Politecnico di Torino

Patterns in encrypted web traffic HTTPS

Master's Degree in Cybersecurity - 2023/2024

Full Name	Student ID
Cornaggia Emanuele	s332073
Marrapodi Francesco	s332062
Sambataro Simone	s331812
Torrisi Alessandro Rosario	s330732

Professor: Luca Vassio

January

Contents

Chapter 1

Data Exploration

In this section, we analyzed the dataset to understand its characteristics. For better explainability we decided to divide this section into 3 main areas, each one of them dedicated to the 3 level that we had to analyze: flow, domain and IP. The full code is visible in the file *project_final.ipynb*.
Before doing any analysis, we performed some preprocessing to enhance the dataset visualization and a better:

- we checked for NaN values to exclude them (there where no NaN values found).
- dropped columns with standard deviation equal to zero because `std=0` means no variation and no information brought to the dataset (for exaple, since we are analyzing TCP traffic, the port will always be 443). We removed the columns: `_s_port`, `_s_pkts_unfs`, `_c_pkts_unfs`, `_c_sack_opt`, `_s_win_0`, `_s_syn_retx`, `_c_pkts_fc`, `_s_sack_opt`, `_c_syn_retx`,
- we also decided to remove the column `time`, since it will bring only overfitting.
- we also scaled data in order to perform PCA and to improve the performance of certain models and overall computation.

1.1 | Flow level analysis

The dataset has been provided grouped by flow (flow = protocol,ip address of the destination,source port, destination port, ip address of the source). Analyzing the dataset in this way gives us a wide view of the data, but with the risk of not being able to focus on particular aspect.

1.1.1 | Flow level: ECDF and EPDF visualization

ECDF (Empirical Cumulative Distribution Function) and EPDF (Empirical Probability Density Function) visualizations, which offer a comprehensive view of the flow-level data. While analyzing the entire dataset at the flow level provides a broad perspective, these visualizations allow us to uncover specific nuances and patterns within the data. The whole figures are present in the file where the whole file is contained, in this section and in the other ones, we will analyze some of the most significant graphics.
In this plot 1.1.1, we highlighted the EPDF and the ECDF of the `c_rtt_avg` feature; this feature tells us the average round-trip time (RTT) for packets sent by the client.This data set shows that the average round-trip time (c_rtt_avg) for a sample of network connections is 7.09 milliseconds. The median c_rtt_avg is 4.00 milliseconds, which is close to the 75th percentile (9.98 ms). This indicates that most of the connections in the sample had an average round-trip time of 4-9 ms. However, there is a long tail to the right of the distribution, which suggests that there are a few connections with very high c_rtt_avg values (up to 1125.39 ms). These outliers may be due to network congestion or other issues.Overall, the network performance is good as the average c_rtt_avg is relatively low.
Regarding _c_bytes_all (Figure 1.1.2), the average value is 5,060.9 KB, suggesting a moderate amount of data exchange on the client prospective. The distribution is right-skewed, indicating a few connections transferring a very large number of bytes, potentially causing congestion.

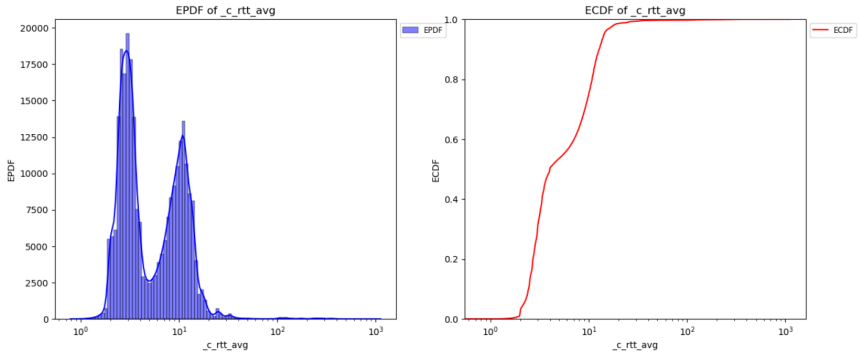


Figure 1.1.1: ECDF and EPDF for average for the RTT of the client

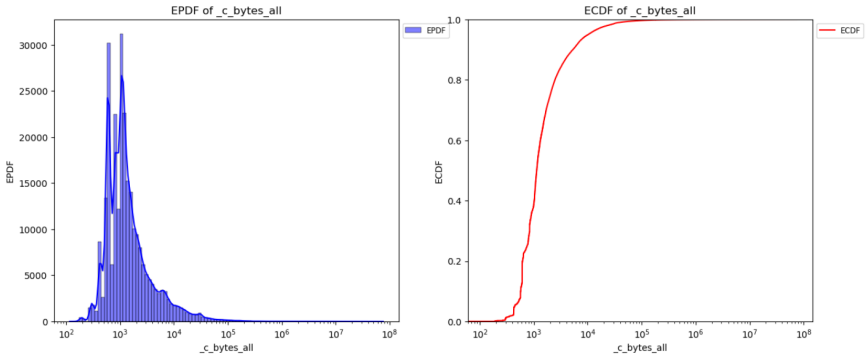


Figure 1.1.2: ECDF and EPDF for average for bytes sent by the client

There is a weak positive correlation between _c_bytes_all and c_rtt_avg. This means that connections that transferred more bytes tended to have slightly longer round-trip times. However, the correlation is weak, so it is not a reliable predictor of c_rtt_avg.

1.1.2 | Flow level: correlation analysis

These are the top three correlated features:

- Features: _c_pkts_data-_c_pktsize_count, Value: 1.0
- Features: _s_pkts_dup-_s_syn_cnt, Value: 0.99
- Features: _s_pkts_data-_s_seg_cnt, Value: 0.99
- Features: _s_ack_cnt-_s_pkts_all, Value: 0.99
- Features: _c_ack_cnt-_c_pkts_all, Value: 0.99

Going through this section we will notice that some feature will be correlated across the the 3 level, like the 2 feature _c_pkts_data and _c_pktsize_count: the first one refers to the number of packets sizes sent by the client, the other one represent the number of data packets sent by the client.The correlation between these two features can be attributed to the fact that when a client sends data packets, these packets will have various sizes. The size of a packet is determined by factors like the amount of data being sent, network protocols, and other related parameters. Therefore, as the number of data packets sent by the client increases, the count of different packet sizes (_c_pktsize_count) is likely to increase as well. It important to remind that correlation does not imply causality, and even if we have a large amount of data, these are just assumption. While the correlation between _s_ack_cnt and _s_pkts_all might be indicative of the reliable nature of the communication, where acknowledgments are consistently sent for each packet, ensuring that the data is successfully received

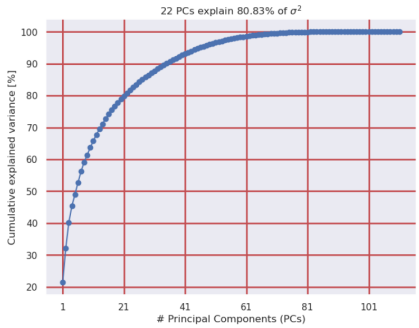
Statistic	_c_bytes_all	_c_rtt_avg
Count	2.62443×10^5	262443.000
Mean	5.060923×10^3	7.092755
Standard Deviation	1.890236×10^5	12.520561
Minimum	1.17×10^2	0.780184
25% (Q1)	7.52×10^2	2.850531
50% (Median)	1.133×10^3	4.004200
75% (Q3)	2.016×10^3	9.981834
Maximum	7.553852×10^7	1125.393487

Table 1.1.1: Descriptive Statistics for _c_bytes_all and _c_rtt_avg

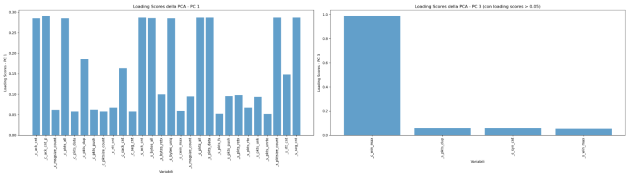
by the intended recipient. When server/client send a packet during a tcp conncetion they both wait for an ack response, and so these 2 feature depend from one another.

1.1.3 | Flow level: PCA and t-SNE

Principal component analysis (PCA) is a linear dimensionality reduction technique with applications in exploratory data analysis, visualization and data preprocessing . The data is linearly transformed onto a new coordinate system such that the directions (principal components) capturing the largest variation in the data can be easily identified (https://en.wikipedia.org/wiki/Principal_component_analysis). Before applying the PCA we standardized the to ensures that all variables contribute equally to the analysis. We found pca really useful because it allowed us to perform computation in a faster way, since we are reducing the number of rows, without loosing much of the information to reduce time of computation. For all the level we tried to find the minimum amount of principal components in order to arrive with at least 80% of the explained variance in order to avoid overfitting : for the flow level we found that the we need 22 PC (figure 1.1.3a). Throughout the notebook, we have demonstrated how each Principal Component (PC) is influenced by the individual contributions of the original variables (features), here we just put 2 of them (Figure 1.1.3).



(a) Flow- PCA



(b) Principal components for the flow

Figure 1.1.3: Caption for the combined figures

Applying t-SNE to the whole dataset we can see that the cluster are not well separeted and so looking at the dataset from this view we can just notice that majority of points fall into the same space , and all the clusters tent to overlap. This could be cause also by the fact that different services might respond and act in the same way.

Figure 1.1.4: Enter Caption

1.2 | IP level analysis

Some consideration may also be valid also for the domain level, and this is our reasons behind the choices of aggregation; given the presence of outliers in the dataset, despite efforts in outlier detection, we opted for the median as our aggregation function. The choice of the median contributes to a more robust measure, less susceptible to extreme values that might skew the results. However, it's essential to acknowledge that using the median comes with limitations. As we will see some outliers are so big that they will make the mean an outlier itself, producing significant errors. In our context, prioritizing resistance to outliers outweighs the need for capturing the mean, aligning with the goal of ensuring the reliability of our aggregated results. While aggregating at IP level for median we noticed some columns got the same value, after exploring the meaning of the columns in the starting dataset, we decided to drop these feature with all the same values.
For example, we take in consideration the column _c_ack_count from the scaled dataframe, the mean value is around -0.002506, this value is influenced by a single sample that has a value of 686. The Q1 - Q3 range is -0.1065 - -0.079, the min value is in the 89.9 percentile. If we drop this sample the absolute value of the mean doubles (-0.005121), it's still an high value, because of other outliers. That is why we chosed to use the median value.

1.2.1 | IP-Level Analysis: ECDf and EPDF

The ECDF (empirical cumulative distribution function) and EPDF (empirical probability density function) of the **_s_rtt_avg** variable for the IP level (Figure 1.2.1) provide valuable insights into the distribution of round-trip times (RTTs) across different IP addresses.

Statistic	**_s_rtt_avg**
Count	1526.000
Mean	689.996444
Standard Deviation	3317.793871
Minimum	0.009625
25% (Q1)	3.997172
50% (Median)	11.014725
75% (Q3)	28.573660
Maximum	41944.805880

Table 1.2.1: Descriptive Statistics for _s_rtt_avg

- **Average RTT:** The average RTT is approximately 689.99 milliseconds, suggesting a general low latency across IP addresses.
- **High Standard Deviation:** The standard deviation is 3317.79 milliseconds, indicating significant variability in RTT values. This suggests that there is a wide range of RTTs observed for different IP addresses.
- **Minimum RTT:** The minimum recorded RTT is 9.625 milliseconds, suggesting cases where latency is negligible or absent. This could occur in a local test environment or with high-performance servers near the client.
- **Maximum RTT:** The maximum RTT value of 41,944.81 milliseconds points to the presence of outliers or cases with extremely high latency. These values could stem from unfavorable network conditions, traffic congestion, or other anomalies.
- **Quartiles and IQR:** The 25th percentile (Q1) is 3.997 milliseconds, the 50th percentile (median) is 11.015 milliseconds, and the 75th percentile (Q3) is 28.574 milliseconds. These values indicate that the majority of RTT measurements fall within a relatively narrow range (3.997 - 28.574 milliseconds).
- **Interquartile Range (IQR):** The IQR (Q3 - Q1) of 24.577 milliseconds suggests that the central 50% of the data (between Q1 and Q3) is enclosed in a relatively small range. This implies that, despite the existence of outliers, the majority of RTT measurements are concentrated within a predictable range.

While the majority of IP addresses exhibit low and predictable RTTs, there are significant exceptions with extremely high latency. These outliers warrant further investigation to identify and address the underlying causes of such high latencies. It is crucial to consider other potential factors, such as network topology, server load, and traffic patterns, when interpreting these RTT data.

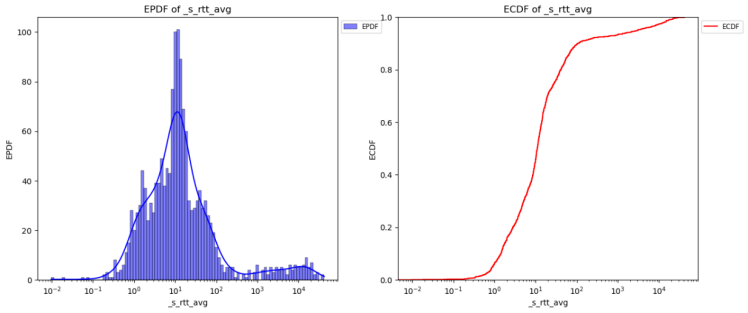


Figure 1.2.1: Empirical Probability Density Function (EPDF) and Empirical Cumulative Distribution Function (ECDF) at the IP level for the variable _s_rtt_avg.

1.2.2 | IP level: correlation analysys

- Features: _c_pkts_data-_c_pktsize_count, Value: 1.0
- Features: _s_pkts_dup-_s_syn_cnt, Value: 0.99
- Features: _c_msgsize-_c_pkt_size1, Value: 0.99
- Features: _s_pkts_data-_s_seg_cnt, Value: 0.99
- Features: _s_ack_cnt-_s_pkts_all, Value: 0.99
- Features: _c_ack_cnt-_c_pkts_all, Value: 0.99

Here we notice that the correlated feature are really similar to the one made before. The two variables, _s_syn_cnt and _s_pkts_dup, might be highly correlated because they measure the same outcome: the number of duplicate packets received by the server during the TCP connection establishment process. This correlation exists because the server sends only one SYN packet for each connection establishment, eliminating the possibility of duplicate SYN packets originating from clients. As a result, any duplicate packets received by the server must have originated from other sources on the network. This explains why the correlation between _s_syn_cnt and _s_pkts_dup is so high. Throughout the dataset there were a lot of columns that similarly maesured the same quantity, this is one of the cases.

1.2.3 | IP level: PCA and t-SNE

To achieve an explained variance of 80%, only 6 Principal Components (PC) are required (Figure 1.2.2. This is likely attributed to the use of the median as the aggregation function, which resulted in a more homogeneous dataset. This homogeneity allows each feature to contribute in the same way to the PC.

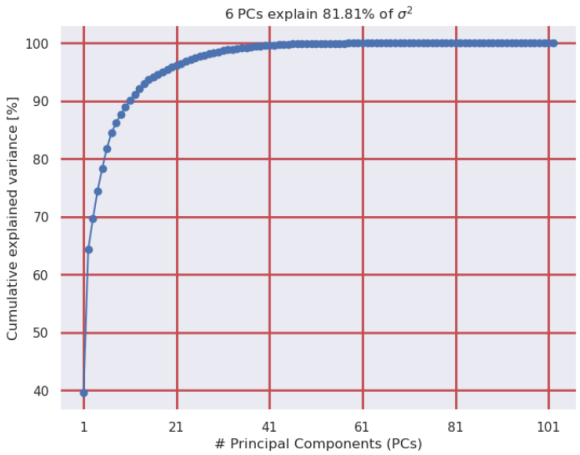
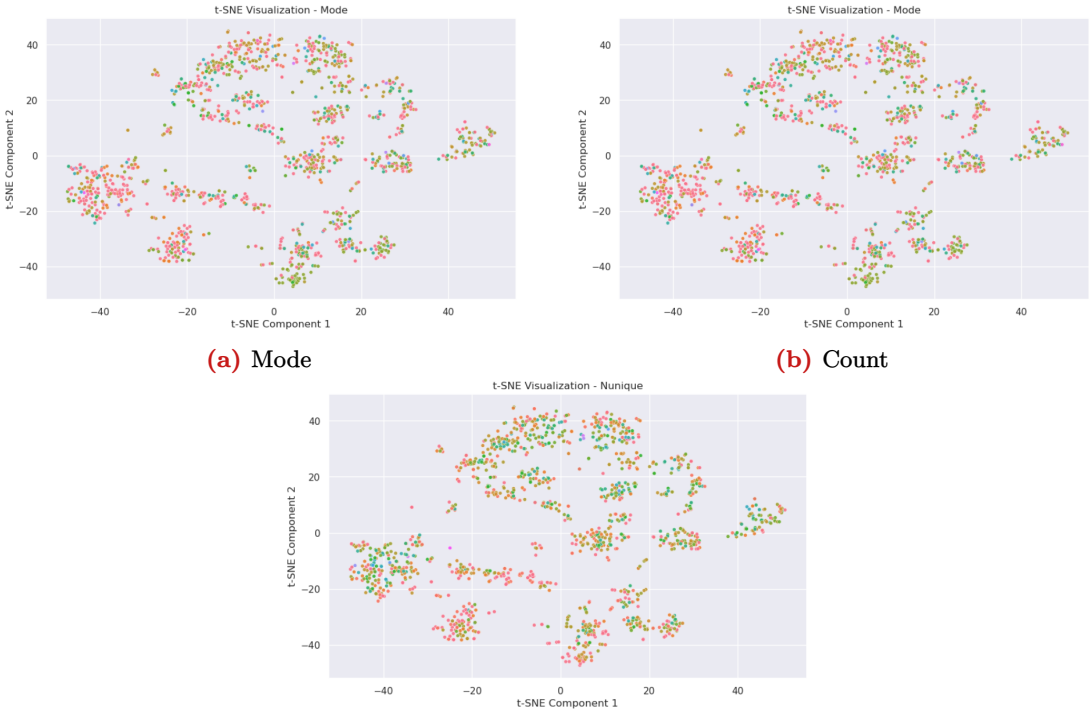


Figure 1.2.2: Enter Caption

For the t-SNE we made 3 different plots :

- in the figure 1.2.3a each points represent the mode of the label for that IP (if an IP connects 4 time to scdn.co and 2 times to twitch.tv, it will be colored with the color of scdn.co)
- in the figure 1.2.3b we made some intervals, and we showed how many connection each IP had
- in the figure 1.2.3c we showed the number of unique connection that each ip had (if an ip connected to twitch.tv and to www.fastly.com it would have made 2 unqiue connection)

With these plots, we can observe that the majority of the IPs established more connections with the label 'other' indicating a higher frequency of connections to that service. Additionally, it is noticeable that IPs generally make between 40 to 100 connections. Furthermore, each IP appears to have had between 3 to 5 distinct connections to different services.



