Politecnico di Torino

Content Delivery Network of YouTube traffic

CESTA INCANI LUCA 305963
CUGLIARI ALEX 343766
DI PEDE ANTONELLO 347895
INSALACO GIUSEPPE 343410

Author's address: Cesta Incani Luca 305963
Cugliari Alex 343766
Di Pede Antonello 347895
Insalaco Giuseppe 343410.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Contents

0 INTRODUCTION

The aim of this project is to design an unsupervised machine learning technique to track changes in the YouTube CDN and analyze the differences between the most critical features during stable conditions and periods of significant alterations.

0.1 Explanation of the Code's configuration

In the first part we can find all the imports of the libraries that we will use during the entire execution of the program.
In the next part there are the Flags, where we are able to set if we want to apply the filter for the "outlier", in fact if we see some data that have a distance greater then the mean (and median) ± 2 * std, we consider it as a outlier.

```
RUN_OUTLIER_FILTER = True        # Se True, abilita il filtraggio degli outlier in base a min_rtt
USE_IP = True                    # Se True, scompone gli IP in ottetti
RUN_PCA = True                   # Se True, esegue PCA
```

In the second part of the Flags we can activate or deactivate some of the machine learning algorithms, to better understand which of them can be useful to solve the problem. The same is for the part of the clustering algorithm in which you can manually choose a number of clusters or let the "computer" understand which number is the best.

```
RUN_ML = True
RUN_LINEAR_REGRESSION = True
RUN_KNN = True
RUN_RANDOM_FOREST = False         # Probabilmente RF non    la scelta migliore per questo task
RUN_LASSO = True
TRAIN_ON_WEEK1_VALIDATE_OTHERS = True
RUN_TEST_WEEKS = True

# --- Clustering (K-Means) ---
RUN_CLUSTERING = True
KMEANS_AUTO = True
KMEANS_N_CLUSTERS = 12
CLUSTER_OTHER_WEEKS = True
```

In the third part of the Flags we can initialize which of the different plots we want to see.

```
RUN_PLOTS = True
RUN_PLOTS_ECDF = True
RUN_PLOTS_SCATTER = True
RUN_PLOTS_CLUSTER = True
RUN_PLOTS_IP_2D = True
RUN_PLOTS_PCA_3D = True
RUN_PLOTS_PCA_3D_EDGENODE = True
```

In the last part of the Flags we have to decide how much power from our pc we want to use to calculate the machine learning algorithms.

```
PARALLEL_LEVEL = 8
MODEL_COMPLEXITY = 2
```

1 SECTION 1 – DATA EXPLORATION AND PRE-PROCESSING

In this first section we investigated the provided dataset, in particular the first week which was indicated as baseline. We analyzed the behavior of some specific features (the ones playing significant roles) on different flow levels. We used data visualization techniques and statistical analysis to understand the behavior of the features and then we filtered out the useless information (outliers). We considered an outlier to be any data point that lies significantly far from the mean or median of the dataset.

