PROJECT 3: HTTPS Domain Names

Riccardo Bruno, Federico Civitareale, Giuseppe Giordano, Vincenzo Longo

January 28th 2024

Contents

1 Data exploration and pre-processing

1.1 Data Exploration and Preprocessing

In this section data has been analysed at three different levels. Here the three working datasets used:

- *flow level* (initial dataset) **df_flow**
- *IP level* (clients) **df_client**
- *domain name level* (labels) **df_DomainName**

The latter two datasets have been derived from the initial dataset by merging the following features (particularly a subset of the total features has been selected):

- **_s_ack_cnt** whose sum has been performed
- **_c_appdataB** whose sum has been performed
- **_s_appdataB** whose sum has been performed
- **_c_bytes_all** whose sum has been performed
- **_s_bytes_all** whose sum has been performed
- **_c_bytes_retx** whose sum has been performed
- **_s_bytes_retx** whose sum has been performed
- **_c_cwin_max** whose max has been performed
- **_s_cwin_max** whose max has been performed
- **_c_cwin_min** whose min has been performed
- **_s_cwin_min** whose min has been performed
- **_c_fin_cnt** whose sum has been performed
- **_s_fin_cnt** whose sum has been performed
- **_s_rtt_avg** whose mean has been performed
- **_c_rtt_avg** whose mean has been performed
- **_c_rtt_cnt** whose sum has been performed
- **_s_rtt_cnt** whose sum has been performed
- **_c_rtt_max** whose max has been performed
- **_s_rtt_max** whose max has been performed
- **_c_rtt_min** whose min has been performed
- **_s_rtt_min** whose min has been performed
- **_c_rtt_std** whose std has been performed
- **_s_rtt_std** whose std has been performed
- **_c_syn_cnt** whose sum has been performed
- **_s_syn_cnt** whose sum has been performed
- **_durat_all** whose sum has been performed
- **_durat_mean** whose mean has been performed

1.2 Features Visualisation

For the visualization of the features the focus is placed on the ones related to the amount of the bytes exchanged by client and server (in both directions), the round-trip time and the duration of the connections. Here the analysed features:

Client

- **_c_appdataB** number of bytes of application data sent by the client
- **_c_bytes_all** number of bytes transmitted by the client
- **_c_rtt_avg** the average RTT from client to server

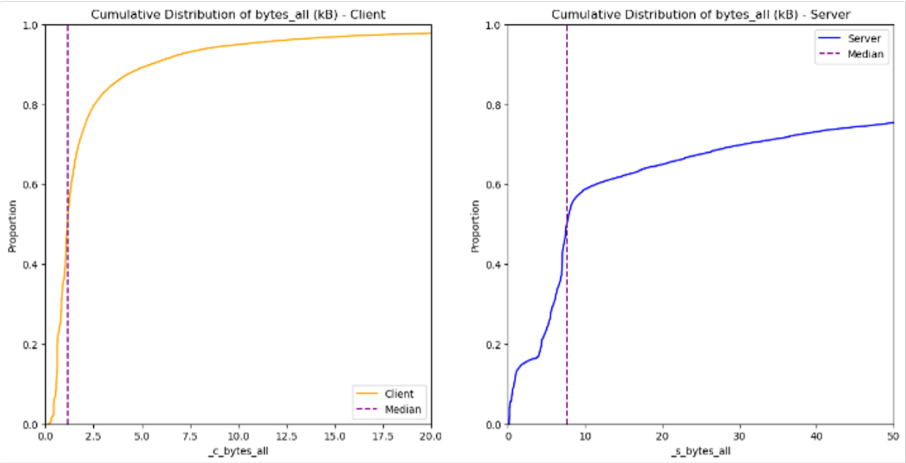
Server

- **_s_appdataB** number of bytes of application data sent by the server
- **_s_bytes_all** number of bytes transmitted by the server
- **_s_rtt_avg** the average RTT from server to client

General

- **_durat** duration of connections

ECDF and boxplots of client and server's transmitted bytes To understand how the communication between clients and servers works, the number of transmitted bytes was analysed by plotting the Empirical Cumulative Distribution Functions (ECDFs) of the features **appdataB** and **bytes_all** were plotted to visualize the distribution of data points. This analysis aimed to discern distinct behaviours between clients and servers by determining the median (the point at which 50% of the data lies below, it serves as a robust measure of central tendency) of each distribution.



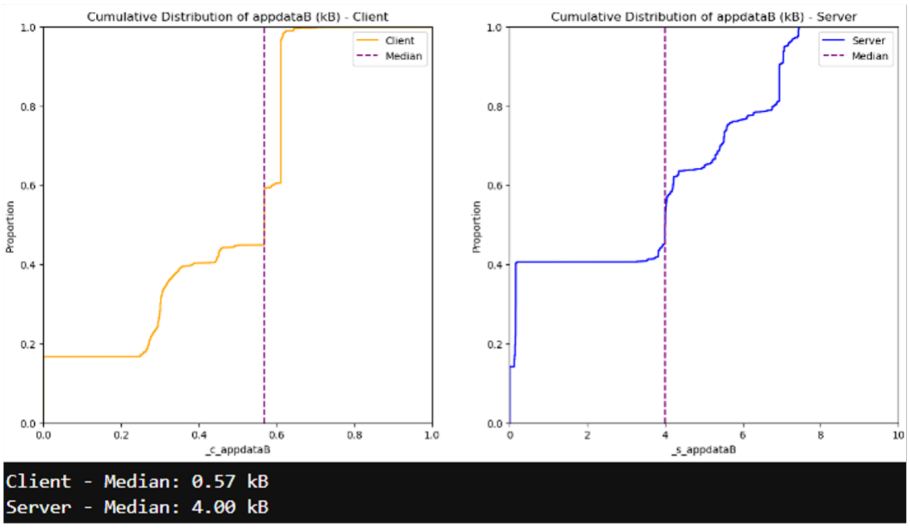
***ECDF:** bytes transmitted by client and server*

_c_bytes_all

- The median for the total bytes transmitted by clients is 1.14 kB. It means that approximately 50% of client interactions transmit 1.14 kB or less, emphasizing a concentration of smaller total data exchanges
- The ECDF illustrates a relatively steep initial ascent, suggesting that a significant proportion of client interactions involve smaller amounts of total data transmission

_s_bytes_all ECDF

- The median for the total bytes transmitted by servers is 7.62 kB. It means that around 50% of server interactions involve total data transmissions of 7.62 kB or less, indicating a larger typical payload size compared to clients
- The server ECDF exhibits a more gradual slope, indicating a broader range of total data transmission compared to clients



***ECDF**: bytes transmitted at application level by client and server*

_c_appdataB ECDF

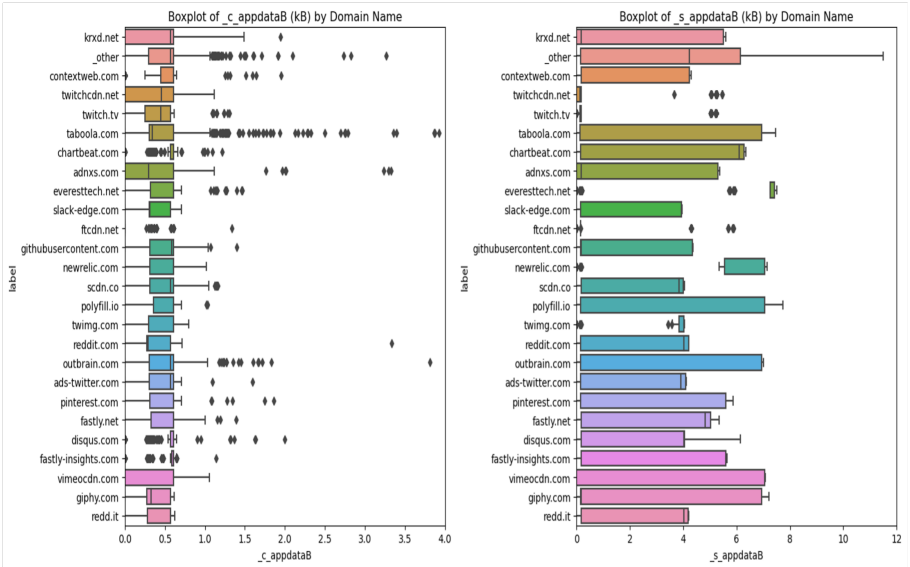
- The median for the bytes transmitted by clients at the application level is 0.57 kB. It means that approximately 50% of client interactions transmit 0.57 kB or less, highlighting a concentration of smaller data exchanges
- The ECDF shows a steep initial ascent, indicating that a significant proportion of client interactions involve relatively small amounts of data transmission

_s_appdataB ECDF

- The median for the bytes transmitted by servers at the application level is 4.00 kB: it means that around 50% of server interactions involve data transmissions of 4.00 kB or less, indicating a larger typical payload size compared to clients
- The server ECDF demonstrates a more gradual slope, suggesting a broader range of data transmission compared to clients

Summarizing, the difference in medians underscores the contrasting roles of clients and servers in the communication process, with clients often initiating smaller requests and servers responding with larger data payloads.

Subsequently, attention shifted to a more granular analysis by examining features per domain name. Particularly boxplots were generated by grouping the datapoints belonging to different labels in order to understand the behaviour of each service:



Boxplots** of the **number of bytes transmitted at application level** sent by clients and servers by **Domain Name

This kind of visualization provides a clear representation of the central tendency, spread, and potential outliers in the data for each domain name (label). The medians and interquartile ranges (IQR) offer insights into the 50% of the distribution, while the whiskers help identify the range of the majority of the data. Outliers, if present, can be identified beyond the whiskers.

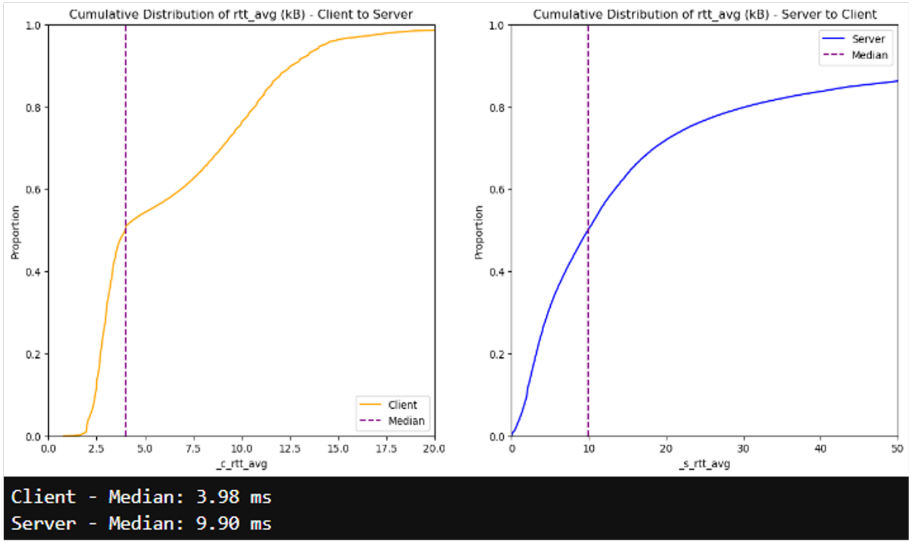
_c_appdataB boxplots per Domain Name

- These boxplots visvisualize the distribution of the application-level bytes transmitted by client (**_c_appdataB**) categorized by the domain name
- It can be noticed that clients behave quite in the same way when they contact each server. There are no significant differences in the amount of data sent into requests

_s_appdataB boxplots per Domain Name

- These boxplots illustrate the distribution of the application-level bytes transmitted by servers (**_s_appdataB**) categorized by the domain name
- It is evident that here the number of bytes at application level sent by the server to the clients depend on the type of the service offered (i.e. on the domain name). For example, social networks' servers such as *redd.it* (alias *reddit.com*) have a broader range of data transmission compared to hosting servers such as *twimg.com* (image-hosting domain) where there is a more specific range of data transmission, in this case geared towards efficiently delivering static image content

ECDF and boxplots of client and server's round-trip time The same kind of analysis was performed on the **rtt_avg** value from client to server and vice versa. Again, particular attention was given to the median value of each distribution, which provides valuable insights into the distribution of response times in both directions.



***ECDF**: average RTT from client to server and vice versa*

Client to Server avg_rtt ECDF

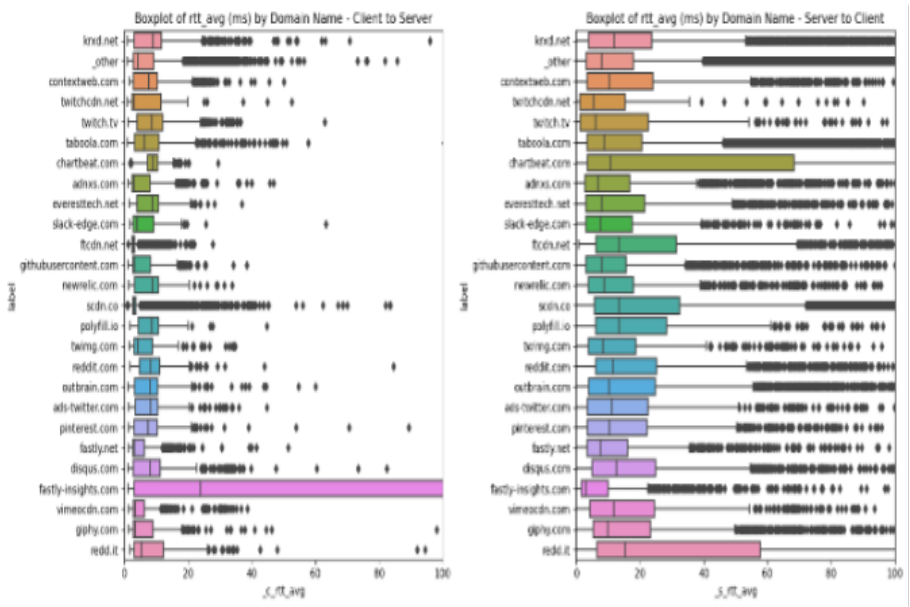
- The median for the **_c_avg_rtt** is 3.98 ms. It means that approximately 50% of client-to-server interactions have an **avg_rtt** of 3.98 ms or less, emphasizing a concentration of quicker response times.
- The ECDF indicates a relatively steep ascent initially, suggesting that a significant proportion of client-to-server interactions experience shorter round-trip times

Server to Client avg_rtt ECDF

- The median for the **_s_avg_rtt** is 9.90 ms. It means that around 50% of server-to-client interactions have an avg_rtt of 9.90 ms or less, suggesting a larger typical response time compared to the client-to-server direction
- The ECDF for server-to-client communication shows a more gradual slope, indicating a broader range of round-trip times compared to the client-to-server direction

The difference in medians emphasizes the asymmetry in round-trip times between client-to-server and server-to-client communication, which could be influenced by factors such as server processing and network latency.

Then, the average RTT in both the directions was analysed more in depth by plotting the boxplots per each domain name in order to better visualize the response times for each types of server:



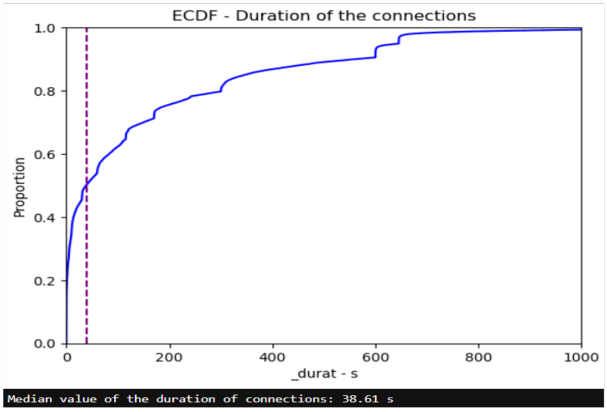
Client to Server

- The boxplots illustrate the distribution of average round-trip times (RTT) from clients to servers across the different domain names
- The majority of client-to-server RTTs appear to be concentrated within the lower range, with some variability observed among domain names
- The rightward skewness of some boxplots (e.g. *fastly-insights.com*) suggests that few domain names may experience longer RTTs in client-to-server communication

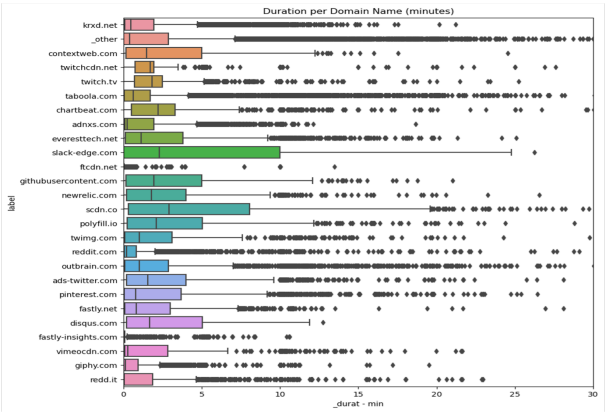
Server to Client

- The boxplot visualizes the distribution of average RTT from servers to clients for various domain names
- The majority of server-to-client RTTs fall in a broader range compared to the client-to-server one, with variations among domain names
- The rightward skewness indicates potential outliers and services with longer server-to-client RTTs

Duration of connections analysis



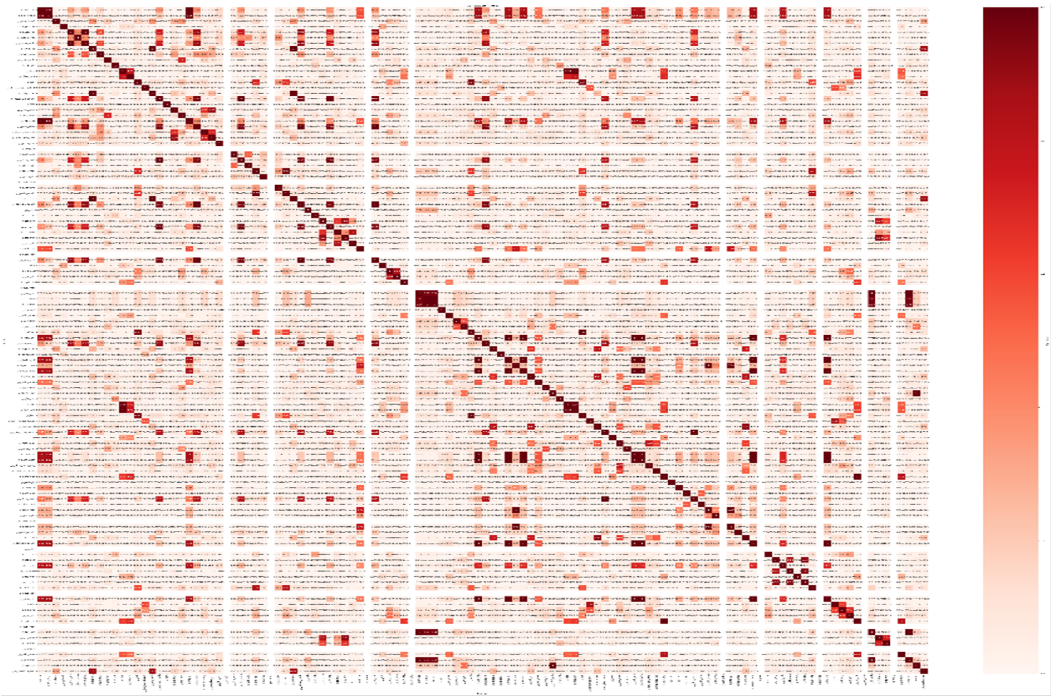
ECDF: *duration of connections*



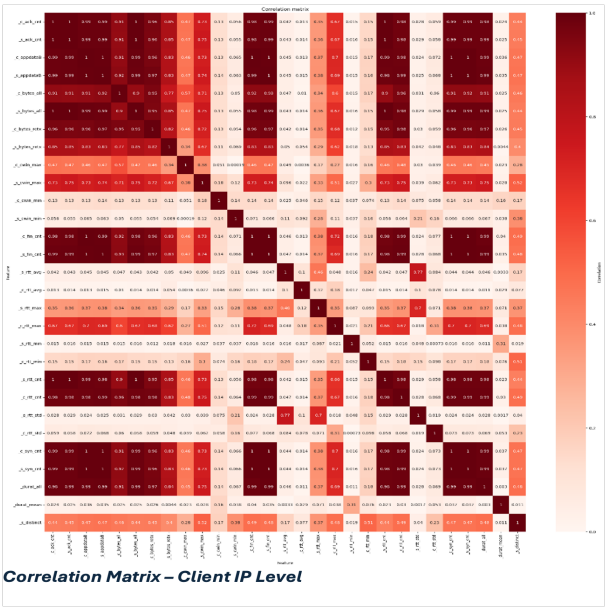
Boxplots per Domain Name – _durat

- The median for the duration of connections (**_durat**) is 38.61 sseconds. It means that approximately 50% of client-server interactions have a duration of 38.61 seconds or less, highlighting the prevalence of quick connections
- The ECDF shows a steep ascent initially, suggesting that a significant proportion of the connections have a short duration
- Subsequently, the curve starts to grow slowly suggesting that there is also a significant part of the connections that last longer
- Specifically, nearly the entire dataset displays durations of 1000 seconds or less
- The boxplots visualize the distribution of the duration of connections (now expressed in minutes) per domain names
- Most of the servers open connection with a duration that falls in a broader range (from few seconds to several minutes)
- A minority of all the servers opens connections which lasts few seconds and no more (except for some outliers). This is the case of some CDN's servers such as *fastly-insights.com* and *ftcdn.net*
- The rightward skewness indicates potential outliers and services with longer duration of connection

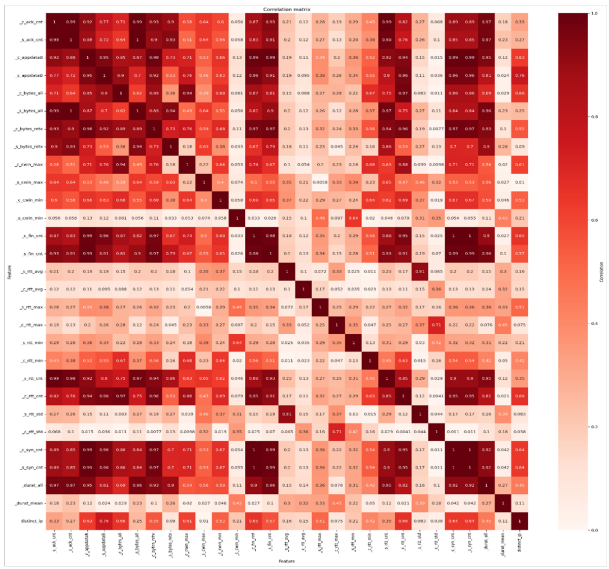
Most correlated fetures at the three levels For this purpose, each dataset has been first standardized and then its correlation matrix has been plotted. After the method *getMostCorrelatedFeature* has been implemented in order to get the pairs of highly correlated features (the threshold was fixed at 0.95). Finally the most correlated features have been printed (see on Project 3 – Section 1.ipynb file). Here the obtained correlation matrixes.



