffling in order to get balance representative of websites traces in every fold such that each fold could acquire maximum accuracy.

As shown in Figure 18 in line 153, the library function KFold uses the parameter called n_splits and splits dataset into 10 consecutive folds.

In line number 157 and 158, each fold is then used once as a validation which is used the data set of variable called X_test and y_test while the 9 (i.e k – 1) remaining folds form the training set which is used the data set variable called X_train and y_train.

```
153 kf = KFold(n_splits=10)
154 index = 1
155 scores = []
156 for train_index, test_index in kf.split(X):
157     X_train, X_test = np.array(X)[train_index.astype(int)], np.array(X)[test_index.astype(int)]
158     y_train, y_test = np.array(y)[train_index.astype(int)], np.array(y)[test_index.astype(int)]
```

Figure 18: The python code for KFold using K-Folds cross-validation library

5.2 Support Vector Machine

SVM can be used for both regression and classification tasks. In our case it will be used in classification purposes. The objective of the support vector machine algorithm [30] is to find a *"hyper plane"* in an N-dimensional space, that distinctly classify the data points. The dimension of hyperplane depends from the number of features which

is in our case 54, such that the hyperplane becomes 53-dimensional plane.

"Support vectors" [40] are data points which are closest to the decision surface "hyper plane". The position and orientation of hyper plane is influenced by the support vectors. By deleting the support vectors, the position of hyper plane can be changed. These support vectors are points which used to build SVM model.

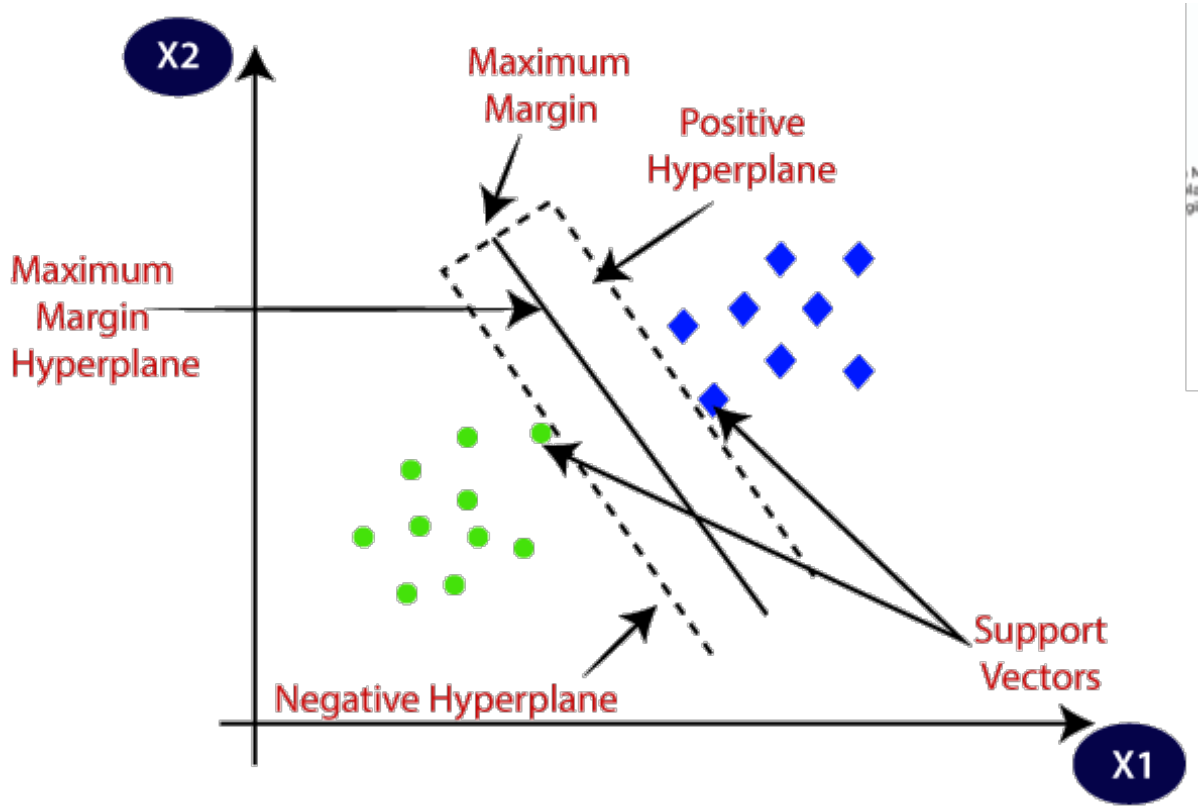


Figure 19: SVM explanation[7]

Figure 19 shows the splitting of two different classes by the SVM algorithm. SVM builds the hyperplane with the highest possible margin from the support vectors, such that support vectors are closest distants from the plane.

For implementing the SVM classifier the simplest approach is to use library function from sklearn.svm[4,5], it called “SVC” module. As shown in Figure 20, our kernel function parameter is “linear”[27], by default the regularization parameter "C" is equal to 1.0. "C" is a valuation of "how badly" we want to classify our data[27]. For the study project, the SVM is using the logic, which is mentioned in classification Algorithm 1:

Algorithm 1: Support Vector Machine

1. Open " $joined\_df.csv$" For Input as Input Files in Read Mode
2. Divide the Data into 2 parts for Features(X) and Label(y)
3. Shuffle the data Sets using Function $unison\_shuffled\_copies(X,y)$
4. Call the Function svm()
5. Call the Function run_algo() (As shown in Figure 20, Line 212)
6. Loop: If(KFold!= 10){
7. Split Data in Train and Test Sets
8. Save the Trained Model as $KFold\_svm\_model.sav$
9. Increment KFold and Goto Loop
10. }
11. Load the Saved Trained Model and Test the Accuracy Score of Test Data Sets

```
210  def svm(X, y, algo_name):
211      clf = SVC(kernel='linear')
212      run_algo(clf, X, y, algo_name)
```

Figure 20: The python code for SVM using SVC library

5.3 Random Forest

Random forest is one among of the classification algorithm used to implement machine learning techniques. The parameters which are used to implement the random forest classifiers is total number of decision trees and which have minimum split and support the split criteria. Random forest classifier initially creates some set of decision trees from subset of training set.

As shown on the Figure 21, each particular tree in the random forest give a class prediction, then all similar prediction values are summing and the class with the highest majority of votes becomes our model's prediction. By which we can decide the final class of test object.

The simplest approach of implementation the random forest classifier is to import "$RandomForestClassifier$" class of the sklearn.ensemble [26] to solve classification problems. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset, and uses averaging to improve the predictive accuracy and control over-fitting[37]. As shown in Figure 22, our max_depth parameter is 100 and this parameter defines the maximum depth of trees in the random forest[37]. Before choosing 100 we have tried with 50, 60, 70, 90, 110. We found under fitting if the depth was less than 100 and over fitting above 100. So, 100 was the optimal parameter value, for this study project. Again, we must know about the parameter,