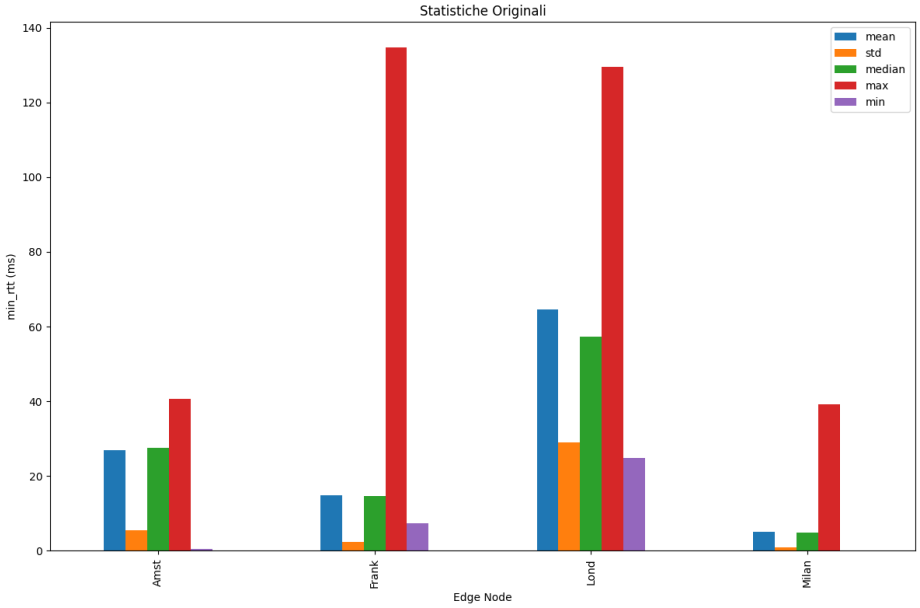
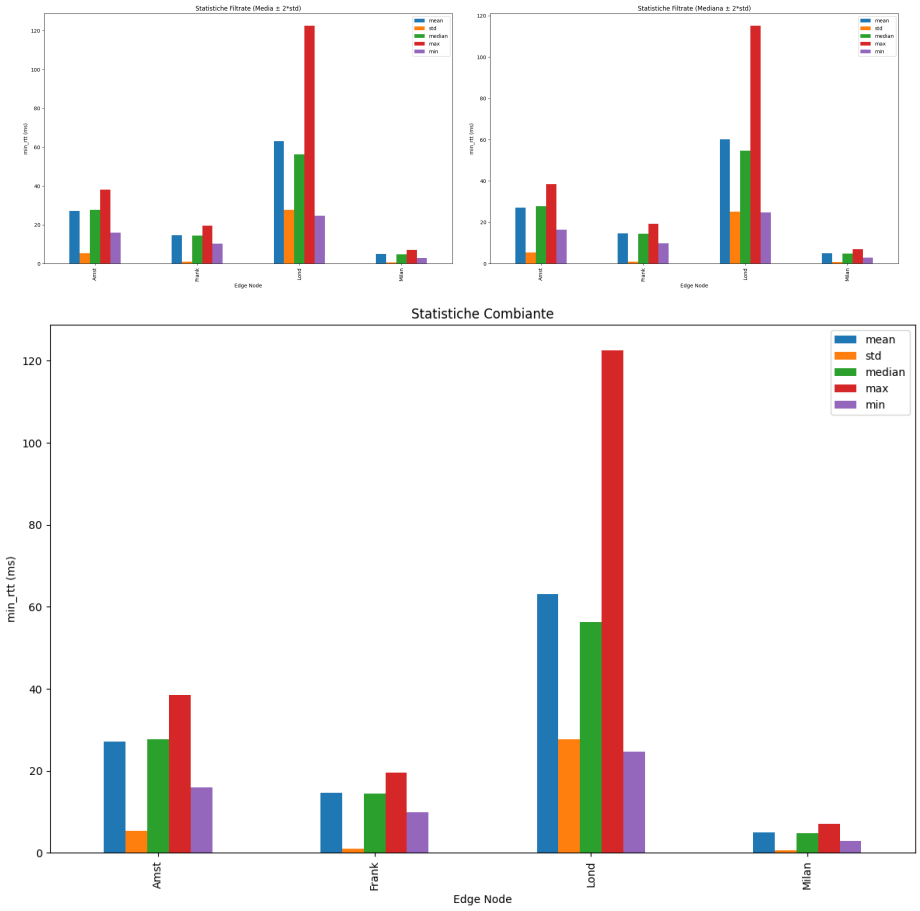


Fig. 1

1.1 Filtering of the outlier

In fact we can clearly see that in the Fig. 1, in particular in Frank, there are some "samples" which have some uncommon characteristics that depict some anomalies. As a result of our tests, we were able to confirm that using the formulas $mean \pm 2(std)$ and $median \pm 2(std)$, the data included in the range for each edge-node were probably correct and concordant with the theory, while we considered every point outside the interval as an outlier. All the considerations we will make from here on will be applied on data filtered by min_rtt and throughput.

Below the graphs of the filtered data are displayed:



After filtering we got :

Week1 original length: 3024852
Week1 filtered (mean) length:2915542
Week1 filtered (median) length: 2915752

So we were able to remove approximately 100,000 samples from our dataset. Additionally, it is important to note our approach to eliminating duplicates. The database contained numerous duplicate records, identifiable by instances where a host and a server exchanged the same amount of data at the same time, an occurrence that is, at best, questionable. Hence, these duplicate entries were removed.

When exploring data at a cache level, we aggregated all flows that shared the same server_ip. Although the provided script does not explicitly show a “groupby server_ip and compute aggregated stats” step for each cache, the concept is straightforward: we could group by server_ip, then compute average min_rtt, average throughput, the total or average uniq_byte, and so on. This aggregated data would represent a single cache line. Similarly, to get the edge-node level, we would group by edge_node. Since edge_node is a textual identifier that indicates a bigger group of caches (the entire edge-node), we could again compute overall statistics like the distribution of RTT or throughput across all flows within that edge-node.

1.2 Creation of the correlation matrix

Subsequently, we constructed a correlation matrix that enabled us to identify which features were correlated. If two features exhibit a high degree of correlation, they essentially convey the same information, making it unnecessary to retain both; the same reasoning applies to features with a strong negative correlation.