(a) Mode

(b) Count

(c) Unique

Figure 1.2.3: t-SNE at the IP level

1.3 | Domain level analysis

Our objective for domain-level analysis was to identify the intrinsic characteristics of each label, revealing the defining features that principally represent each label. We decided to use the median as the aggregation function due to its resistance to outliers, which is the same reason we used it for IP-level analysis. While aggregating at domain level for median we noticed some columns got the same value, after exploring the meaning of the columns in the starting dataset, we decided to drop all the feature with a standard deviation smaller than a certain threshold.

1.3.1 | Domain level: ECDF and EPDF

Since each label offers different service ('twitch.tv' is a straming service ,'scdn.co' is a dns server for Spotify, 'reddit.com' is social network, 'pinterest.com' is a website used to look for picture, and finally 'giphy.com' is a website used to download gifs), analyzing the distribution of the _s_cwin_max variable can be interesting for data exploration, especially when comparing different services with distinct traffic patterns and resource usage. For instance, we might expect streaming services like Twitch.tv to exhibit a higher average _s_cwin_max value than social media platforms like Reddit.com, reflecting the higher bandwidth demands of video streaming. By comparing the distributions across different services, we can gain insights into their relative performance characteristics and make informed decisions about resource allocation.

- The maximum congestion window size distribution indicates that most servers utilize a range of 3,946 to 6,799 bytes, with a mean of 10,740 bytes. However, there is a significant skew towards larger cwnd sizes, with a maximum value of 89,166 bytes. This suggests that some servers are using larger cwnd sizes to handle more data-intensive traffic.
- The high standard deviation of 17,840 bytes and skewness of 0.93 further confirm that there is a wide range of cwnd sizes used by different servers. This variation may be attributed to factors such as server configurations, network conditions, and the type of traffic being transmitted.

Statistic	Value
Count	26.000
Mean	10740.000
Standard Deviation	17840.159239
Minimum	424.000
25% (Q1)	3946.250
50% (Median)	5325.000
75% (Q3)	6798.750
Maximum	89166.000

Table 1.3.1: Descriptive Statistics for the Data

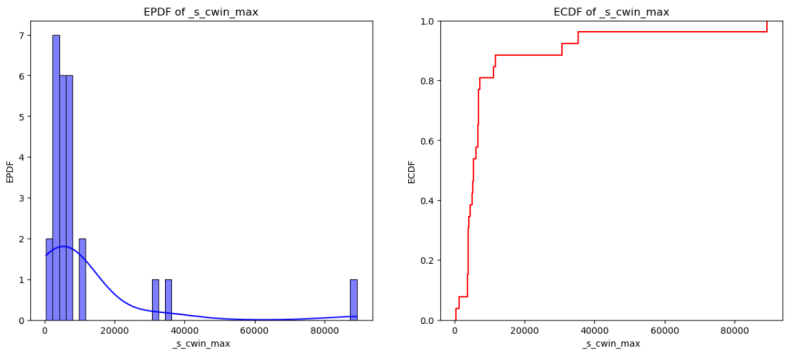


Figure 1.3.1: Enter Caption

1.3.2 | Domain level: correlation analysis

- Features: _c_ttl_max-_c_ttl_min, Value: 1.0
- Features: _c_pkts_data-_c_pktsize_count, Value: 1.0
- Features: _c_pkts_data-_c_seg_cnt, Value: 1.0
- Features: _c_pktsize_count-_c_seg_cnt, Value: 1.0
- Features: _c_tm_opt-_s_tm_opt, Value: 1.0

Like we said before, some of the correlated feature repeat them self, here where we try to find the characteristic the characterize each domain, and some correlation tent to repeat them self. This is also caused by the choice of aggregation, in fact, if we look at each single row for _c_ttl_max and _c_ttl_min, the values are really similar in terms of value that they assume.

1.3.3 | Domain level: PCA and t-SNE

The graph shows that the first few PCs explain a relatively large amount of the variance in the dataset. This is because the first few PCs capture the most important patterns in the data. As we add more PCs, the amount of additional variance explained decreases. This is because the remaining PCs capture less important patterns in the data, caused by the way we aggregated data.
We can notice from the t-SNE distribution of Figure 1.3.3 that there are several distinct clusters of websites, each of which is associated with a different TLD. For example, the cluster of websites in the top left corner of the visualization is mostly composed of ad networks and social media websites, while the cluster in the bottom right corner is mostly composed of video streaming websites or websites where you need to download stuff (twittch, github, krxd.net).

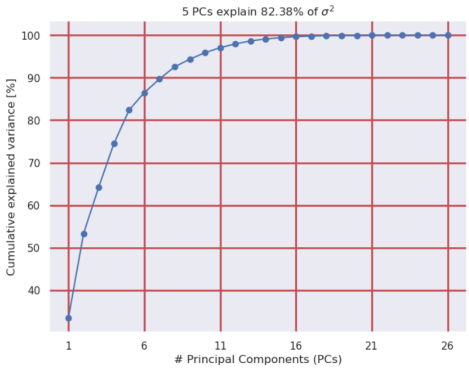


Figure 1.3.2: PCA at domain level

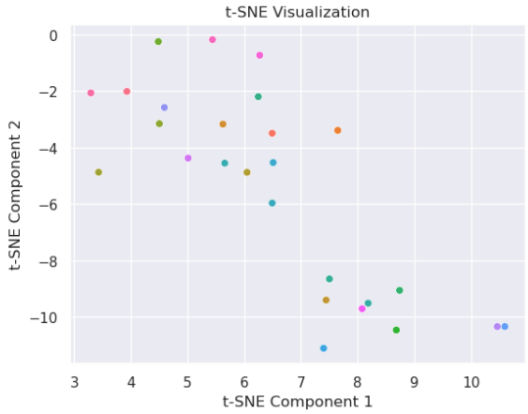


Figure 1.3.3: t-SNE at domain level

1.4 | Statistics on Bytes Transmitted

In this section, we analyze the bytes transmitted by the server for each label. Observing the plot 1.4.1 and understanding the dataset structure, it becomes apparent that some labels consistently transmit a large amount of bytes, even though they are less prevalent in the dataset. This suggests that certain services, despite their lower frequency, have to transmit a lot more more bytes compared to other services, in particular, giphy.com, that has around 7000 entry has sent a lot of bytes compared to other labels that are more present (like krxd.net that has around 20 000 entry). To have a better view of what are the labels that sent the most bytes (by both the client and the server) we used a bar plot that showed us this results (figure 1.4.2).

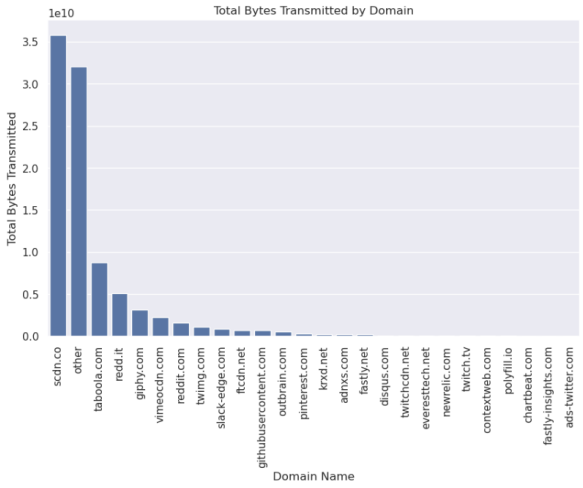
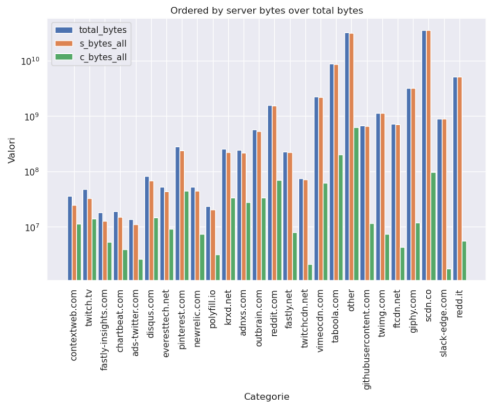


Figure 1.4.1: Bytes sent by each label

We've also explored the relationship between the bytes sent by the client and the bytes sent by the server using a scatter plot. For certain services, such as the service labeled scdn.co (see Figure 1.4.3a), distinct patterns emerged. While investigating the relationship between the bytes sent by the client and those sent by the server based on the service label, we observed a notable pattern for standard services like DNS. These patterns appear more "dense" in certain areas, suggesting a reasonable hypothesis: when clients seek access to specific information, the steps required to obtain that information may be similar. Consequently, the server responds with similar byte patterns to all users attempting to access the same information.



From this chart we can notice that the two services where client and server send the most bytes are scdn.co and taboola.com. The first one seems to be a server for Spotify while the other one is a website used for advertisement.

Figure 1.4.2: Total bytes sent by client and server

(a) Left: Scatter plot for scdn.co; Right: Bytes sent for reddit.com

Figure 1.4.3: Labels with similar patterns

1.5 | Round Trip Time (RTT) Analysis

In our analysis of Round Trip Time (RTT), we aimed to visualize the labels with the highest average RTT from both the server and client perspectives. Figure 1.5.2 illustrates the results, highlighting the two labels with the highest server-side RTT values: "twitch.tv," representing the streaming service itself, and "twitchcdn.net," a DNS associated with Twitch. This observation may suggest that these two service providers are located far from the data source.

Upon analyzing the client-side RTT, we observed a general trend of lower RTT values compared to the server-side. This aligns with the goal of minimizing client wait times to enhance the user experience. However, there was one notable exception: "fastly-insights.com," an optional service deployed by some Fastly customers for network and performance monitoring and research purposes (https://insights.fastlylabs.com/).

Given that this website collects detailed information about HTTP and HTTPS network transactions, including network routing, performance timing, and equipment characteristics, it is reasonable to expect a longer response time. The computational load on the end-point could contribute to the extended RTT experienced by clients interacting with "fastly-insights.com."

To better visualize the difference between fastly-insights.com we used another plot that defines a rateo between client rtt and server rtt, and as we expected, this one came on top (Figure 1.5.1).

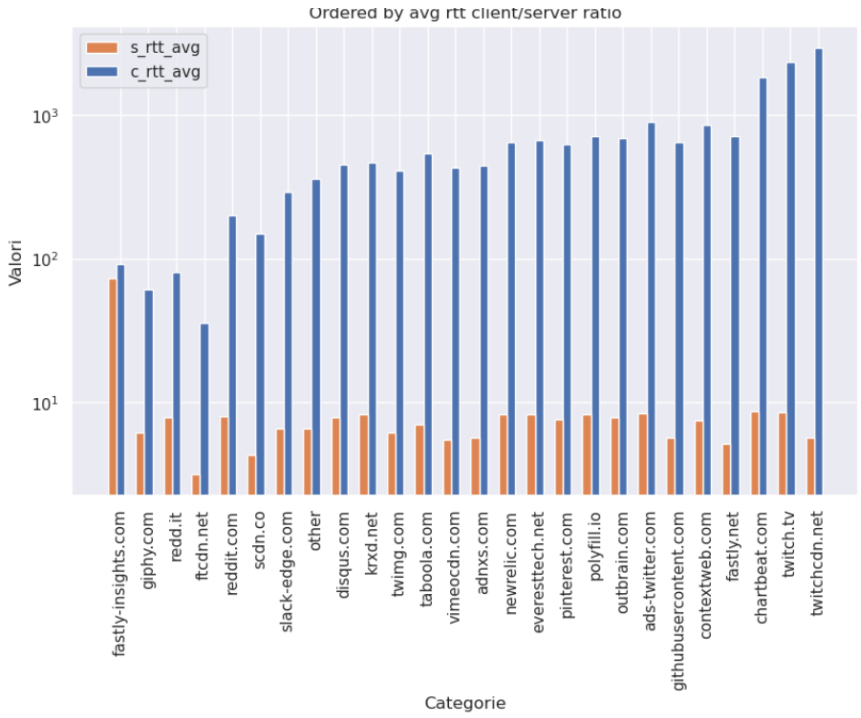


Figure 1.5.1: Rateo between s_rtt and c_rtt

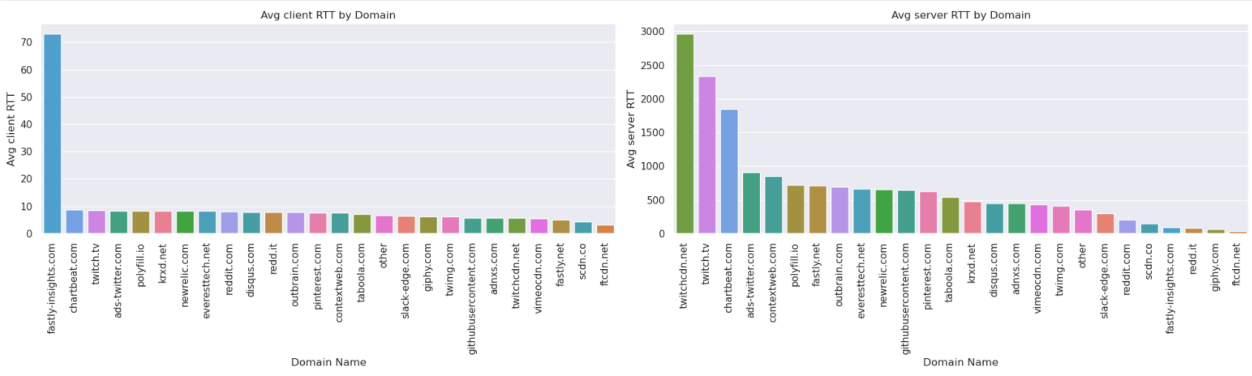


Figure 1.5.2: Client and Server Rtt for each label

Chapter 2

Classification

We have chosen the following three methods for classification:

- Random Forest
- k-Nearest Neighbors
- Neural Network

2.1 | Preparation of the dataset

We have used the scaled dataset created in the first exercise and we have then dropped the column related to client ip because we want to classify the data based on traffic and not on users, and keeping that feature would have produced some overfitting. We have then dropped 46 features in a way that in our correlation matrix there are no relationship between features with a correlation higher than 0.8. We have also applied our best model to the dataset obtained in exercise 1, using PCA and keeping PCs such as explained variance is greater than 80% and compared the results on both reduced dataset.
After a lot of test, we found out that a way that gave us better accuracy was analyzing feature and manually selecting them, performing the tests. Remarkably we obtained better performance on a model with just 4 features instead of the model trained with the dataset that has 65 features with no features correlated with each other for more than 0.80.
We are going to show our work done on the "bigger" dataset, we will later compare our results with the results obtained by just taking in consideration the subset of features.

2.2 | Random Forest

First we have created a random forest classifier using default settings, the train set performance resulted in a correct classification of every single row of the dataset, while the test set performance had worse results, with an accuracy of 83%.