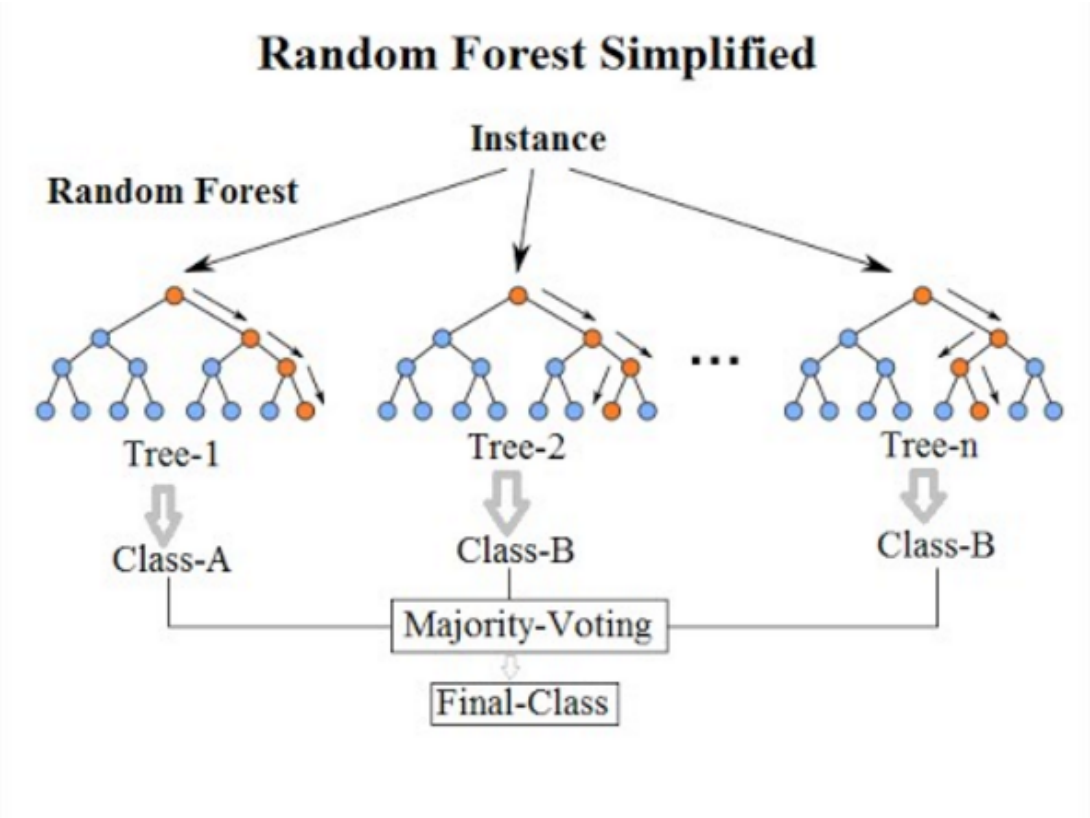


Figure 21: Random Forest algorithm[6]

random_state, which controls both the randomness of the bootstrapping of the training data set's samples used for building trees (bootstrap=True, by default) and the sampling of the features to consider when looking for the best split at each node[32]. Generally, when training a model, the random_state can set to "0" which means the results will be the same each time we run the split for reproducible results[32]. For the study project, the random forest is using the logic, which is mentioned in classification Algorithm 2:

Algorithm 2: Random Forest

1. Open " $joined\_df.csv$" For Input as Input Files in Read Mode
2. Divide the Data into 2 parts for Features(X) and Label(y)
3. Shuffle the data Sets using Function $unison\_shuffled\_copies(X, y)$
4. Call the Function $random\_forest()$
5. Call the Function $run\_algo()$ (As shown in Figure 22, Line 48)
6. Loop: If(KFold!= 10){
7. Split Data in Train and Test Sets
8. Save the Trained Model as $KFold\_random\_forest\_model.sav$
9. Increment KFold and Goto Loop
10. }
11. Load the Saved Trained Model and Test the Accuracy Score of Test Data Sets

```
46  def random_forest(X, y, algo_name):
47      clf = RandomForestClassifier(max_depth=100, random_state=0)
48      run_algo(clf, X, y, algo_name)
```

Figure 22: Random forest classifier using library function

5.4 K-Nearest Neighbor

K-nearest neighbors a classification algorithm, which stores all the available data points based on similarity measure. In our case data points is a final cumulative sum of the packets sizes collected during browsing of each website. Hence it is mainly used for pattern recognition and measuring the statistical data. Based on the distance function it chooses the nearest neighbors. The value k should be chosen carefully, if the k value is "*too small*": it is highly sensitive for outliers, and if it is "*too large*": the objects of the other classes will be in the decision set. So, the k value should be medium.

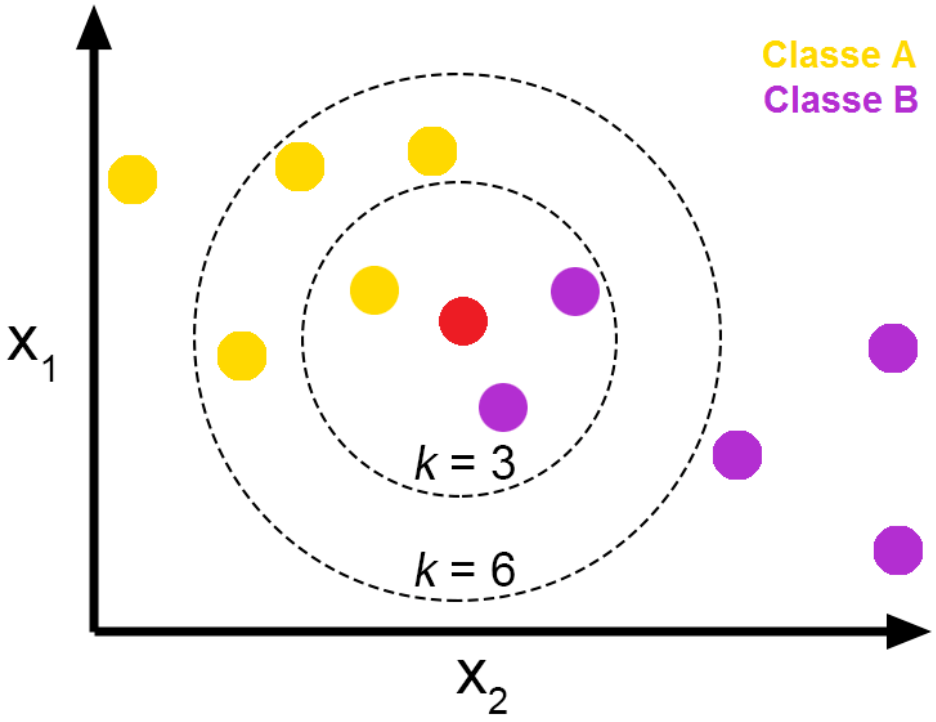


Figure 23: Knn algorithm[15]

As shown on the Figure 23, we can see the importance of a proper k-value selection. The new red point needs to be defined among two classes: yellow and purple. If we chose k as 3 a new red point will be defined as purple, otherwise will belong to yellow class.

The algorithm for the k-nearest neighbor classifier is among the simplest of all machine learning algorithms [31]. The optimal choice of the value k is highly data-dependent and classification can be computed by a majority vote of the nearest neighbors of the

unknown sample[31]. Now mathematically compared to other classification methods, the k-nearest neighbor classifier works directly on the learned of the training samples, closest in distance to the new point without creating rules. The most common choice for distance metric measure is a standard Euclidean distance. So, implementing the k-nearest neighbor classifier is very simple by importing “KNeighborsClassifier” class of the sklearn.neighbors for classification of data sets.

As shown in Figure 24, our n_neighbors parameter is “5” which defines the minimum number of neighbors needed to classify a new data. Again, we must know about the parameter, weights and metric. Parameter weights is used in prediction and in this study project, we used the weights values: ‘uniform’ (by default) which means all points in each neighborhood are weighted equally. Nonetheless, parameter metric is also needed to be tuned. In our study project, the default metric is ‘minkowski’ distance metric which is equivalent to the standard Euclidean metric. For the study project, the k-nearest neighbor is using the logic, which is mentioned in classification Algorithm 3:

Algorithm 3: K-Nearest Neighbour

1. Open "$joined\_df.csv$ " For Input as Input Files in Read Mode
2. Divide the Data into 2 parts for Features(X) and Label(y)
3. Shuffle the data sets using Function $unison\_shuffled\_copies(X, y)$
4. Call the Function $k\_nearest\_neighbor()$
5. Call the Function $run\_algo()$ (As shown in Figure 24, Line 216)
6. Loop: if (KFold!= 10) {
7. Split data in Train and Test Sets
8. Save the Trained Model as $KFold\_knn\_model.sav$ as Output Files
9. Increment KFold and Goto Loop
10. }
11. Load the Saved Trained Model and Test the Accuracy Score of Test Data

```
214  def k_nearest_neighbor(X, y, algo_name):
215      clf = KNeighborsClassifier(n_neighbors=5)
216      run_algo(clf, X, y, algo_name)
```

Figure 24: K-nearest neighbor using library function

5.5 Neural Network

The artificial neural networks is a machine learning model, inspired by the networks of biological neurons found in our brains . Here, we use the neural networks to classify websites. All the artificial neurons are interconnected and data travels through neurons from the input layer to the output layer. As in Figure 25 shown a simple neural network

with five inputs, five outputs, and 2 hidden layers of neurons (first hidden layer of 2 blue neurons and second hidden layer of 5 magenta neurons).