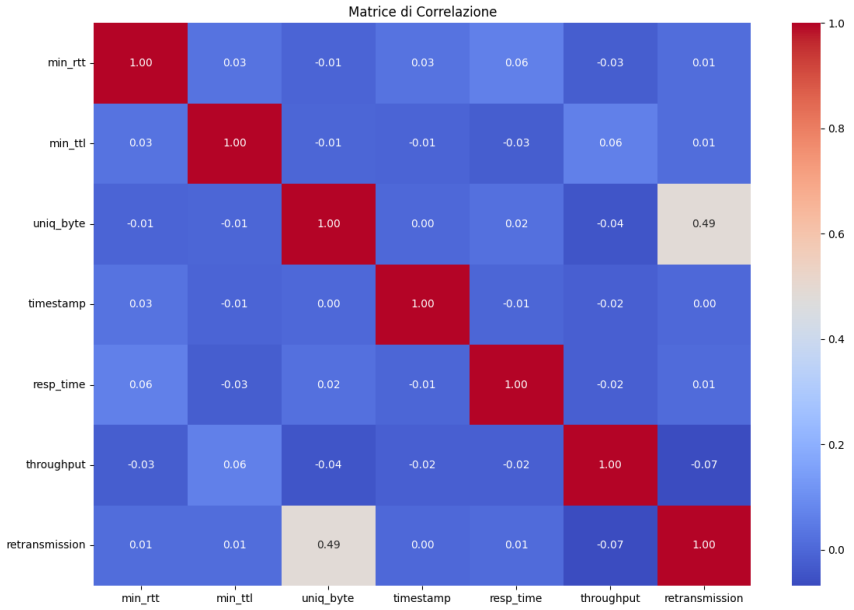


Fig. 2

From the matrix, we observe that each feature is, as expected, perfectly correlated with itself. Overall, the features appear to be independent with one notable exception. The correlation between *uniq_byte* and *retransmission* is 0.49. Based on our hypothesis, this relationship likely arises because some TCP connections encountered errors, prompting the protocol to retransmit information. Consequently, it does not necessitate any adjustments since the features are largely independent overall.

1.3 PCA

PCA (Principal Component Analysis) is a technique that simplifies large data sets by reducing their size. This process consists of transforming the initial variables, often correlated with each other, into a more compact set of principal components, capable of preserving most of the original information. In our case we set the threshold to the 90% of the cumulative variance. Since the first eleven components represent this percentage we decided to remove the remain four features having a very small impact on the overall information. First of all we decided to convert an ip from one feature to 4 because, according to us, that gives the possibility to the PCA algorithm to have more freedom in the job of removing the "useless" features. What we have done is :

server_ip= 127.0.0.1 -> *s_ip*1=127, *s_p*2=0, *s_ip*3=0, *s_ip*4 = 1;

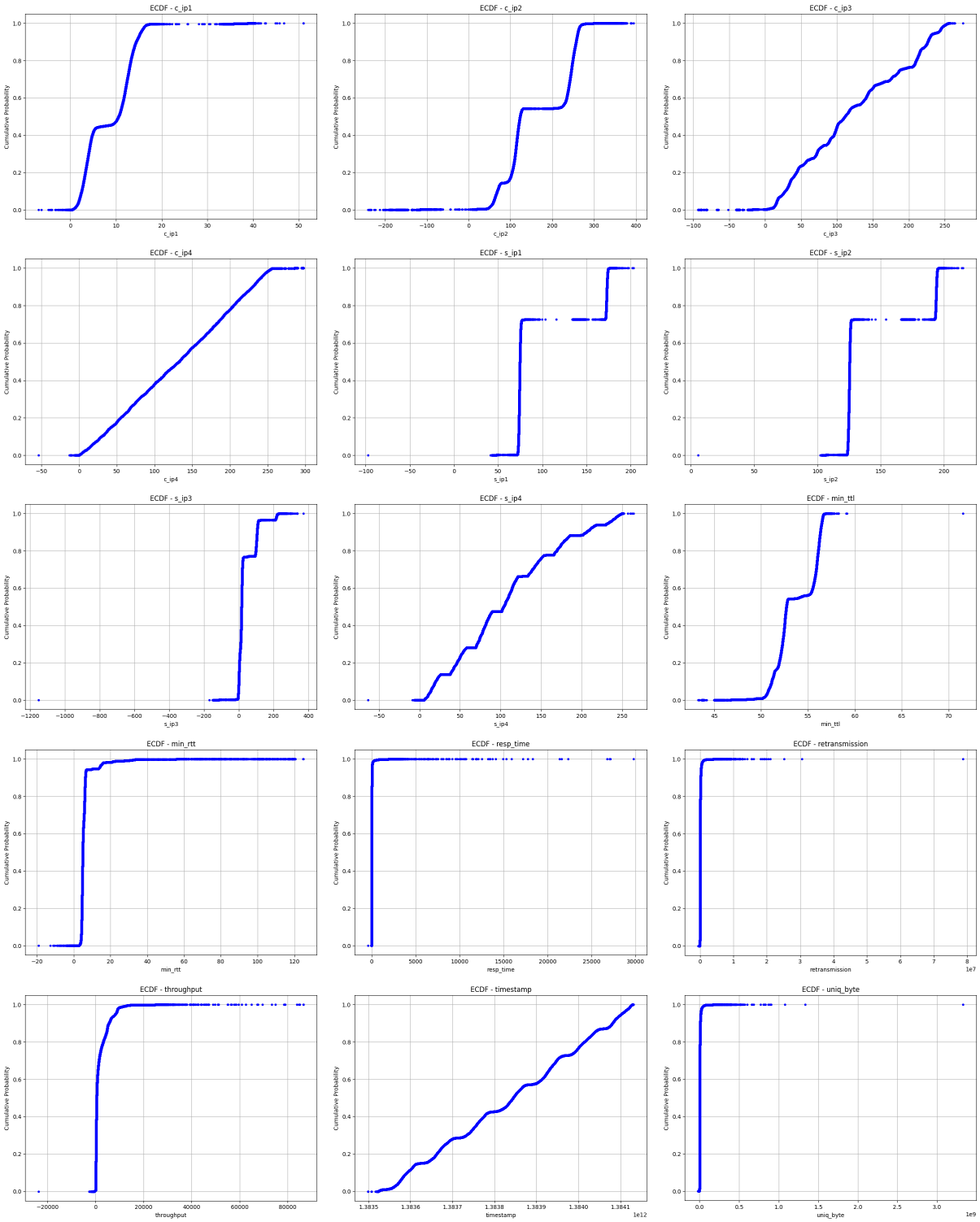
As we well know the first octet is strongly linked with the geographical position and the relationship becomes less pronounced as we move towards the fourth octet.