Correlation Matrix – Flow Level



Correlation Matrix – Client IP Level



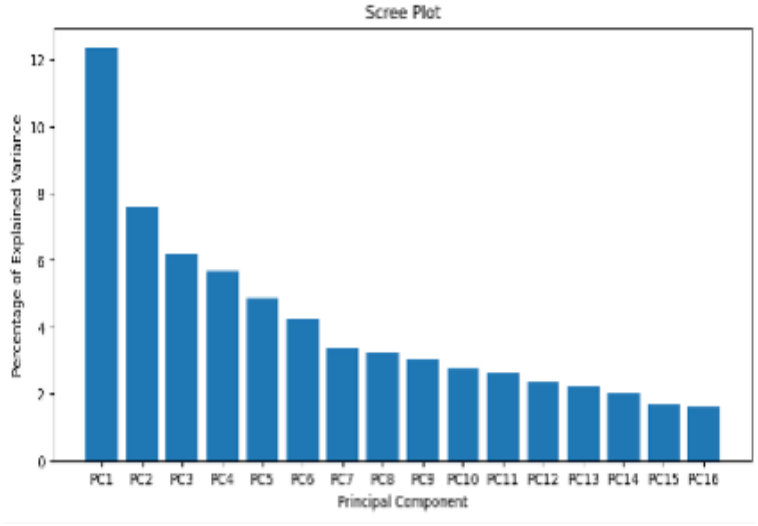
Correlation Matrix – Domain Name Level

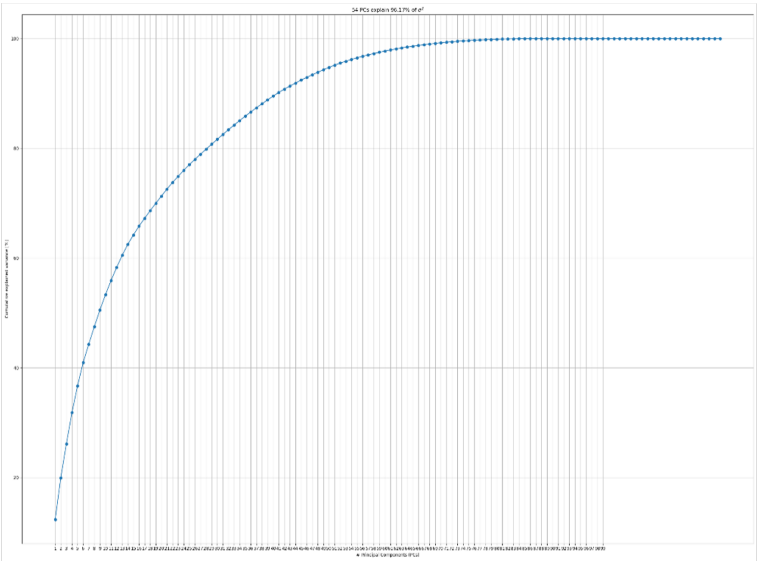
1.2.1 PCA at the three levels

At this point the PCA has been performed at the three levels, to reduce the dimensionality of the dataset. The dataset has been scaled in order to comply with the PCA's requirements, then a PCA object has been initialized with *random_state=15* and in the end the fit method was applied to it. After that the *explained_variance* has been retrieved from the output with the relative percentage, these values has been visualized by plotting a scree plot thanks to which it's possible to undestand the variance drop ratio among the principal components. Next, the *cum_exp_var* (cumulative explained variance) has been computed and visualized by plotting a simple curve. This kind of visualization allows to determine the number of principal components needed to retain an acceptable variance ratio (minimum 90%) of the original features of the dataset. In particular, this value has been chosen based on where there is a significant drop of the slope (elbow method). Once

decided the number of the principal components, the new dataset has been created.
Since then, the principal components are a linear combination of the original features, the loading scores of the first three PCs has been visualized. The procedure just described has been followed at each of the three levels:

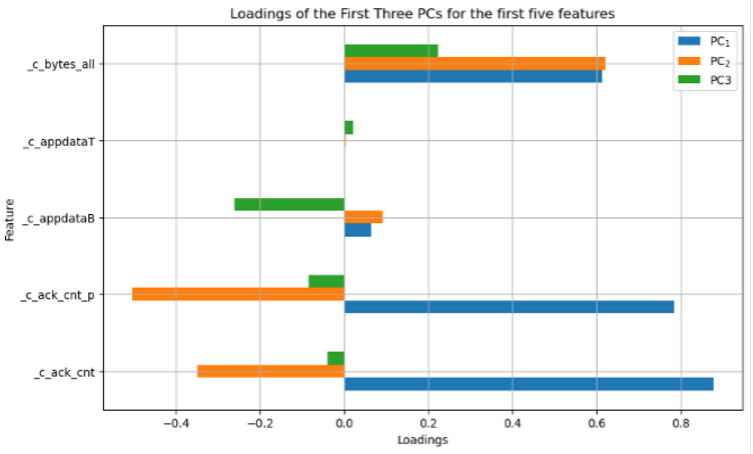
Flow level



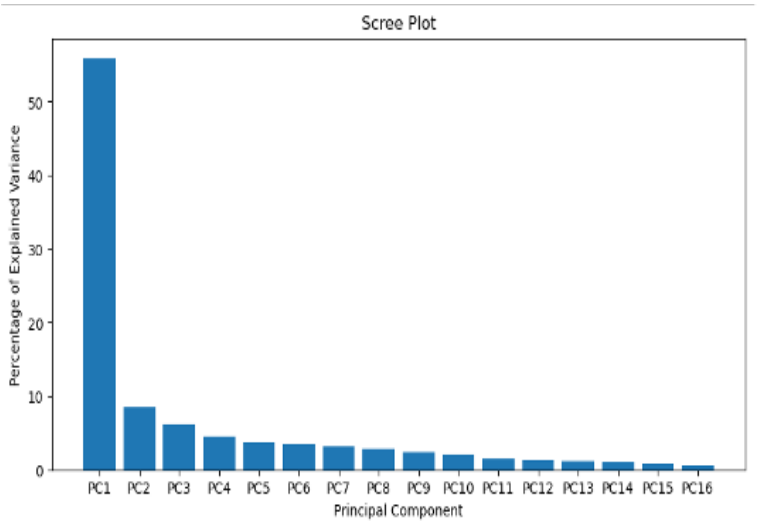


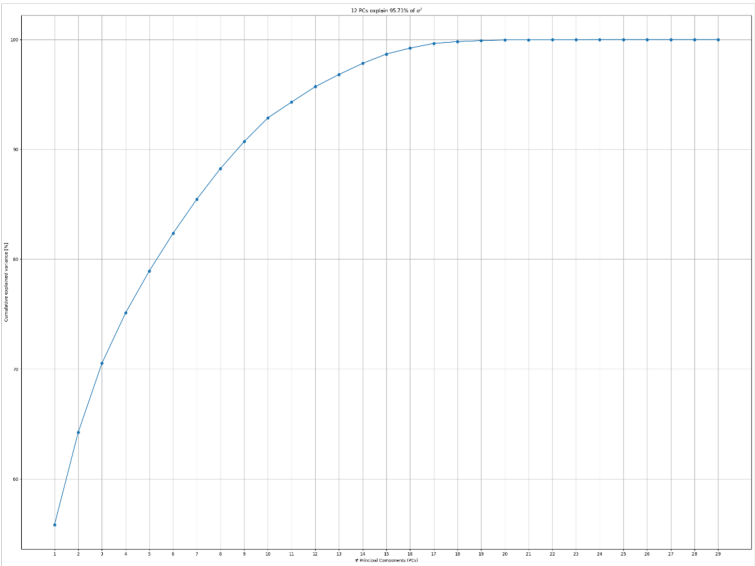
Cumulative explained variance visualization

After the cumulative explained variance analysis, 54 PCs has been selected to create the new dataset (they explain about 96% of the cumulative variance). In this case, the loading scores have been plotted only for the first five original features for a better visualization:



Client IP level

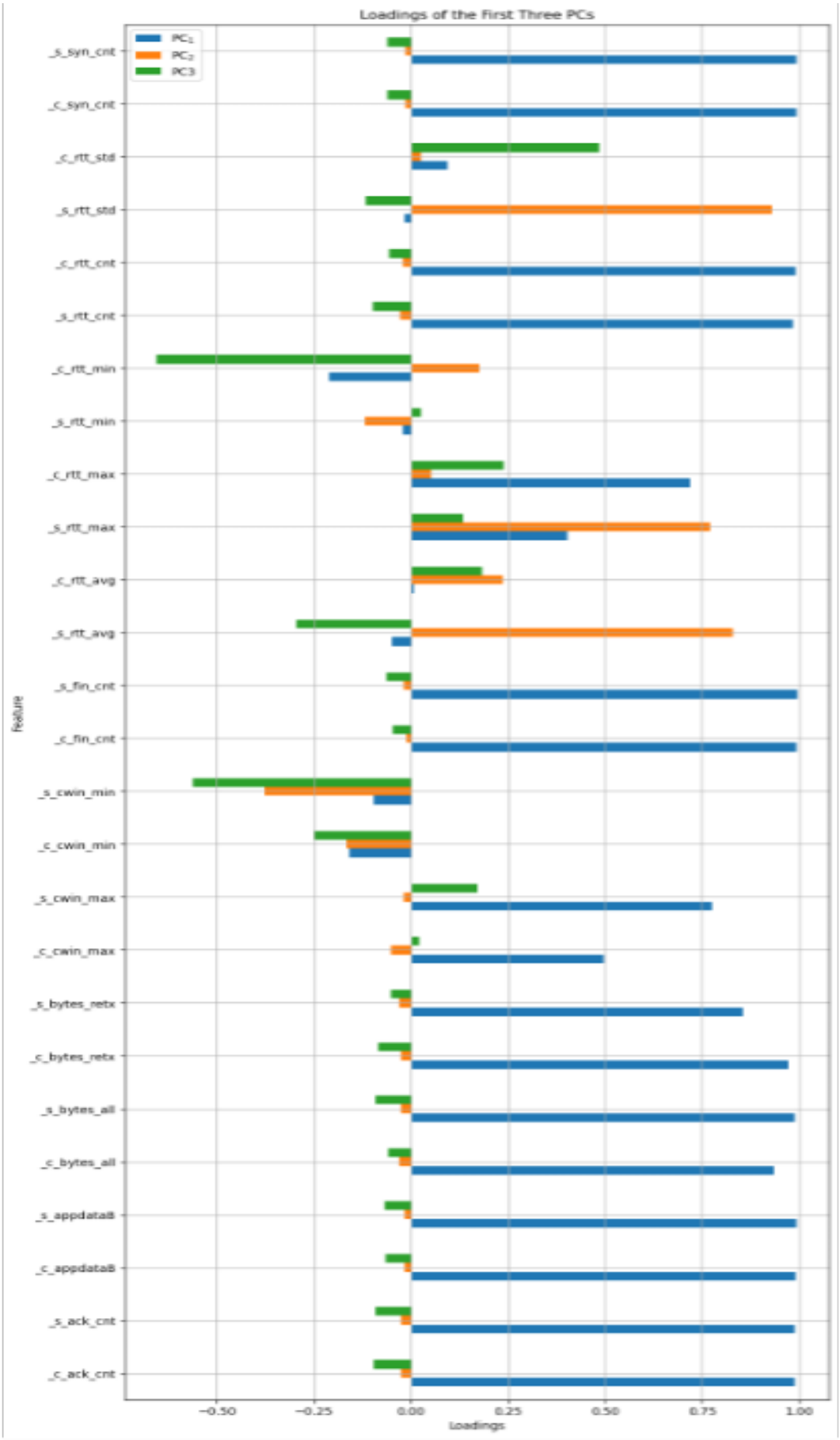




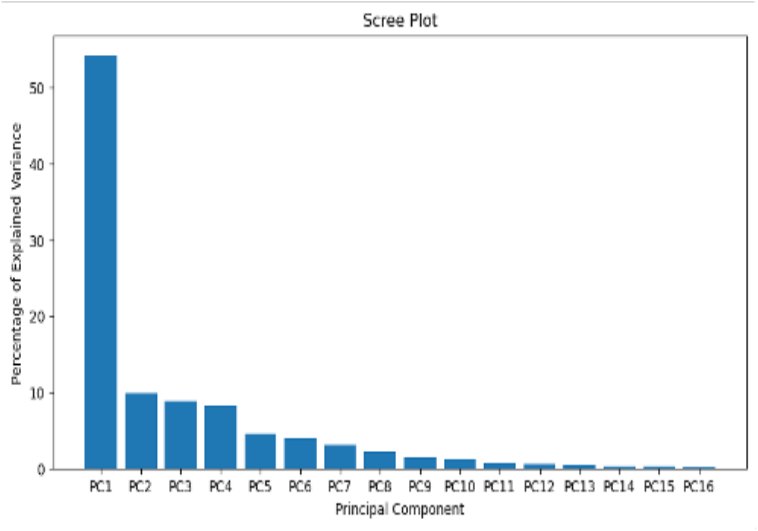
Cumulative explained variance visualization

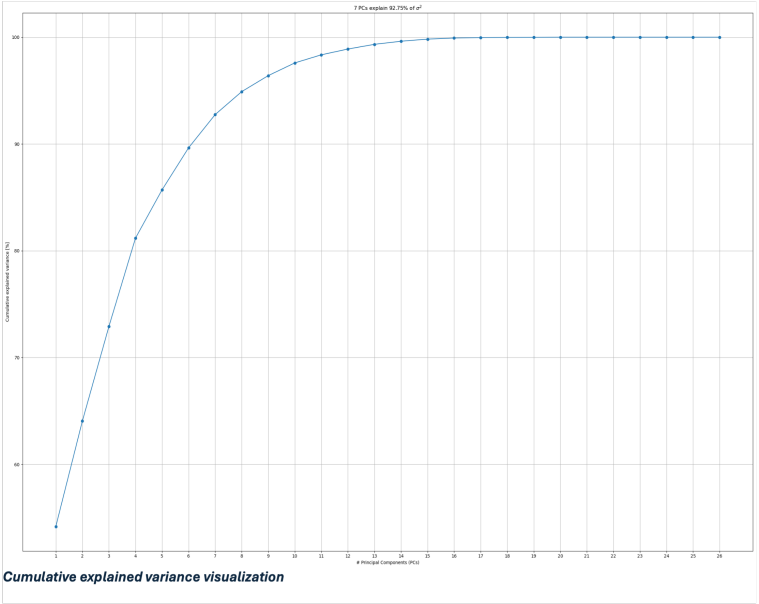
After the cumulative explained variance analysis, 12 PCs has been selected to create the new dataset (they explain about 96% of the cumulative variance).

Then the loading scores for the first three PCs has been plotted for all the origibnal features of the dataset:



Domain Name Level

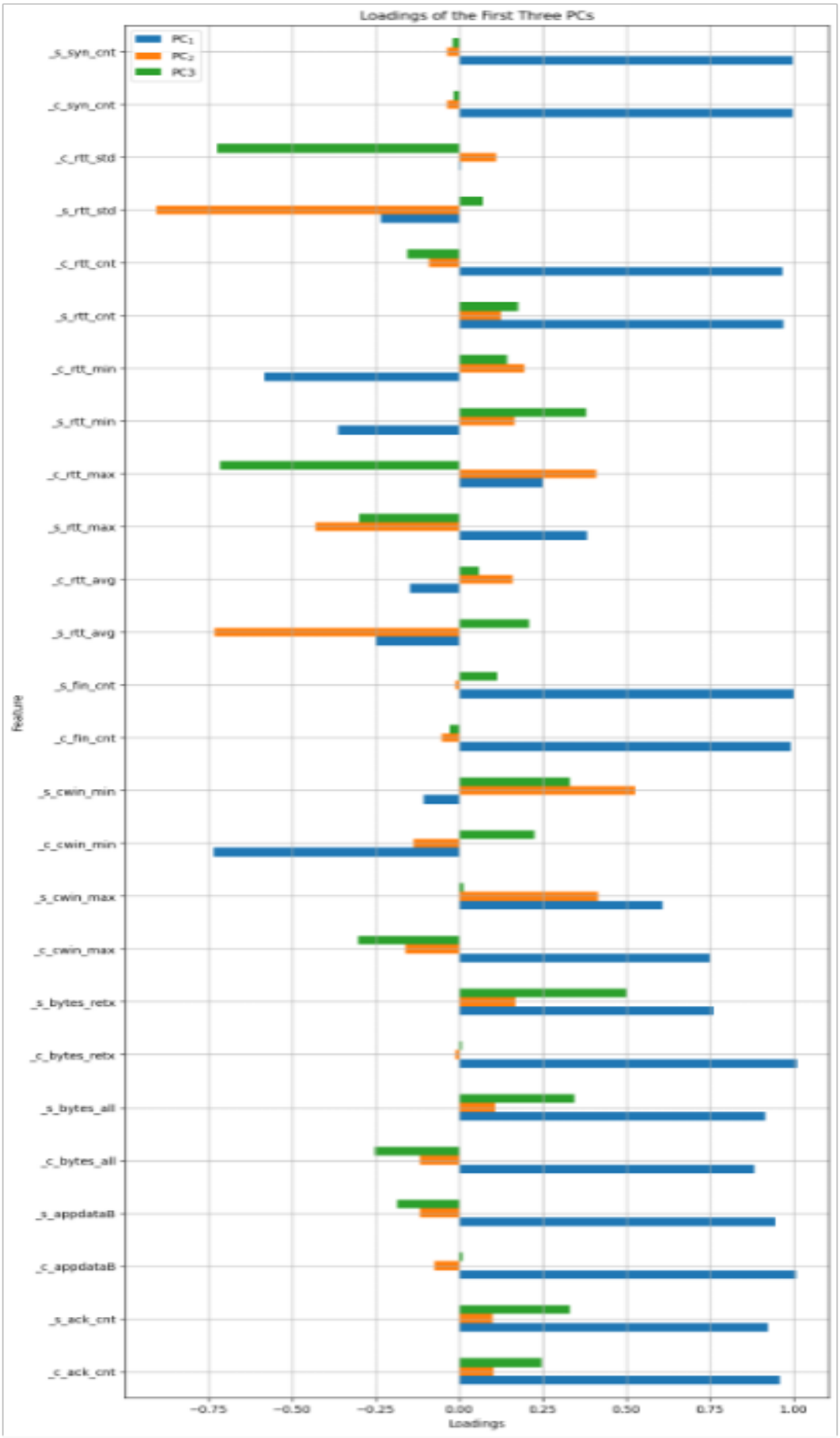




Cumulative explained variance visualization

After the cumulative explained variance analysis, 7 PCs has been selected to create the new dataset (they explain about 93% of the cumulative variance).