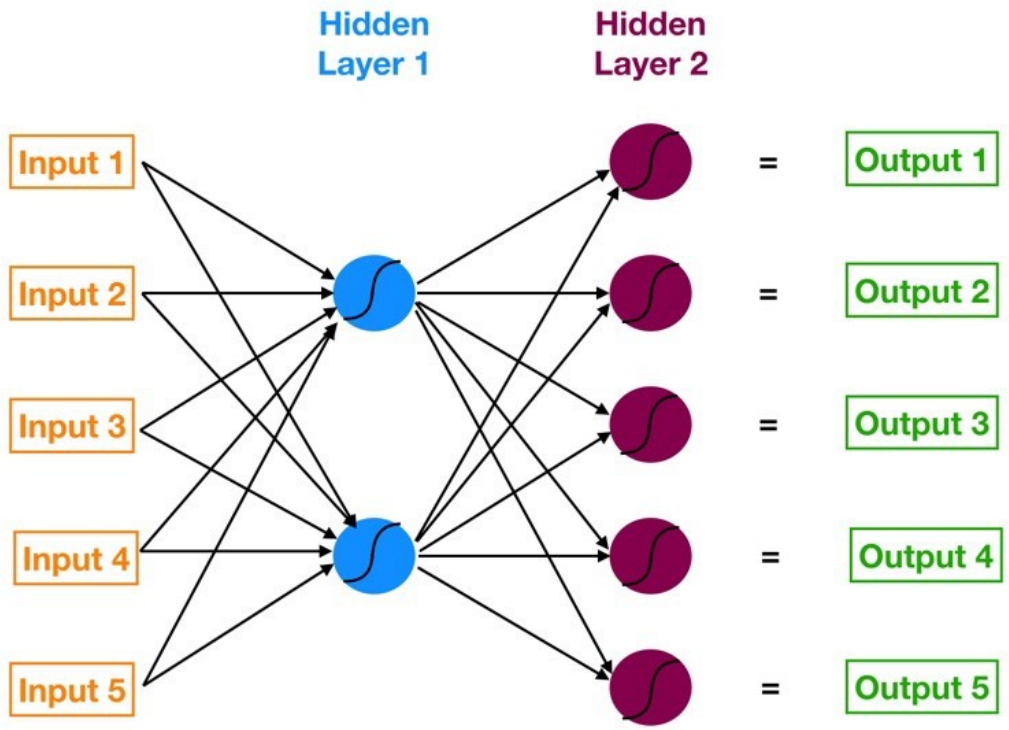


Figure 25: Neural network with 2 hidden layers

As shown in Figure 26, a fundamental building block of neural network consists of just the following components:

- The multiple connections, each with its own weight (W), going into a particular neuron, with a weight transforms the input (e.g. X) and gives it to the neuron.
- With forward propagation of a neuron that includes a bias term (e.g.B_0) and calculating,

$$[Z] = [W] * [X] + [Bias] \tag{4}$$

  and applying the activation function (e.g sigmoid) for each successive layer.
- The cost function (e.g. Mean Squared Error) is an approximation of how erroneous the target outcome is relative to the predictions results.
- To minimize a cost function with back propagation (e.g. gradient descent function)

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns by training on a dataset, given a set of input features:

$$X = x_1, x_2, ..., x_m$$

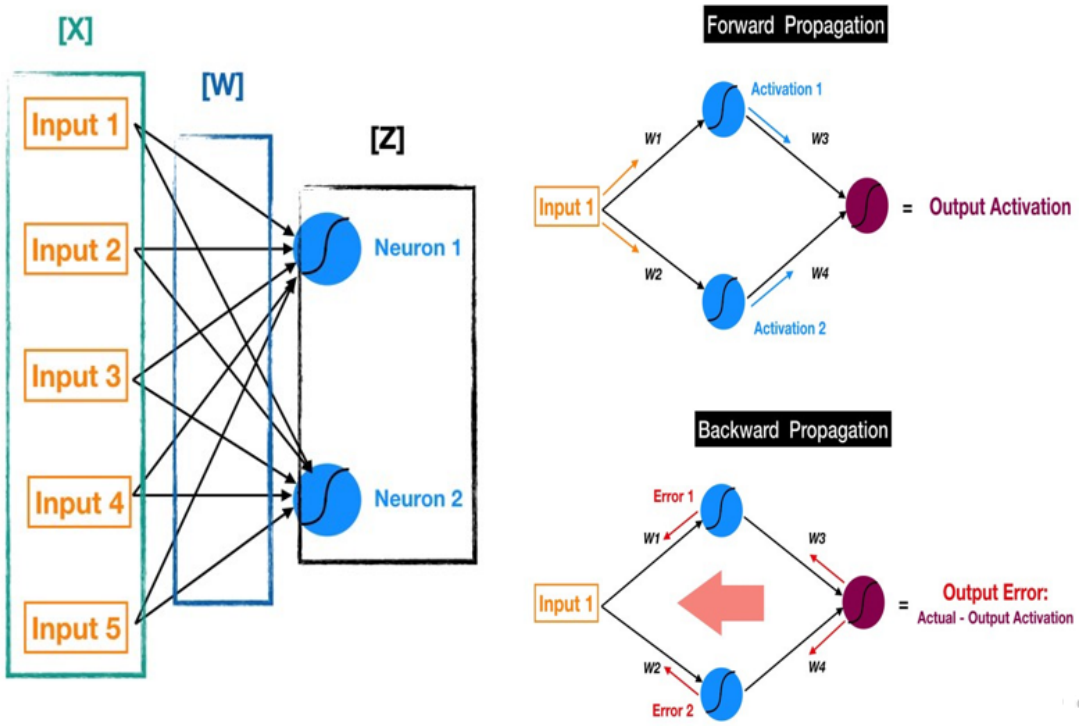


Figure 26: Connections between input and hidden layer to visualize [*W*], [*X*], and [*Z*]

known as the input layer, consists of a set of neurons. And a target y values (class labels), the output layer receives the values from the last hidden layer and transforms them into output values[29].

Mathematical representation is: a function *f(.):* $R^m \rightarrow R^o$, where,*'m'* is the number of dimensions for input and *'o'* is the number of dimensions for output.

So, by importing "MLPClassifier" class of the sklearn.neural_network the implementation of the MPL classifier is very simple and easy[31] for classification of data sets. Finally, to build the Multilayer Perceptron classifier, as shown in Figure 27, we have to tune some of the parameters which are as follows [23][31]:

- hidden_layer_sizes : this parameter allows us to set the number of layers and the number of nodes we wish to have in the Neural Network Classifier. In our study project, we used 3 hidden layers in the network represents the number of nodes 1000, 500, 300 consecutively.
- max_iter: it denotes the number of epochs. In our study project, we used 30 iterations with 30 epochs, the MLP model is passed through the whole dataset 30 times.
- activation: The activation function for the hidden layers. We have selected 'tanh', the hyperbolic tan function, returns f(x) = tanh(x) because it gave us better results against other (such as 'identity', 'logistic' and 'relu') activation function.