Table 1. Variance (Week_1)

Principal Components	**PCA1**	**PCA2**	**PCA3**	**PCA4**	**PCA5**	**PCA6**	**PCA7**	**PCA8**	**PCA9**	**PCA10**	**PCA11**
Variance (%)	17.00	15.82	9.99	7.91	6.80	6.68	6.57	6.33	6.10	4.98	3.88

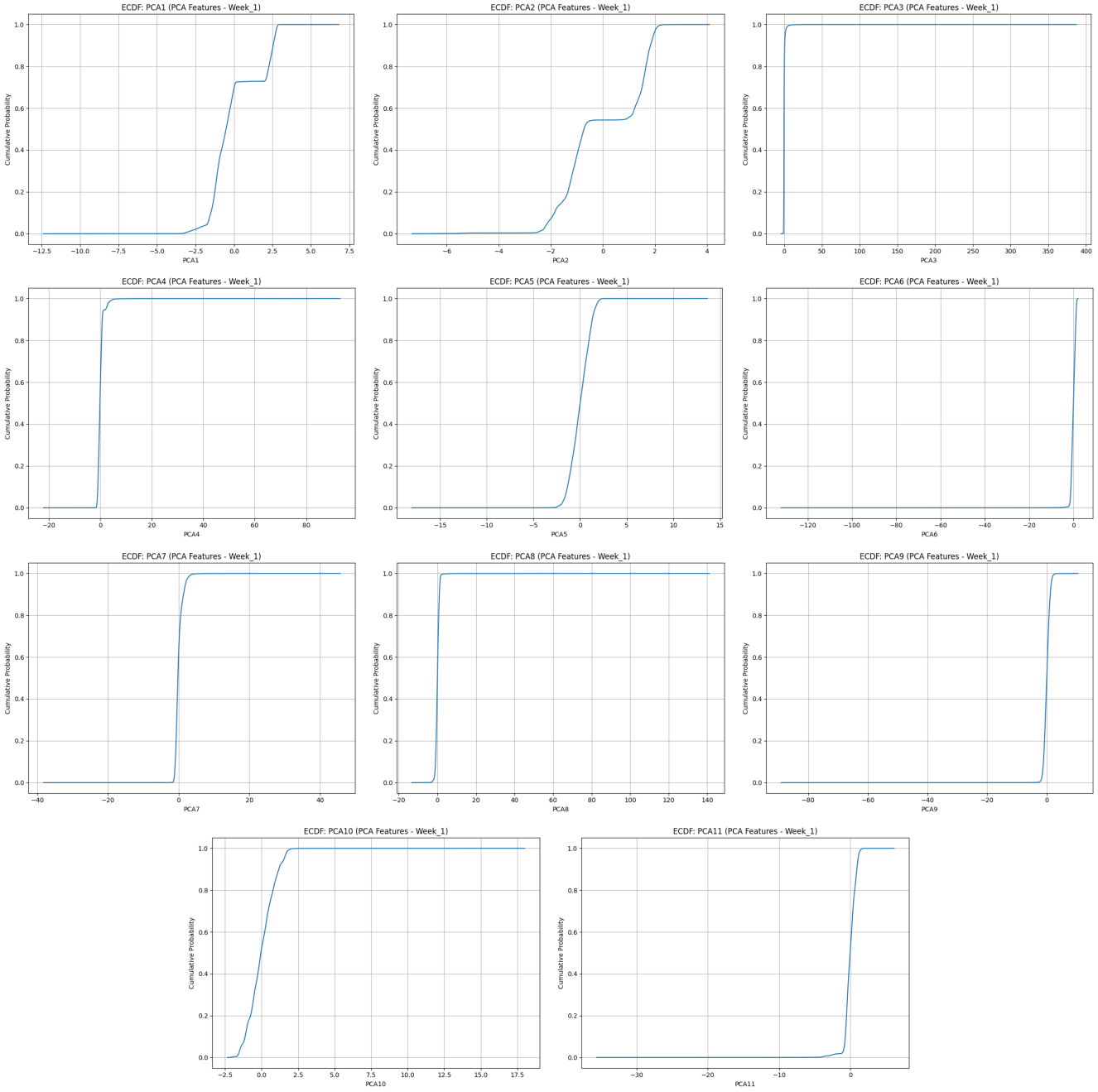
1.4 ECDF original features

The following figures display the reconstructed ECDF of the features composing the flow.