Then the loading scores for the first three PCs has been plotted for all the origibnal features of the dataset:



1.2.2 t-SNE at the three levels

At this point the t-SNE has been applied on the three datasets in order to visualize the data in two dimensions preserving its local structure.
First, the original dataset has been standardized to ensure uniformity and to prevent the domination of the features with larger scales. After scaling the dataset, the t-SNE method is applied to generate a two-dimensional dataset and is provided with the following parameters:

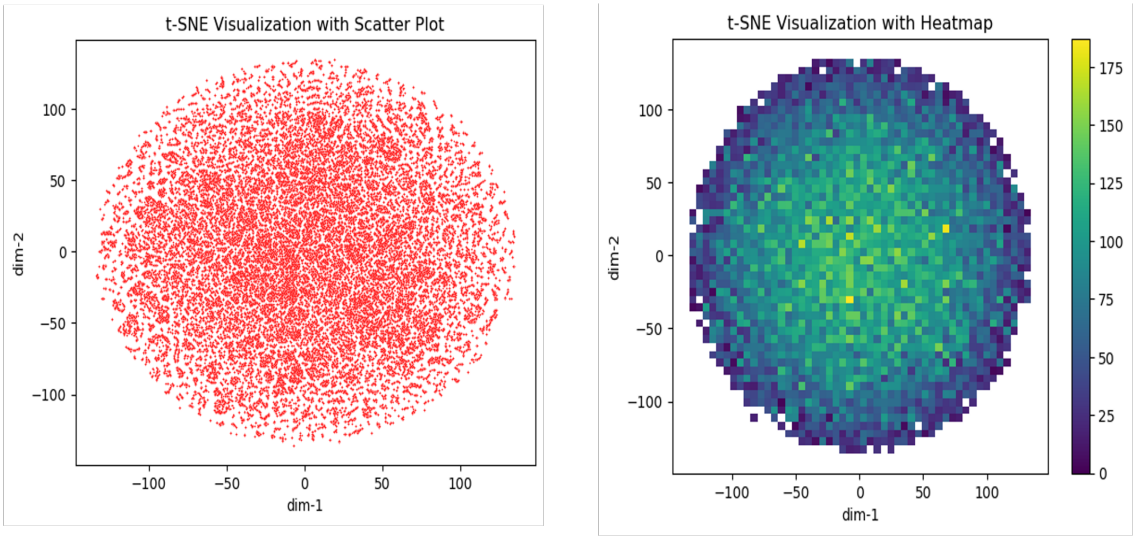
- *n_components=2*
- *learning_rate="auto"* to make t-SNE automatically adapt to data
- *init="random"* to randomly initialize the embedding
- *perplexiy=30* (default value) which influences the balance between global and local aspects of the data, affecting the reulting embedding

Later, the obtained array has been saved in dataset **df_tSNE** to plot:

1. a scatterplot for visualising the obtained points in the 2-dimensional space and the presence of some possible clusters
2. a heatmap by dividing the space in several bins for visualising the frequency of the points in each bins

The procedure just described has been followed at each of the three levels:

Flow Level

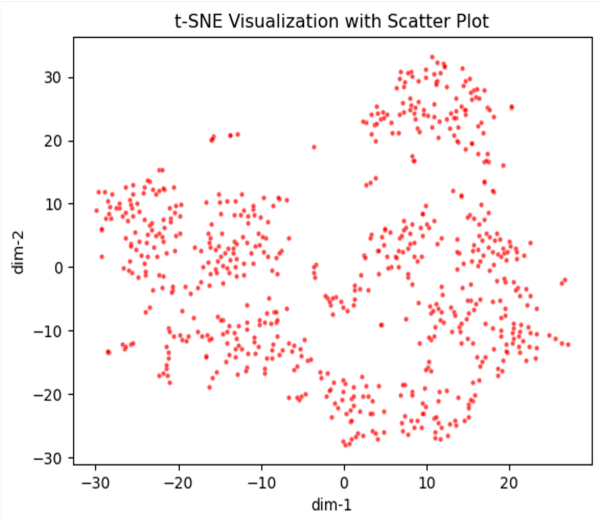


t-SNE flow level visualization

The t-SNE on the original dataset produces a two-dimensional space where datapoints are closed into a "ball". Particularly, groups of similar data points appear as dense regions in the scatter plot. The proximity of points indicates their similarity in the original high-dimensional space.
The heatmap helps to better visualize the density of points into the new space; it can be noticed that the densest areas are in the middle where some clusters can be identified.
Externally points are sparser, and they exhibit less similarities.

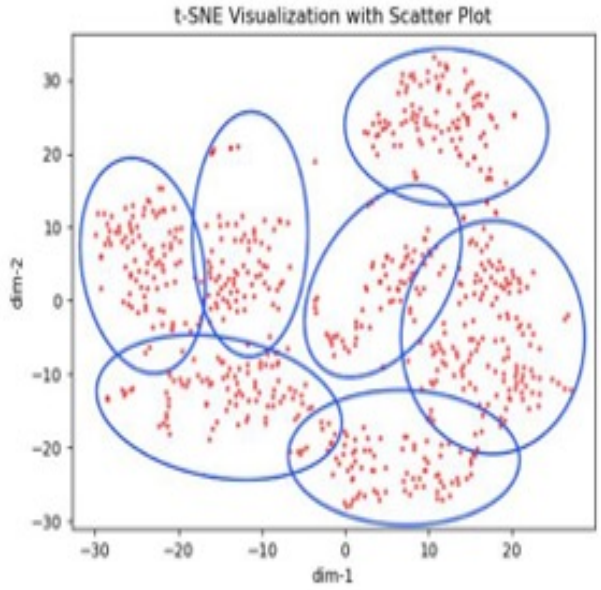
At the next levels only the scatter plot has been visualized because the heatmap does not provide any further information.

Client IP Level

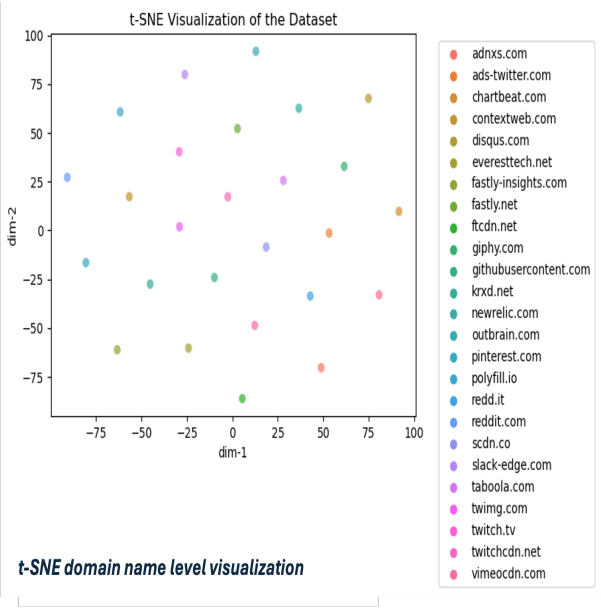


t-SNE client IP level visualization

After the t-SNE on the dataset at client IP level, all the points resulted easily visualizable on the new space. Again, the denser areas suggest the similarity of those points. In this case we can observe approximately the presence of seven clusters.



Domain Name Level



t-SNE domain name level visualization

The t-SNE on the original dataset produces a two-dimensional space where datapoints are all well separated. It is the expected result because in this case each domain name (i.e. each server) has different behavioural patterns without having any kind of similarity with other datapoints.

2 Supervised learning: Classification

In this section, three ML methods for classification are provided. After a first analysis (*a fast check on classification reports among the most famous classification models with default parameters*) the following models have been chosen in order of performances:

- Random Forest Classifier (*the best one*)
- K-Nearest Neighbors (*a trade-off between the other two*)
- Gaussian Naive Bayes (*the worst one*)

2.1 Data Preprocessing

Before starting with the ML models' analysis, data was processed by normalizing it and by eliminating highly correlated features (considering a threshold of 0.8) in order to reduce the dimensionality of the dataset (gaining in computational overhead).
At the end, the following datasets are available: the train set **X_s**, the validation set **X_val_s** (both recovered by splitting the initial training dataset) and the test set **X_test_s** (entirely recovered from the test dataset). The corresponding label arrays are **y_s**, **y_val_s**, and **y_test_s**.
From this point on, the analysis of each method has been performed by following these steps:

1. Model Training with default Parameters
2. Hyper-parameters Tuning
3. Performance Evaluation

2.2 Random Forest Classifier

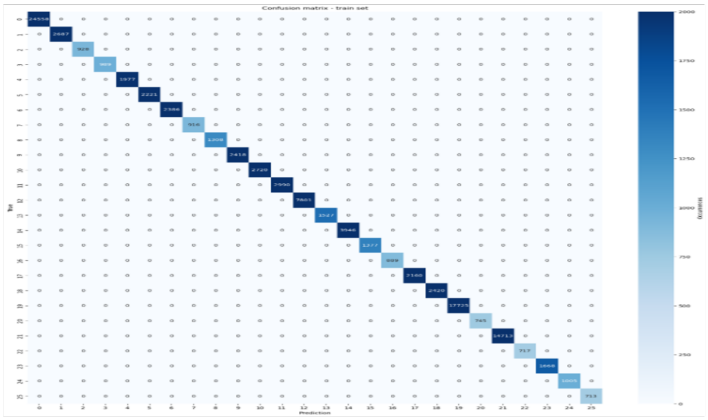
It is an extension of the Decision Tree model where k DTs (**n_estimators**) and a subset of $m' < m$ (**max_features**) are used.

2.2.1 Model Training with default Parameters

First, an initial evaluation of the model is performed by training it on **X_s** with the default parameter. Then predictions have computed over each set **X_s**, **X_val_s** and **X_test_s**, classification reports have been retrieved, and confusion matrixes plotted. Here the obtained results:

CLASSIFICATION REPORT RANDOM FOREST CLASSIFIER (TRAIN-SET):

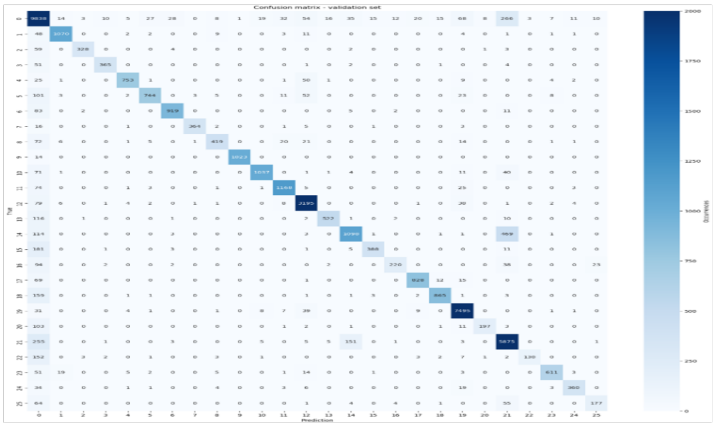
	precision	recall	f1-score	support
0	1.00	1.00	1.00	24558
1	1.00	1.00	1.00	2687
2	1.00	1.00	1.00	928
3	1.00	1.00	1.00	989
4	1.00	1.00	1.00	1977
5	1.00	1.00	1.00	2221
6	1.00	1.00	1.00	2386
7	1.00	1.00	1.00	916
8	1.00	1.00	1.00	1308
9	1.00	1.00	1.00	2418
10	1.00	1.00	1.00	2720
11	1.00	1.00	1.00	2930
12	1.00	1.00	1.00	7801
13	1.00	1.00	1.00	1527
14	1.00	1.00	1.00	3946
15	1.00	1.00	1.00	1377
16	1.00	1.00	1.00	889
17	1.00	1.00	1.00	2160
18	1.00	1.00	1.00	2420
19	1.00	1.00	1.00	17725
20	1.00	1.00	1.00	2745
21	1.00	1.00	1.00	14713
22	1.00	1.00	1.00	717
23	1.00	1.00	1.00	1668
24	1.00	1.00	1.00	1005
25	1.00	1.00	1.00	713
accuracy			1.00	103504
macro avg	1.00	1.00	1.00	103504
weighted avg	1.00	1.00	1.00	103504



Training Set Prediction – Classification Report and Confusion matrix

CLASSIFICATION REPORT RANDOM FOREST CLASSIFIER (VAL-SET):

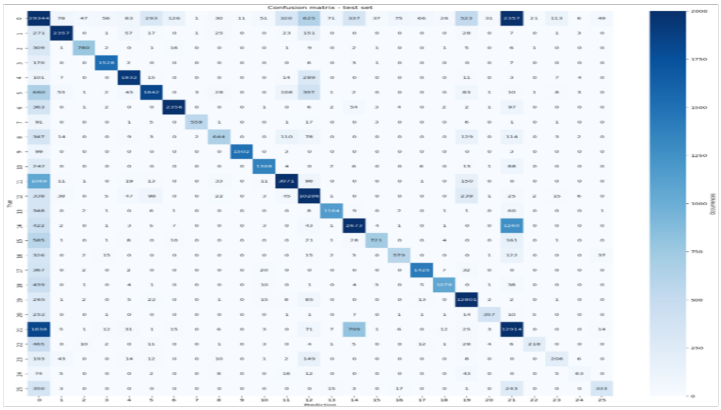
	precision	recall	f1-score	support
0	0.82	0.93	0.88	10525
1	0.96	0.93	0.94	1152
2	0.97	0.83	0.89	397
3	0.96	0.86	0.91	424
4	0.97	0.89	0.93	847
5	0.94	0.78	0.85	952
6	0.95	0.90	0.93	1022
7	0.99	0.93	0.96	393
8	0.91	0.75	0.82	561
9	1.00	0.99	0.99	1037
10	0.97	0.89	0.93	1166
11	0.93	0.91	0.92	1281
12	0.92	0.96	0.94	3343
13	0.95	0.80	0.87	655
14	0.84	0.65	0.73	1691
15	0.95	0.66	0.78	590
16	0.91	0.58	0.71	381
17	0.96	0.90	0.93	925
18	0.96	0.83	0.89	1037
19	0.97	0.99	0.98	7597
20	0.95	0.62	0.75	319
21	0.86	0.93	0.90	6305
22	0.98	0.42	0.59	307
23	0.96	0.85	0.90	715
24	0.93	0.84	0.88	431
25	0.84	0.58	0.68	306
accuracy			0.90	44359
macro avg	0.94	0.81	0.86	44359
weighted avg	0.91	0.90	0.90	44359



Validation Set Prediction – Classification Report and Confusion matrix

CLASSIFICATION REPORT RANDOM FOREST CLASSIFIER (TEST-SET):

	precision	recall	f1-score	support
0	0.75	0.84	0.79	34777
1	0.90	0.80	0.85	2942
2	0.92	0.69	0.79	1135
3	0.94	0.89	0.91	1717
4	0.85	0.81	0.83	2383
5	0.78	0.57	0.66	3246
6	0.93	0.81	0.87	2894
7	0.99	0.81	0.89	686
8	0.80	0.44	0.57	1455
9	0.99	0.93	0.96	1607
10	0.92	0.79	0.85	1730
11	0.82	0.69	0.75	4477
12	0.83	0.92	0.87	11171
13	0.92	0.73	0.81	1604
14	0.68	0.60	0.64	4426
15	0.93	0.47	0.62	1542
16	0.85	0.53	0.65	1102
17	0.93	0.77	0.84	1855
18	0.95	0.68	0.79	1585
19	0.91	0.97	0.94	13223
20	0.89	0.55	0.68	651
21	0.74	0.82	0.78	15764
22	0.88	0.28	0.43	771
23	0.57	0.32	0.41	644
24	0.68	0.28	0.39	228
25	0.77	0.35	0.48	965
accuracy			0.80	114580
macro avg	0.85	0.67	0.73	114580
weighted avg	0.81	0.80	0.80	114580



Test Set Prediction – Classification Report and Confusion matrix

Rapidly, it can be noticed that the model achieves perfect performances (*accuracy*, *precision*, *recall* and *f1-score*) on the training set and there are drops in performances both on validation and test set where the accuracy values are lower (respectively 0.90 and 0.80). Concerning other metrics (precision, f1-score and recall) the drop is more significant for some classes (for example for labels 14 *outbrain.com* and 23 *twitch.tv*) and it has less impact on others. Based on these

observations the model shows signs of **overfitting** because it performs extremely well on the training set but does not work in the same way on the validation and test sets.

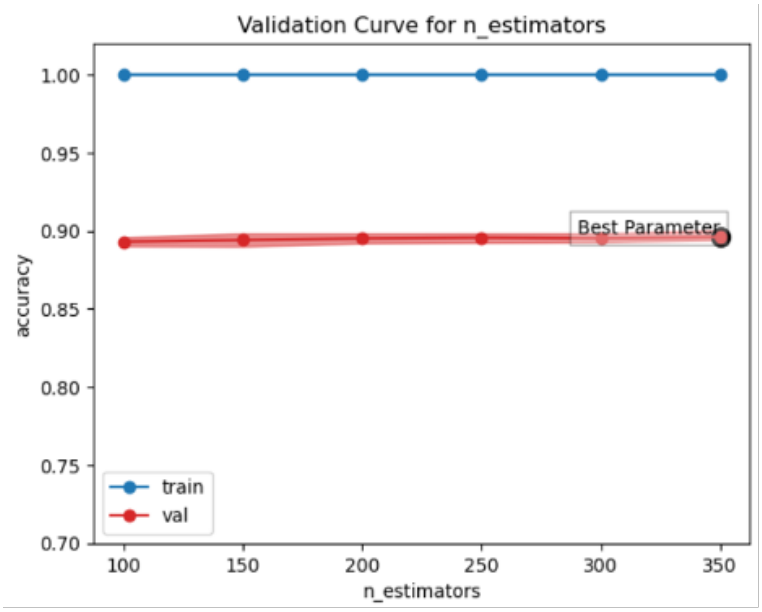
2.2.2 Hyper-parameters Tuning

For improving model's performances, the tuning of the following hyper-parameters was performed:

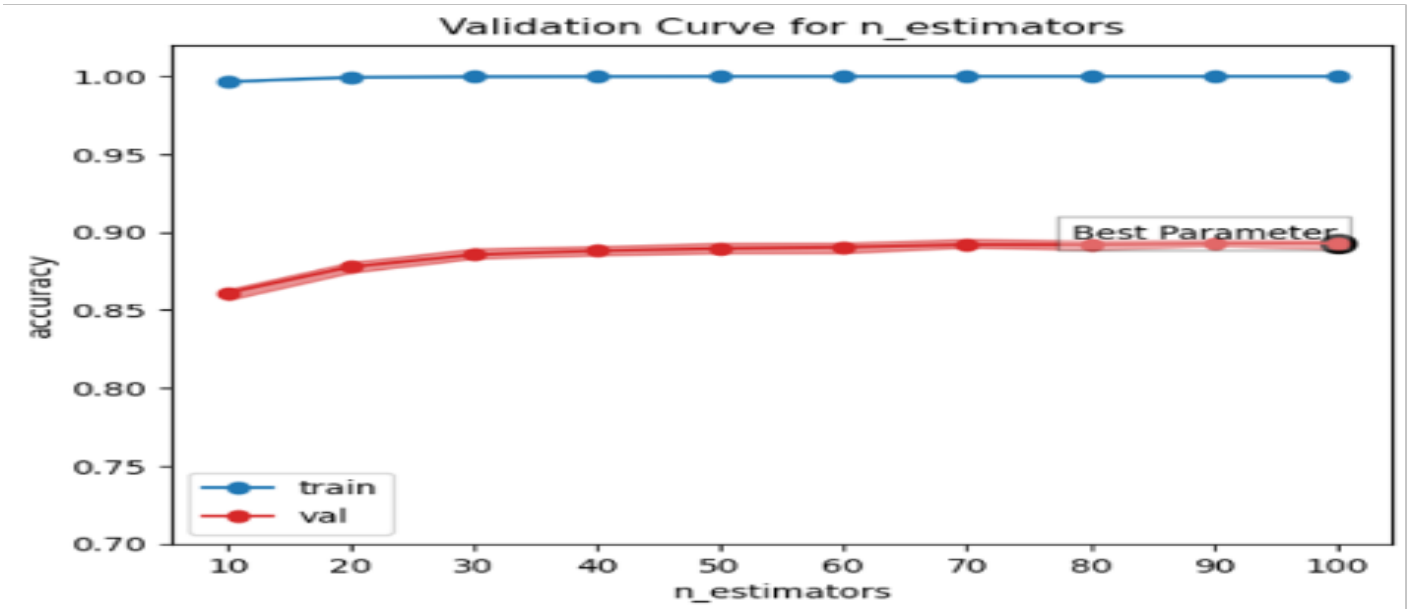
- **n_estimators**: it indicates the number k of DTs used for making the prediction
- **max_features**: it indicates the number of considered features ($m' < m$)

These parameters have been chosen because they have more impact on the model's performances.