- solver: this parameter specifies the algorithm for weight optimization across the nodes as cost function. We used here 'sgd' refers to stochastic gradient descent rather than default one called 'adam'.
- alpha: for regularization term which helps in avoiding overfitting by penalizing weights with large magnitudes. We used here the default=0.0001 as L2 penalty (regularization term) parameter.
- batch_size: size of minibatches for stochastic optimizers. Here, we set to batch_size=32, which is a popular batch sizes in the case of minibatch gradient descent.
- learning_rate_init: the initial learning rate used. It controls the step-size in updating the weights. Here we used the default learning_rate_init =0.001 to support the solver='sgd'.

For the study project, the Neural Network is using the logic, which is mentioned in classification Algorithm 4:

Algorithm 4: Neural Network

1. Open " $cnn\_dataframe\_29021.csv$" For Input as Input Files in Read Mode
2. Divide the Data into 2 parts for Features(X) and Label(y)
3. Shuffle the data Sets using Function $unison\_shuffled\_copies(X,y)$
4. Call the Function $neural\_network()$
5. Call the Function $run\_algo()$ (As shown in Figure 27, Line 39)
6. Loop: If(KFold!= 10){
7. Split Data in Train and Test Sets
8. Save the Trained Model as $KFold\_neural\_network\_model.sav$
9. Increment KFold and Goto Loop
10. }
11. Load the Saved Trained Model and Test the Accuracy Score of Test Data Sets

```
35  def neural_network(X, y, algo_name):
36      clf = MLPClassifier(activation='tanh', solver='sgd', alpha=0.0001,
37          hidden_layer_sizes=(1000, 500, 300), batch_size=32,
38          learning_rate_init=0.001, max_iter=30)
39      run_algo(clf, X, y, algo_name)
```

Figure 27: The python code for neural network using MLPClassifier library

6 Evaluation of Classifier Algorithms

This is the final step of our study projects. Here, we have evaluated our results in the form of tabular and graphical representation. The results are evaluated using the following methods:

- We have used tabular representation to visualize of accuracy for all the classifier algorithm in respect of 10-Fold cross validation
- We represent our data in the bar graph to measure the high accuracy against the classifier algorithm
- The evaluation from the line graph made us to understand the deviation between each point from the accuracy of each fold.
- The confusion matrix is evaluated the rates of True positive and False Positive in both with and without normalized format
- We evaluated accuracy with confidence intervals for all the classifier algorithm
- We developed a bar graph for comparing between how easily or difficult to finger-print of websites across our all classifiers algorithm.

6.1 Results

In Table 1, we have the results of all classifier algorithm for k-Fold cross validation. After getting the accuracy from all classifier we have calculated the average accuracy as well as the variance among the 10-Fold cross validation. The important observation we have figure out that is for “random forest” classifier we achieved highest accuracy of 91.94% whereas the “neural network” classifier provide us the worst result of 43.0% (average). It might be happened that the parameter tuning or the data preparation for the “neural network” was not properly worked for us. Nonetheless, for the “neural network” may need bigger training data sets than our smaller trained data sets.

K-Fold Cross Validation	Random Forest	KNN	SVM	Neural Network
K-Fold 1	0.9032	0.7043	0.5887	0.4328
K-Fold 2	0.8952	0.7124	0.6237	0.4195
K-Fold 3	0.9005	0.7339	**0.6371**	0.4403
K-Fold 4	0.9086	0.7177	0.5699	0.4235
K-Fold 5	**0.9194**	0.7527	0.5780	0.4333
K-Fold 6	0.8952	0.7554	0.5081	0.4313
K-Fold 7	0.8925	0.7527	0.5672	0.4249
K-Fold 8	0.9113	0.7446	0.6210	0.4317
K-Fold 9	0.9003	0.7439	0.5957	0.4347
K-Fold 10	0.9030	**0.7574**	0.6092	**0.4415**
Average accuracy:	0.90	0.74	0.59	0.43
Variance	(+/- 0.02)	(+/- 0.04)	(+/- 0.07)	(+/- 0.02)