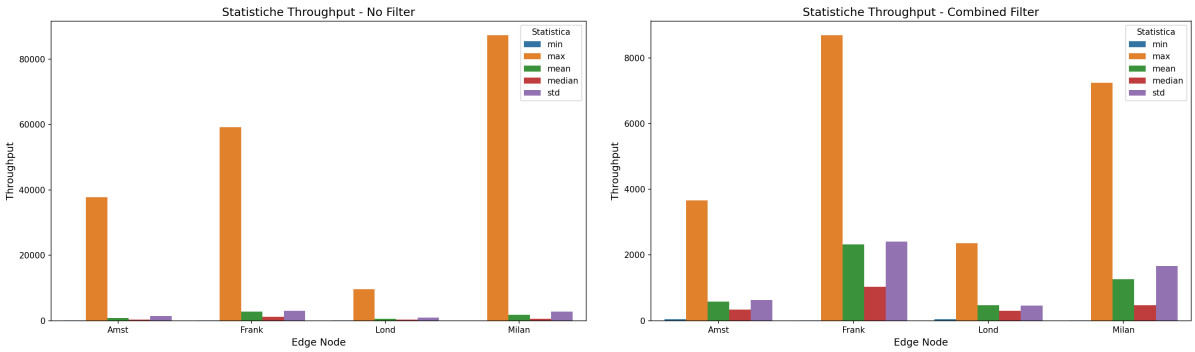
1.5 ECDF PCA features

The plots below show the ECDF of different PCA related to week 1.



2 SECTION 2 – SUPERVISED LEARNING – REGRESSION – ESTIMATE THROUGHPUT OF THE FLOW

Throughput represents the amount of data actually transmitted through the network in a certain time interval, and is usually measured in bits per second (bps) or its variants (Mbps, Gbps). It indicates the actual rate at which information is delivered between devices, taking into account any inefficiencies, delays and network overhead. High throughput suggests a high-performing network capable of handling large amounts of traffic, while low throughput may indicate problems with congestion, delays or packet loss.



Then we used the throughput as a target column and we chose which machine learning algorithm to use. Among the most popular we have:

- Linear Regression: a regression model that tries to find a linear relationship between the independent variables (input) and the dependent variable (output) through a straight line that minimizes the mean square error.
- Lasso Regression: a variant of linear regression that includes an hyperparameter alpha to reduce the absolute value of the coefficients, favoring the selection of the most important features and reducing the risk of overfitting.
- Random Forest: supervised learning model based on a set of decision trees. It combines the predictions of multiple trees to improve accuracy, reduce the risk of overfitting and handle nonlinearity in the data.
- K-Nearest Neighbors (KNN): a proximity-based algorithm that predicts the value of a new observation using the values of the K closest data points in the dataset, considering a distance measure, such as Euclidean.

Initially, we planned to use a Linear Regressor, Random Forest, and KNN. However, we soon realized that the computational resources required for the Random Forest exceeded what was available to us and we decided to substitute Random Forest with Lasso Regression. For each ML method we selected 2 distinct metrics: MSE and R^2.

Algorithm	Pros	Cons
Linear Regression	- Simple and easy to interpret. - Computationally efficient. - Provides a good baseline for complex models.	- Performs poorly with non-linear relationships. - Sensitive to outliers. - Requires certain assumptions (linearity, independence).
Lasso Regression	- Automatic feature selection. - Reduces overfitting. - Improves interpretability over complex models.	- May exclude too many variables with high correlation. - Sensitive to the choice of regularization parameter. - Not suitable for highly non-linear data.
Random Forest	- Robust and accurate for complex and non-linear problems. - Handles both categorical and continuous data well. - Reduces overfitting with bagging.	- Computationally more expensive. - Less interpretable compared to linear models. - Requires parameter tuning for optimal performance.
K-Nearest Neighbors (KNN)	- Easy to implement. - Suitable for non-linear data. - No assumptions about data distribution.	- Computationally expensive on large datasets. - Sensitive to the choice of k and distance metric. - Vulnerable to noisy data and outliers.

The document below summarizes the validation results for several weeks. The table reports the original and filtered data sizes along with performance metrics (MSE and R^2) for three algorithms.

Week	Original Data	Filtered Data	Algorithm	Metrics
Week 2	2,669,770	1,268,144	Linear Regression	MSE = 6906.662, $R^2 = 0.998$
			KNN Regressor	MSE = 248627.904, $R^2 = 0.924$
			Lasso	MSE = 6793.361, $R^2 = 0.998$
Week 3	1,301,176	1,253,008	Linear Regression	MSE = 8093.497, $R^2 = 0.998$
			KNN Regressor	MSE = 425496.657, $R^2 = 0.882$
			Lasso	MSE = 7920.960, $R^2 = 0.998$
Week 4	1,034,994	992,168	Linear Regression	MSE = 25536.057, $R^2 = 0.991$
			KNN Regressor	MSE = 2979864.186, $R^2 = -0.080$
			Lasso	MSE = 22553.711, $R^2 = 0.992$
Week 5	5,505,454	1,177,328	Linear Regression	MSE = 37810.551, $R^2 = 0.991$
			KNN Regressor	MSE = 4896250.281, $R^2 = -0.106$
			Lasso	MSE = 35171.982, $R^2 = 0.992$
Week 6	1,387,605	1,384,750	Linear Regression	MSE = 18693.429, $R^2 = 0.997$
			KNN Regressor	MSE = 5244797.909, $R^2 = 0.118$
			Lasso	MSE = 16542.179, $R^2 = 0.997$