Validation Curve At this point, the impact of **n_estimators** has been analysed by plotting the validation curve considering a **param_range** = *range(100, 400, 50)*, accuracy as scoring and a 5-fold cross-validation. The output is the following:



In this range the accuracy has already stabilized its mean value. For this reason a second validation curve has plotted by considering **param_range** = *range(10, 110, 10)* (shorter than the previous one). The output is the following:

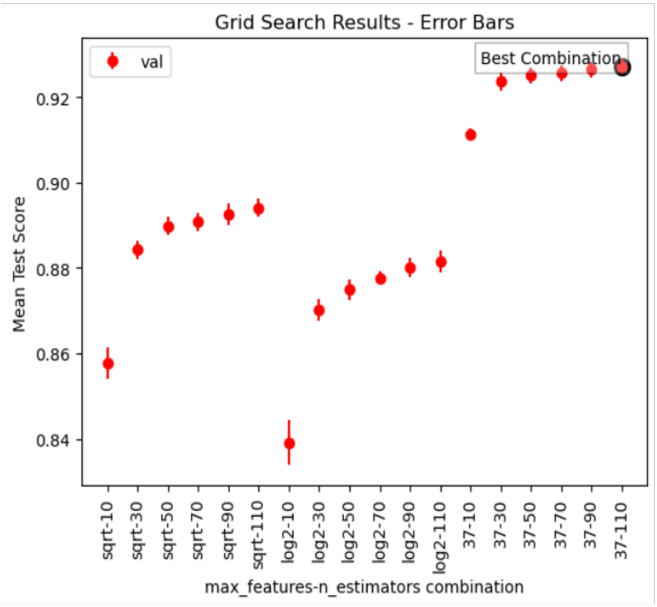


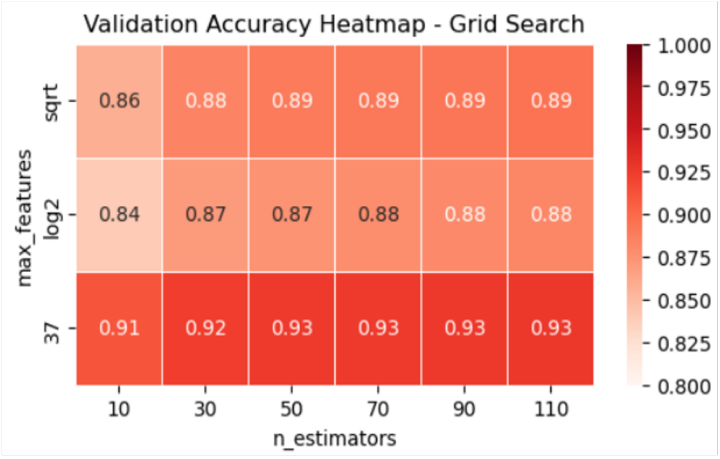
It can be noticed that the accuracy value on the validation set slightly grows up in the range between 10 and 30, then immediately starts to stabilize its value to about 0.90 by keeping on increasing it very slowly.

Grid Search At this point, also the **max_features** parameter is considered by choosing these possible values:

- *sqrt*
- *log2*
- *n_features*

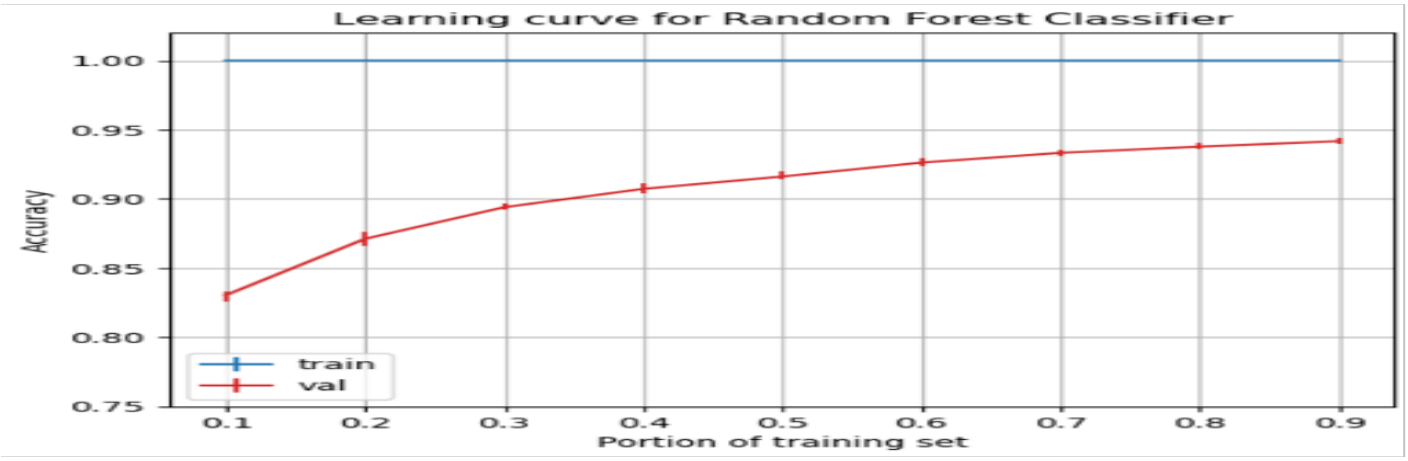
A Grid Search has performed by maintaining also the previous range for **n_estimators**. Note: the function *GridSearchCV* from sklearn.model_selection uses as default a 5-fold cross validation and accuracy as score. The obtained results have been visualized by plotting both an error bar and a heatmap:





The best combination obtained is the one with **n_estimators** = 110 and **max_features** = 37 (**n_features** / 2).

Learning Curve At this point, a learning curve has plotted to study the amount of training sample needed to obtain acceptable performance. First, a Random Forest Classifier object was initialized with the previously retrieved best combination of parameters. In this case the entire training dataset is needed, so **X_learningCurve_s** (*where _s stands for scaled*) and **y_learningCurve** have retrieved from **df_training**. The obtained learning curve is the following:



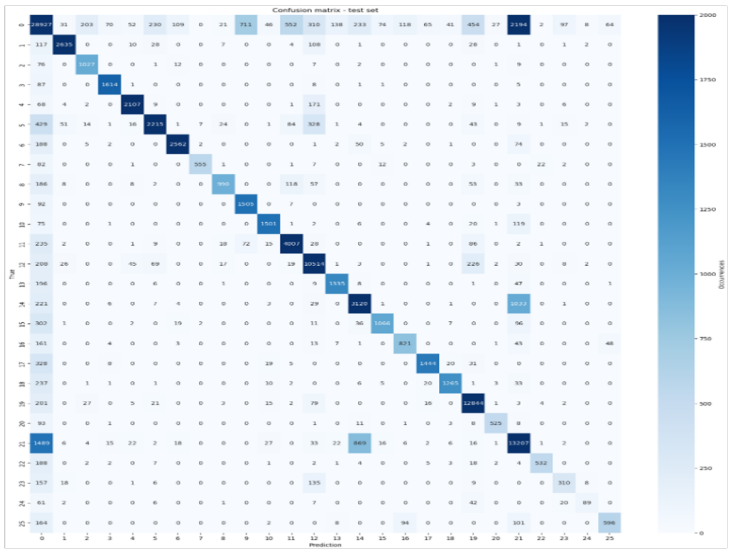
It can be noticed that the validation accuracy grows up with the growing of the training set portion. Particularly starting from the 70% of the dataset the accuracy validation score starts to stabilize its value at around of **0.94**, continuing to show a slight overfitting.

2.2.3 Performance Evaluation

Finally, the obtained best model has been tested on unseen data (**test set**) and on **validation set** as well, to observe if some improvements in performance have been reached. Here the obtained results:

CLASSIFICATION REPORT RANDOM FOREST CLASSIFIER (TEST-SET):

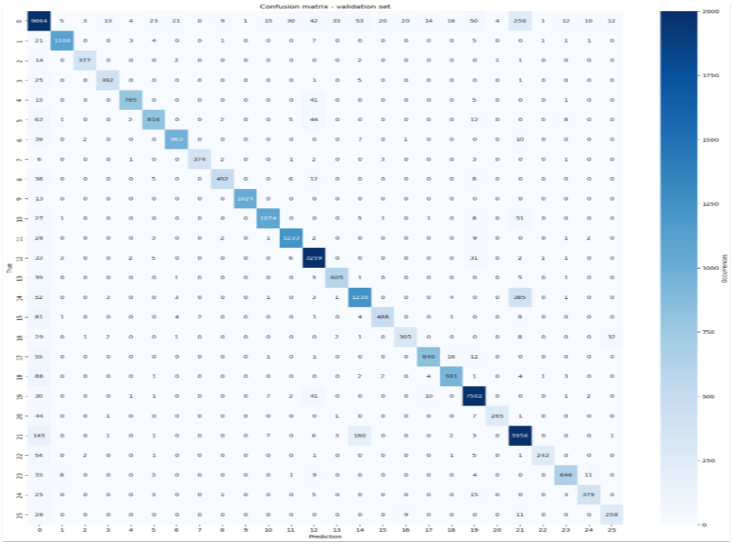
	precision	recall	f1-score	support
0	0.84	0.83	0.83	34777
1	0.95	0.90	0.92	2942
2	0.80	0.90	0.85	1135
3	0.94	0.94	0.94	1717
4	0.93	0.88	0.91	2383
5	0.85	0.68	0.76	3246
6	0.94	0.89	0.91	2894
7	0.98	0.81	0.89	686
8	0.91	0.68	0.78	1455
9	0.66	0.94	0.77	1607
10	0.92	0.87	0.89	1730
11	0.83	0.90	0.86	4477
12	0.89	0.94	0.91	11171
13	0.88	0.83	0.86	1604
14	0.72	0.70	0.71	4426
15	0.90	0.69	0.78	1542
16	0.79	0.75	0.77	1102
17	0.93	0.78	0.85	1855
18	0.94	0.80	0.86	1585
19	0.92	0.97	0.95	13223
20	0.93	0.81	0.86	651
21	0.77	0.84	0.80	15764
22	0.94	0.69	0.80	771
23	0.67	0.48	0.56	644
24	0.80	0.39	0.53	228
25	0.84	0.62	0.71	965
accuracy			0.85	114580
macro avg	0.86	0.79	0.82	114580
weighted avg	0.85	0.85	0.85	114580



Test Set Prediction – Classification Report and Confusion matrix

CLASSIFICATION REPORT RANDOM FOREST CLASSIFIER (VAL-SET):

	precision	recall	f1-score	support
0	0.91	0.94	0.92	10525
1	0.98	0.96	0.97	1152
2	0.98	0.95	0.96	397
3	0.95	0.92	0.94	424
4	0.98	0.93	0.95	847
5	0.94	0.86	0.90	952
6	0.97	0.94	0.95	1022
7	0.99	0.95	0.97	393
8	0.97	0.88	0.92	561
9	1.00	0.99	0.99	1037
10	0.97	0.92	0.95	1166
11	0.96	0.96	0.96	1281
12	0.94	0.97	0.96	3343
13	0.94	0.92	0.93	655
14	0.83	0.73	0.78	1691
15	0.95	0.83	0.88	590
16	0.91	0.80	0.85	381
17	0.97	0.91	0.94	925
18	0.96	0.90	0.93	1037
19	0.98	0.99	0.98	7597
20	0.98	0.83	0.90	319
21	0.89	0.94	0.92	6305
22	0.98	0.79	0.88	307
23	0.95	0.90	0.93	715
24	0.94	0.88	0.91	431
25	0.85	0.84	0.85	306
accuracy			0.93	44359
macro avg	0.95	0.90	0.92	44359
weighted avg	0.93	0.93	0.93	44359



Validation Set Prediction – Classification Report and Confusion matrix

There are clear improvements in performance concerning accuracy (from 0.90 to 0.93 for validation set, from 0.80 to 0.85 for test set). Precision, recall and f1-score values for most classes have improved as well. This suggests that the model is making more balanced predictions reducing both false positives and false negatives.
For example, considering the labels 14 and 23 there are significant improvements in their precision compared to the classification report retrieved without parameters tuning.

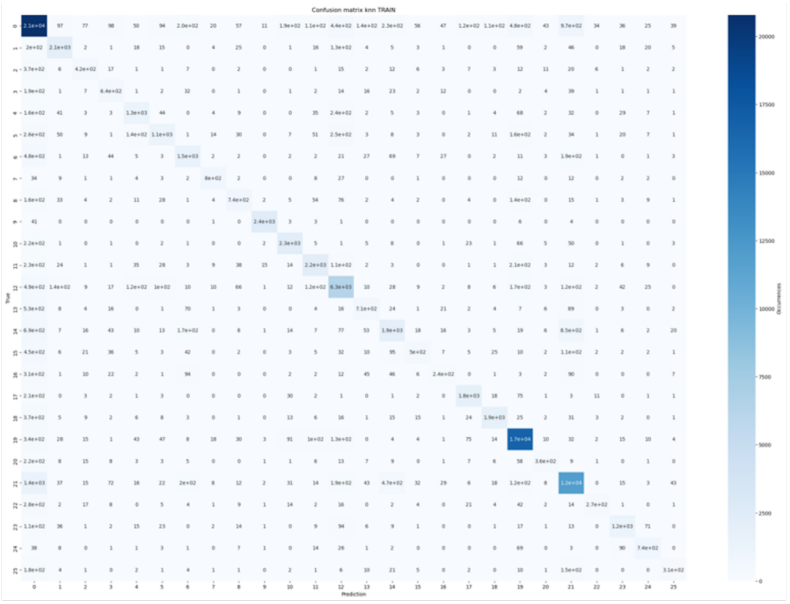
2.3 k-Nearest Neighbors Classifier

2.3.1 Model Training with default Parameters

After dividing the dataset and obtaining **X_s** and **X_val_s**, an initial classification is performed using default parameters to assess the model's performance. Following the training on **X_s**, predictions were made on both the training set and the validation set to evaluate the model's accuracy with its default parameters. At the conclusion of the experiment, an accuracy of 78% was achieved for the training set, while the validation set show an accuracy of 69%. Are show below the results obtained from the confusion matrix and the classification report are presented.

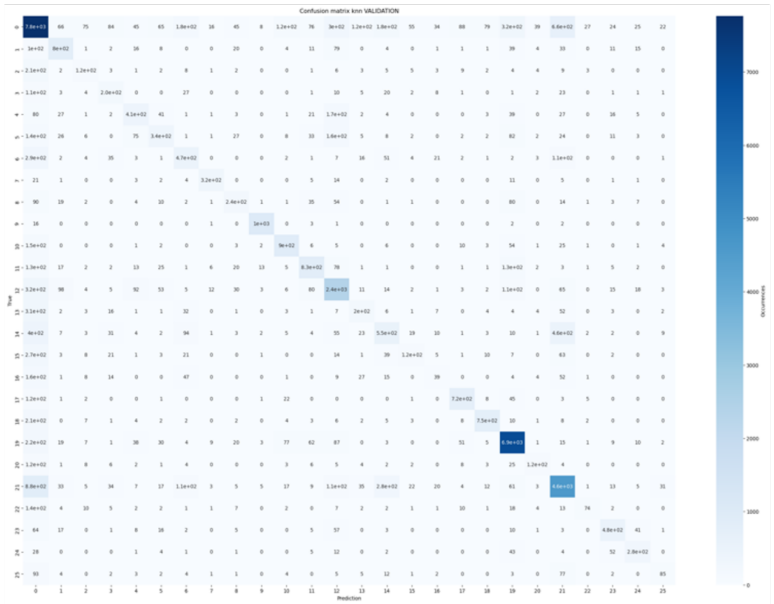
CLASSIFICATION REPORT K-Nearest Neighbors CLASSIFIER (TRAIN-SET):

	precision	recall	f1-score	support
0	0.72	0.85	0.78	24558
1	0.79	0.79	0.79	2687
2	0.63	0.46	0.53	928
3	0.61	0.64	0.63	989
4	0.72	0.65	0.68	1977
5	0.72	0.52	0.60	2221
6	0.63	0.62	0.63	2386
7	0.89	0.87	0.88	916
8	0.70	0.56	0.62	1308
9	0.98	0.98	0.98	2418
10	0.84	0.86	0.85	2720
11	0.79	0.75	0.77	2990
12	0.76	0.80	0.78	7801
13	0.65	0.47	0.54	1527
14	0.63	0.48	0.55	3946
15	0.74	0.37	0.49	1377
16	0.58	0.27	0.36	889
17	0.85	0.83	0.84	2160
18	0.89	0.77	0.83	2420
19	0.90	0.94	0.92	17725
20	0.75	0.49	0.59	745
21	0.80	0.81	0.81	14713
22	0.80	0.37	0.51	717
23	0.81	0.75	0.78	1668
24	0.79	0.74	0.76	1005
25	0.69	0.43	0.53	713
accuracy			0.78	103504
macro avg	0.76	0.66	0.69	103504
weighted avg	0.78	0.78	0.78	103504



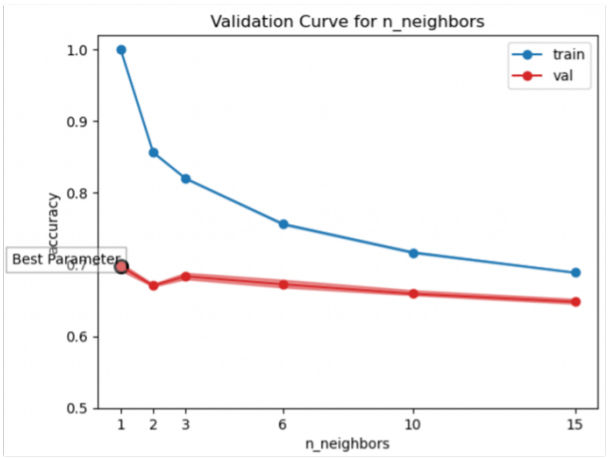
CLASSIFICATION REPORT K-Nearest Neighbors CLASSIFIER (VALIDATION-SET):

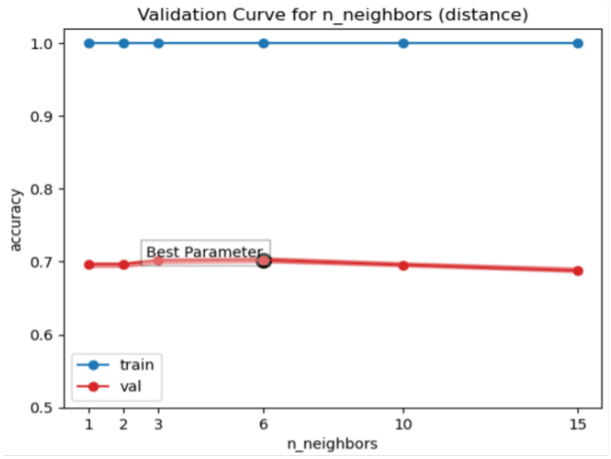
	precision	recall	f1-score	support
0	0.62	0.74	0.68	10525
1	0.69	0.69	0.69	1152
2	0.42	0.30	0.35	397
3	0.44	0.48	0.46	424
4	0.56	0.48	0.51	847
5	0.54	0.35	0.42	952
6	0.46	0.46	0.46	1022
7	0.86	0.82	0.84	393
8	0.55	0.42	0.47	561
9	0.96	0.98	0.97	1037
10	0.76	0.77	0.76	1166
11	0.69	0.65	0.67	1281
12	0.66	0.71	0.68	3343
13	0.43	0.30	0.35	655
14	0.45	0.32	0.38	1691
15	0.50	0.21	0.29	590
16	0.26	0.10	0.15	381
17	0.78	0.77	0.78	925
18	0.84	0.73	0.78	1037
19	0.86	0.91	0.89	7597
20	0.61	0.37	0.46	319
21	0.72	0.73	0.73	6305
22	0.62	0.24	0.35	307
23	0.74	0.67	0.70	715
24	0.67	0.65	0.66	431
25	0.53	0.28	0.36	306
accuracy			0.69	44359
macro avg	0.62	0.54	0.57	44359
weighted avg	0.68	0.69	0.68	44359



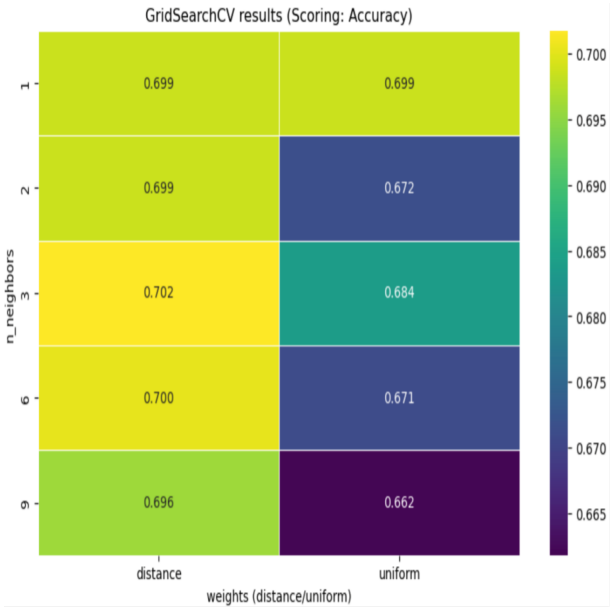
2.3.2 Hyper-parameters Tuning

At this point the validation curve is performed in order to analyze how the accuracy varies with changes in the parameter **n_neighbors**. The curve was computed twice, using two different weights ("uniform" and "distance"). Observing the graphs, a distinct difference in model performance is noticeable with varying weights. Using "uniform" weight, there is a sharp decline in precision as the number of neighbors increases for the training set, and a similar decline occurs in the validation set, albeit less prominently. In the second case (using "distance" weight), it is evident that overall performance in the training set improves significantly, with no marked difference within a range of neighbors from 1 to 15.