Table 1: Comparison of machine learning algorithms with 10-Fold cross validation

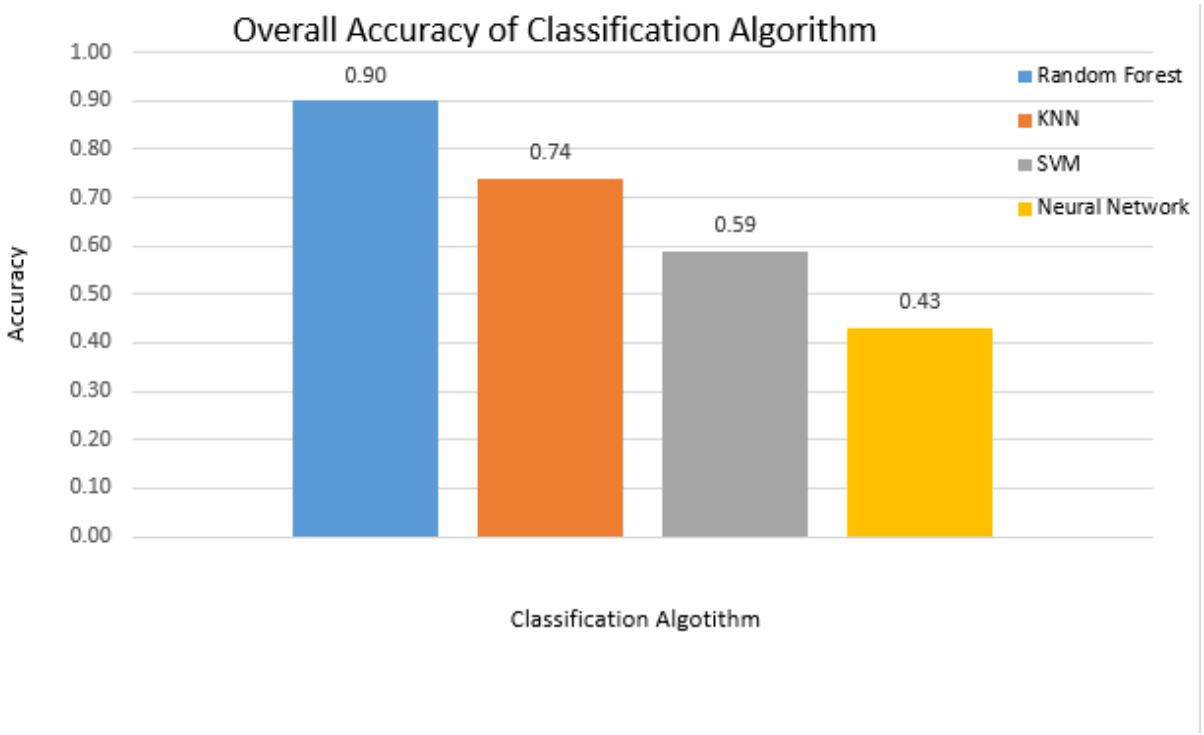


Figure 28: Classification accuracy for each machine learning algorithm

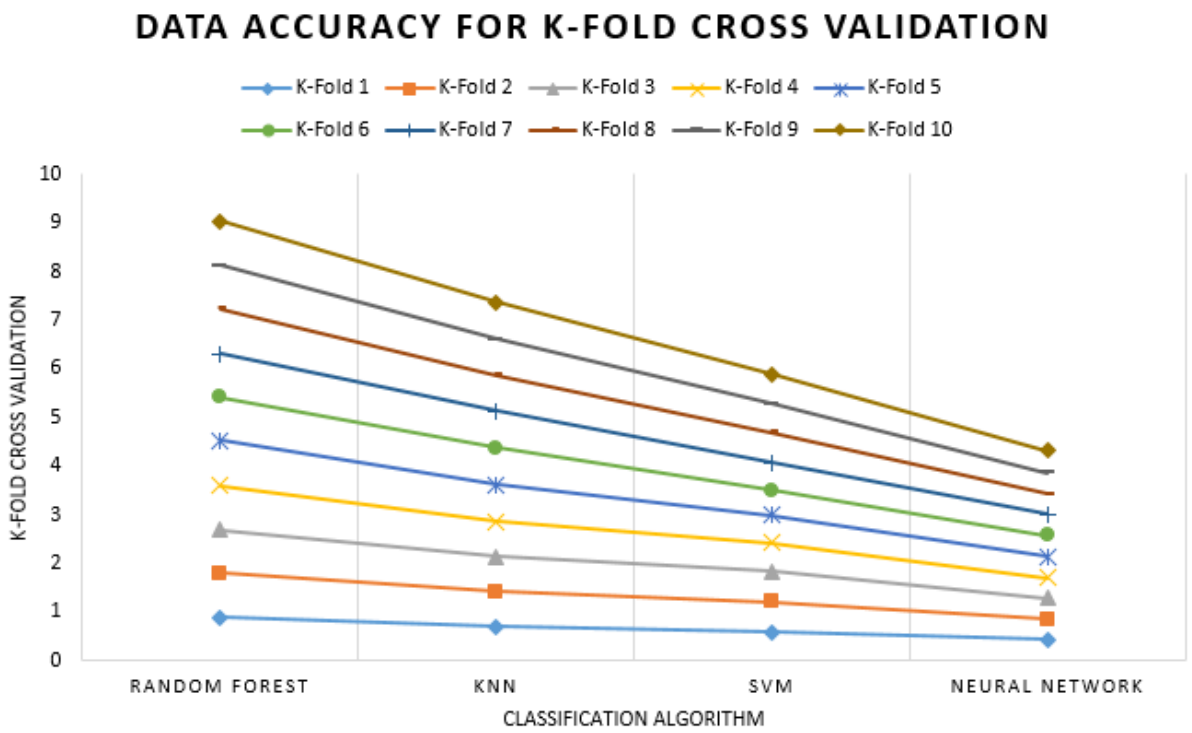


Figure 29: Classification accuracy of 10 – Fold cross validation for each classifiers

Figure 28 shows the overall classification accuracy for each machine learning algorithm, in which we can evaluate the average accuracy of the classifiers at a glance. Using the random forest classifier, we have achieved the best accuracy and it was high-speed classifier as well.

However, the neural network was provide us worst average results with compare to other classifiers and it was very slow to train the model during the training for us. Maybe the problem with tuning the number of hidden layer, number of perceptron's, activation function or cost function for back propagation.

Using the random forest and 10-Fold cross validation was deemed the best performance and it was evaluated the sharp decrease of accuracy in every consecutive fold for each machine learning algorithm. Figure 29 shows the classification of data set accuracy of 10 – Fold cross validation for each classifiers.

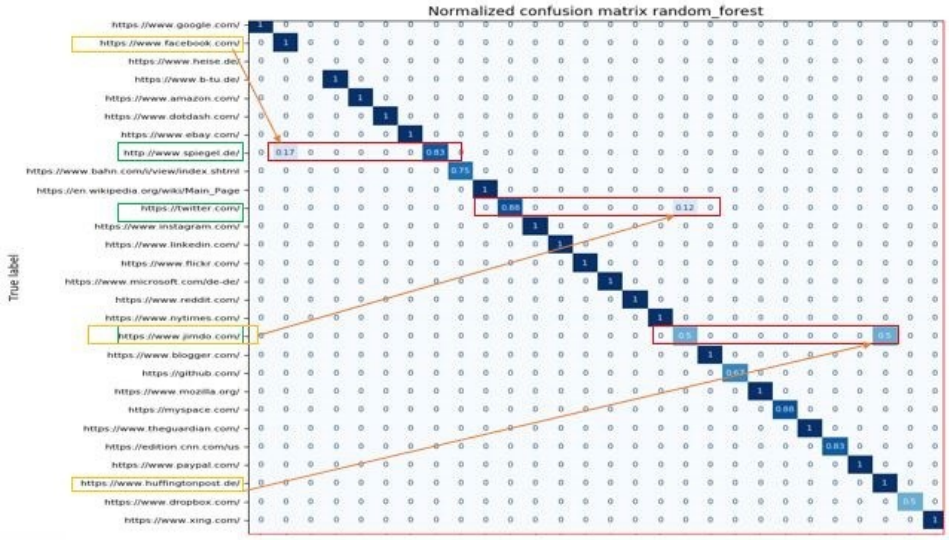


Figure 30: Confusion matrix for the random forest classifier with normalization

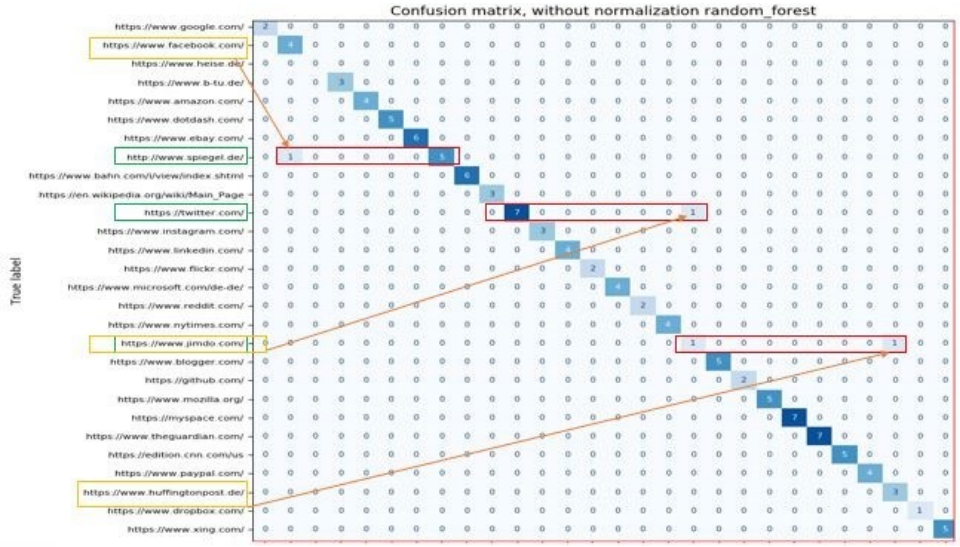


Figure 31: Confusion matrix for the random forest classifier without normalization

Figure 30 and 31 are shown, the confusion matrix for the random forest classifier with normalization, numeric data as predicted labels and without normalization, fractional data as predicted labels with confusion matrix respectively. Confusion matrix without normalization does not provide comparability of the traces, so we normalized confusion matrix from non-normalized on Figure 31 to normalized on Figure 30. We have used confusion matrix to visualize and compare the quality of the results of all classifier algorithm. The column showed the predicted results whereas the rows showed the actual outputs. The diagonal elements represent True Positive i.e. correctly classify all data for specific webpages URL for which the true label and predicted label is equal with each other. Again the mislabeled URLs, i.e. False Negative, URL is actually positive but the classifier recognize it as negative, have shown as off-diagonal elements.