The validation metrics reveal that both Linear Regression and Lasso Regression perform excellently without any sign of overfitting, as evidenced by their high R^2 and low MSE values. On the other hand, the KNN Regressor appears to underfit the data, likely due to suboptimal parameter selection or the unsuitability of the KNN approach for this linear dataset (greater susceptibility to outliers and noise). According to us a possible cause could be due to the fact that after 3 weeks there are new edge nodes and the server cache is constantly uploaded since first week.
Summarizing:

(1) We trained three models (Linear Regression, KNN, and Lasso) on Week_1.
(2) On the training set, all three models have a very low MSE (below 34k) and a very high R^2 (0.996–0.999), indicating that they fit Week_1 thoroughly.
(3) Linear Regression and Lasso maintain good performance on Weeks 2–3: MSE stays relatively low (about 6k–8k), and R^2 remains extremely high (0.998).
(4) Even in Weeks 4–5, although their MSE increases (25k–37k), R^2 hovers around 0.99, still indicating a decent generalization despite the shift in data.
(5) KNN, however, does well on Week_1 (MSE 33k, R^2=0.996) but sees its MSE surge in Weeks 4–5 (above 2.9M and 4.8M), with R^2 going below zero, this strongly indicates overfitting or a mismatch with the new data.
(6) Linear Regression and Lasso only show moderate performance drops in Weeks 4–5, implying they are more robust and do not severely overfit Week_1.
(7) Consequently, KNN is likely overfitting to Week_1, fantastic on training but struggling to generalize in later weeks.
(8) Linear Regression and Lasso exhibit stable MSE and high R^2 across most weeks, suggesting they generalize well and avoid underfitting or severe overfitting.

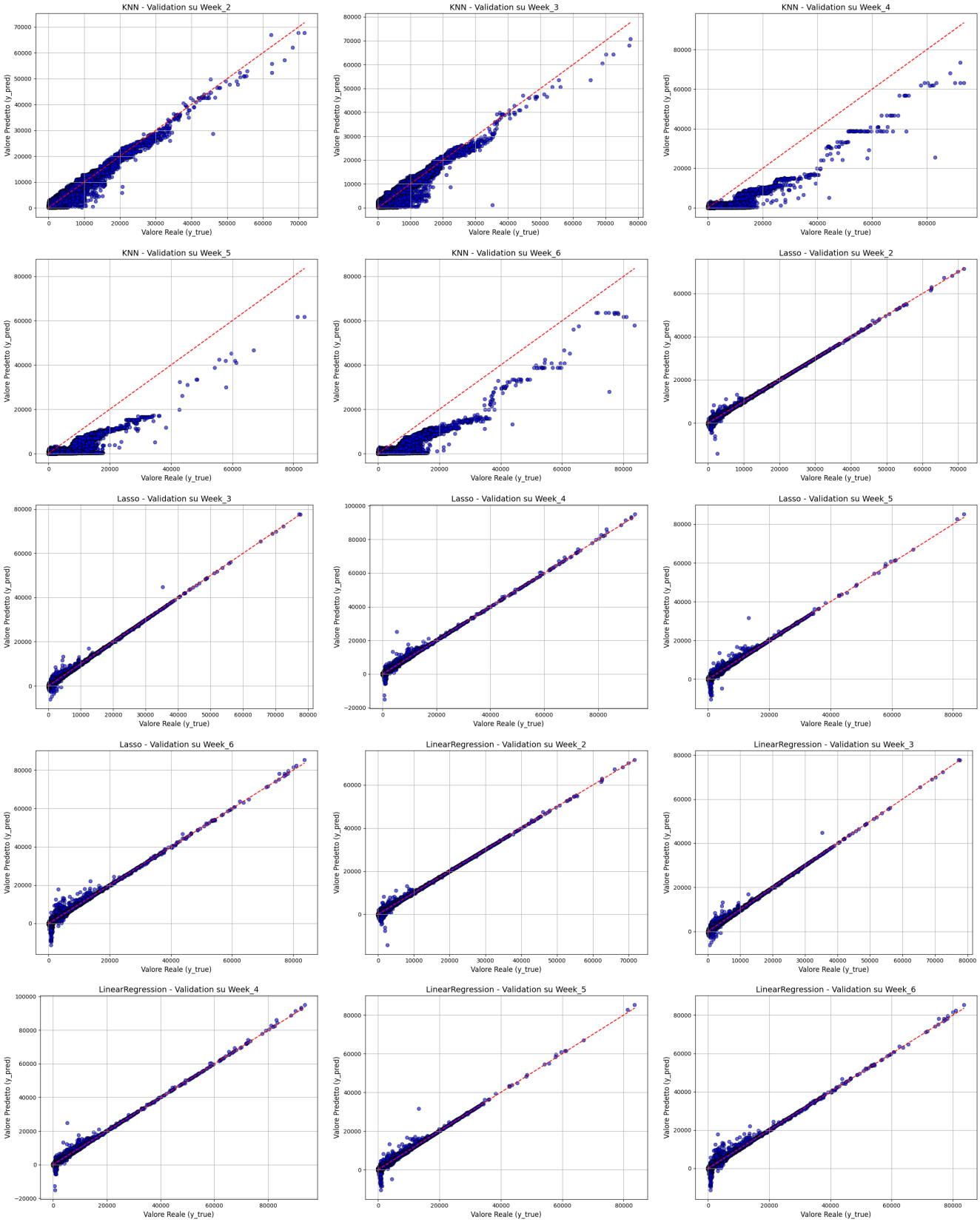
In the code we have a Flag called Model_Complexity that accepts an integer value from 1 to 3. As a result of its configuration it is possible to modify the hyper parameters for machine learning algorithms, in particular it allows us to modify KNN_neighbors for KNN and alpha for Lasso.