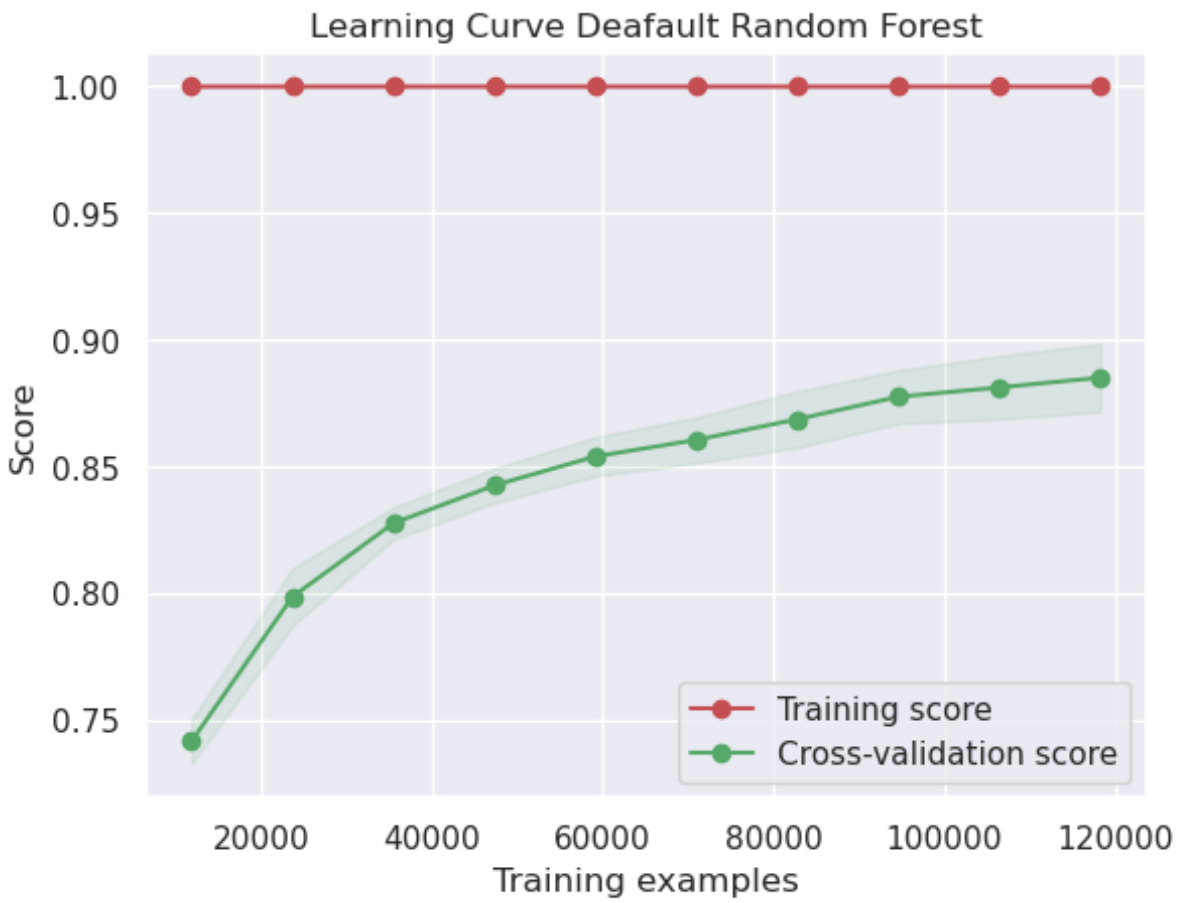


Figure 2.2.1

Judging from the learning curve we can observe overfitting, we can also see that the cross-validation score improves with the number of row in the dataset, meaning that an increased dataset will help solve part of the problem. We can try to improve the result with a better feature selection and hyperparameter tuning.
We have tried tuning the hyperparameters, the results had a very minimal improvement, that could be even better if more combinations of hyperparameters were to be tried.

Figure 2.2.2

As a final test we have tried to ”play” with hyperparameters and we found some models that the cross validation discarded because of a worse score, but are interesting to observe because of the little overfitting. This result has been obtained by limiting the max depth of the decision trees inside the forest. This parameter has been important to improve the overall prediction of the classifier, but it’s also the main culprit for the overfitting since having no maximum depth can create some leaf based only on specific train set data.

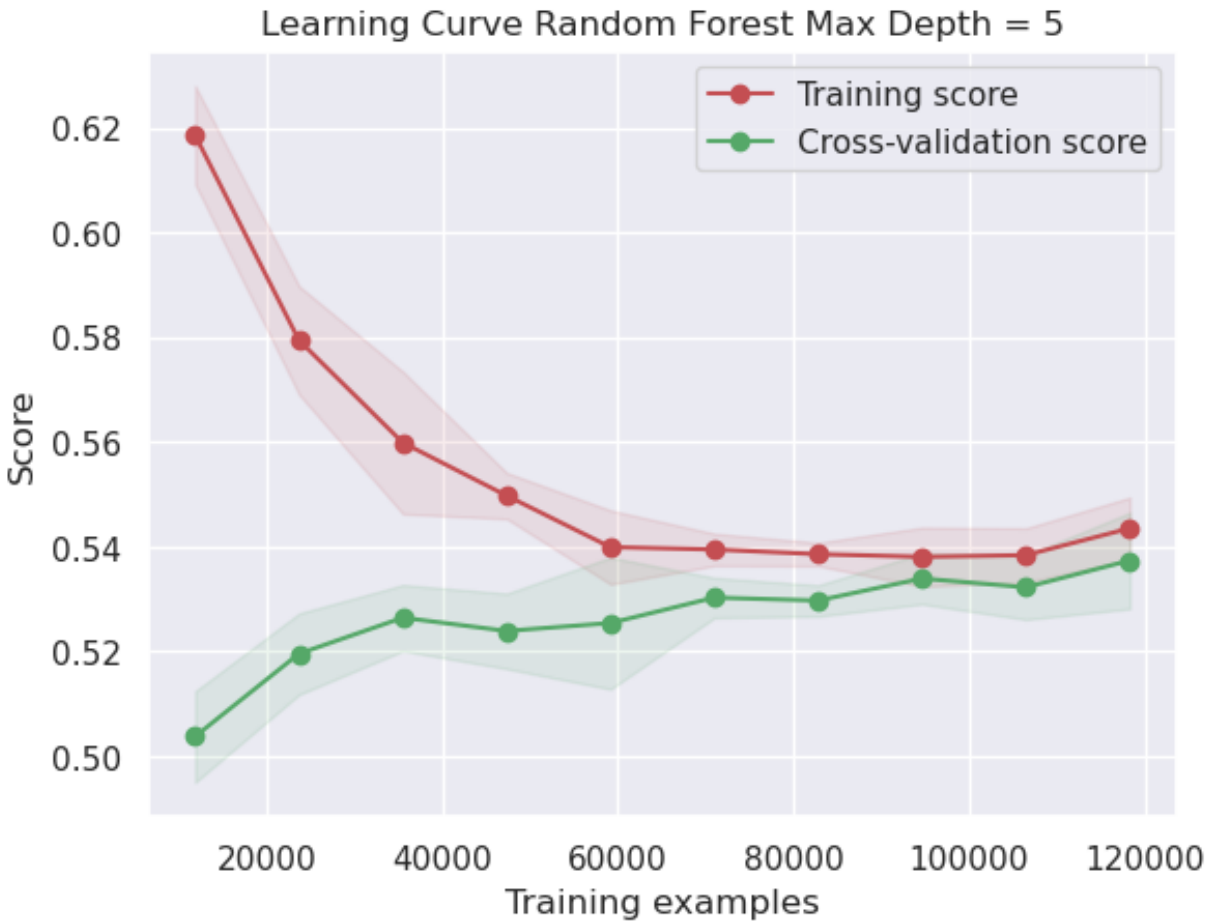

Learning Curve Random Forest Max Depth = 5
Training score
Cross-validation score
0.62
0.60
0.58
0.56
0.54
0.52
0.50
Score
20000
40000
60000
80000
100000
120000
Training examples

Figure 2.2.3

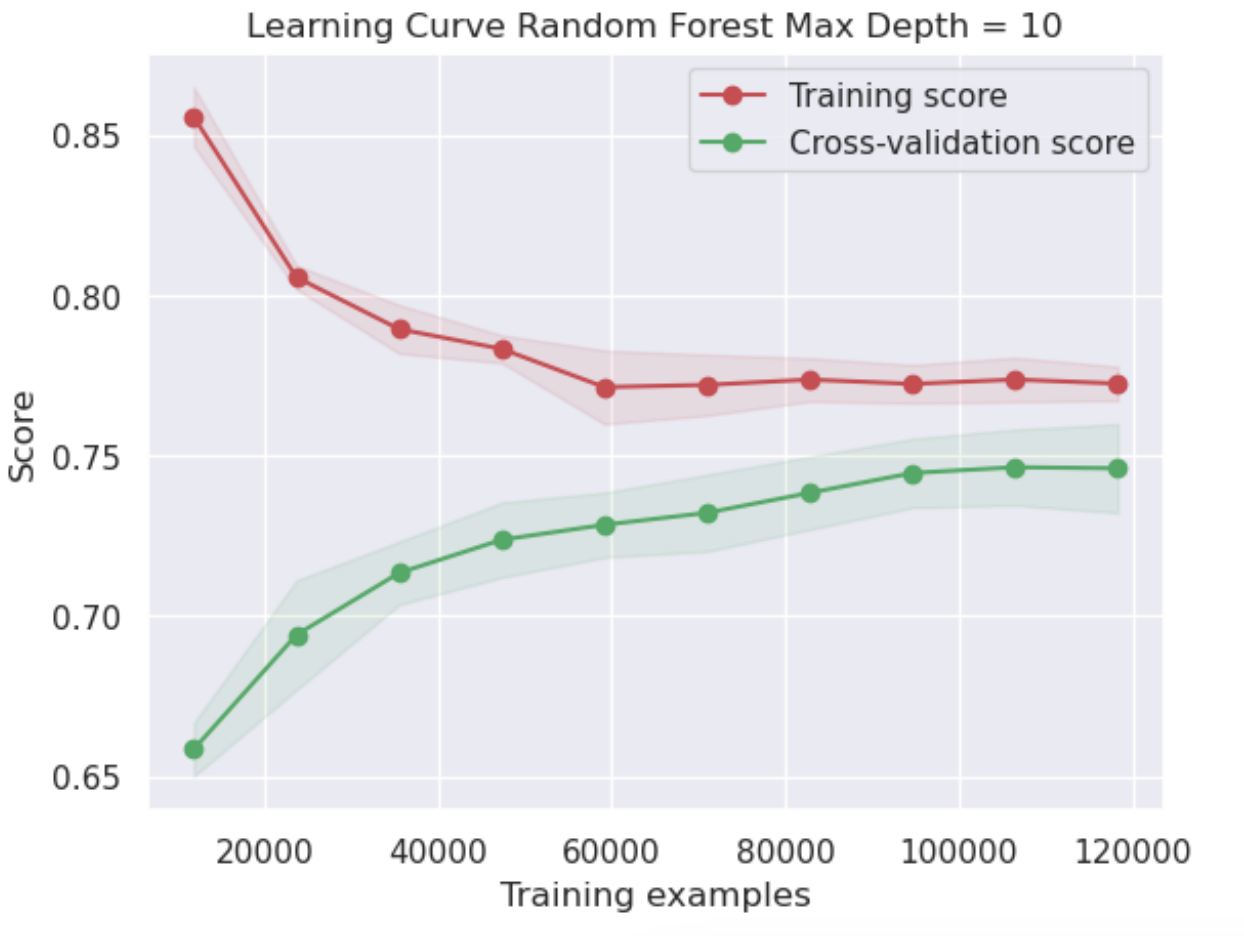

Learning Curve Random Forest Max Depth = 10
Training score
Cross-validation score
Score
0.85
0.80
0.75
0.70
0.65
20000
40000
60000
80000
100000
120000
Training examples