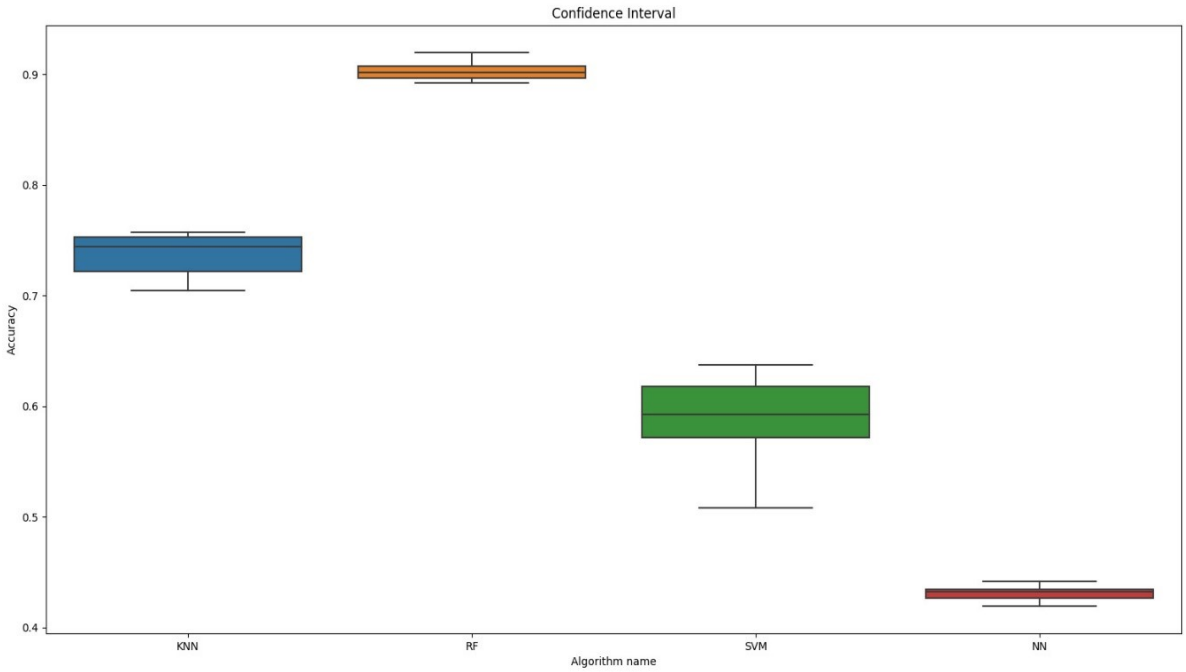


Figure 32: Confidence interval for the classifier algorithm

Figure 32 shows the confidence interval for all the classifier and the values at 95% confidence are shown in Table 2. The interval of small box plot in Figure 32 is deemed to plot the inter quartile range which shows the 95% confidence for the mean value. We have interpreted the confidence interval that we were 95% confident that the mean accuracy of all the samples data sets of URLs in between upper bound and lower bound in given Table 2 for all classification algorithm.

	Random Forest	**KNN**	**SVM**	**Neural Network**
Average	0.9029	0.7375	0.5898	0.4313
Standard Deviation	0.0083	0.0195	0.0373	0.0070
Sample Size	10.0000	10.0000	10.0000	10.0000
Confidence Coefficient	1.9600	1.9600	1.9600	1.9600
Margin of Error	0.0051	0.0121	0.0231	0.0044
Upper Bound	0.9080	0.7496	0.6129	0.4357
Lower Bound	0.8978	0.7254	0.5667	0.4270
Max	0.9194	0.7574	0.6371	0.4415
Min	0.8925	0.7043	0.5081	0.4195
Range	0.0269	0.0531	0.1290	0.0220

Table 2: Confidence interval calculation at 95% confidence

The different classification algorithm were tested for evaluating how correctly the classifier can recognized the URLs. Figure 33-36 are shown in Appendix A, the comparison of detecting the URLs between easy or hard across all classifiers algorithm. It is deemed from the Figure 36 that the random forest detect all URLs very smoothly, then there is a little fluctuation for KNN and the fluctuation is higher for SVM than KNN, are shown in Figure 33-34. Last but not the least the neural network, in Figure 35, could not classify the URLs for most of the cases. So we got a ranking of classifiers by accuracy classification of all the URLs.

7 Conclusion

In conclusion our project showed that website fingerprinting attack has a real threat for anonymity in topology with proxy server. Security TLS protocol which was implemented in stunnel does not protect from website fingerprinting attack. Possible attacker can potentially perform such attack between client and proxy server, and get good results. Encryption does not protect from traffic analysis and packet's headers inspection. So, it became possible to successfully analyze the website traffic by observing the pattern flow of incoming and outgoing packet.

For the future research we recommending to try website fingerprinting attack on larger dataset, since neural network showed an unexpectedly low results. In our point of view, the perfect size of dataset for neural network should have more crawlings. Also it is possible to tune neural network with other parameters which uses more computation power.

Also, we noticed that number of outliers for each website generally more then five percent. In future maybe we need to clean our dataset more precisely and gather new data for replacing of outliers. In total, this measure should increase the accuracy of machine learning classifiers.

Since our project showed an anonymity vulnerability during connection to a proxy server which uses TLS tunneling, the appropriate security measures should be taken by the client. System administrator should be aware of the possibility of being successfully attacked and use different countermeasures techniques at different network layers.