Flag	KNN_neighbors	alpha
1	50	10
2	5	1
3	1	0.01

The next table is created with the Flag equal to 1.

Week	Original Data	Filtered Data	Algorithm	Metrics
Week 2	2,669,770	1,268,144	Linear Regression	MSE = 6906.662, $R^2 = 0.998$
			KNN Regressor	MSE = 250238.382, $R^2 = 0.924$
			Lasso	MSE = 7428.923, $R^2 = 0.998$
Week 3	1,301,176	1,253,008	Linear Regression	MSE = 8093.497, $R^2 = 0.998$
			KNN Regressor	MSE = 380491.570, $R^2 = 0.895$
			Lasso	MSE = 9286.498, $R^2 = 0.997$
Week 4	1,034,994	992,168	Linear Regression	MSE = 25536.057, $R^2 = 0.991$
			KNN Regressor	MSE = 2286644.250, $R^2 = 0.171$
			Lasso	MSE = 13018.374, $R^2 = 0.995$
Week 5	5,505,454	1,177,328	Linear Regression	MSE = 37810.551, $R^2 = 0.991$
			KNN Regressor	MSE = 3577468.102, $R^2 = 0.192$
			Lasso	MSE = 26905.051, $R^2 = 0.994$
Week 6	1,387,605	1,384,750	Linear Regression	MSE = 18693.429, $R^2 = 0.997$
			KNN Regressor	MSE = 4476220.831, $R^2 = 0.247$
			Lasso	MSE = 10438.314, $R^2 = 0.998$

Table 2. Validation metrics for different algorithms across multiple weeks.


KNN - Validation su Week_2
KNN - Validation su Week_3
KNN - Validation su Week_4
KNN - Validation su Week_5
KNN - Validation su Week_6
Lasso - Validation su Week_2
Lasso - Validation su Week_3
Lasso - Validation su Week_4
Lasso - Validation su Week_5
Lasso - Validation su Week_6
LinearRegression - Validation su Week_2
LinearRegression - Validation su Week_3
LinearRegression - Validation su Week_4
LinearRegression - Validation su Week_5
LinearRegression - Validation su Week_6
Valore Predetto (y_pred)
Valore Reale (y_true)

2.1 Consideration about the possible estimation of the Throughput

Based on the observed results, it appears that throughput estimation can indeed be achieved using similar linear modeling techniques, provided that the additional data captures the relevant aspects of network performance. The excellent performance of both Linear Regression and Lasso Regression in this context supports the idea that when given high-quality, appropriately filtered data, the underlying linear relationships can be effectively exploited to predict throughput. On the other hand, methods that are not well-suited to capture these linear relationships (such as the current implementation of KNN) may need further parameter tuning or may be less appropriate for this task.

3 SECTION 3 - UNSUPERVISED LEARNING – CLUSTERING

We grouped the caches into clusters representing the edge nodes and we implemented a clustering algorithm to group caches into edge nodes for identifying different edge-node configurations. In order to represent cache by means of features we aggregated them for each unique server IP. To identify the edge-nodes we implemented a clustering algorithm that used 3 numeric features (*min_rtt*, *min_ttl* and *uniq_byte*) instead of the textual code (edge-node).

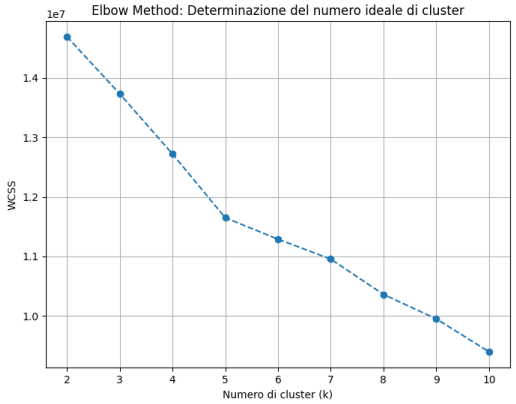
We used K-means because is an unsupervised clustering algorithm particularly valued for its simplicity and efficiency. It allows large amounts of data to be grouped into homogeneous clusters, making it easier to detect hidden patterns and reduce the complexity of the dataset. In addition, the centroids calculated for each cluster represent an “average profile” of the group, making interpretation of the results immediate and intuitive.