Figure 2.2.4

2.2.1 | False Positive/Negative Analysis

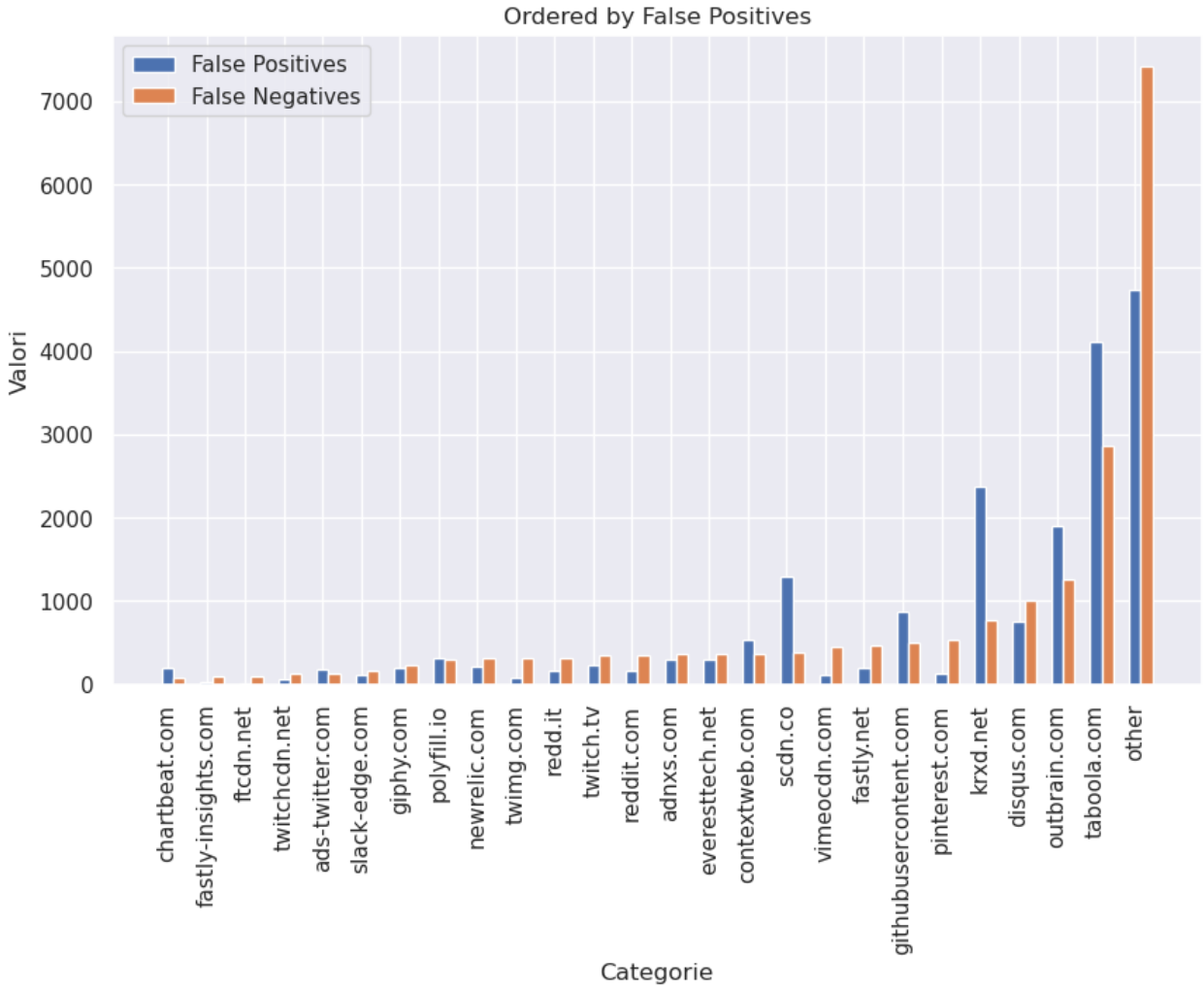


Figure 2.2.5

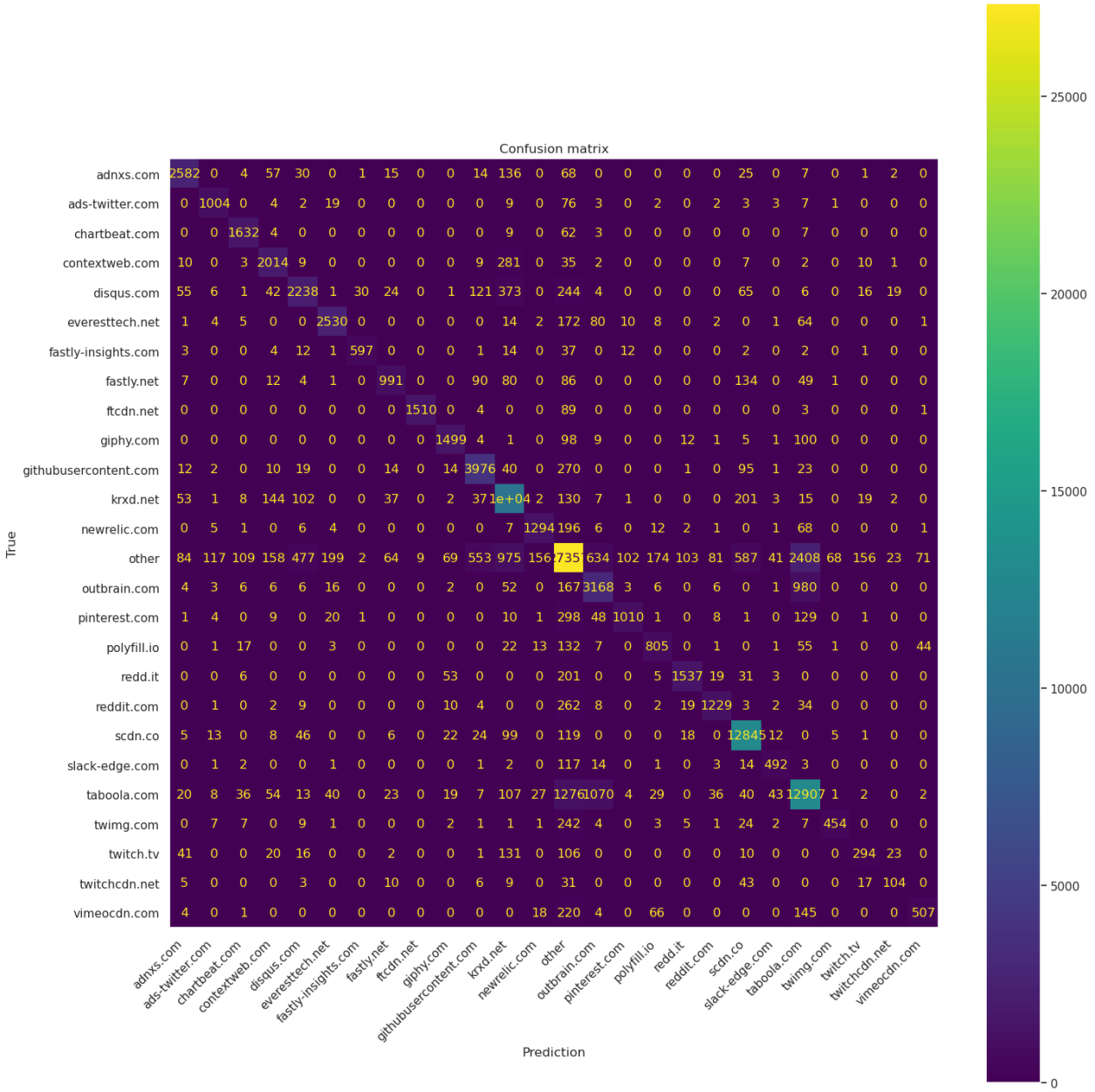


Figure 2.2.6

We can observe that false positive are more than false negative, in particular there are a lot of FN on the label "other". We can hypothesize that there are a lot of website that has similar characteristic to the ones that have a proper label, so the classifier will predict this traffic to be another label rather than the "other" label that has a lot of mixed information. The false positive are also the higher in the "other" label, this is probably due to the fact that the connections related to "other" label are 69860 while most of the other label have less than 10000 connections, also the fact that "other" is derived from a large mix of website that will have very different characteristic from each others, will produce a bigger hypothesis space than the other classes.
The sample 218560 has the label "other" while the classifier has predicted "taboola.com". We can observe that the value of _c_cwin_max for the sample, is 645.031636, if we consider the subset of the test where our classifier has misclassified and predicted "taboola.com" we can see it's an extreme value since the 25% percentile is -0.144258, the 75% percentile is 0.220692 and the mean is 0.400993, mean that is also highly influenced by this value, cause in this subset there are 4114 occurrences. If we analyze the train set we can see that the top 10

features with higher _c_cwin_max ranging from 230 to 28), are all labaled as "other". Also the random forest classifier, consider _c_cwin_max as the fourth best feature, this makes this misclassification particular, why would the classifier predict "taboola.com"? The choice made by the classifier are the ones in the following picture:

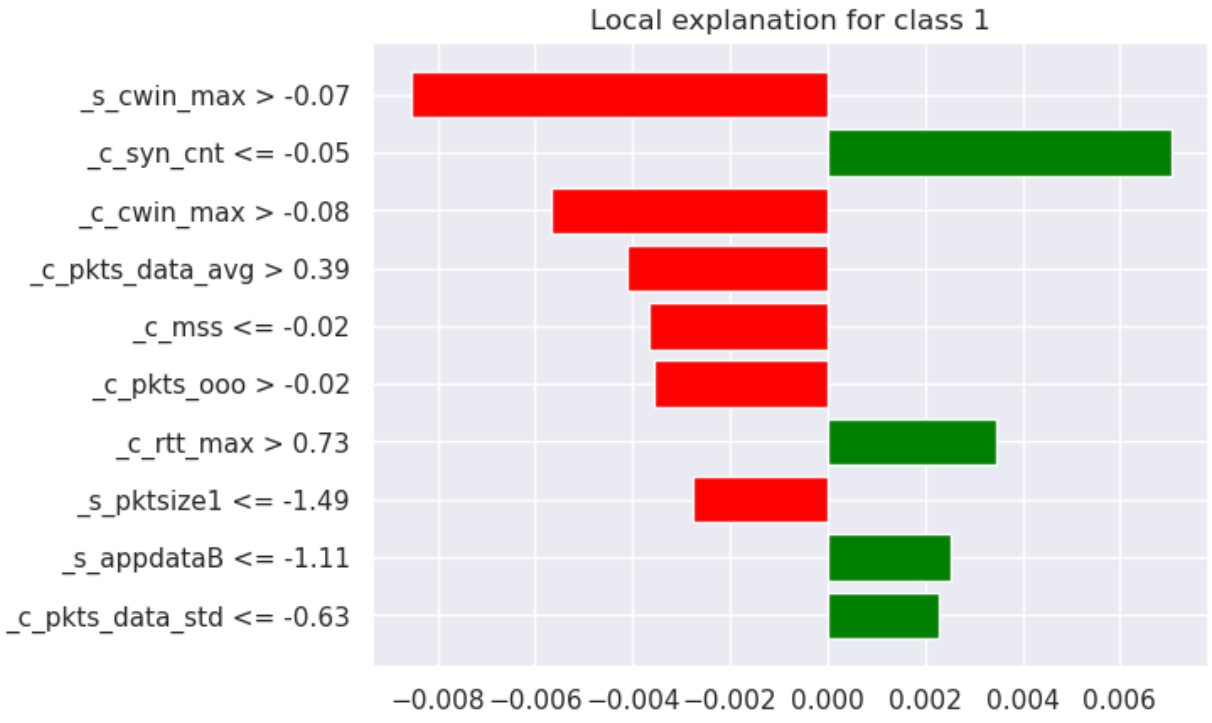


Figure 2.2.7

The percentile of -0.07 for _s_cwin_max for "other" as label is 78.45%, most of it's value are lower, while our classifier looks for value that are bigger. For the _c_cwin_max feature, we can see that the classifier looks for value that are bigger than -0.08, so the fact that our value is so extreme, isn't taken in consideration. Another interesting feature to consider it's _c_pkts_data_avg, the value of our sample is 6.37, and it exceeds the maximum value for "other" flow in the train set, which is maximum value is 6.24. Furthermore the value 0.39 searched by our classifier is in the 83.24 percentile of the _c_pkts_data_avg seen for "other" in the train set, meaning that most of their value are lower. This is part of the reason why sample 218560 has been misclassified. The probability of being "taboola.com" for the model is 0.40, while 0.25 for "other", the third highest chance is "outbrain.com" at 0.08, the classifier would have been right with it's second guess.
For sample 250110 we have considered the best 20 features found by the random forest classifier:

- `_s_appdataB`
- `_s_cwin_ini`
- `_s_cwin_max`
- `_c_cwin_max`
- `_s_pkts_data_avg`
- `_c_pkts_data_std`
- `_c_pkts_data_avg`
- `_s_pkts_data_std`

- _s_pktsize1
- _c_appdataB
- _s_rtt_cnt
- _s_mss_max
- _c_rtt_max
- _s_appdataT
- _s_win_max
- _c_appdataT
- _c_cwin_min
- _s_sit_avg
- _c_rtt_avg
- _c_mss_min

Sample 250110 is labelled "fastly.net", classifier prediction is "contextweb.com". We have based our analysis by comparing the median values of the training dataset of the respective labels. Out of the 20 features, 12 features have less difference with contextweb.com median, and 4 with fastly.net median, 4 other features have the same absolute difference. The 2 features that had the most impact are:

Data	_c_cwin_min	_s_sit_avg
sample	-0.091699	-0.326469
fastly.net	-0.091699	-0.329184
contextweb.com	0.769064	-0.042056

2.3 | k-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a powerful and intuitive machine learning technique used for classification tasks. The model performs well on the training set with an overall accuracy of 80%, the weighted average metrics are also balanced, indicating good performance across different classes. However, the model's performance drops on the test set with an overall accuracy of 53%. The macro average metrics are lower, indicating that the model struggles with some classes, like: *pinterest.com*, *polyfill.io*, *twimg.com*, *twitch.tv*, *twitchcdn.net*, and *vimeocdn.com*.
The cross-validation process has been performed, and the best hyperparameters for the k-Nearest Neighbors (KNN) model have been identified as follows:

- Best Hyperparameters: n_neighbors: 8, p: 1, weights: distance

The model can now learn well the training set, achieving 100% accuracy and F1-scores for each class. On the test set the accuracy has improved from 53% to 61%, indicating progress in the model's predictive capabilities also on the test set, although the macro and weighted averages for precision, recall, and F1-scores are still lower compared to the training set.

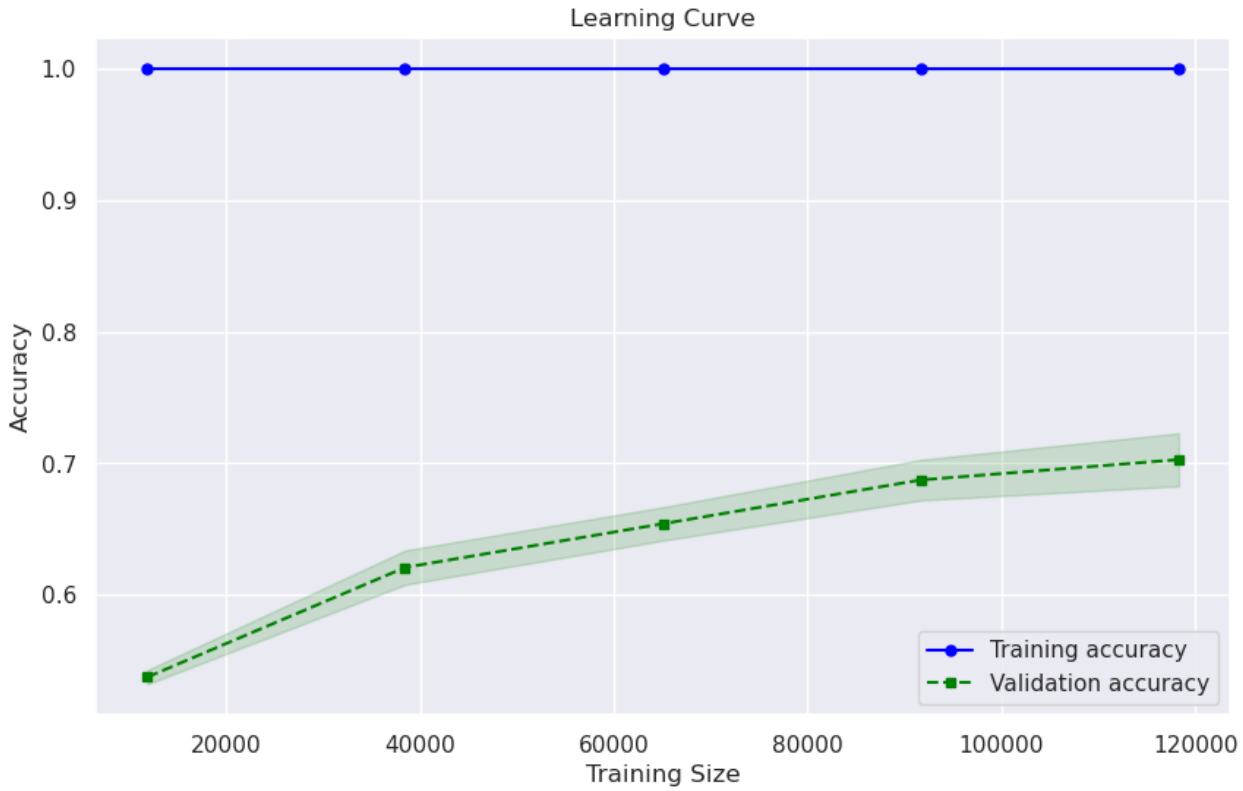


Figure 2.3.1

The learning curve suggests overfitting. As was previously observed in the case of random forest, we can see that the cross-validation score improves with the number of rows in the dataset, indicating that an increased dataset could help to mitigate the problem. To further improve the results, we can try to improve feature selection and hyperparameter tuning.

2.3.1 | False Positive/Negative Analysis

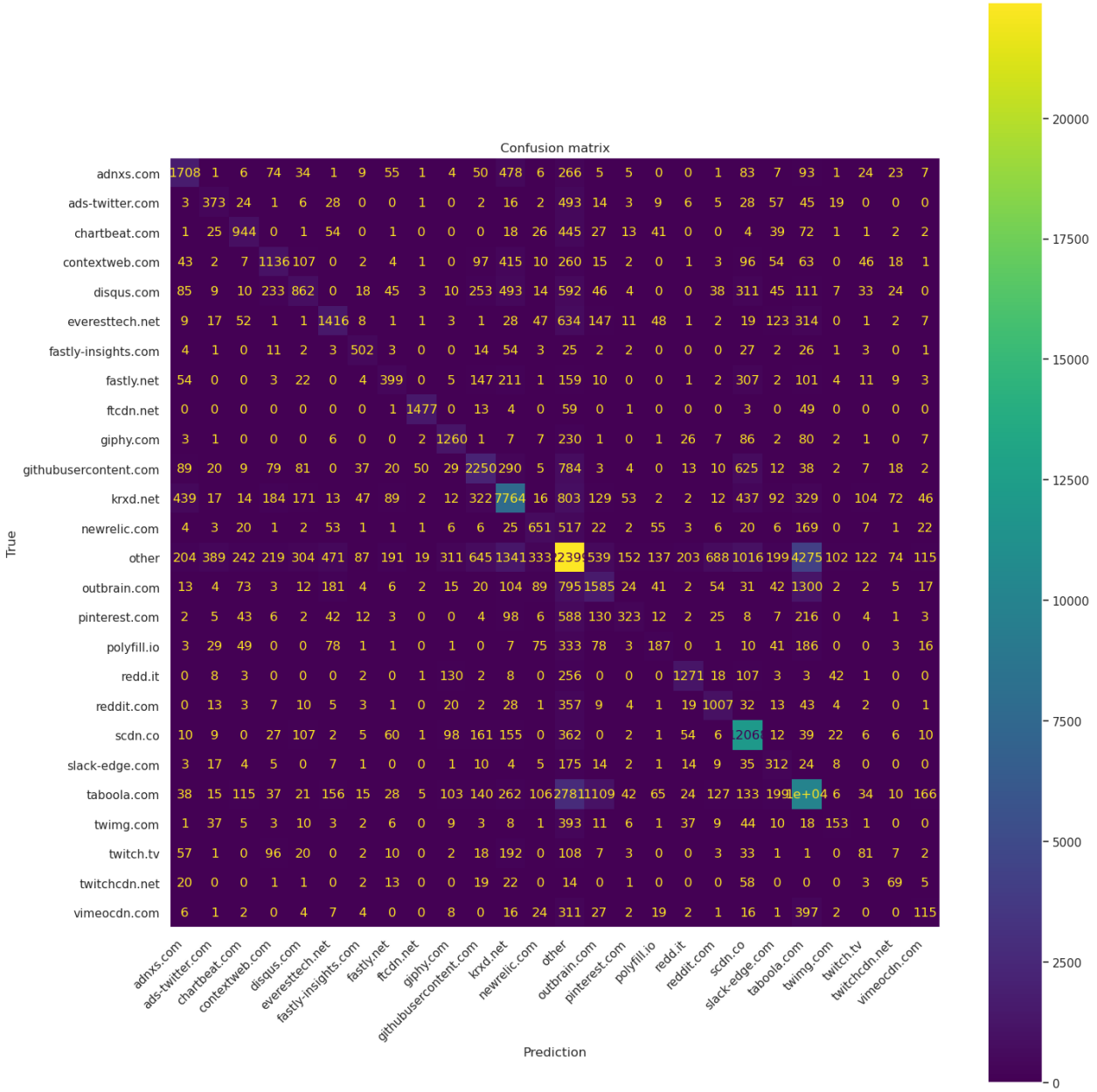


Figure 2.3.2

Having lower overall accuracy than random forest, it means that we also have more false positive and negative, in particular we notice that "taboola.com", "outbrain.com" and other have more misclassification, often intertwining with each other.
We can check sample 218560 that has prevoulsy been misclassified in random forest this time has the correct label, we cannot say the same for sample 250110, that this time has been predicted as "krxd.net". If we run the same test we did for random forest, we can see that 9 features have less difference with krxd.net median, 7 with fastly.net median, 4 other features have the same absolute difference. _c_cappdataT and _c_win_min are closer to krxd.net, while _s_sit_avg is now closer to fastly.net.
We now analyze sample 222290, it has been predicted as everesttech.net, while its label is outbrain.com. The

list of neighbors generated by the model it's the following:

ID	Website
47114	ads-twitter.com
10658	everesttech.net
119432	everesttech.net
37007	outbrain.com
2271	everesttech.net
126141	everesttech.net
60623	everesttech.net
88975	polyfill.io

Table 2.3.1: Neighbors

We then analyze the median value of this 5 points and the median values of the data in the train set with label outbrain.com. We computed the absolute distance from our sample and the medians of the 2 dataset in consideration, and we observed the biggest change in the following features:

Feature	Neighbors	222290	outbrain.com	abs-niegh	abs-out
_c_appdataB	0.120600	0.096524	0.574035	0.024076	0.477510
_c_pkts_data_avg	-0.356982	-0.595119	-0.099467	0.238137	0.495652
_c_pkts_data_std	-0.379246	-0.497133	0.190731	0.117888	0.687865
_c_port	0.602439	0.442706	0.085250	0.159733	0.357456
_c_rtt_avg	0.177933	0.200500	0.080708	0.022566	0.119792
_c_win_max	0.078051	0.078051	-0.171849	0.000000	0.249900
_s_appdataB	-1.120015	-1.120015	1.282304	0.000000	2.402319
_s_cwin_ini	-1.157706	-1.157706	1.253332	0.000000	2.411038
_s_mss_max	-2.327852	-2.279130	0.433471	0.048721	2.712601
_s_pkts_data_avg	-1.947554	-1.926540	-0.017293	0.021014	1.909247
_s_pkts_data_std	-1.611003	-1.532365	0.735412	0.078638	2.267778
_s_pktsize1	-1.555366	-1.555366	0.655572	0.000000	2.210938
_s_sit_avg	0.122095	0.075359	-0.333937	0.046736	0.409296
_s_sit_std	0.084346	0.026890	-0.451618	0.057456	0.478508
_tls_session_stat	-1.606309	-1.606309	0.608787	0.000000	2.215096

Table 2.3.2: Most distant features

We can notice the presence of both _c_appdataB, _c_pkts_data_avg, _c_pkts_data_std and _s_appdataB, _s_pkts_data_avg, _s_pkts_data_std

2.4 | Neural Network

The neural network is designed as a sequential model in Keras, consisting of three layers, a dense layer with 128 neurons and ReLU activation, a dropout layer with a rate of 0.2 to reduce overfitting by randomly disabling some neurons during training, and a dense output layer of 26 neurons with softmax activation. The model is compiled with the Adam optimizer, 'sparse_categorical_crossentropy' loss function for multi classification, and accuracy as the metric.
After 10 epochs of training, the model achieved an accuracy of 0.66. The model performed on the train set resulted in an accuracy of 0.70. Classes such as "ftcdn.net" and "scdn.co" show good performance with precision, recall, and F1-score above 0.90. Some classes, such as "polyfill.io" and "vimeocdn.com", have lower performance, with F1-score close to 0. The model's performance on the test set was 0.60. The classes "ftcdn.net" and "scdn.co" still show good performance, but in general, the metrics are slightly lower than on the training set. This indicate a presence of overfitting. Increasing the epoches to 10, improves the accuracy from 0.66 to 0.69, this is a good indicator meaning that the classifier has more room to learn. While

validating the model on 10 epochs, we noticed a slightly higher validation accuracy than training accuracy, 0.7076 and 0.6838. This might happens due to the low variance of the dataset and the presence of dropout: During training, dropout layers are active, but they are disabled (skipped) during validation and testing. This behavior is automatic and intentional. While dropout may slightly impact training error, it is a known and expected outcome. Increasing the epoch to 100 we obtain a training accuracy of 0.7436 and a validation accuracy of 0.7740.

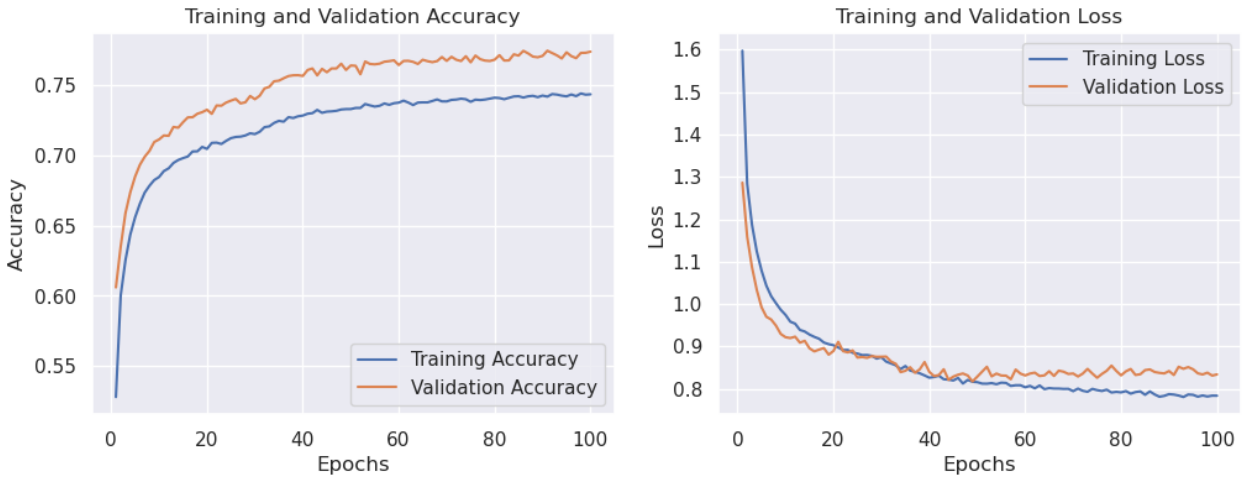


Figure 2.4.1

The cross-validation was performed manually with the following hyperparameters:

- Optimizer: The method used to update the parameters of the model during training.
- Activation function: The function used to calculate the output of the units of the model.
- Dropout rate: The percentage of units of the model to be eliminated during training.

Five-fold cross-validation was employed for robust model evaluation .

Optimizer	Activation Function	Dropout Rate	Mean Validation Accuracy
Adam	ReLU	0.2	0.680375
Adam	ReLU	0.3	0.674539
Adam	Tanh	0.2	0.648082
Adam	Tanh	0.3	0.634313
SGD	ReLU	0.2	0.594479
SGD	ReLU	0.3	0.587209
SGD	Tanh	0.2	0.563349
SGD	Tanh	0.3	0.554131

The best parameters chosen by the cross validation ended up being the ones chosen for initialization.

2.4.1 | False Positive/Negative Analysis

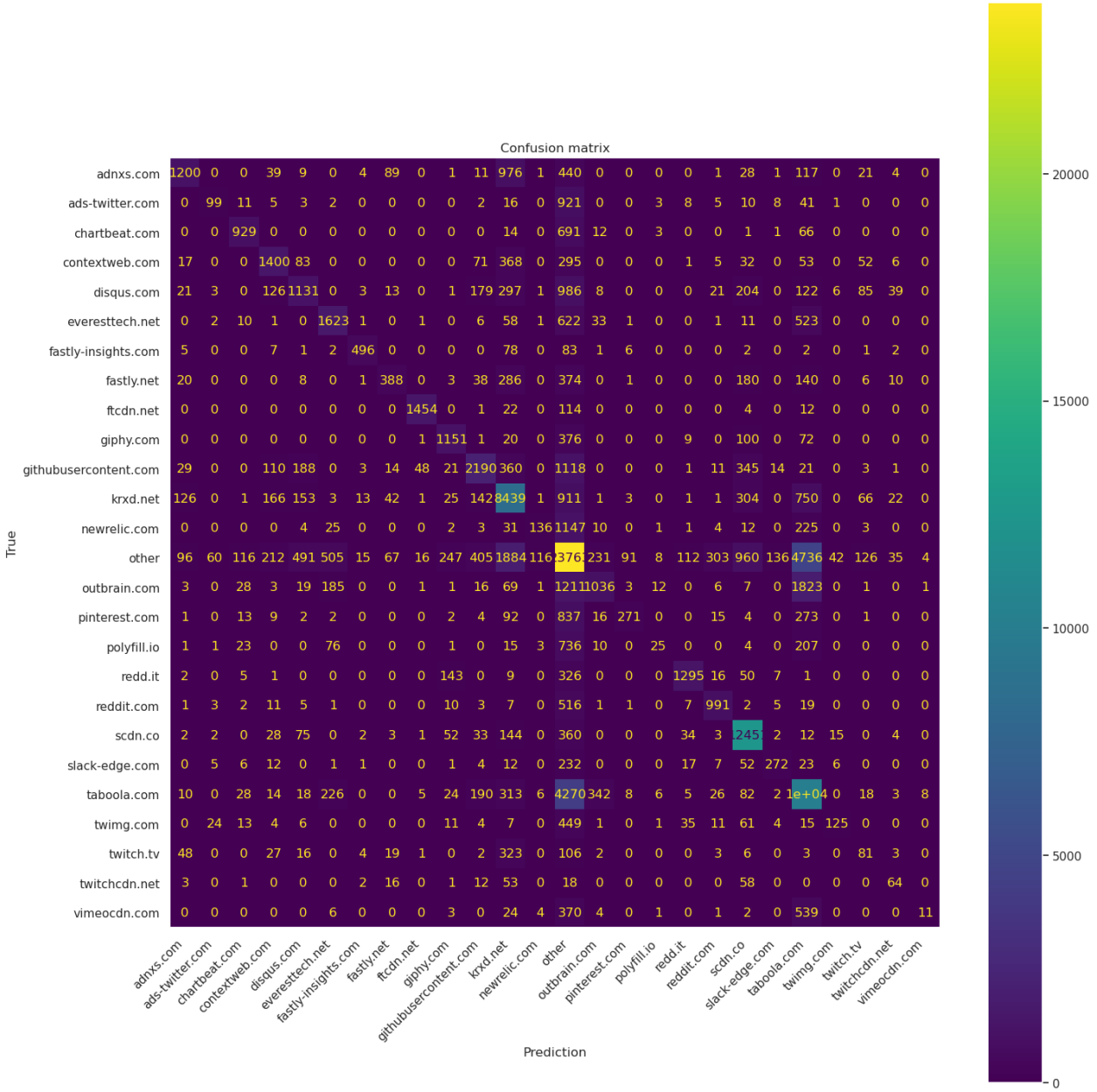


Figure 2.4.2

Examining the 3 samples we have seen in the previous classifier, we can interestingly notice that two of the three are evaluated like KNN classifier. So 218560 gets labelled correctly into ”other”, 250110 gets misclassified in the same label ”krxd.net”, while 222290 is now being predicted ”taboola.com”.

Class	Probability
taboola.com	0.5009
everesttech.net	0.1819
outbrain.com	0.1718

Table 2.4.1: Probability of classes

2.5 | Subset of Features

To get to the result of utilising only 4 features, we have analyzed all the features and then we dropped some that we didn't consider interesting and than we evaluated the results on the test set with a random forest model. We noticed the accuracy was initially slightly dropping (from 0.83 to 0.80), then it started increasing again and we got back 0.83 when we reached the 4 features.
The list of 4 features is the following:

- _c_appdataB
- _s_appdataB
- _s_pkts_data_avg
- _c_pkts_data_avg

After further research for the best features we have also found this 6 features that performed 0.86 on the default random forest:

- _c_cwin_max
- _c_pkts_all
- _c_pktsize1
- _s_appdataB
- _s_cwin_max
- _s_msgsize1

But the best accuracy we have seen (0.87) it has been on this subset of this 16 features:

- `_c_bytes_all`
- `_c_bytes_uniq`
- `_c_cwin_max`
- `_c_pkts_all`
- `_c_pkts_data`
- `_c_pktsize1`
- `_c_pktsize_count`
- `_s_ack_cnt`
- `_s_appdataB`
- `_s_bytes_all`
- `_s_bytes_uniq`

- _s_cwin_max
- _s_msgsize1
- _s_pkts_all
- _s_pktsize_count
- _s_seg_cnt

The code related to this part are in the files *4features.ipnyb*, *16features.ipnyb* and *PCA.ipnyb*.
"No correlations" is referred to the dataset with most correlated columns dropped so that there aren't a pair of features with a correlation $> .80$, "All Features" is the whole dataset with time, c_ip and 0 std columns dropped (for reason we have already discussed), consisting of 111 features.

Classifier	No Correlation	4 Features	16 Features	All Features	22Pcs
RF Default	0.83	0.83	0.87	0.85	0.62
RF Tuned	0.83	0.83	0.86	-	0.62
KNN Default	0.53	0.66	0.73	0.55	0.55
KNN Tuned	0.61	0.69	0.76	-	0.57
NN Default	0.62	0.48	0.54	0.66	0.52
NN Tuned	0.62	0.48	0.54	-	0.52

Table 2.5.1: Performance metrics of different classifiers on various feature sets.

As for random forest and knn, we have seen better results in the 16-features data set, with significant improvements for the knn. It's also very interesting to see that just 4 features gave us a better performance than the "no correlation" data set which performed slightly worse than the whole dataset. Reducing size is a key part in machine learning because of computational times, there are certain algorithms, or certain tasks, that might not be possible perform on big data set due to excessive computational time, in fact we found really hard to perform cross validation on the "All Features" dataset. Working with just 4 or 16 features allowed us to improve the model.
As for the neural network instead, we can see that the more features we used the better the accuracy we obtained.
We can see the worse performance are obtained in the data set with 22 PCs, PCA didn't work as well as our choice of features. An higher number of PCs could have been considered, but we have already worse accuracy with more columns of the data set compared to 16 features. Interestingly on default k-nn the dataset with 22 PCs had the same accuracy of the data set with all features.

2.6 | Conclusion

The dataset itself has a lot of of correlated features and a lot of features that have a very low significance. They would require an in depth analysis, faster and simple approach like keeping PCs until variance is $> 80\%$ or just dropping most correlated features can still get an acceptable result. Most of our model have some overfitting, that it can be reduced but it'd also reduce the accuracy of the classifier.
Our choice of selecting manually the features seems it's very good choice for implemented models like Random Forest and k-nearest neighbour, while it lost a lot of accuracy on our neural network. It might be interesting to look if there are other features that we can add to our subsets, that would result in a better classifier.
The model that has performed the best it's random forest, but we can say that the neural network has some potential. We choose a very basic one with only a few epochs. The fact that the NN performed better with more features, indicates that it could still extract some data, so there is room of improvement for the "building" of the NN itself: adding more layer and increasing the number of epochs would improve the classifier. An improved version might reach better performance than random forest or k-nearest neighbours.

Chapter 3

Unsupervised learning – clustering

For this unsupervised task, the selected clustering algorithms are:

- K-Means
- Spectral Clustering
- Optics_DBSCAN

To cluster domain names based on similar patterns using unsupervised machine learning techniques, we utilized the domain-level aggregation of our dataset, as previously detailed in Section 1 under the "Domain level analysis" paragraph.

3.1 | Feature selection

In response to the directive to find a representation for each domain name (features), LASSO (Least Absolute Shrinkage and Selection Operator) feature selection was employed as a strategic approach. LASSO feature selection serves the purpose of identifying the most influential features for domain clustering while mitigating the impact of irrelevant or redundant ones. This ensures a more focused and meaningful representation of domain patterns.
LASSO regression was applied with an alpha parameter set at 0.1, allowing for the selection of relevant features while introducing a level of regularization. The feature selection mask was obtained by identifying non-zero coefficients from the regression, the result was applied to the original dataset, resulting in a subset of features deemed relevant for domain clustering.
The selected features include:

- _c_appdataB
- _c_cwin_ini
- _c_msgsize1
- _c_mss_min
- _c_pkts_data_avg
- _c_pktsize1
- _c_rst_cnt
- _c_rtt_std
- _c_ttl_max
- _c_win_min

- _s_appdataB
- _s_msgsize1
- _s_mss_max
- _s_pkts_data_avg
- _s_pktsize1
- _s_sit_std
- _tls_session_stat

3.2 | K-Means

In leveraging clustering algorithms such as K-Means, understanding the influence of different algorithm initialization approaches on clustering outcomes is imperative. In this initial phase, we focused on the analysis between two distinct initialization methods for the K-Means algorithm: 'k-means++' and 'random'. The primary objective is to evaluate how these initialization strategies impact the resulting cluster structure. This evaluation is conducted through a comprehensive examination of both inertia and silhouette scores across a spectrum of cluster numbers.