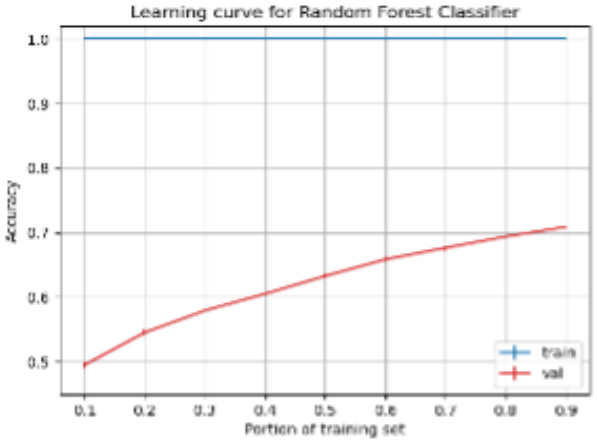




After the validation curve, the research of the best parameter to use for the model is computed. This is performed thanks GridSearchCV to assess how the model's accuracy varies with different parameter combinations.



From the heat-map, it is observed that, although there is no distinct difference, the optimal combination of parameters is achieved by choosing "distance" as the weight and setting the number of neighbors to "3".
Subsequently, a learning curve is generated using the original dataset and the recently identified parameters to assess how the model's accuracy varies with changes in the training set size and to determine if there are signs of overfitting or underfitting. It is observed that the model fits perfectly to the test set, and the accuracy of the validation improves with increasing size. A clear discrepancy between the two curves is noted, it indicates the presence of overfitting.

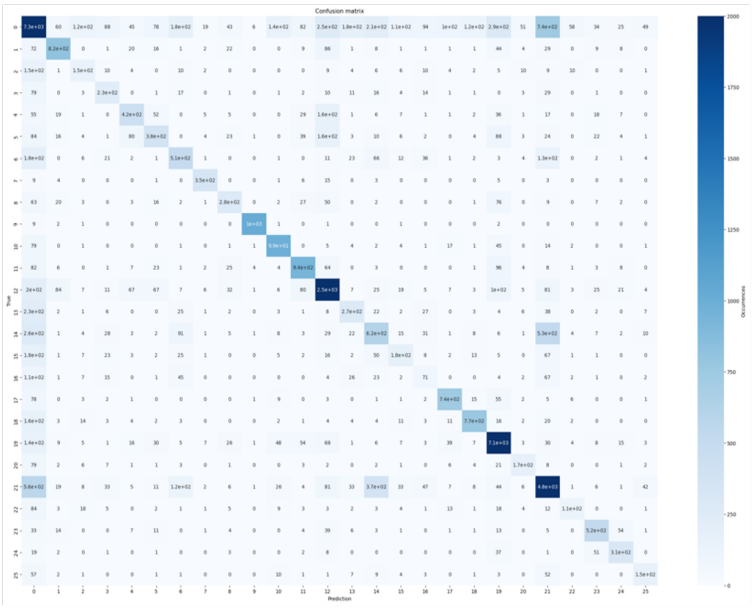


2.3.3 Performance Evaluation

Finally, the obtained best model has been tested on unseen data (**test set**) and on **validation set** as well, to observe if some improvements in performance have been reached. Here the obtained results:

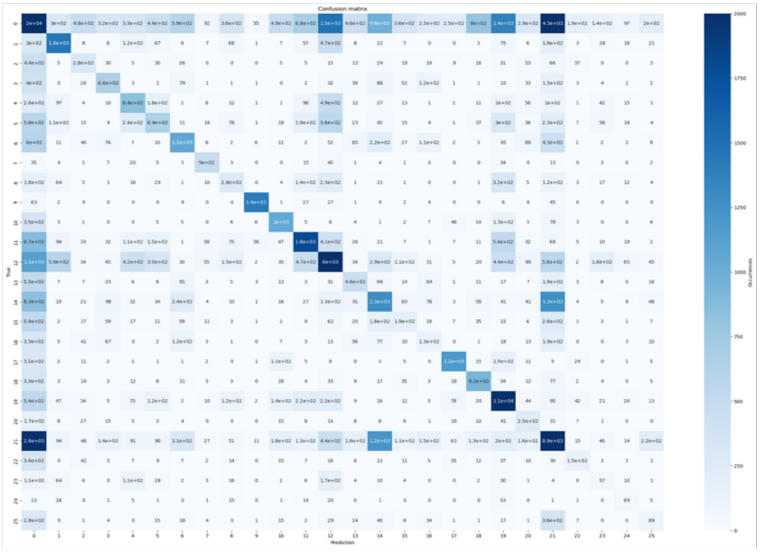
CLASSIFICATION REPORT K-NEAREST NEIGHBORS CLASSIFIER (VALIDATION-SET):

	precision	recall	f1-score	support
0	0.71	0.70	0.70	10525
1	0.75	0.71	0.73	1152
2	0.40	0.37	0.39	397
3	0.47	0.54	0.50	424
4	0.61	0.50	0.55	847
5	0.54	0.40	0.46	952
6	0.49	0.50	0.49	1022
7	0.86	0.88	0.87	393
8	0.58	0.49	0.53	561
9	0.98	0.98	0.98	1037
10	0.78	0.85	0.81	1166
11	0.73	0.73	0.73	1281
12	0.69	0.74	0.72	3343
13	0.44	0.41	0.42	655
14	0.42	0.36	0.39	1691
15	0.42	0.30	0.35	590
16	0.20	0.19	0.19	381
17	0.77	0.80	0.79	925
18	0.79	0.74	0.77	1037
19	0.87	0.93	0.90	7597
20	0.60	0.53	0.56	319
21	0.71	0.77	0.74	6305
22	0.55	0.37	0.44	307
23	0.72	0.72	0.72	715
24	0.67	0.71	0.69	431
25	0.54	0.50	0.52	306
accuracy			0.71	44359
macro avg	0.63	0.60	0.61	44359
weighted avg	0.71	0.71	0.71	44359



CLASSIFICATION REPORT K-NEAREST NEIGHBORS CLASSIFIER (TEST-SET)

	precision	recall	f1-score	support
0	0.61	0.56	0.59	34777
1	0.48	0.49	0.49	2942
2	0.24	0.25	0.24	1135
3	0.41	0.39	0.40	1717
4	0.34	0.35	0.35	2383
5	0.29	0.20	0.23	3246
6	0.39	0.36	0.38	2894
7	0.60	0.72	0.66	686
8	0.22	0.19	0.21	1455
9	0.91	0.88	0.89	1607
10	0.48	0.60	0.53	1730
11	0.46	0.40	0.43	4477
12	0.54	0.54	0.54	11171
13	0.30	0.29	0.29	1604
14	0.27	0.29	0.28	4426
15	0.18	0.13	0.15	1542
16	0.13	0.12	0.12	1102
17	0.68	0.63	0.66	1855
18	0.43	0.58	0.50	1585
19	0.74	0.86	0.79	13223
20	0.20	0.38	0.26	651
21	0.50	0.56	0.53	15764
22	0.29	0.19	0.23	771
23	0.09	0.09	0.09	644
24	0.18	0.30	0.23	228
25	0.12	0.09	0.11	965
accuracy			0.53	114580
macro avg	0.39	0.40	0.39	114580
weighted avg	0.52	0.53	0.52	114580



Despite the improvement in model performance, a significant disparity in accuracy is observed, which could have been anticipated. As can be seen there is an improvement between the starting validation test with default parameter and the validation test that has been done after the Hyper-parameters Tuning (accuracy from 69% to 71%). The same thing happens with the test set. There is an important drop on test performance. It indicates overfitting. Upon reviewing the final classification report, a notable discrepancy is observed between label 09 (*ftcdn.net*), which achieved a precision of 91%, and label 23 (*twitch.tv*), which, conversely, obtained a precision of 9%.

2.4 Gaussian Naive Bayes

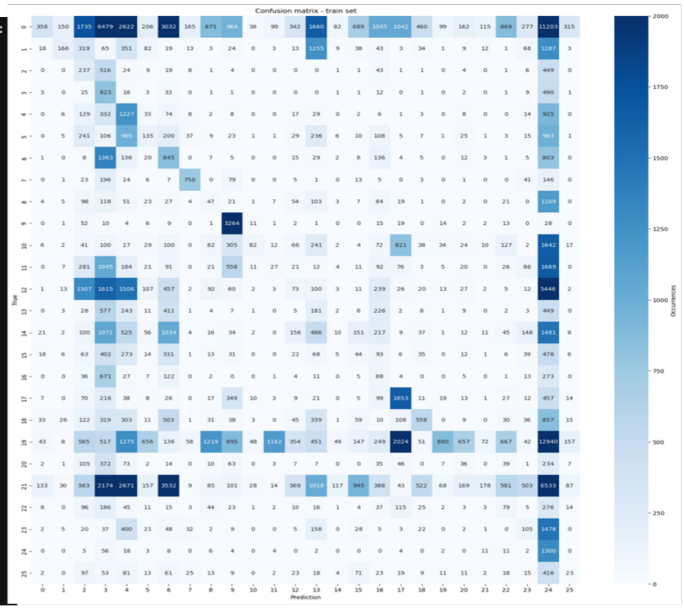
Naive Bayes classifier aims at minimizing 0/1 loss, namely it tries to maximize the probability that $h(x) = x$.

2.4.1 Model Training with default Parameters

First, the model is tried as is on the train and test set previously scaled. As you may notice, this model is the worst one among the three in terms of accuracy, since the training set has an accuracy of 0.09 while the test set 0.07. The thing that really jumps out at you is the drop of precision, especially with label 7, 9 and 19. The model overall accuracy is bad, anyway it seems performing **underfitting**, since both training and test set have about the same low accuracy.

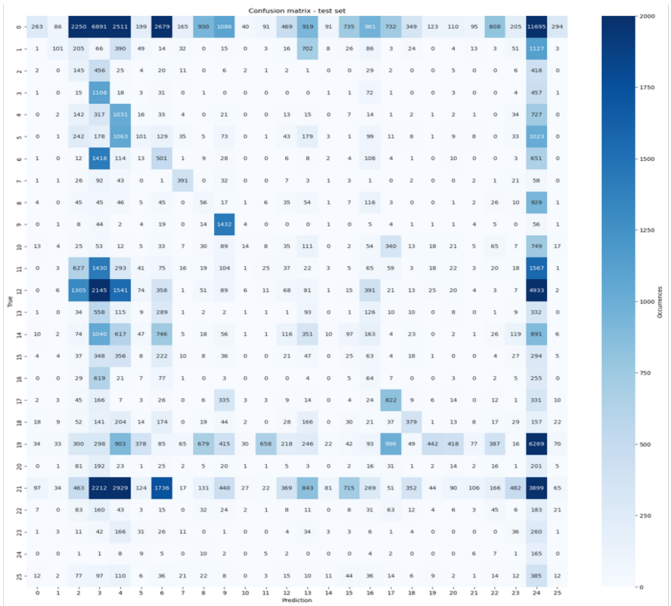
CLASSIFICATION REPORT GAUSSIAN NAIVE BAYES CLASSIFIER (TRAIN-SET):

	precision	recall	f1-score	support
0	0.54	0.01	0.02	35083
1	0.38	0.04	0.08	3839
2	0.04	0.18	0.06	1325
3	0.04	0.58	0.08	1413
4	0.09	0.43	0.15	2824
5	0.08	0.04	0.06	3173
6	0.08	0.25	0.12	3408
7	0.67	0.58	0.62	1309
8	0.02	0.03	0.02	1869
9	0.47	0.94	0.63	3455
10	0.34	0.02	0.04	3886
11	0.02	0.01	0.01	4271
12	0.04	0.01	0.01	11144
13	0.03	0.08	0.04	2182
14	0.03	0.00	0.00	5637
15	0.02	0.02	0.02	1967
16	0.03	0.07	0.04	1270
17	0.27	0.54	0.36	3085
18	0.30	0.16	0.21	3457
19	0.76	0.03	0.07	25322
20	0.03	0.03	0.03	1064
21	0.42	0.01	0.02	21018
22	0.03	0.08	0.04	1024
23	0.07	0.04	0.05	2383
24	0.02	0.91	0.05	1436
25	0.03	0.02	0.03	1019
accuracy			0.09	147863
macro avg	0.19	0.20	0.11	147863
weighted avg	0.38	0.09	0.07	147863



CLASSIFICATION REPORT GAUSSIAN NAIVE BAYES CLASSIFIER (TEST-SET):

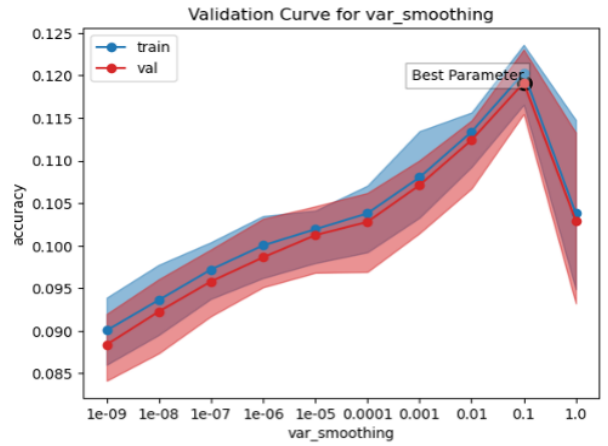
	precision	recall	f1-score	support
0	0.56	0.01	0.01	34777
1	0.34	0.03	0.06	2942
2	0.02	0.13	0.04	1135
3	0.06	0.65	0.10	1717
4	0.08	0.43	0.14	2383
5	0.09	0.03	0.05	3246
6	0.07	0.17	0.10	2894
7	0.49	0.57	0.53	686
8	0.03	0.04	0.03	1455
9	0.33	0.89	0.48	1607
10	0.10	0.01	0.02	1730
11	0.03	0.01	0.01	4477
12	0.04	0.01	0.01	11171
13	0.02	0.06	0.03	1604
14	0.04	0.00	0.00	4426
15	0.01	0.02	0.02	1542
16	0.02	0.06	0.03	1102
17	0.26	0.44	0.32	1855
18	0.30	0.24	0.26	1585
19	0.64	0.03	0.06	13223
20	0.02	0.02	0.02	651
21	0.31	0.01	0.01	15764
22	0.03	0.06	0.04	771
23	0.03	0.06	0.04	644
24	0.00	0.72	0.01	228
25	0.02	0.01	0.02	965
accuracy			0.07	114580
macro avg	0.15	0.18	0.09	114580
weighted avg	0.33	0.07	0.05	114580



2.4.2 Hyper-parameters Tuning

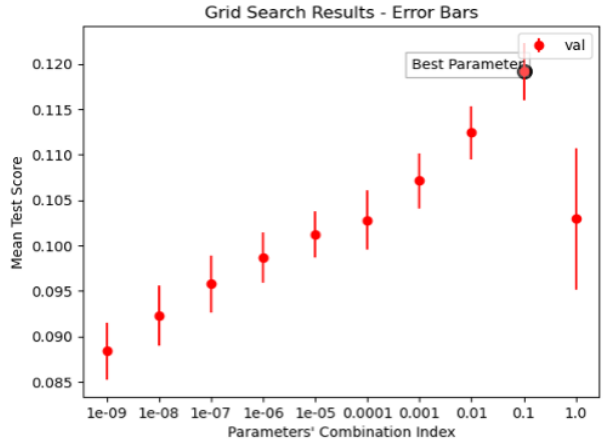
In order to try to improve performance, it has been fulfilled the tuning of the hyper-parameters. The only parameter present is **var_smoothing**, that indicates the portion of the largest variance of all features that is added to variances for calculation stability.
It has been split as follows:
parameter_range = $'var\_smoothing' : np.logspace(-9, 0, 10)$ Since there is only one parameter, Validation Curve has been used to seek the best value for the variable.

Validation Curve



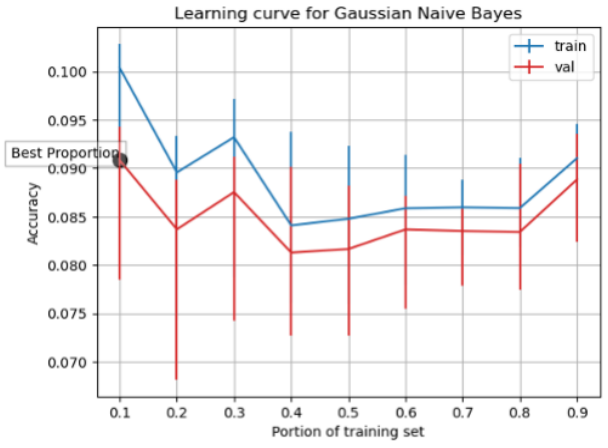
It is clear to see that the best value for the parameter is 0.1, that results in an accuracy of 0.12 for what concerns the training set and slightly lower for the validation set.

Grid Search After the Validation Curve, to be sure that it has been found the best value, a Grid Search has been performed with the same range of the parameter.



As you may notice, the result on the validation set are the same as the previous test on validation curve, confirming the same outcomes.

Learning Curve Last of all, a learning curve has been plotted to see which is the best combination between training and validation set.



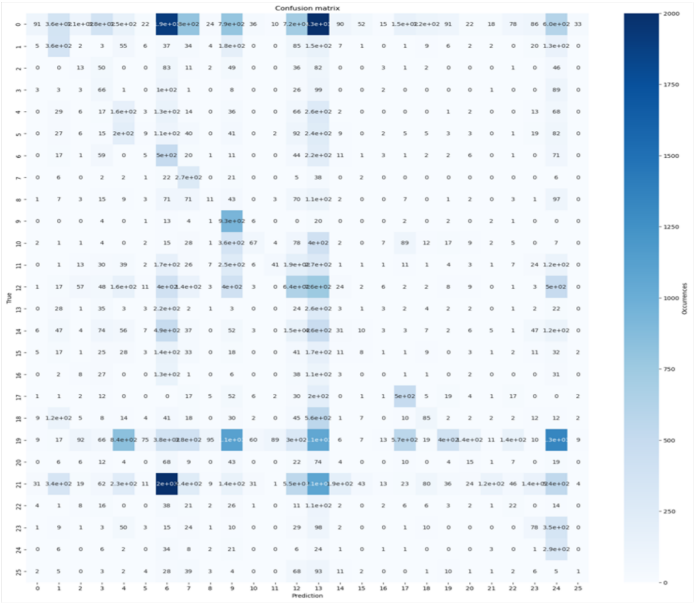
The best combination is the one with the highest validation set accuracy, that is 1/10 for the training set and 9/10 for the validation set, although it's even the combination in which there is more discrepancy, hence overfitting. However, choosing the 90% of data as training set shows a still good performance on validation set but with less deficit between the 2 sets.

2.4.3 Performance Evaluation

After the hyper-parameter tuning, the best combination of parameter, in this case **var_smoothing** = 0.1, has been used for testing on a previously unknown dataset, the initial test set.