8 Appendix A

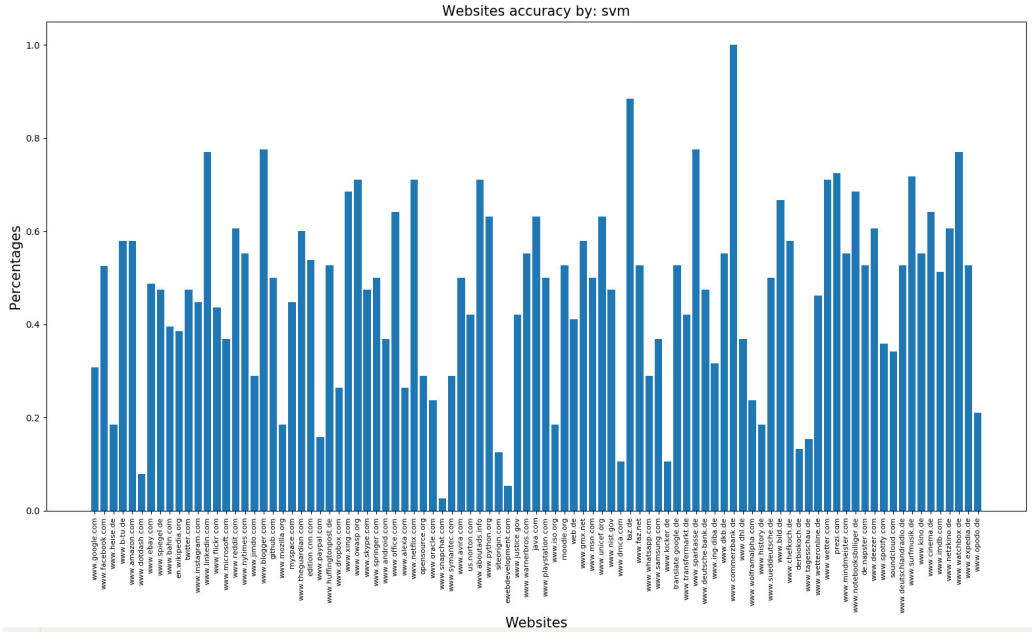


Figure 33: Accuracy Results for SVM

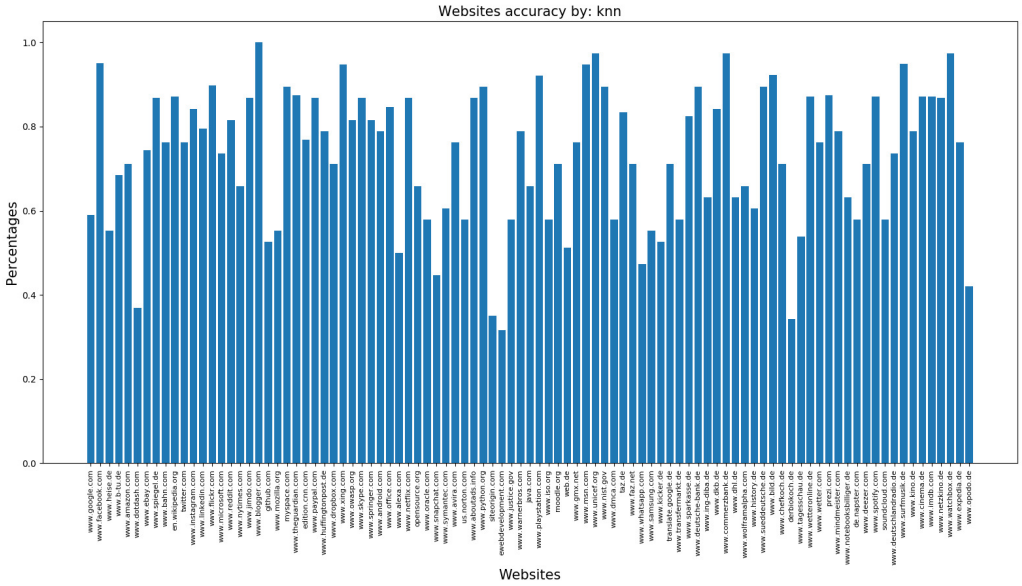


Figure 34: Accuracy result for KNN

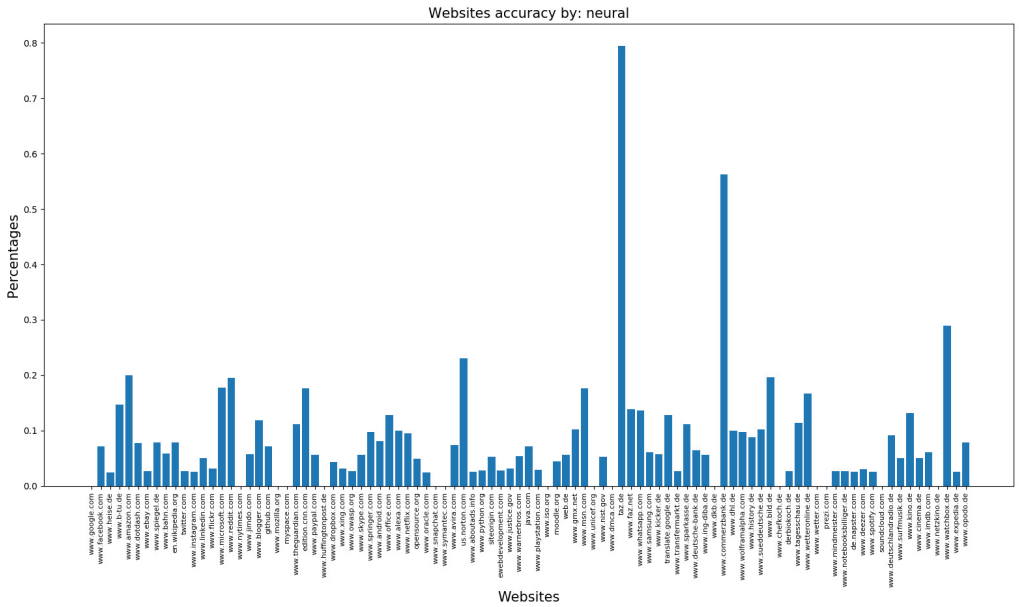


Figure 35: Accuracy result for NN

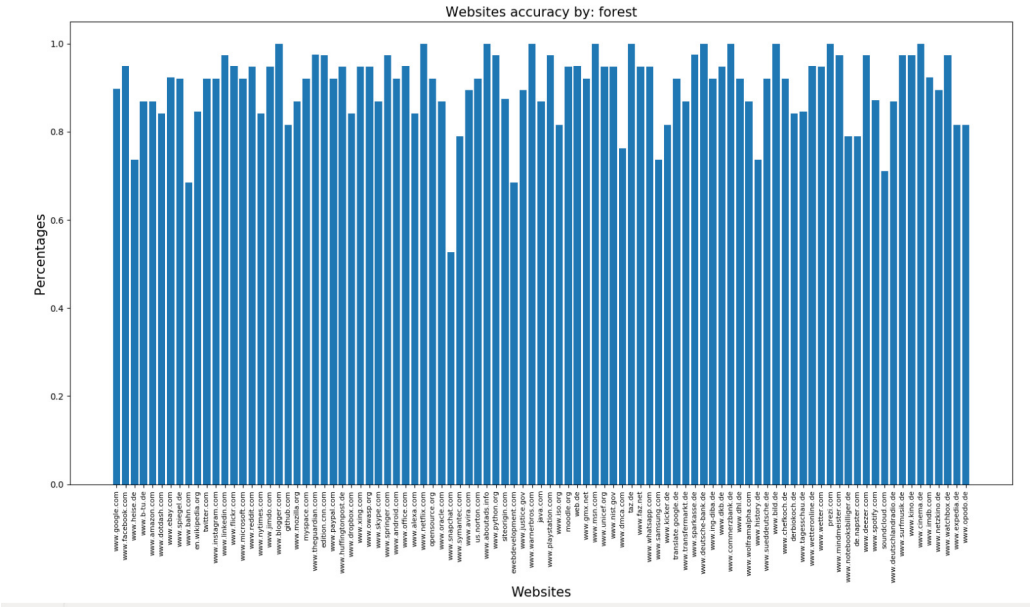


Figure 36: Accuracy result for RandomForest

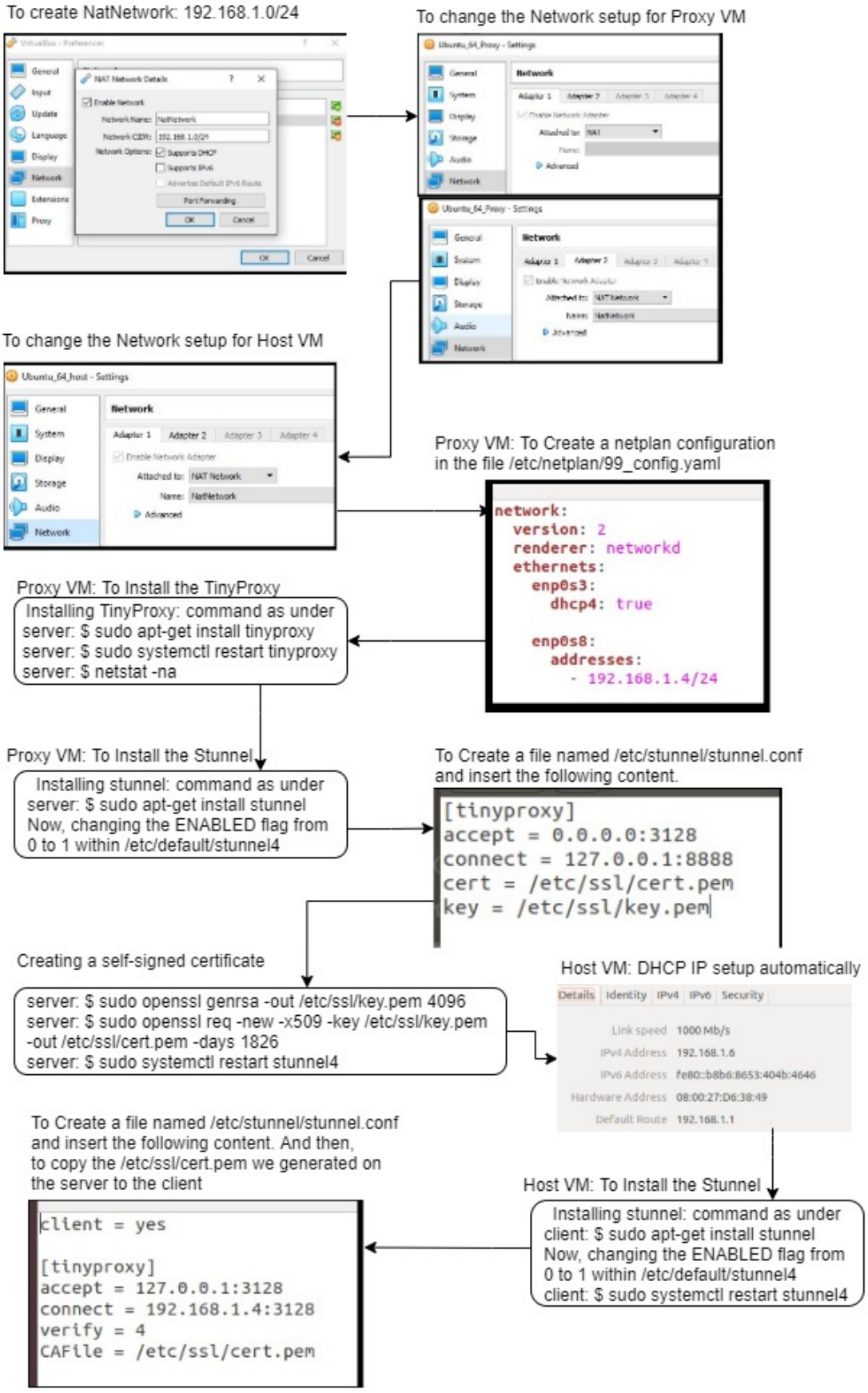


Figure 37: Network configuration of two virtual boxes (client, proxy server)