Figure 3.2.1: Elbow analysis and Silhouette (K-Means++)

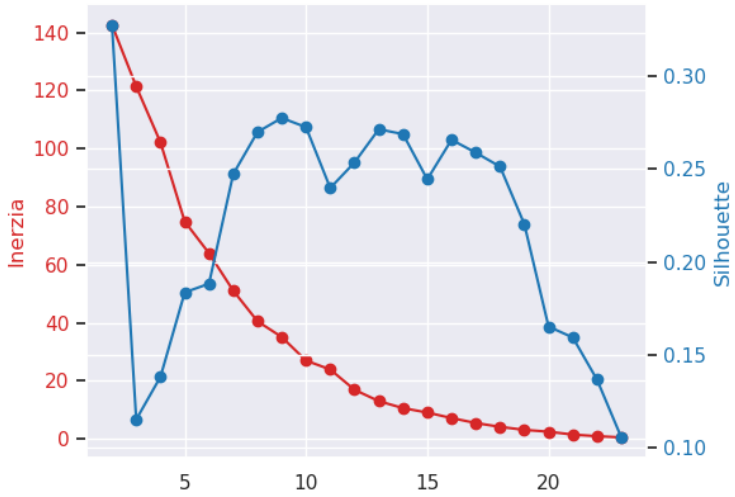


Figure 3.2.2: Number of clusters

Figure 3.2.3: Elbow analysis and Silhouette (Random)

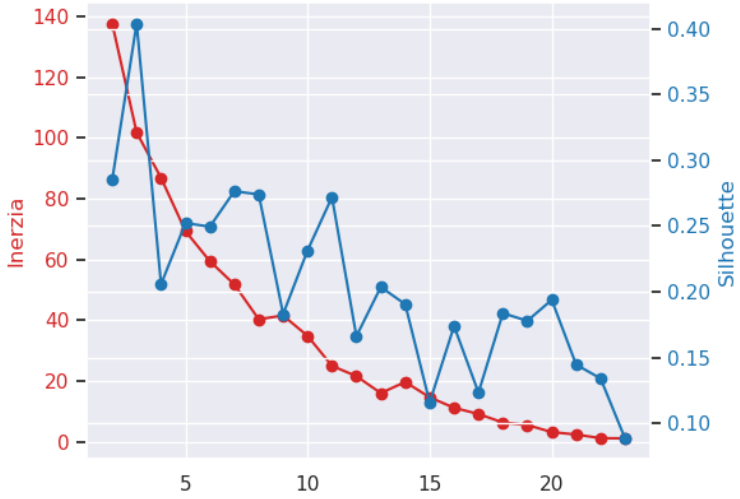


Figure 3.2.4: Number of clusters

From the above graphs, it can be inferred that the trend of inertia values is nearly similar in the case of using kMeans++ and Random. However, due to the shape of the elbow curve, identifying the elbow point is not straightforward. Regarding the silhouette values for the two initializers, they exhibit a generally different behavior, random shows a good value for silhouette at number of cluster equal to 3, while kmeans++ show a particular behaviour when it has a good value for 2 cluster but it than instantly drop to a bad value at 3 clusters, for then improving again.
Below an exhaustive search for the optimal parameters for the KMeans clustering algorithm is conducted. Various combinations of parameters, including the number of clusters (`n_clusters`), initialization method (`init`), number of initializations (`n_init`), maximum number of iterations (`max_iter`), tolerance value (`tol`), and algorithm (`algorithm`), are explored . Therefore, it was possible to find the **best parameters**: {'n_clusters': 3, 'init': 'k-means++', 'n_init': 10, 'max_iter': 300, 'tol': 0.001, 'algorithm': 'lloyd'} and the **best silhouette score**: 0.40396646567546524. This is interesting compared to what said before, silhouette for k-means++, 3 number of clusters and default parameters gave us the lowest silhouette score, while a search for the best parameter, improved significantly giving us the best score.
Additionally, a KMeans instance (best_kmeans) is created using the best parameters, and the data (X) is clustered with these optimal parameters. The resulting cluster labels are stored in the labels variable.

3.2.1 | Detected Clusters

Below, we present a coarse analysis of the detected clusters. As the first step, the ECDF of domain names per cluster was analyzed, provides a visual representation of data variability across the different groups identified by the clustering algorithm. The plot allows to analyze the distribution patterns and to understand how domain names are spread within each cluster.

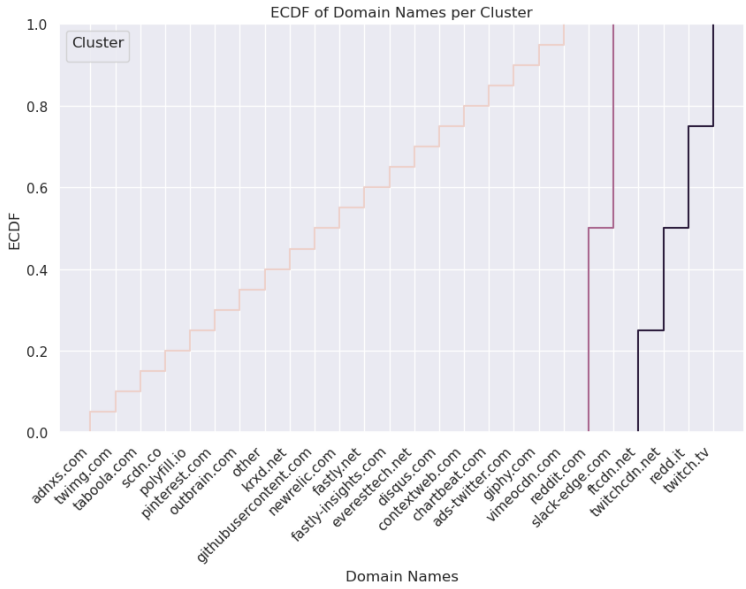


Figure 3.2.5

As observed from the figure above, three distinct clusters can be identified. Additionally, we can approximately classify 'families' of domain names; for example, the cluster consisting of sites from 'adnxs.com' to 'vimeocdn.com,' inclusive of the domain name 'other,' can be categorized as generic traffic, encompassing diverse types. Furthermore, within the cluster composed of 'reddit.com' and 'slack-edge.com,' a potential classification related to messaging sites emerges. This is supported by Slack being a business collaboration tool for instant messaging within teams, and Reddit serving as a platform primarily for reading and contributing relevant content to various discussions. Lastly, the cluster comprising 'fdcdn.net,' 'twitchcdn.net,' 'redd.it,' and 'twitch.tv' is associated with the entertainment domain: 'fdcdn.net' and 'twitchcdn.net' are both multimedia content distribution sites, while 'redd.it' and 'twitch.tv' are social media platforms focusing on multimedia content.

Then a Principal Component Analysis (PCA) was performed on the dataset for clustering analysis. The PCA is configured to reduce the dataset to two principal components, providing a lower-dimensional representation while retaining as much of the original variability as possible. The cumulative explained variance is calculated to understand how much of the dataset's variability is retained by the selected number of components.

With two principal components, 54.09% of the variance is explained. The transformed PCA results are then organized into a Dataframe, where each domain name is associated with its corresponding cluster assignment and the values of the two principal components. The Dataframe is visualized using a scatter plot, where each point represents a domain name in the reduced feature space defined by the first two principal components. The points are color-coded based on their assigned cluster, allowing for a visual inspection of the separation and distribution of clusters in the reduced feature space.

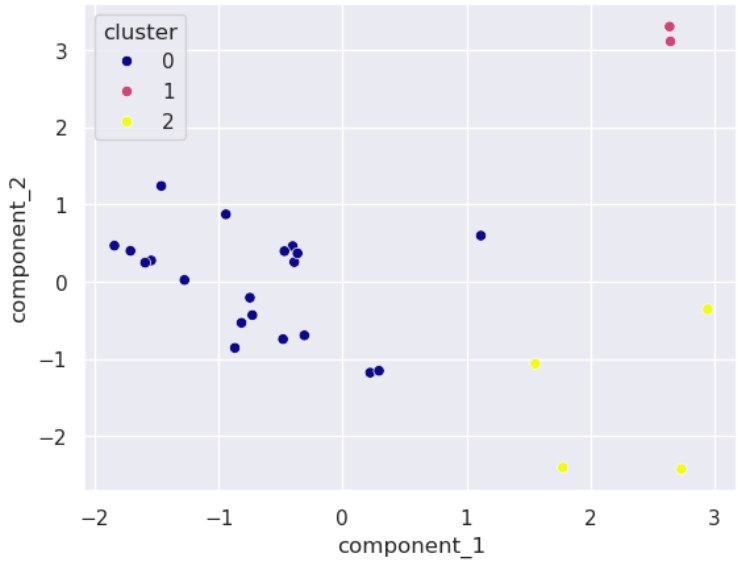


Figure 3.2.6

As evident from the figure above, it is easy to identify the separation into three distinct clusters. These clusters appear reasonably separated from each other. However, it is noteworthy that especially in the cluster with the highest number of elements (Cluster 0), the points do not appear entirely compact. In fact, some points seem to be distant from each other.

3.2.2 | Cluster Evaluation Metrics

The evaluation metrics provide insights into the performance of the clustering algorithm on the dataset:

- Davies-Bouldin index: 1.0581546277061933
- Silhouette index: 0.40396646567546524
- Calinski-Harabasz index: 9.123843297228461

The silhouette score of 0.40396646567546524 indicates a commendable level of cohesion within clusters, suggesting that the data points within each cluster are well-matched. Simultaneously, the Davies Bouldin score of 1.0581546277061933, being relatively low, signifies effective separation between the clusters, providing confidence in the robustness of the clustering results.
In fact, examining the cluster composition, it's evident that each cluster encapsulates a distinct set of domain names. The separation is most notable in the cluster with the highest number of elements, where data points, while cohesive within the cluster, exhibit noticeable distances from each other (Figure 3.2.6). This aligns with the silhouette and Davies Bouldin scores, emphasizing the trade-off between intra-cluster cohesion and inter-cluster separation.
We have taken the best 10 features considered by the random forest and we have created a pair plot, in some of this graph we can observe the distinction, for example in the graph related to _s_pcksize1 we can see the all the points contained in the second cluster are on the bottom of the graph, while the outer points are at the top. Another feature where we are able to see some separation is _s_msgsize1. It can be observed that the distribution of points from different clusters appears well separated and exhibits a clearly defined behavior. It remains consistent across _c_msgsize1, _c_win_ini, and c_pktsize1 in relation to _tls_session_stat. This suggests that the characteristics related to message sizes, packet sizes, and initial congestion windows of clients maintain a consistent relationship with the state of the TLS (Transport Layer Security) session. It indicates a significant correlation between these specific network parameters and the TLS security protocol.

3.3 | Spectral Clustering

For the selection of the number of the cluster for our spectral clustering who analyzed the silhouette score and the gap statistics. A low gap value indicates that the within-cluster dispersion of the data is close to what would be expected by random chance. Looking at the plot (Fig: 3.3.1) we can see that the numbers 2 gives us the best silhouette scores and one of the lowest gap values.

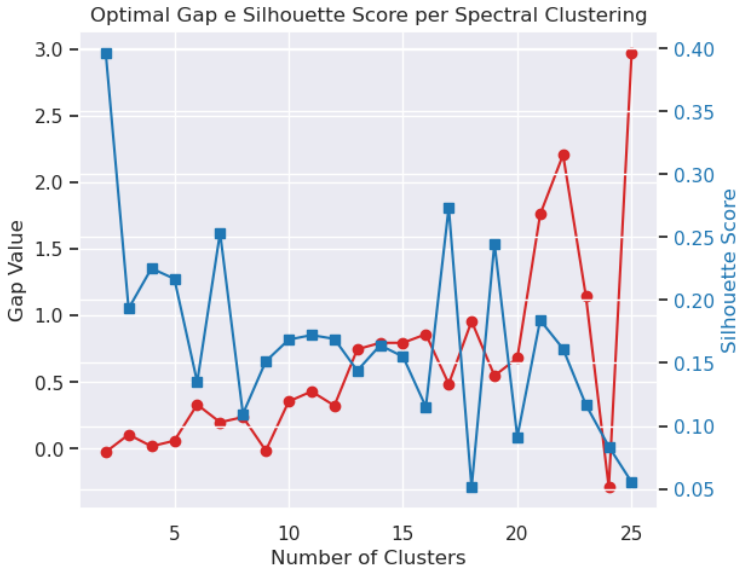


Figure 3.3.1

We have performed a tuning of the parameters but the value of the silhouette score hasn't increased. Obtaining the best results, the optimal parameters are found to be: {'n_clusters': 2, 'n_init': 10, 'gamma': 1.0, 'affinity': 'rbf', 'assign_labels': 'discretize'}, with an highest silhouette score achieved of 0.39640900552644964.

3.3.1 | Detected Clusters

We can see from the ECDF that the the smallest cluster is composed by slack-edge.com and reddit.com, as we said in the previous point, they are website used for exchanging long paragraph of text, while the largest cluster, consisting of all the other domains, can be generically classified as generic traffic.

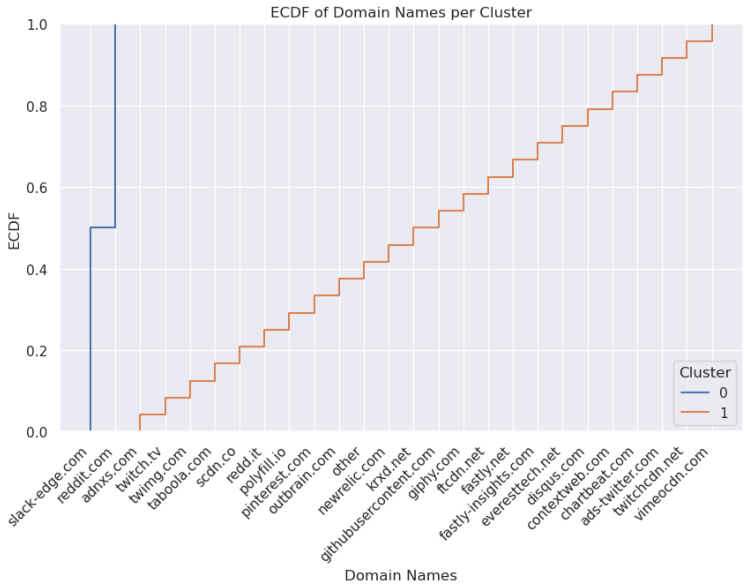


Figure 3.3.2

- Davies Bouldin score: 0.6180228930778464
- Silhouette score: 0.39640900552644964
- Calinski Harabasz score: 6.531913561493328

The scatter plot produced by the 2 principal components underline the clear separation of this cluster from all the other points (Fig: 3.3.3). We can also see how dispersive the other cluster is, so we tried as expressed in the subsection below to run again the clustering algorithm on this cluster to see if we can have a better separation.

Figure 3.3.3

The pair plot shows us some separation in the same feature as k-means cluster and in some other like _c_cwin_ini.

3.3.2 | Subclustering

The silhouette score gives us a better result with a number of cluster of 6 (Fig: 3.3.4), but when we tune the hyperparameters this value change, and the silhouette score for 2 number of cluster goes from 0.15 to 0.30.

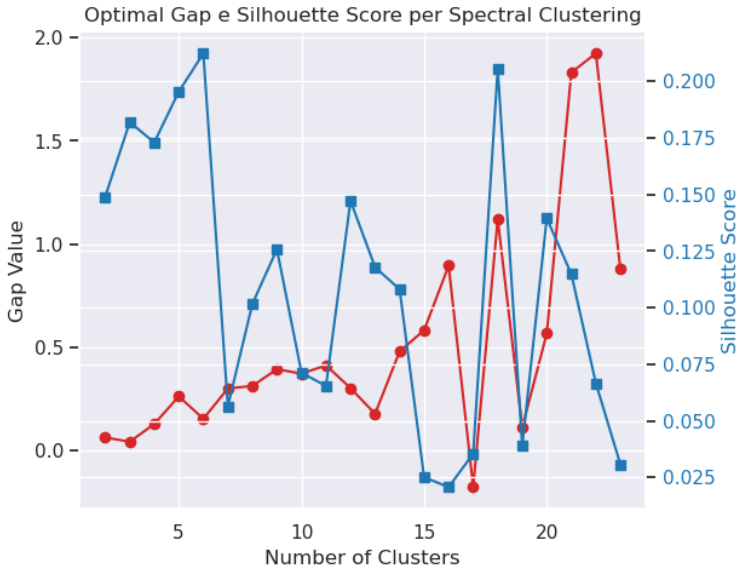


Figure 3.3.4

Our ECDF shows us that one other cluster has been found with twitch.tv, adnxs.com, twitchcdn.net, giphy.com, ftcdn.net, redd.it, vimeocdn.com; based on the analysis, the cluster could be classified as "generic online entertainment" noting that within it, sites like 'adnxs.com' and 'twitchcdn.net' are often associated with advertising and tracking services.