CLASSIFICATION REPORT GAUSSIAN NAIVE BAYES (VALIDATION-SET):

	precision	recall	f1-score	support
0	0.53	0.01	0.02	10024
1	0.25	0.33	0.28	1097
2	0.04	0.03	0.04	379
3	0.06	0.16	0.09	404
4	0.08	0.20	0.12	807
5	0.05	0.01	0.02	907
6	0.07	0.52	0.12	974
7	0.12	0.72	0.20	374
8	0.06	0.02	0.03	534
9	0.20	0.94	0.33	987
10	0.30	0.06	0.10	1110
11	0.27	0.03	0.06	1220
12	0.19	0.20	0.20	3184
13	0.03	0.42	0.05	623
14	0.07	0.02	0.03	1611
15	0.06	0.01	0.02	562
16	0.00	0.00	0.00	363
17	0.36	0.57	0.44	881
18	0.23	0.09	0.12	988
19	0.65	0.05	0.10	7235
20	0.06	0.05	0.05	304
21	0.70	0.02	0.04	6005
22	0.06	0.08	0.07	292
23	0.17	0.11	0.14	681
24	0.06	0.72	0.12	410
25	0.02	0.00	0.01	291
accuracy			0.12	42247
macro avg	0.18	0.21	0.11	42247
weighted avg	0.41	0.12	0.09	42247



CLASSIFICATION REPORT GAUSSIAN NAIVE BAYES (TEST-SET):

	precision	recall	f1-score	support
0	0.59	0.01	0.01	34777
1	0.18	0.23	0.20	2942
2	0.05	0.04	0.04	1135
3	0.07	0.13	0.09	1717
4	0.08	0.24	0.12	2383
5	0.03	0.00	0.01	3246
6	0.08	0.44	0.13	2894
7	0.08	0.68	0.15	686
8	0.03	0.01	0.02	1455
9	0.10	0.90	0.19	1607
10	0.16	0.04	0.07	1730
11	0.41	0.04	0.08	4477
12	0.23	0.27	0.25	11171
13	0.02	0.32	0.04	1604
14	0.10	0.04	0.06	4426
15	0.03	0.01	0.01	1542
16	0.00	0.00	0.00	1102
17	0.34	0.51	0.41	1855
18	0.22	0.11	0.15	1585
19	0.54	0.05	0.09	13223
20	0.02	0.02	0.02	651
21	0.58	0.02	0.03	15764
22	0.04	0.04	0.04	771
23	0.04	0.08	0.06	644
24	0.01	0.38	0.02	228
25	0.02	0.01	0.01	965
accuracy			0.10	114580
macro avg	0.16	0.18	0.09	114580
weighted avg	0.39	0.10	0.08	114580

The overall results are slightly improved, although still really poor. The accuracy has raised of 0.03 (from 0.07 to 0.10), but the results appear to be more homogeneous than before. Indeed, label 7, 9 and 19 that had the worst drop in this parameter, are facing a further drop, while other labels have an enhance. Since the results for both test and validation are both low, there is evidence of underfitting that remains even after hyper-parameters tuning.

2.5 Conclusions

To sum up, the model that generates the best performance is the Random Forest Classifier, even if it slightly shows overfitting (there is a drop of 0.08 in accuracy between validation and test set).
Instead, the k-NN and the Gaussian Naive Bayes' classifiers definitely provide worse performance showing respectively a significant overfitting and underfitting.

3 Unsupervised learning: Clustering

For this part two algorithms were chosen: **KMeans** for a hard one, **Gaussian Mixture Model** for soft clustering. The main difference is that the hard assigns data samples to k clusters -that are scoped in a smart way-, whereas the soft one predicts a probability of assignments.

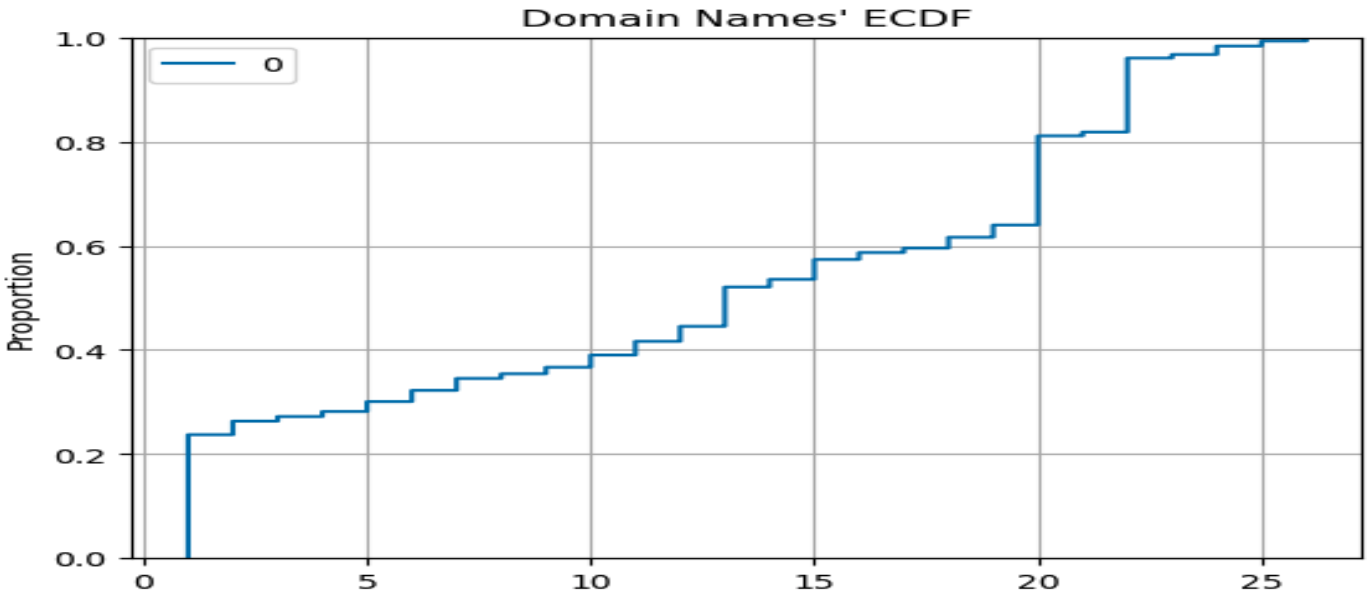
3.1 Data Exploration and Preprocessing

As can be seen in the *jupyter* notebook, section *0.3 Data Exploration* all unique labels (domain names) are grouped in a dictionary as keys and their values are the occurrences. *vimeocdn.com* is the domain name with least occurrences (1029), *_other* domain names represent the majority (35083).

A good approach to get some dataset's *domain knowledge* is to understand the function or service that each domain name is offering. For this purpose a website called netify.ai was consulted and all 26 unique labels could be grouped in around 7 categories (bear in mind that this is a human-made classification, so groups can also be 6, 7 or 8). Therefore number of clusters expectation could be 7:

- 8 ADVERTISING: [*adnxs.com*, *ads-twitter.com*, *chartbeat.com*, *contextweb.com*, *everesttech.net*, *krxd.net*, *outbrain.com*, *taboola.com*]
- 5 CONTENT DELIVERY NETWORK: [*fastly.net*, *fastly-insights.com*, *ftcdn.net*, *scdn.co*, *twitchcdn.net*]
- 5 SOCIAL MEDIA: [*disqus.com*, *githubusercontent.com*, *pinterest.com*, *reddit.com*, *redd.it*]
- 3 SOFTWARE-AS-A-SERVICE: [*newrelic.com*, *polyfill.io*, *slack-edge.com*]
- 2 HOSTING: [*giphy.com*, *twimg.com*]
- 2 STREAMING: [*twitch.tv*, *vimeocdn.com*]
- OTHER

Following in *0.4 Data Visualisation* an ECDF is plotted:



The code snippet in *0.5 Data Preprocessing* shows that all columns' data types are *float64* numbers, except for *c_ip* and *label* columns which are *object* dtype (can hold any Python object, including strings Pandas dtypes).

Finally, those columns must be mapped into numerical values using *LabelEncoder*; it has been chosen to map all unique IPs in [1, 738] and labels in [1, 26].

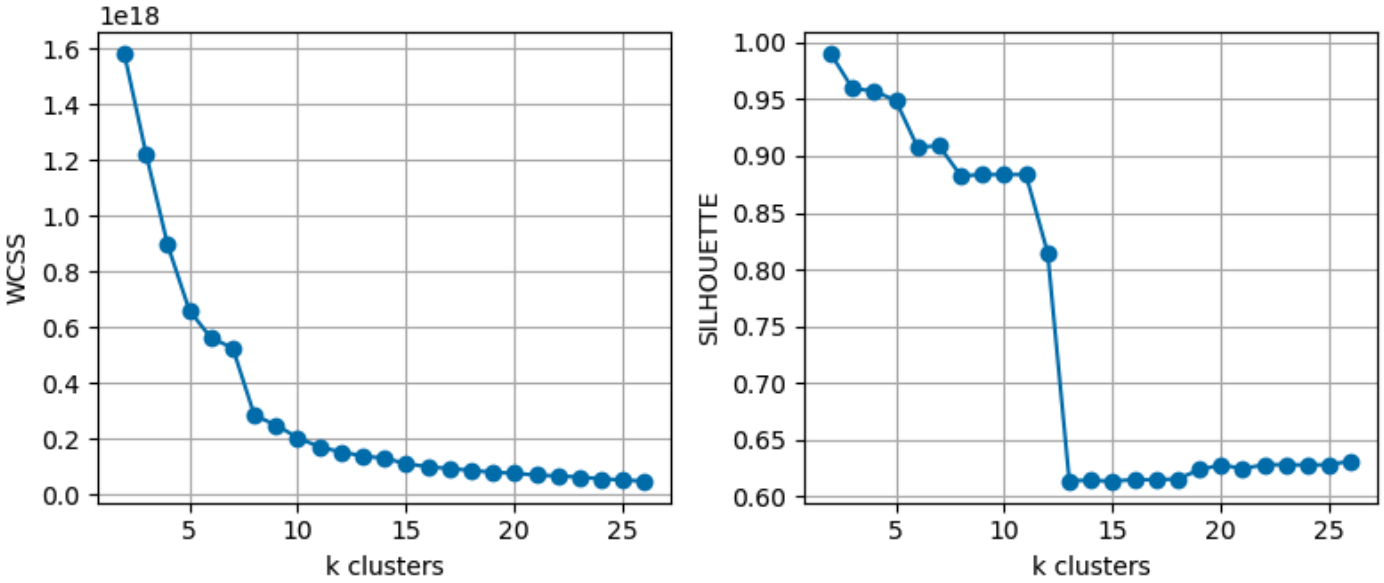
Since both KMeans and GMM operate in an Euclidean space, data scaling is mandatory to standardize the data (*StandardScaler* is used).

3.2 KMeans Clustering

3.2.1 Looping over k

When applying Machine Learning's algorithms it is crucial to wisely choose the hyper-parameters, in this case the best combination is using *n_init=1* (to avoid overhead in looping and as KMeans documentation says too) and *random_state=42* (random number generation for centroid initialization), leaving as default *init='kmeans++'*.
After few combinations of these parameters the best initialization turned out to be the one used. The only hyper-parameter that has to be picked is clusters number k.

For this purpose, the function *try_kmeans(X_s)* loops kmeans' k cluster number with same hyper-parameters initialization over the range $[2, 27]$ and stores the output in a list.
Then that list is used in *find_elbow_silhouette(X_s)* to plot the Elbow Method and Silhouette Score: both relative to k. The output is the following:

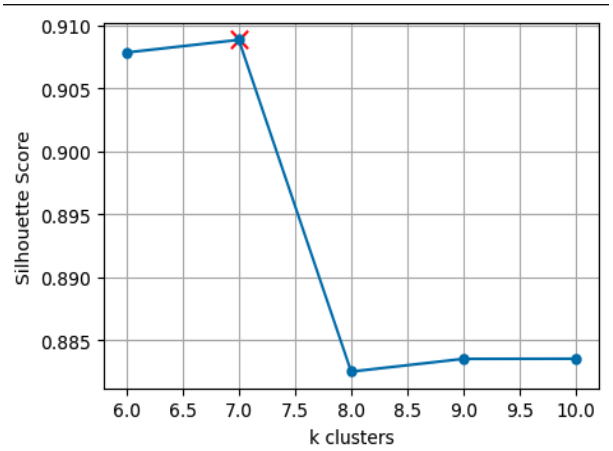


On the left the **Elbow Method** representing the **WCSS**: *Within Cluster Sum of Squares* is the sum of squared (Euclidean space) distances between all data points in each cluster and the relative cluster's centroid (*aka* inertia). The optimal k can be picked on the graph on the first point after which the slope starts decreasing (choosing a greater value for k could lead to overfitting).
On the right picture the **Silhouette Score** is represented, which stands for data points' tightness in the same cluster, possible values of this metric are in $[-1, +1]$: the closest the silhouette to 1, the tighter the data points.

Each value of k in $[5, 11]$ Silhouette Score is really high (tightness of points of the same cluster), whereas from the elbow picture on the left it can't be argued that k is in range $[6, 10]$. So far it is necessary to find the associated k to the highest silhouette score in that range.

3.2.2 Scoping best k

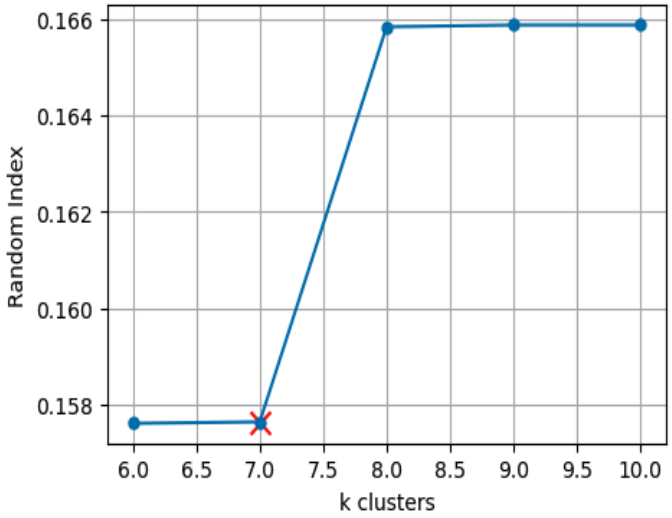
In section *1.2 Scoping best k* in *jupyter* notebook the script was written knowing already the fact that the optimal k is certainly in that range ($[6, 10]$), so at each iteration performance metrics are computed too and appended to pre-declared lists for later analysis.
The output is really good: silhouette score for $k = 7$ is 0.909, that means that data points are tight in their assigned clusters. As soon as the program is finished, the following is plotted:

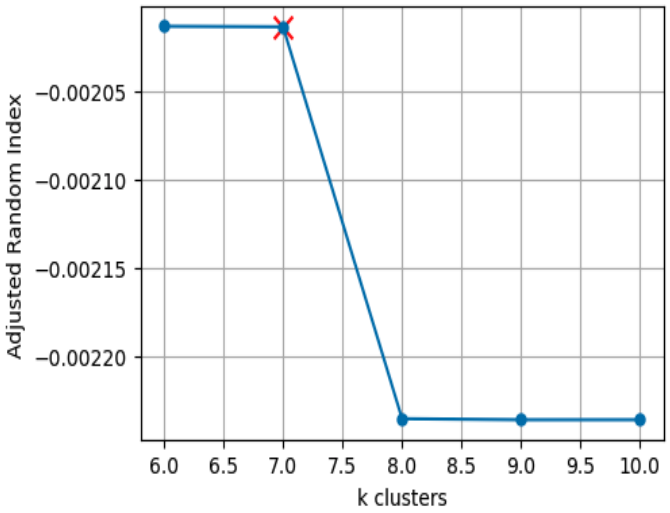


3.2.3 Performance Evaluation

In a clustering context the best metrics for assignments are *Random Index* ($0 \leq RI \leq 1$, indicates level of similarity between true and predicted cluster assignments) and *Adjusted Random Index* ($-1 \leq ARI \leq +1$,), specifically the ladder is more useful in scenarios where RI's value is close to zero. Referring to jupyter's *1.3 Performance evaluation*:

- $RI = 0.158$
- $ARI = -0.002$

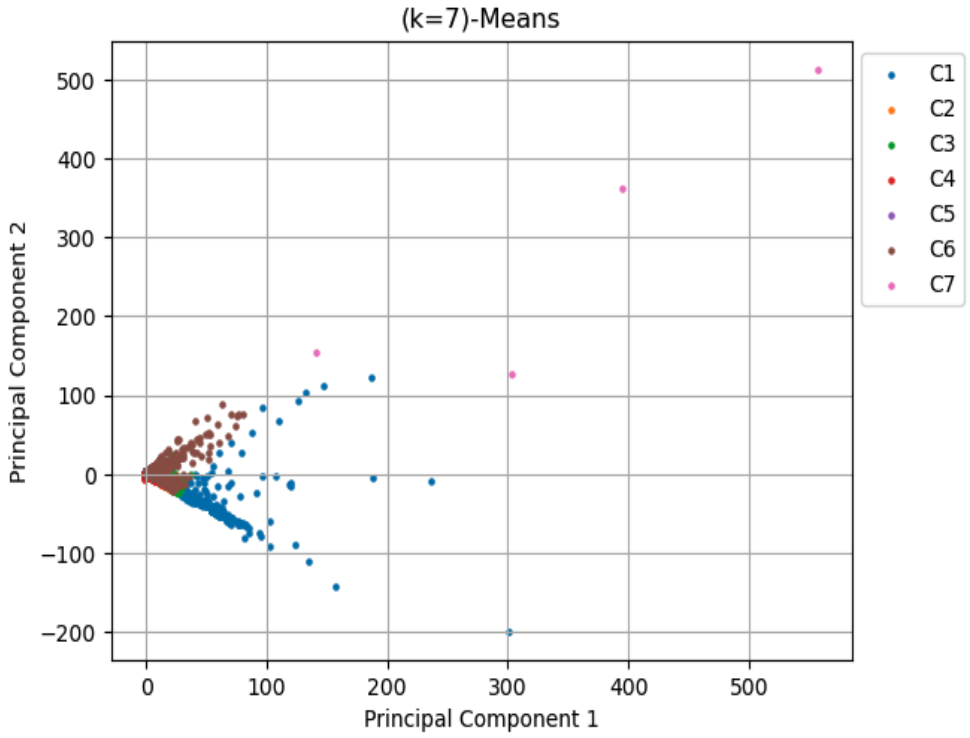




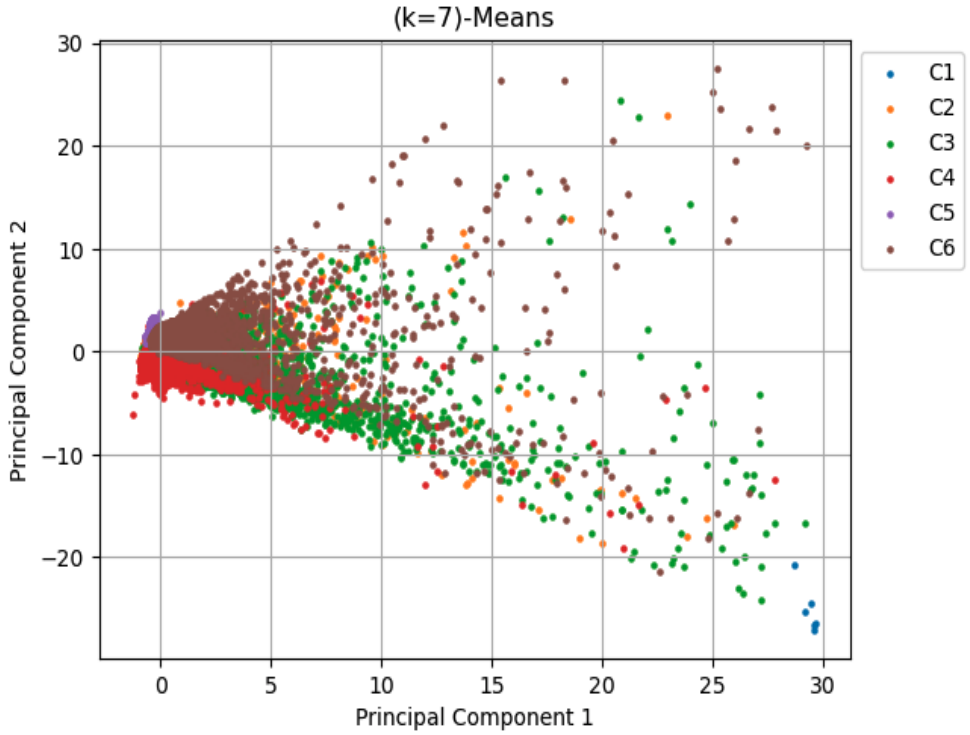
These parameters reveal that KMeans is maybe not detecting the underlying data pattern even though Silhouette's results are really good. At this point data visualisation is required.