List of the URLs using for collection of Traces

https://www.google.com/
https://www.facebook.com/
https://www.heise.de/
https://www.b-tu.de/
https://www.amazon.com/
https://www.dotdash.com/
https://www.ebay.com/
http://www.spiegel.de/
https://www.bahn.com/i/view/index.shtml
https://en.wikipedia.org/wiki/Main_Page
https://twitter.com/
https://www.instagram.com/
https://www.linkedin.com/
https://www.flickr.com/
https://www.microsoft.com/de-de/
https://www.reddit.com/
https://www.nytimes.com/
https://www.jimdo.com/
https://www.blogger.com/
https://github.com/
https://www.mozilla.org/
https://myspace.com/
https://www.theguardian.com/
https://edition.cnn.com/us
https://www.paypal.com/
https://www.huffingtonpost.de/
https://www.dropbox.com/
https://www.xing.com/
https://www.owasp.org/index.php/Main_Page
https://www.skype.com/
https://www.springer.com/
https://www.oracle.com/index.html
https://www.debian.org/
https://www.android.com/
https://www.office.com/
https://www.alexa.com/
https://www.netflix.com/
https://opensource.org/
https://www.icann.org/
https://www.oracle.com/sun/index.html
https://www.snapchat.com/
https://www.symantec.com/
https://www.avira.com/
https://us.norton.com/
http://www.aboutads.info/
https://www.microsoft.com/
https://www.python.org/
https://siteorigin.com/
http://ewebdevelopment.com/
https://www.justice.gov/
https://www.warnerbros.com/
https://java.com/
https://www.playstation.com/
https://www.iso.org/home.html
https://moodle.org/
https://web.de/
https://www.gmx.net/
https://www.msn.com/
https://www.unicef.org/
https://www.nist.gov/
https://www.dmca.com/
http://taz.de/
http://www.faz.net/aktuell/
https://www.whatsapp.com/
https://www.samsung.com/us/
http://www.kicker.de/
https://translate.google.de/
https://www.transfermarkt.de/
https://www.sparkasse.de/
https://www.deutsche-bank.de/
https://www.ing-diba.de/
https://www.dkb.de/
https://www.commerzbank.de/
https://www.dhl.de/
https://www.wolframalpha.com/
https://www.history.de/
https://www.sueddeutsche.de/
https://www.bild.de/
https://www.chefkoch.de/
http://derbiokoch.de/
https://www.tagesschau.de/
https://www.wetteronline.de/
https://www.wetter.com/
https://prezi.com/
https://www.mindmeister.com/
https://www.alternate.de/
https://www.notebooksbilliger.de/
https://de.napster.com/
https://www.deezer.com/de/
https://www.spotify.com/
https://soundcloud.com/
https://www.deutschlandradio.de/
http://www.surfmusik.de/
https://www.kino.de/
https://www.cinema.de/
https://www.imdb.com/
http://www.netzkino.de/
https://www.watchbox.de/
https://www.expedia.de/
https://www.opodo.de/

References

[1] Browser automation defined. *URL:http://www.biodesignautomation.org/browser-automation-defined.html*.

[2] Install scapy. *URL:https://scapy.readthedocs.io/en/latest/installation.html*.

[3] Install selenium. *URL:https://www.srcmake.com/home/selenium-python-chromedriver-ubuntu*.

[4] Install stunnel and tinyproxy. *URL:https://bencane.com/2017/04/15/using-stunnel-and-tinyproxy-to-hide-http-traffic/*.

[5] Inter quartile rangeoutlier detection. *URL:https://wissenschafts-thurm.de/grundlagen-der-statistik-dispersionsparameter-spannweite-und-interquartilsabstand/*.

[6] Random forest simple explanation. *URL: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d*.

[7] Support vector machine algorithm. *URL: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm*.

[8] Tcpdump tool. *URL:https://www.tcpdump.org/*.

[9] Z-score table. *URL:http://www.z-table.com/*.

[10] The efficient randomness testing using boolean functions. *URL: https://www.researchgate.net/publication/318873062*$_{T}he_{E}fficient_{R}andomness_{T}esting_{u}sing_{B}o$

[11] Fingerprinting keywords in search queries over tor. *URL: https://www-users.cs.umn.edu/ hoppernj/fp*$_{k}w_{p}opets17.pd,$ 2017.

[12] Understanding boxplots. *URL: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51*, 2018.

[13] Unsupervised anomaly detection with isolation forest. *URL: https://www.slideshare.net/PyData/unsupervised-anomaly-detection-with-isolation-forest-elena-sharova*, 2018.

[14] Definition of the standard normal distribution. *URL: http://www.ltcconline.net/greenl/courses/201/probdist/zScore.htm*, 2019.

[15] Knn. *URL: https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d*, 2019.

[16] Z-score outlier detection. *URL:https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/z-score/*, 2019.

[17] google. *URL:https://support.google.com/chrome/answer/95346?co=GENIE.Platform*, 2.10.2019.

[18] Ubuntu linux. *URL:https://ubuntu.com/download/desktop*, 2.10.2019.

[19] Browser automation. *URL:https://realpython.com/modern-web-automation-with-python-and-selenium/*, 2.3.2020.

[20] Wireshark · go deep. *URL: https://www.wireshark.org/*, 2.5.2019.

[21] Download scipy.org. *URL: https://scapy.net/*, Accesed 2019.10.19.

[22] Matplotlib: Visualization with python. *https://matplotlib.org/*, Accesed 2019.10.19.

[23] Josep Lluis Berral-García. A quick view on current techniques and machine learning algorithms for big data analytics. In *2016 18th international conference on transparent optical networks (ICTON)*, pages 1–4. IEEE, 2016.

[24] Jason Brownlee. A gentle introduction to k-fold cross-validation. *Accessed October*, 7:2018, 2018.

[25] Benjamin Cane. Using stunnel and tiny proxy to obfuscate http traffic. *URL: https://bencane.com/2017/04/15/using-stunnel-and-tinyproxy-to-hide-http-traffic*, 2017.

[26] Random Forest Classifier. Electronic resourse. *URL: http://scikit-learn.org/stable/modules/generated/sklearn. ensemble. RandomForestClassifier. html sklearn. ensemble. RandomForestClassifier*.

[27] Gianluca Corrado. Scikit learn: Machine learning in python.

[28] Hawkins Douglas. *Identification of Outliers*. Chapman and Hall, 1980.

[29] Olga Fuks. Classification of news dataset. *Standford University*, 2018.

[30] Rohith Gandhi. Support vector machine—introduction to machine learning algorithms. *Towards Data Science*, 2018.

[31] Norhidayu Abdul Hamid and Nilam Nur Amir Sjarif. Handwritten recognition using svm, knn and neural network. *arXiv preprint arXiv:1702.00723*, 2017.

[32] Will Koehrsen. Hyperparameter tuning the random forest in python. *Towards Data Science*, 2018.

[33] T. Liu. Isolation forest. *Eighth IEEE International Conference*, 2008.

[34] John McCarthy and Edward A Feigenbaum. In memoriam: Arthur samuel: Pioneer in machine learning. *AI Magazine*, 11(3):10–10, 1990.

[35] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[36] Dan Peleg. *Mastering Sublime Text*. Packt Publishing Ltd, 2013.

[37] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[38] SCIKIT-LEARN. Pdf documentation-scikit-learn, estimation,performance.

[39] Guido Van Rossum et al. Python programming language. In *USENIX annual technical conference*, volume 41, page 36, 2007.

[40] Dengsheng Zhang. Support vector machine. In *Fundamentals of Image Data Mining*, pages 179–205. Springer, 2019.