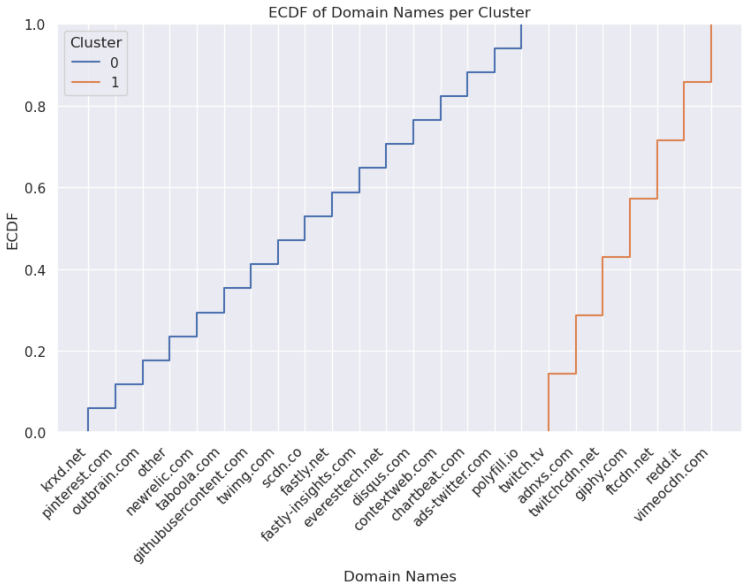


Figure 3.3.5

Observing the scatter plot generated by the 2 principal components we have a clear separation of the clusters as suggested by the Silhouette values and the low Davies Bouldin scores (although worse than before the

subclustering operation), even the points are still quite sparse.

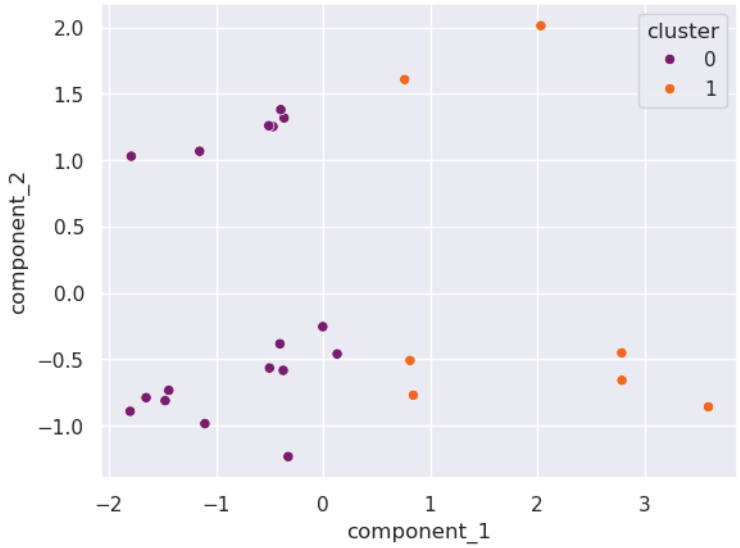


Figure 3.3.6

- Davies Bouldin index: 1.5755885690001772
- Silhouette index: 0.3008403329947371
- Calinski Harabasz index: 8.260467782751327

The pair plot of this subclustering is most interesting in the feature _s_appdataB, we notice that for (Number of bytes of application data sent by the server), the clusters appear well-separated, generally occupying half of the plot each.

3.4 | OPTIC-DBSCAN

Spectral and K-means are clustering algorithms that require the number of parameters as input. We choose a clustering algorithm that give us the number of cluster as output so we could confront our results.
We used the `compute_optics_graph` function to calculate crucial components of the OPTICS algorithm:

- Ordering: The ordering of points based on their density.
- Core Distances: The distance to the k-th nearest neighbor for each point.
- Reachability: A measure of the connectivity between points.
- Predecessor: Information about the predecessor of each point in the reachability plot.

We have chosen the `eps` parameter, representing the maximum distance for two points to be considered neighbors, to be set at 3.0, in order to obtain the maximum value of the average cluster density.

3.4.1 | Detected Clusters

The number of cluster selected by the algorithm is 2 plus a noise cluster where it puts the data that can't classify (Fig: 3.4.1). We can observe that it has divided the data in the exact same cluster we have identified with k-means clustering, and also spectral clustering, if we exclude the cluster classified as noise.

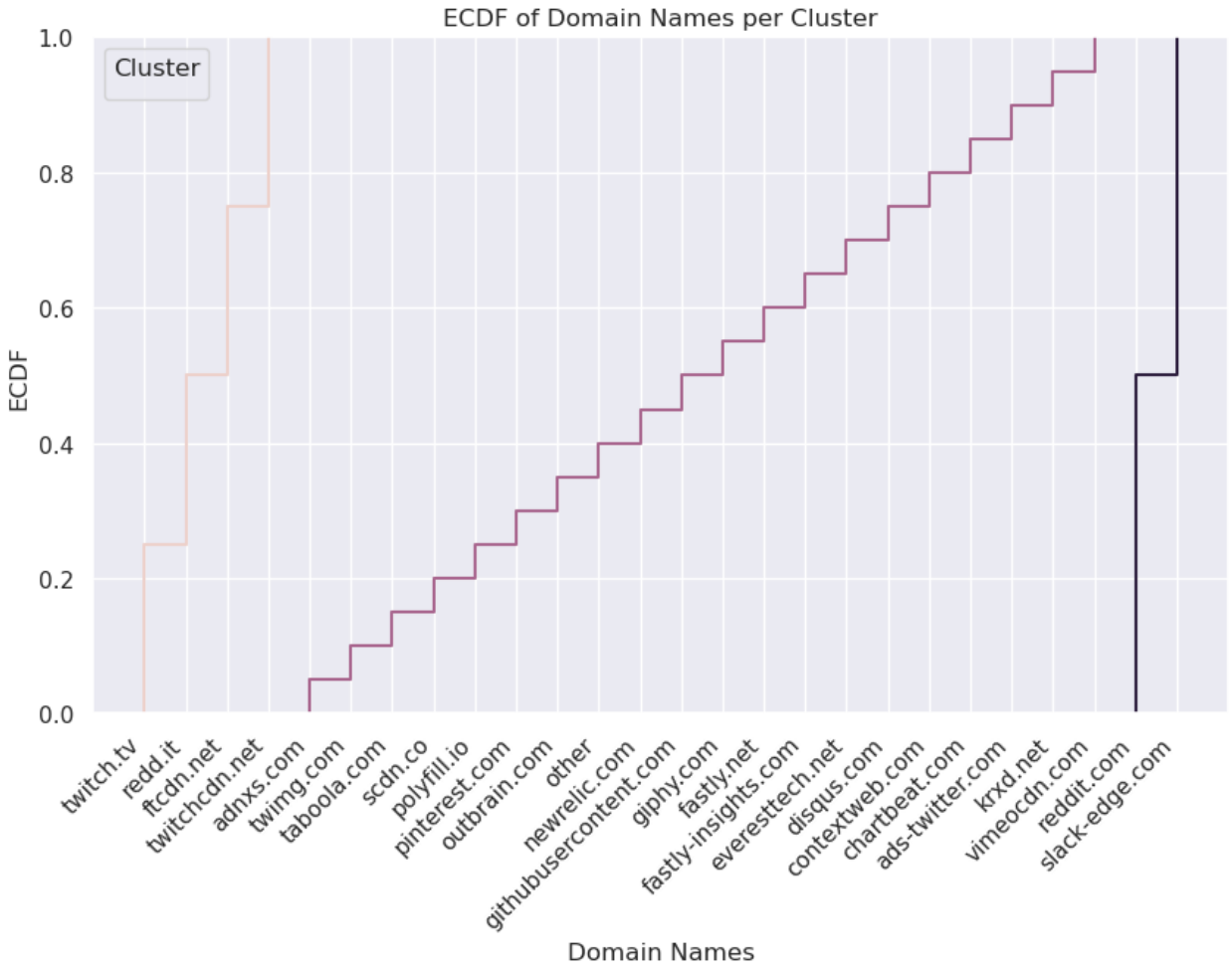

ECDF of Domain Names per Cluster
Cluster
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
twitch.tv
redd.it
ftcdn.net
twitchcdn.net
adnxs.com
twimg.com
taboola.com
scdn.co
polyfill.io
pinterest.com
outbrain.com
other
newrelic.com
githubusercontent.com
giphy.com
fastly.net
fastly-insights.com
everesttech.net
disqus.com
contextweb.com
chartbeat.com
ads-twitter.com
krxd.net
vimeocdn.com
reddit.com
slack-edge.com
Domain Names

Figure 3.4.1

Figure 3.4.2

The scatter plot generated by the 2 principal components highlights, as previously observed in the two analyzed algorithms, the distinct separation of clusters (Fig: 3.4.2). Furthermore, there is an improved compactness of the clusters, as indicated by the optimal Silhouette and Davies Bouldin scores obtained.

- Davies Bouldin index: 1.0581546277061933
- Silhouette index: 0.40396646567546524
- Calinski Harabasz index: 9.123843297228461

3.5 | Cluster evaluations

Davies-Bouldin Index
The Davies-Bouldin Index measures the compactness and separation of clusters obtained through the clustering process. The lower it is, the better the clustering result, indicating that clusters are compact and well-separated. The index is calculated based on pairwise comparisons between clusters. For each cluster, the Davies-Bouldin Index considers the ratio of the sum of the within-cluster distances to the distance between the centroid of that cluster and the centroid of the cluster with which it has the highest similarity.
Silhouette Score
Silhouette provides a measure of how well-separated the clusters are. The silhouette score for a data point is a measure of how similar it is to its own cluster (cohesion) compared to other clusters (separation). The

Table 3.5.1: Metrics values for K Means, Spectral Clustering e OPTIC-DBSCAN

Algorithm	Davies Bouldin	Silhouette	Calinski Harabasz
K Means	1.058	0.404	9.124
Spectral Clustering	0.618	0.396	6.532
OPTIC-DBSCAN	1.058	0.404	9.124

silhouette score ranges from -1 to 1. A high silhouette score indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most data points have a high silhouette score, it suggests that the clustering configuration is appropriate. The overall silhouette score for the entire clustering is then the average of the silhouette score for each instance in the dataset.

Calinski-Harabasz Index

The Calinski-Harabasz Index is another metric used for evaluating the quality of clustering in unsupervised machine learning. It assesses the ratio of between-cluster variance to within-cluster variance, providing a measure of how well-separated the clusters are. The goal is to maximize the Calinski-Harabasz Index, as a higher value indicates better-defined and well-separated clusters.

The first thing that comes to mind looking at the table is the fact that K Means and OPTICS-DBSCAN have the same value for each row; this is due to the fact that we obtained the exact same subdivision into clusters. Now, let's compare K Means and Spectral Clustering. The first thing we have to take into account is that K Means was computed with 3 clusters, while Spectral Clustering with 2.

The Davies-Bouldin Index has a good value in both models, suggesting good cluster separation with some compactness. The better result in Spectral Clustering might be due to better separation with only 2 clusters. As we have seen in the PCA scatter plot, there are 2 points belonging to one cluster, and they are far from all the other points, while K Means has 3 clusters, and clusters 0 and 2 are not as far apart as cluster 1.

The silhouette score is quite similar for both algorithms. We can hypothesize that the difference we have previously seen in DBI can't be observed by the silhouette score. This is because K Means has a higher number of clusters and makes a good division, splitting the bigger cluster into 2. The points are closer to each other, generating a high silhouette score for those points, and being an average, it improves the result.

The values found for Calinski-Harabasz suggest a good partition, with K Means having a considerably better one. This might also explain the difference in DBI and silhouette score.

Chapter 4

Regression – Estimate bytes transmitted and round trip time

This section focuses on regression analysis involving two key variables: *s_bytes_all* and *s_rtt_avg*.

- **s_bytes_all** Represents the total number of bytes transmitted from the server to the client in the payload, including retransmissions (measured in bytes).
- **s_rtt_avg** Indicates the average Round-Trip Time (RTT) from the server to the client, calculated by measuring the time elapsed between the data segment and the corresponding ACK (measured in milliseconds).

For each of the two regression tasks, the following steps are followed:

- **Preprocessing** (normalization and data reduction)
- **Model Training**
- **Hyper-parameter tuning**
- **Performance Analysis**

Let's briefly describe the steps and the models used to perform the tasks.

4.1 | Preprocessing

For each task, we followed some steps in order to prepare the data to the application of the Machine learning models.
First of all, the following rules have been respected:

- for the task related to s_bytes_all, the feature *s_bytes_uniq* has been removed;
- for the task related to s_rtt_avg, all the columns related to other measures of RTT have been removed, in particular: *c_rtt_avg, c_rtt_cnt, c_rttmax, c_rtt_min, c_rtt_std, s_rtt_cnt, s_rtt_max, s_rtt_min, s_rtt_std.* After this procedure, all the samples where the value of s_rtt_avg was equal to 0, have been deleted.

Then, the following steps have been followed for both the tasks:

- **Feature standardization** thanks to sklearn.preprocessing.StandardScaler, that standardize the features by removing the mean and placing the standard deviation equal to one.
- **Feature reduction**: first of all, features with standard deviation equal to zero have been dropped, then a correlation analysis has been performed, and features with a correlation either equal or greater than 0.8 have been removed.

As result of the preprocessing phase, we obtained one dataset for each task respectively with 61 features (for the task related to the RTT) and 65 features (for the task regarding the Bytes). A good result considering that at the beginning we had more than 100 features for each dataset.
Now we are ready to apply the models and perform the regression tasks.

4.2 | Model Training and Hyper-Parameters tuning

In this section we have on a first attempt, evaluated the performance of 3 ML models, with default parameter configuration, on both train and test set, and then we have tuned the hyper-parameters of the models through the analysis of the validation curve, recording the variation of the performances.
Let's now briefly describe the models that we have used.

4.2.1 | The models

Lasso Regressor

Lasso (Least Absolute Shrinkage and Selection Operator) is a linear regression technique used for regression tasks.
An important feature of the LASSO regressor is the so called **regularization technique**, which is a penalty term in the loss function.
This penalty term is used to promote sparsity and preventing overfitting, and it is useful in situations where there is a huge number of irrelevant features, indeed it works also as a feature selection.
The strength of the regularization is controlled by the hyperparameter alpha: a higher alpha leads to stronger regularization, potentially resulting in more coefficients being set to zero.

Decision Tree

A Decision Tree regressor is a powerful machine learning algorithm that uses a tree-like structure to preform decisions. It can be represented as a flow chart where each internal node represents a condition test based on a specific feature, each branch represents the outcome of the condition test and the leafs are the final results. The tree is constructed through a recursive process of selecting the best features to split the data at each node, typically using metrics like mean squared error or mean absolute error. Decision Trees can handle both linear and non-linear relationships in the data, however, they are prone to overfitting.

Neural Network

A neural network is a computational model composed of interconnected nodes, also known as neurons or units, organized into layers. The primary components of a neural network are: input, hidden and output layer, the weights and the activation function.

4.2.2 | s_bytes_all Regression

Model Training with default hyper-parameters

By the application of the models (with the default hyper-parameters), we have obtained the following results:

- **Decision Tree Regressor**:
  - *Training Set*
    - **Mean Squared Error**: $2.32 * 10^{-17}$
    - **Mean Absolute error**: $1.87 * 10^{-10}$
  - *Test Set*
    - **Mean Squared Error**: 3.07
    - **Mean Absolute error**: 0.0078

- **LASSO Regressor**:
  - *Training Set*
    - **Mean Squared Error**: 0.99
    - **Mean Absolute error**: 0.149
  - *Test Set*
    - **Mean Squared Error**: 5.35
    - **Mean Absolute error**: 0.15
- **Neural Network**:
  - *Training Set*
    - **Mean Squared Error**: 0.006
    - **Mean Absolute error**: 0.013
  - *Test Set*
    - **Mean Squared Error**: 0.003
    - **Mean Absolute error**: 0.012

As we can see from the analysis of the metric scores, all the models have good performances both on training and test set, in particular related on mean absolute error.
But in this case and with this hyper-parameters, it is interesting to note that Decision Tree is prone to overfitting, indeed the performance on the training set are much more better than the performances on the test set, but also with the LASSO regressor, we have seen a certain level of overfitting, indeed there is a particular difference in the computation of mean squared error between train and test set.

Hyper-Parameters tuning

In this second part, it has been applied the Cross validation, in order to tune the hyper-parameters of the models and find the best combination that not only minimize the error in the train and validation set, but also reduce overfitting.