3.2.4 Data Visualisation

At this point is mandatory to interpret those metrics through a visualisation of data points and clusters. First of all, PCA is involved to reduce dataset's dimensionality into two principal components: this is the only way to plot in a 2D human being understandable graph. Then using *plot_clustering* function the following representation is printed:

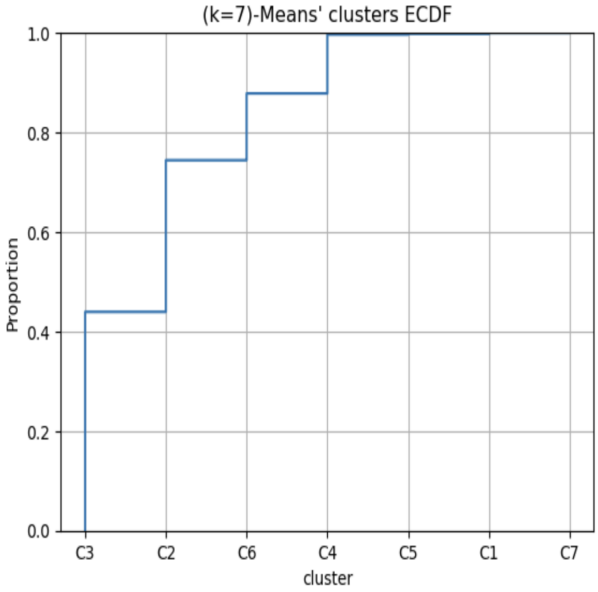


Now it is clear why, despite of a high Silhouette Score, both RI and ARI have low values: data is denser on graph's lower left side. For better visualisation all points having coordinates higher than $(30, 30)$ are excluded:



Points belonging to cluster $C7$ are those that were dropped to focus, all other points are gathered.

3.2.5 KMeans Clusters ECDF

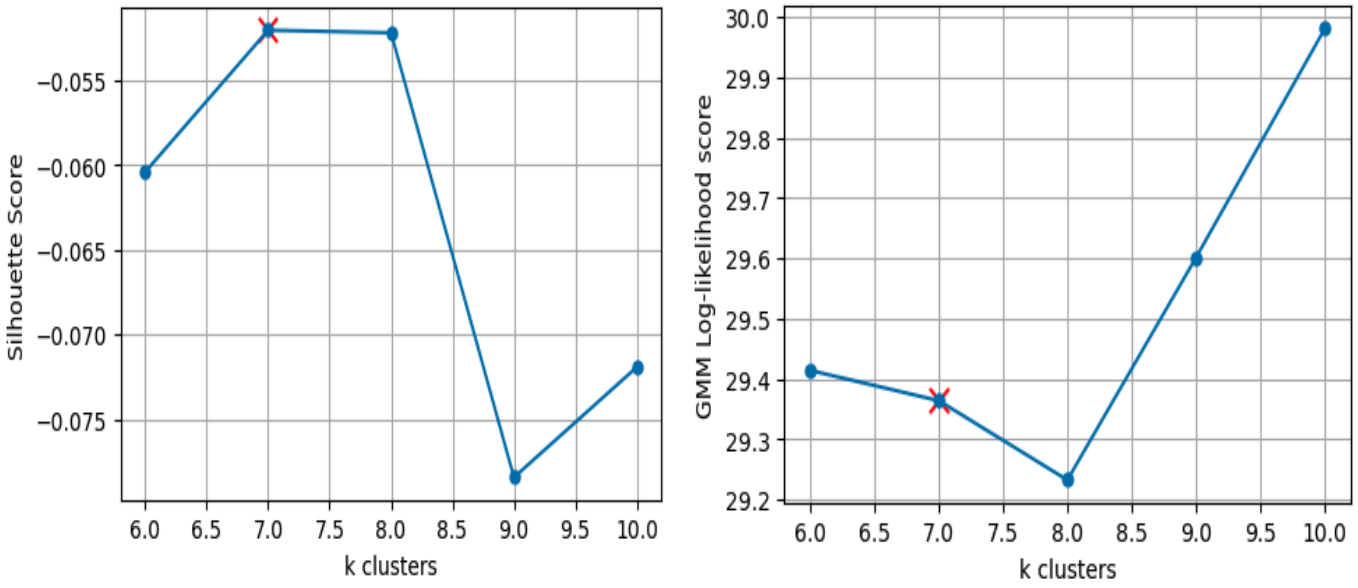


3.3 Gaussian Mixture Model Clustering

At this time a soft clustering method is used. The difference now is that not only silhouette score is useful to metric perfomances, but also *Log-Likelihood Score*: it is a parameter that represents the **probability** of data points' clusters assignments.

3.3.1 Looping over k

GMM in this case was initilized with hyper-parameter *init_params = 'kmeans'* to initialize the weights, the means and the precisions using KMeans, in order to have a better context to benchmark both of the algorithms and their efficiency. Even not considering $k = 7$ from KMeans computed Elbow Method, in jupyter's *2.1 Looping over k* the best result for clusters number is surprisingly still $k = 7$. Silhouette's and Log-Likelihood's plots follow:

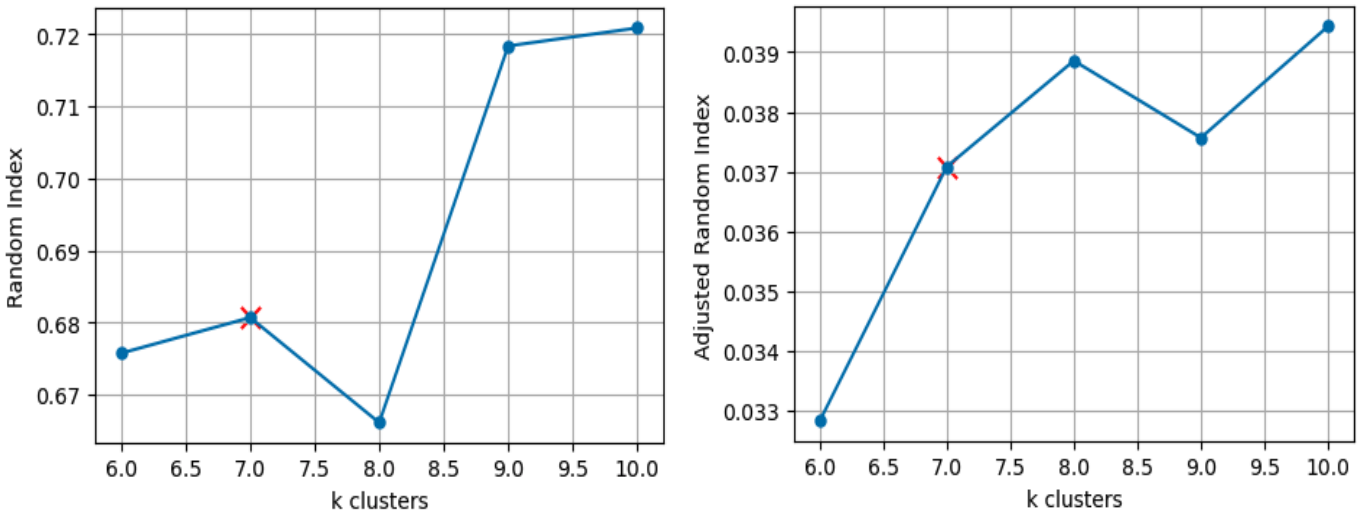


Best Silhouette's value is for $k = 7$ though it is a bad value (-0.052) and what that means is that data points are most likely overlapping. On the other hand, Log-Likelihood must be a trade off between maximization without overfitting data.

3.3.2 Perfomance Evaluation

In this case Random Index and Adjusted Random Index are sligthly better:

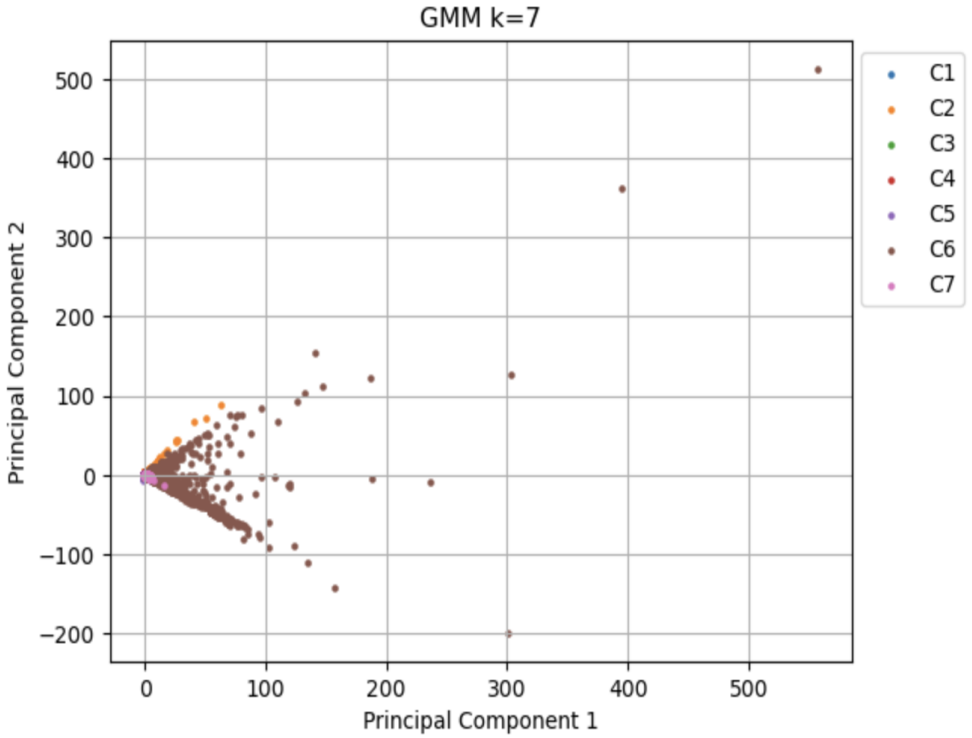
- $RI = 0.681$
- $ARI = 0.037$



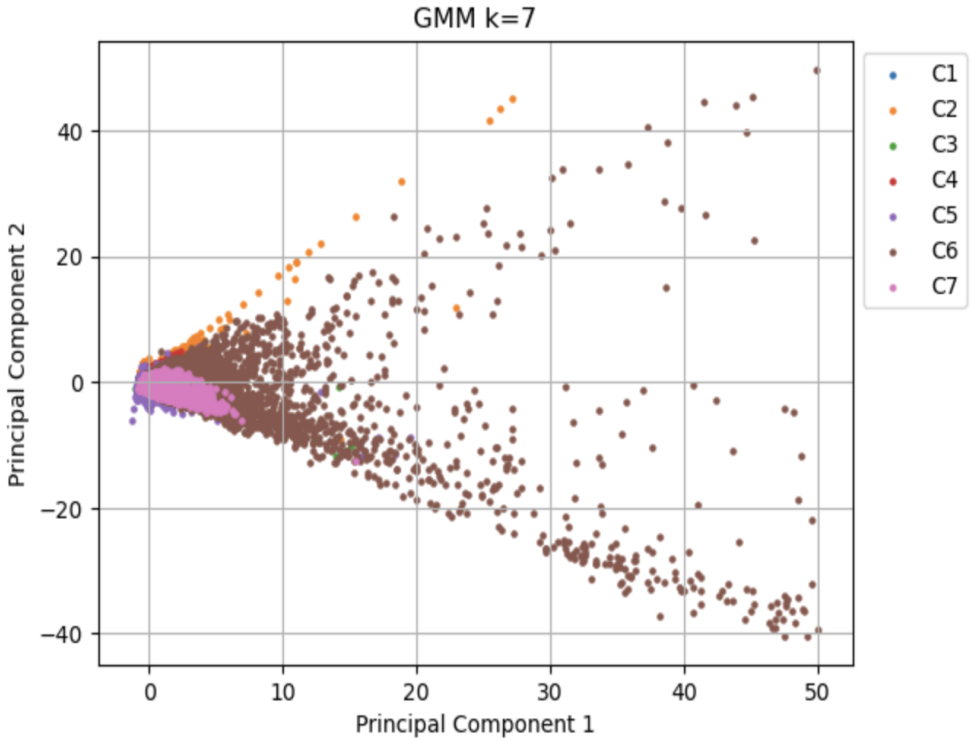
Even though these metrics are higher than KMeans, ARI is suggesting that the accuracy is no better than a random choice.

3.3.3 Data Visualisation

As previously done, a Principal Component Analysis is used for dimensionality reduction, then a 2D graph of datapoints colored by clusters' probability assignment is plotted:

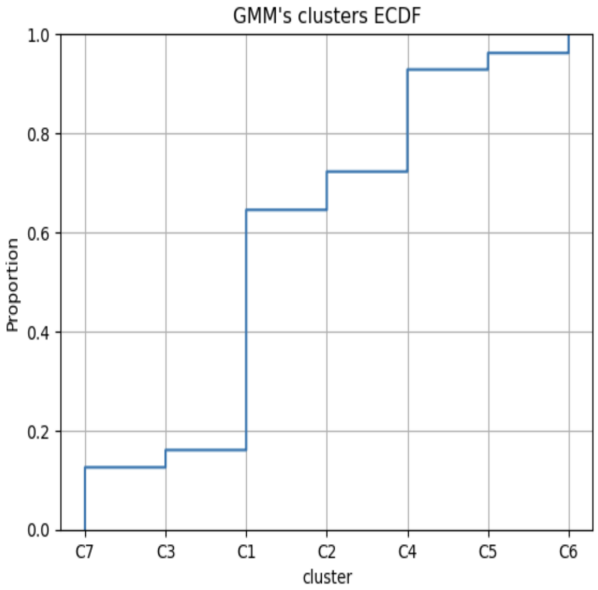


Focusing on data points having coordinates lower than (50, 50):



Clusters are definitely overlapping (as Silhouette showed) and it could be noticed that cluster $C6$ is the most populated one.

3.3.4 GMM Clusters ECDF



4 Regression: Estimate bytes transmitted and RTT

In this section two regression problems are faced: one for estimating the number of transmitted bytes (**s_bytes_all**) and the other for estimating the average round-trip time (**s_rtt_avg**). For each of the regression tasks the following models have been analysed:

1. Linear Regressor
2. Lasso Regressor
3. KNN

Data Preprocessing Before starting with the ML models' analysis, data was processed by normalizing it. Then the "related features" for each regression task are eliminated:

- **_s_bytes_uniq** for the bytes' regression problem
- all the features related to other measures of RTT for the RTT's regression problem

Unlike the classification problem, in this case, the closely related characteristics have been retained instead of eliminating them. This decision was made after observing that there is no increase in computational overload and, instead of improving performance, there is a loss. At the end the following datasets are available: the train set **X_s**, the validation set **X_val_s** (both recovered by splitting the initial training dataset) and the test set **X_test_s** (entirely recovered from the test dataset). The corresponding arrays to predict are **y_s**, **y_val_s**, and **y_test_s**.
From this point on, the analysis of each method has been performed by following these steps:

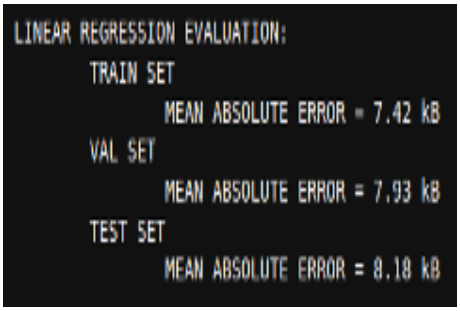
1. Model Training with default parameters
2. Hyper-parameters Tuning
3. Perfomance Evaluation

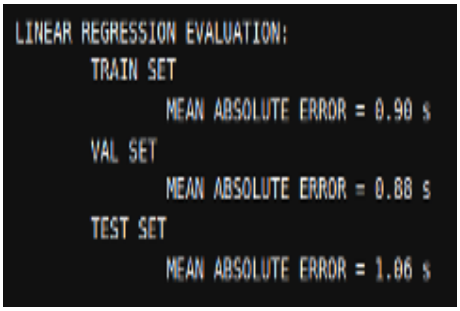
4.1 Linear Regressor

The goal of Linear Regression model is to minimize the sum of the squared differences between the observed and predicted values.

4.1.1 Model Training with default parameters

At first, the model is evaluated on the training set (**X_s**) with the default parameters. Regarding the results, it has been considered only the MAE (Mean Absolute Error), as required by the problem.





Outcomes show that there are light signs of overfitting, as the model could have adapted too much on the training set, both for the **_s_bytes_all** and **_s_rtt_avg**. For the first one, this degree of error of some kB is a good one, considering we are dealing with TCP stream. Instead, for what concerns RTT, a MAE of 1.06 seconds on the test set is quite big, considering RTT is on the order of some milliseconds. Furthermore, the MAE on validation set is lower than the one on the training set.

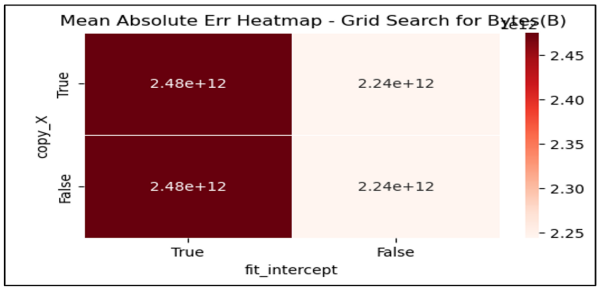
4.1.2 Hyper-parameters Tuning

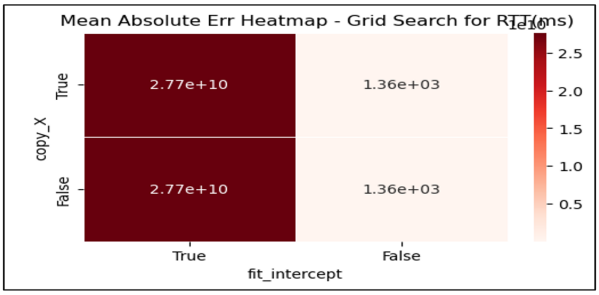
There are 2 main parameters to take into account:

- **fit_intercept**: it is a bool to be set true if you want to calculate the intercept for this model
- **copy_X**: if this bool is set True, X will be copied; else, it may be overwritten

The other 2 parameters, namely *n_jobs* and *positive*, don't affect the results, because the former change only the speed of the execution and the latter the sign of the result.

Grid Search The 2 parameters considered are all booleans, thus it has been decided to go for a grid search, as the validation curve is nonsense since there isn't a range to span. As scoring parameter was chosen *neg_mean_absolute_error*, and 5-fold cross-validation for both.



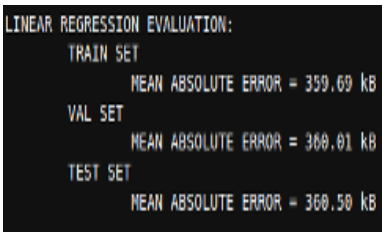


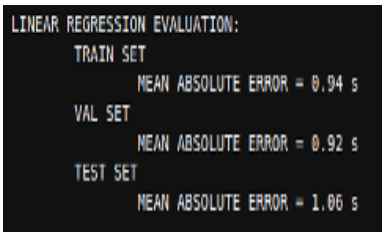
Results show that the parameter **copy_X** is irrelevant for the purposes of the research, so **fit_intercept** is the only parameter changing the error. Hence the best parameter combination in both cases is:

'copy_X': True, 'fit_intercept': False

4.1.3 Performance Evaluation

In the end, we perform an evaluation of the performance on the test set, and we compare it with the performance on the training and validation set.





In the first case, there is the same increasing error trend from training set to test set, although the error rate has increased a lot for every set. In the second case, there is an increasing of 0.04 seconds for training set and validation set, while the test set remain stable.

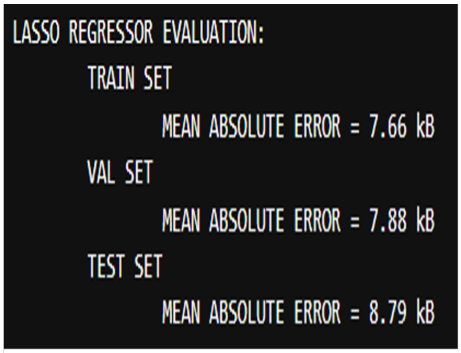
Considerations Since that this model is not robust to noise such as outliers, it was descerned to adopt LASSO Regressor which follows regularization techniques which allows to control the model's complexity, preventing overfitting (encouraging to generalize better to new data).

4.2 Lasso Regressor

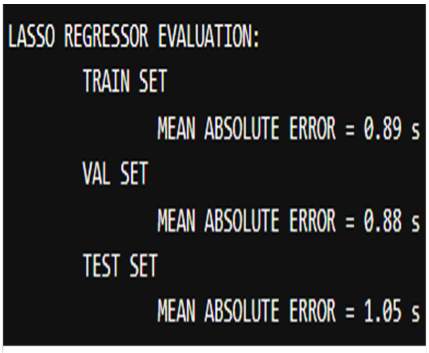
It is equivalent to the linear regressor where a penalty term (regularization term) is added in the squared loss in order to choose the simplest model. The regularization term is the hyper-parameter alpha declared in the sklearn library

4.2.1 Model Training with default parameters

First, an initial evaluation of the model is performed by training it on **X_s** with the default parameters. Then predictions have computed over each set **X_s**, **X_val_s** and **X_test_s**. The mean absolute error was considered for the evaluation (more significant in this case). Here the obtained results:



_s_bytes_all



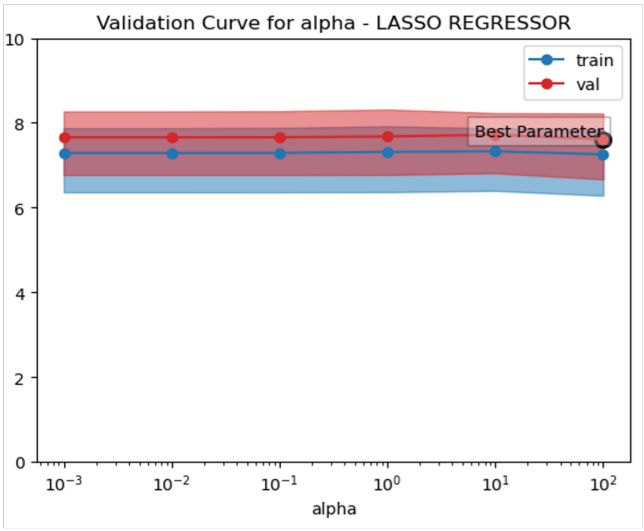
_s_rtt_avg

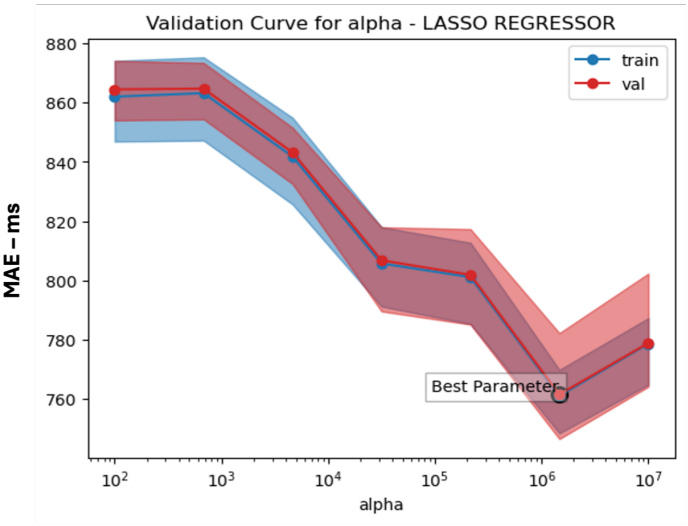
Regarding **_s_bytes_all** it seems that the model is performing well on both the training and validation sets. But there is a small drop in performance on the test set. This indicates a slight degree of overfitting which means that the model is not generalizing well what it has previously learnt on unseen data.
Regarding **_s_rtt_avg** it seems that the model does not show good performance (acceptable values for the RTT's MAE should be in the order of some ms). There are significant drops in performances on both validation (where the difference is about 1000 ms) and test set (where the difference is more relevant). This indicates a high degree of overfitting which means that the model is not generalizing well what it has previously learnt on unseen data.

4.2.2 Hyper-parameters tuning

To imporve model's performances, the tuning of the regularization term *alpha* was performed (obviously the hyper-parameter that has more impact on model's performance)

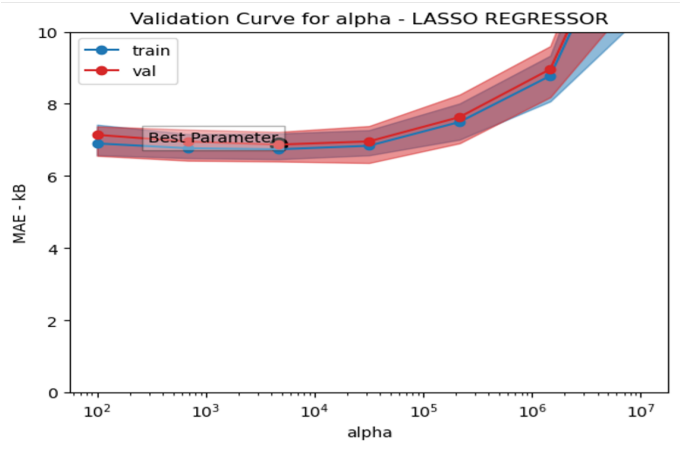
Validation Curve At this point, the impact of alpha has analysed by plotting the validation curve by considering an *alpha_range = [0.001, 0.01, 0.1, 1, 10, 100]* for the bytes' problem and an *alpha_range = np.logspace(2, 7, 7)* for the RTT's problem, *neg_mean_absolute_error* as scoring and a 5-fold cross-validation. The output is the following:

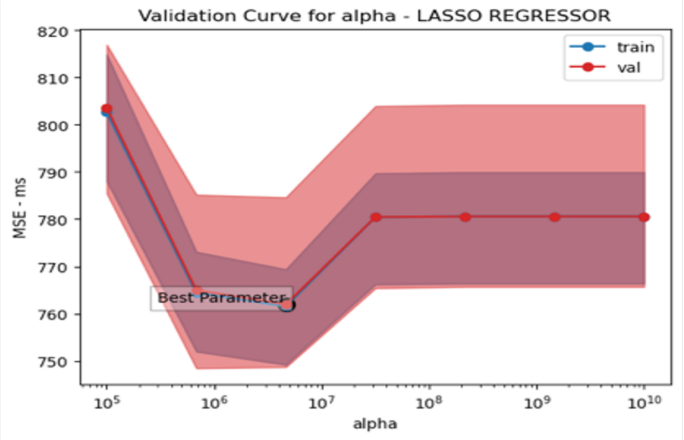




Regarding **_s_bytes_all** the *best_parameter* and the *best_val_score* obtained are respectively equal to 100 and 7.61 kB. In the selected range there is no significant variability of the errors. For this reason, the range has then been expanded to better analyse the behaviour of the model.
Regarding **_s_rtt_avg** it seems that the model does not show good performance (acceptable values for the RTT's MAE should be in the order of some ms). There are significant drops in performances on both validation (where the difference is about 1000 ms) and test set (where the difference is more relevant). This indicates a high degree of overfitting which means that the model is not generalizing well what it has previously learnt on unseen data.

The new considered alpha ranges for both the problems (**_s_bytes_all** and **_s_rtt_avg** ones) are respectively equal to *np.logspace(2, 7, 7)* and *np.logspace(5, 10, 7)*. The obtained curves are the following:

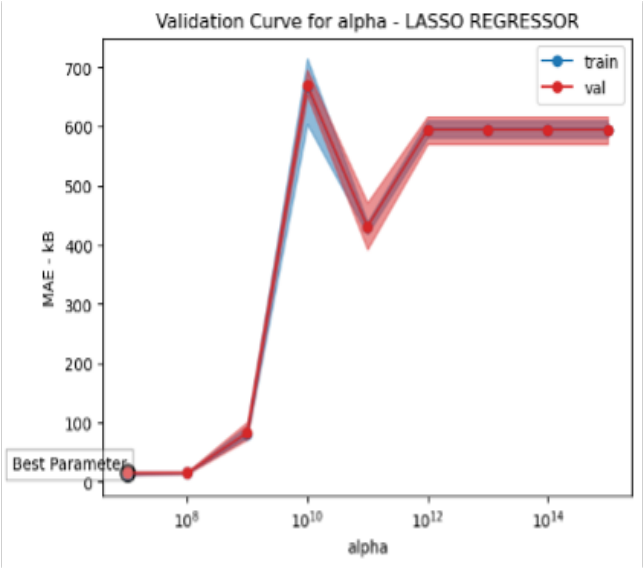




Regarding **_s_bytes_all** the *best_parameter* and the *best_val_score* obtained are respectively equal to 4641.59 and 6.87 kB. It is noticed that starting from a value of alpha close to 105 both training and validation's errors starts to growing up.
In both the first and second validation curves concerning the bytes problem, there is no significant difference between model's performance on training and validation set. This difference becomes smaller with the increasing of the alpha's value. It may be due to the Lasso's regularization technique which allows to control the model's complexity, preventing overfitting (encouraging to generalize better to new data).

Regarding **_s_rtt_avg** the best parameter and the best validation score obtained are respectively equal to 10^6 and 762 ms. After the error starts growing up until it reaches a stable value close to 780 kB, showing that the retrieved parameter is the best one.

A third validation curve for the **_s_bytes_all** problem has been plotted now in order to observe the model performance when alpha tends to infinite.



In this section the error keeps on growing until it reaches a stable value close to 600 kB, showing evidence that the previous minimum found isn't only the local one, but the global one. Indeed, the error after alpha equal to 10^{12} is stable.

4.2.3 Performance Evaluation

Finally, the obtained best model has been tested on the three sets (train, validation and test), to observe if some improvements in performance have been reached. Here the obtained results:

```
LASSO REGRESSOR EVALUATION:
        TRAIN SET
                MEAN ABSOLUTE ERROR = 5.97 kB
        VAL SET
                MEAN ABSOLUTE ERROR = 5.90 kB
        TEST SET
                MEAN ABSOLUTE ERROR = 5.86 kB
```

_s_bytes_all

```
LASSO REGRESSOR EVALUATION:
        TRAIN SET
                MEAN ABSOLUTE ERROR = 780.58 ms
        VAL SET
                MEAN ABSOLUTE ERROR = 769.07 ms
        TEST SET
                MEAN ABSOLUTE ERROR = 852.13 ms
```

_s_rtt_avg

Regarding **_s_bytes_all** the model seems to be well-tuned (the error before the tuning was reduced of a value close to 3 kB) and capable of generalizing effectively to both the validation and test sets, with small differences in mean absolute error's values between the three sets. The model appears to be more robust, and there is a reduced risk of overfitting.
Regarding **_s_rtt_avg** the model is continuing to show bad performance, but the errors obtained are slightly better. Particularly, the validation set's MAE is lower than that of the training set, while the test set's error shows a significant increase. This discrepancy suggests potential overfitting issues and highlights the difficulty of the regression problem.

4.3 KNN Regressor

4.3.1 Model Training with default parameters

First, an initial evaluation of the model is performed by training it on **X_s** with the default parameters. The mean absolute error was considered for the evaluation (more significant in this case). Here the obtained results on the two regression task (**_s_bytes_all** and **_s_rtt_avg**):

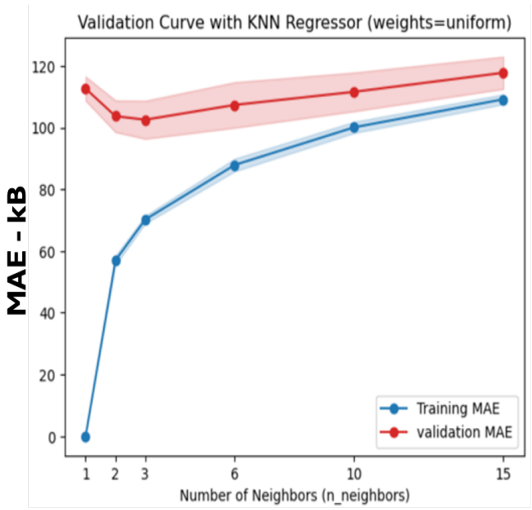
```
K-NEAREST NEIGHBORS REGRESSOR EVALUETION:
        TRAIN SET:
                MEAN ABSOLUTE ERROR: 80.74 kB
        VAL SET:
                MEAN ABSOLUTE ERROR: 93.97 kB
        TEST SET:
                MEAN ABSOLUTE ERROR: 136.18 kB
```

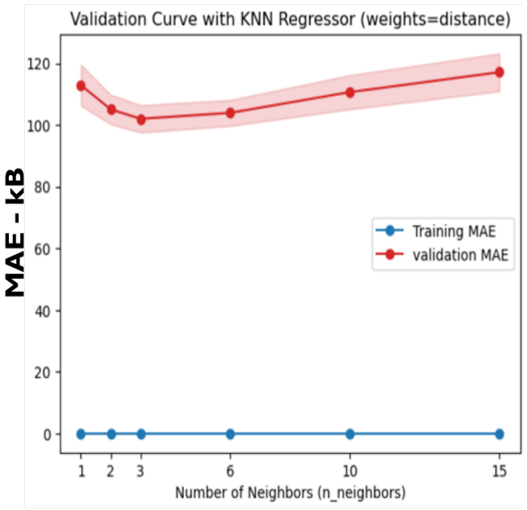
```
K-NEAREST NEIGHBORS REGRESSOR EVALUETION:
        TRAIN SET:
                MEAN ABSOLUTE ERROR: 0.34 s
        VAL SET:
                MEAN ABSOLUTE ERROR: 0.43 s
        TEST SET:
                MEAN ABSOLUTE ERROR: 0.68 s
```

The result over the **_s_rtt_avg** does not show good results. There is a really high MAE, the value **_s_rtt_avg** should be in the order of milliseconds. High error could be indicative of underfitting.
Looking at the results over **_s_bytes_all** it can see that there is a difference between train and test set, but the error is less relevant compared to the previous case.

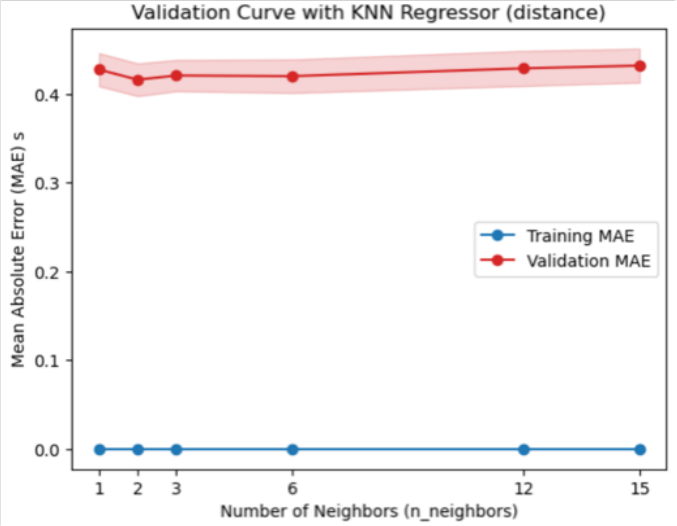
4.3.2 Hyper-parameters tuning

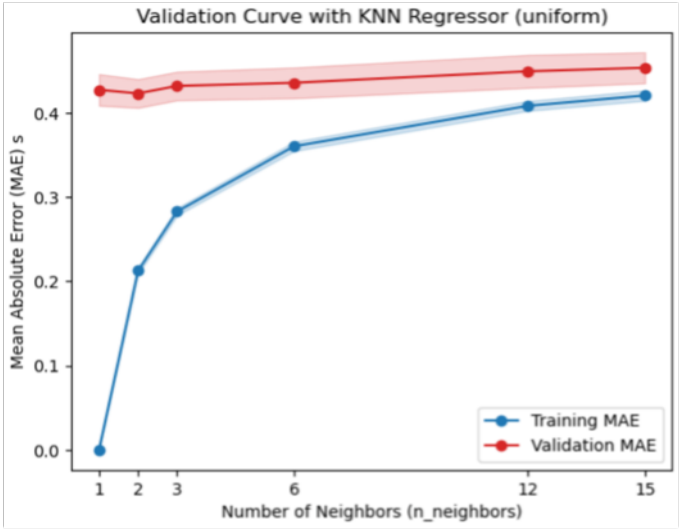
To improve the model, a validation curve is now conducted to observe how the error varies with changes in the *n_neighbors* parameter for both regression tasks. In both cases, a learning curve has been generated with two different weights. The first iteration considers the weight as *distance* and the second iteration employs the *uniform* weight. The obtained validations curve for **_s_bytes_all** are





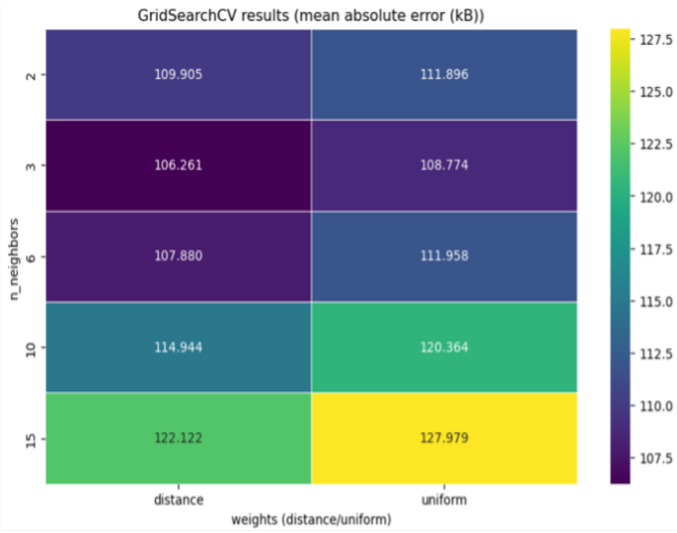
The parameter *n_neighbors* vary in the range $[1, 15]$. The decision was made to vary the parameter within the range [1;15] because as widening the range of parameters would not have been useful for the purposes of the problem and would have increased computational costs. As observed in the graphs, the error initially decreases and than it start to increase with an increasing number of neighbors for both cases.
In the graph obtained by choosing distance as weight, a flat curve is evident for the training set, with a very low error. In contrast, the validation set exhibits a higher error that varies with the parameter, indicating the presence of overfitting.
Examining the graph associated with uniform weight it is observed that the difference between the two curves is higher, but decrease with the increasing of the number of neighbor. The validation curve reach the minimum error when the number of neighbors is 3. We now present the results obtained for the **_s_rtt_avg** task:

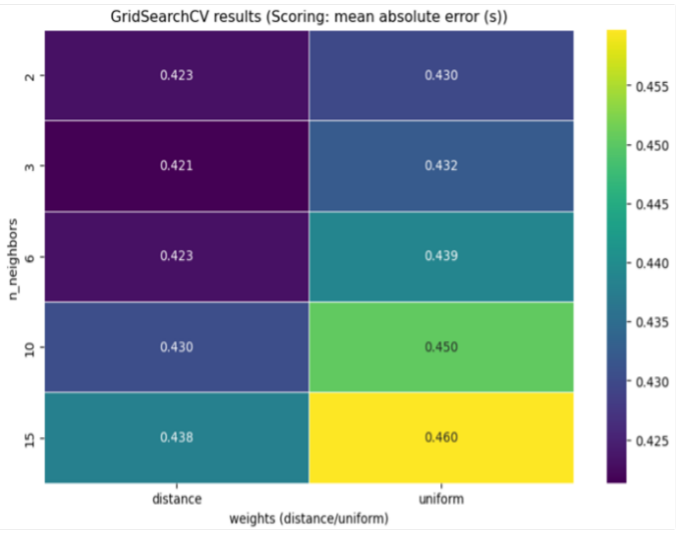




In general the validation curve follows the same behaviour of the previous case. In this scenario the validation curve has less variation in the error in the range $[1, 15]$, however it can see that the error reach the lower error in 2 and, from that point, it begins to increase.

After the validation curve, the research of the best parameter to use for the model is computed. This is performed thanks GridSearchCV to assess how the model's mean absolute error varies with different parameter combinations





As validation curve showed, error increases as long as neighbors number does. From the graphs it is clear that the best combination is *n_neighbors=3* and *weight='distance'*.

4.3.3 Performance Evaluation

Finally, the obtained best model has been tested on the three sets (train, validation and test), to observe if some improvements in performance have been reached. Here the obtained results:

```
K-NEAREST NEIGHBORS REGRESSOR EVALUETION:
        TRAIN SET:
                MEAN ABSOLUTE ERROR: 0.00 kB
        VAL SET:
                MEAN ABSOLUTE ERROR: 99.20 kB
        TEST SET:
                MEAN ABSOLUTE ERROR: 268.27 kB
```

```
K-NEAREST NEIGHBORS REGRESSOR EVALUETION:
        TRAIN SET:
                MEAN ABSOLUTE ERROR: 0.00 s
        VAL SET:
                MEAN ABSOLUTE ERROR: 0.40 s
        TEST SET:
                MEAN ABSOLUTE ERROR: 0.67 s
```

After the grid search the model ended up to be too adapted to the test set. This shows a high level of overfitting. Indeed the error increase significantly in the test set.

Conclusions Summarizing, the model that generates the best performance is the LASSO Regressor, which shows good performance in bytes' regression task and a slight degree of overfitting in the RTT regression task. Instead, the k-NN and the Linear Regressors provide worse performance showing respectively a high degree of overfitting and a too much sensitivity to outliers.

The problem related to estimating the average round-trip time (**_s_rtt_avg**) shows a higher MAE compared to the problem of estimating the number of transmitted bytes (**_s_bytes_all**). This exhibits that the round-trip time regression task is more challenging and less accurate compared to the bytes' regression task. Therefore, this problem has proven to be more difficult to be solved.