- **Decision Tree Regressor**
  - Validation curve:

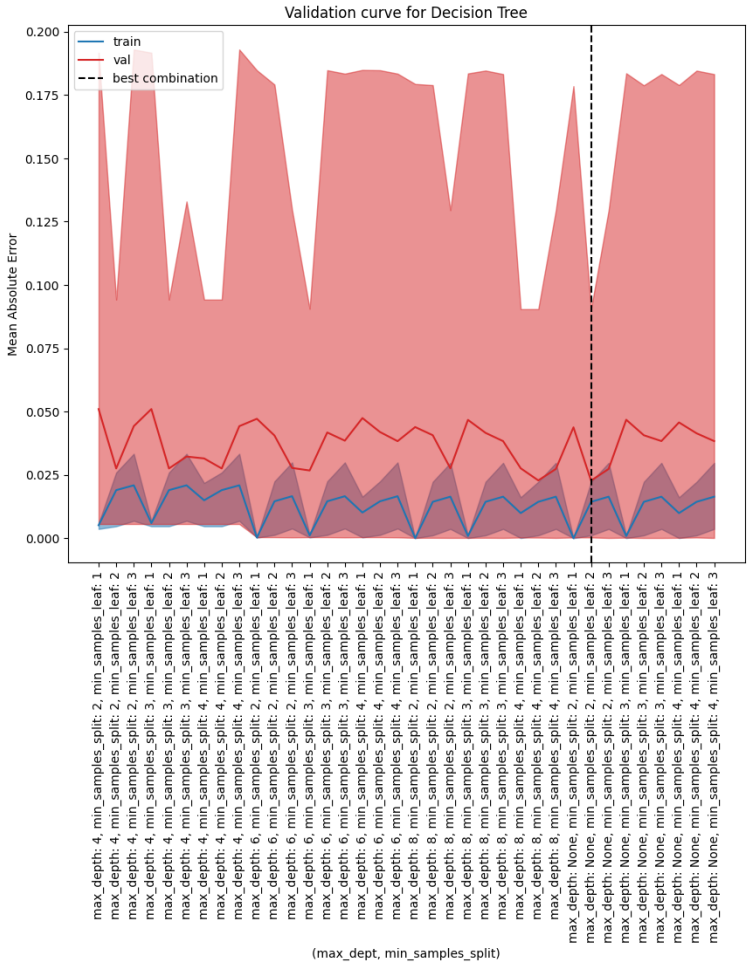


Figure 4.2.1

As we can see from the validation curve, There is a set couple of *max_depth* and *min_samples_split* and *min_samples_leaf*that not only reduce the error(in the validation set), but on the other hand, reduce also the overfitting. In order to support this assumption we can show the regression scores of the Decision Tree after the setting of the Hyper-parameters.

- *Training Set*
  - **Mean Squared Error**: 0.007
  - **Mean Absolute error**: 0.0004
- *Test Set*
  - **Mean Squared Error**: 2.89
  - **Mean Absolute error**: 0.0074

In this second case, the difference of performances between training and test set decrease, and in addition, the values of Mean squared error and Mean absolute error are still acceptable.

- **LASSO Regressor**
  - Validation curve:

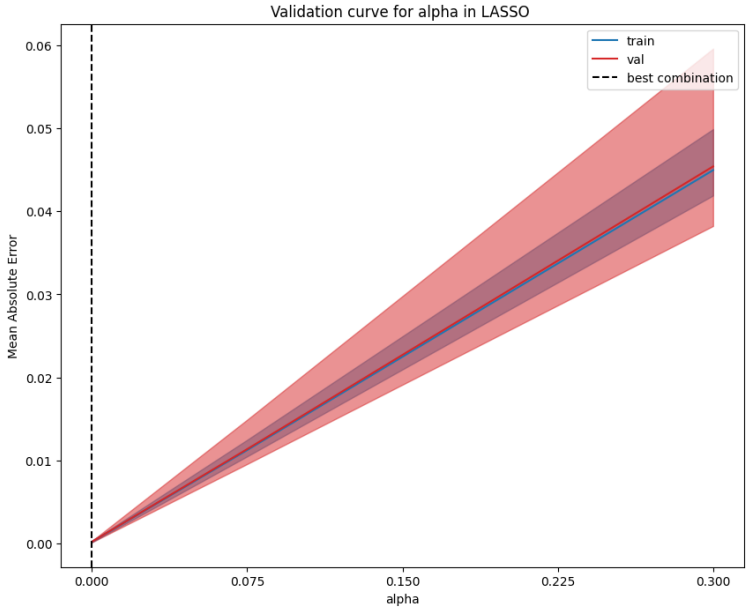


Figure 4.2.2

Here we have found a specific value of the hyper-parameter *alpha*, equals to 0, that minimize the errors as we can notice in the following regression scores, obtained after the application of LASSO regressor with that specific alpha. This is interesting because placing the parameter alpha equal to zero, means that we don't need a regularization parameter to improve the performances of the model and so, we can obtain better results just applying a linear regression.

  - *Training Set*
    - **Mean Squared Error**: $8.9 * 10^{-8}$
    - **Mean Absolute error**: 0.0001
  - *Test Set*
    - **Mean Squared Error**: $9.4 * 10^{-8}$
    - **Mean Absolute error**: 0.0001

Here not only the mean absolute error has improved, but also the Mean Squared error reach its minimum value and also it has been solved a problem related to the overfitting obtained in the previous application of the model with default settings.

- **Neural Network**
  - Validation curve:

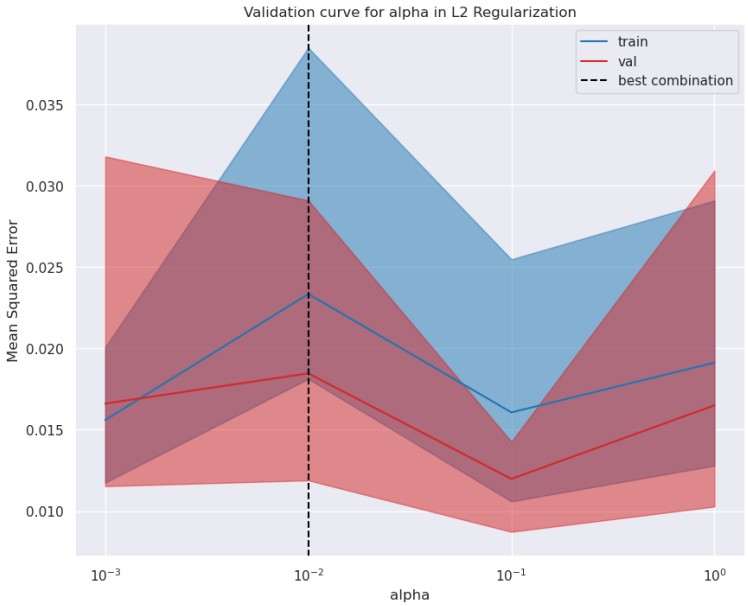


Figure 4.2.3

- *Training Set*
  - **Mean Squared Error**: 0.02
  - **Mean Absolute error**: 0.02
- *Test Set*
  - **Mean Squared Error**: 0.085
  - **Mean Absolute error**: 0.019

In the case of Neural Network, there are no improvements in terms of mean absolute and mean squared error, on the contrary, there are better performances with default parameters.

4.2.3 | s_rtt_avg Regression

By the application of the models (with the default hyper-parameters), we have obtained the following results:

- **Decision Tree Regressor**:
  - *Training Set*
    - **Mean Squared Error**: $2.59 * 10^{-17}$
    - **Mean Absolute error**: $2.46 * 10^{-9}$
  - *Test Set*
    - **Mean Squared Error**: 0.001
    - **Mean Absolute error**: 0.0005
- **LASSO Regressor**:
  - *Training Set*
    - **Mean Squared Error**: 0.99
    - **Mean Absolute error**: 0.25
  - *Test Set*
    - **Mean Squared Error**: 1.08
    - **Mean Absolute error**: 0.27
- **Neural Network**:
  - *Training Set*
    - **Mean Squared Error**: 0.002
    - **Mean Absolute Error**: 0.009
  - *Test Set*
    - **Mean Squared Error**: 0.03
    - **Mean Absolute Error**: 0.01

Similar to the Bytes regression task, here we have good performances from all models, with values of mean squared error and mean absolute error close to zero. Also in this case and with default hyper-parameters, we ave noticed an high level of overfitting with the Decision Tree, indeed the performance on the training set are much better than the performances on the test set.

Hyper-Parameters tuning

Also In this second part, in the same way of the previous regression task, it has been applied the Cross validation, in order to tune the hyper-parameters of the models and find the best combination not only minimize the values of mean absolute error and mean squared error, but also to minimize the overfitting as in the case of Decision Tree.

- **Decision Tree Regressor**
  - Validation curve:

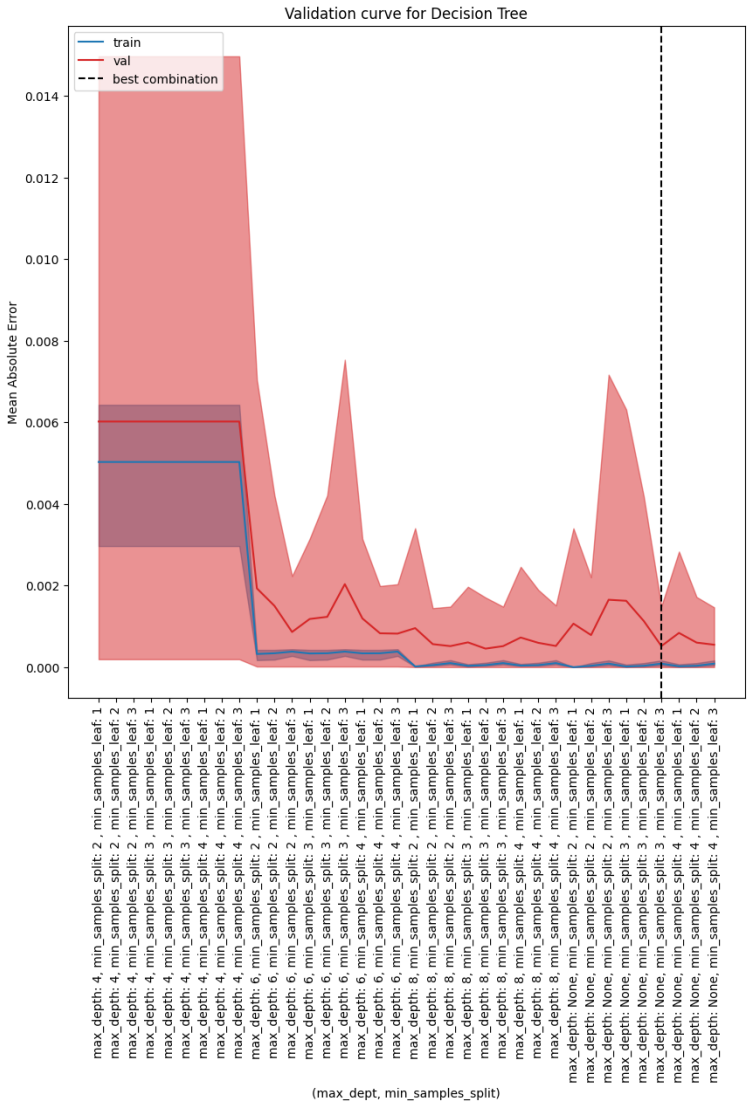


Figure 4.2.4

By the analysis of the validation curve, we can notice that there is a set of *max_depth* and *min_samples_split* and *min_samples_leaf*, respectively equals to "None", 3 and 3, not only reduce the overfitting, but also, improve the performances in terms of Mean squared error and mean absolute error, on the test set, as we can see in the following values.

  - *Training Set*
    - **Mean Squared Error**: $3.57 * 10^{-5}$

- **Mean Absolute error**: 0.0001

- *Test Set*
  - **Mean Squared Error**: 0.001
  - **Mean Absolute error**: 0.0004

In this case, even if the performances on the training set are not improved (but they are still very good), we have a lower level of overfitting and better values of Mean squared error and mean absolute error on the test set.

- **LASSO Regressor**
  - Validation curve:

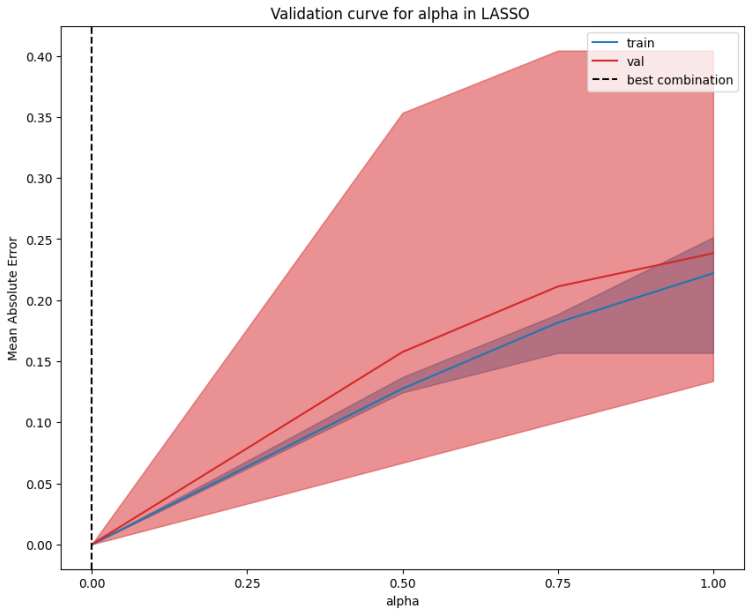


Figure 4.2.5

Thanks to cross validation, also in this task, we have a minimum point in the validation curve, in correspondence on a value of alpha equal to 0, So we can conclude that the performances of a Logistic Regression are better.

  - *Training Set*
    - **Mean Squared Error**: $1.2 * 10^{-7}$
    - **Mean Absolute error**: 0.0001
  - *Test Set*
    - **Mean Squared Error**: $1.15 * 10^{-7}$
    - **Mean Absolute error**: 0.0001

- **Neural Network**
  - Validation curve:

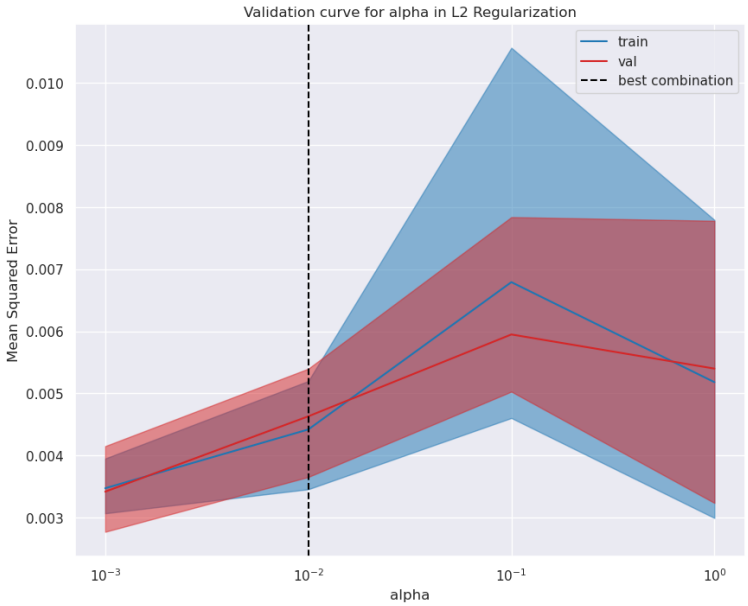


Figure 4.2.6

- *Training Set*
  - **Mean Squared Error**: 0.002
  - **Mean Absolute error**: 0.014
- *Test Set*
  - **Mean Squared Error**: 0.006
  - **Mean Absolute error**: 0.01

4.3 | Conclusion

In conclusion, the models performed very well in both problems, demonstrating mean absolute error values close to zero. From a comprehensive analysis of the results, it can be asserted that the prediction of s_rtt_avg seems to be an easier problem to solve, as the errors were slightly lower compared to the other task. In addition, the Linear Regression model(LASSO with alpha=0), emerged as the most effective, showing better performances than the other models for both regression problems. It is interesting that, for Lasso, the mean squared error is better than mean absolute error, while the opposite holds true for the other two models. Going into the specifics of each model:

- **Decision Tree**: While showing a relatively high Mean Squared Error (in the task related to s_bytes_all), it demonstrated reasonable absolute errors. Particularly, in the context of s_rtt_avg, the Decision Tree displayed minimal errors;
- **Lasso Regressor**: This model consistently showcased exceptional performance with extremely low values for both mean squared error and mean absolute error, indicating a robust adaptation of the model to the test data;
- **Neural Network**: this model performed moderately in the context of s_rtt_avg, with errors slightly higher than the other models; while in the further task, the model's performance fell between the other two, showing higher errors than Lasso, but lower than Decision Tree;

Metric	Value
Decision Tree	
Mean Squared Error	2.89
Mean Absolute Error	0.0074
Lasso Regressor	
Mean Squared Error	8.0×10^{-8}
Mean Absolute Error	0.0001
Neural Network	
Mean Squared Error	0.085
Mean Absolute Error	0.019

Figure 4.3.1: Metrics on Test Set on s_bytes_all

Metric	Value
Decision Tree	
Mean Squared Error	0.001
Mean Absolute Error	0.0004
Lasso Regressor	
Mean Squared Error	1.15×10^{-7}
Mean Absolute Error	0.0001
Neural Network	
Mean Squared Error	0.006
Mean Absolute Error	0.01

Figure 4.3.2: Metrics on Test Set on s_rtt_avg