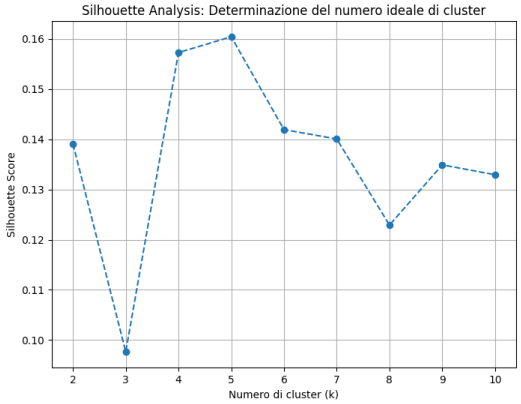
In addition to this KMEANS_AUTO is a flag that allows us to turn on or off the setting that allows the program to use one of our inputs as the number of clusters, or for it to calculate itself through the best algorithms.

To evaluate the best number of clusters we can use the Silhouette and Elbow methods. They are two widely used techniques in cluster analysis for determining the optimal number of clusters. The Elbow method involves plotting the within-cluster sum of squares (or error) against the number of clusters and identifying the "elbow" point, where the reduction in error begins to diminish significantly, as the optimal cluster count. In contrast, the Silhouette method calculates a score for each data point that measures how well it fits within its assigned cluster compared to other clusters; this score ranges from -1 to 1, with higher values indicating better-defined clusters. Together, these methods provide complementary insights that help in validating the cluster structure and selecting the most appropriate number of clusters for the data.

At the end of the calculation we proceed to choose the maximum of the two algorithms, in fact elbow gave 4 while Silhouette 5.

Suggested number of cluster (Elbow): 4
Suggested number of cluster (Silhouette): 5
Final number of clusters selected: 5





Regarding the choice of hyper parameters we chose the standard initialized values since looking at the graphs we could not see a substantial difference, or at least we could not find better ones. It should be noted that because Silhouette requires a lot of computing power and a lot of processing time, to speed up run times, we reduced the samples to 10 000 and 100 000; not noticing much difference, we subsequently left 10 000.
To evaluate cluster performance, we used three metrics:

- Silhouette Score: This metric measures how close each point is to the points in its own cluster relative to those in other clusters. A higher value indicates better internal cohesion and separation between clusters, which is why high values are preferable.

- Davies-Bouldin Index: assesses the similarity between clusters by considering the ratio of the sum of dispersions within clusters to the distance between cluster centroids. A lower index suggests that clusters are well separated and less similar to each other, so low values are desirable.

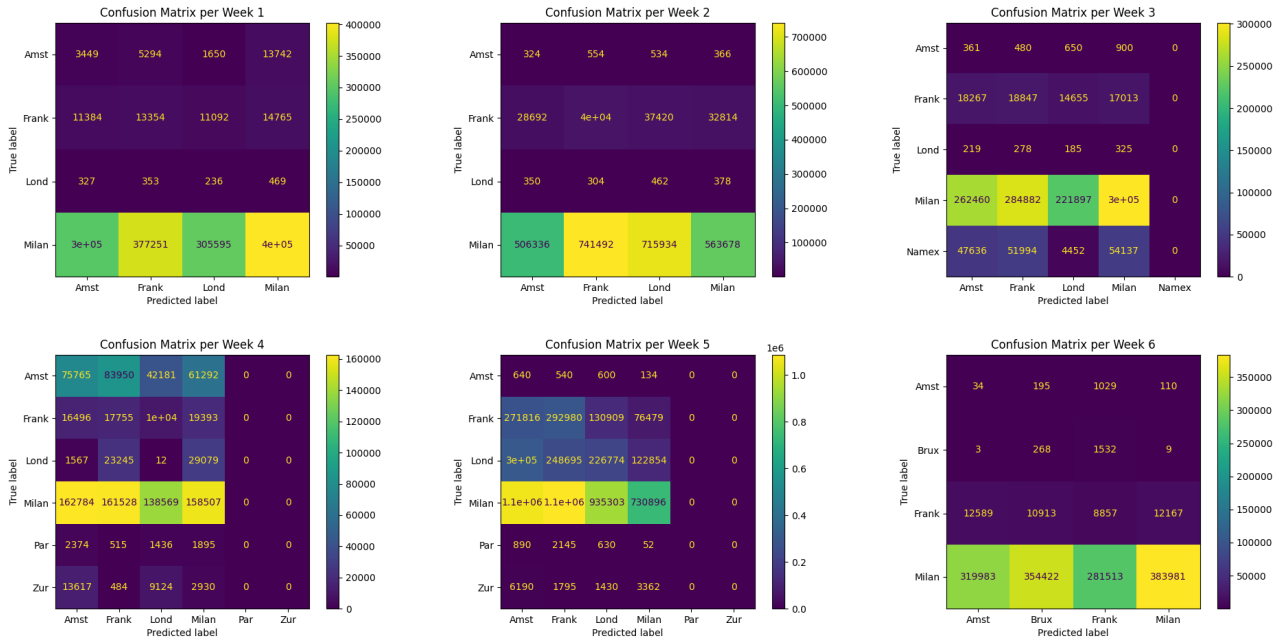
- Calinski-Harabasz Index: also known as Variance Ratio Criterion, this metric compares the dispersion of data within clusters with the dispersion between clusters. A high value indicates that clusters are dense and well separated; therefore, higher values represent better clustering structure.

These three metrics provide a complementary view of clustering quality: the Silhouette Score and Calinski-Harabasz Index emphasize compactness and separation of clusters, while the Davies-Bouldin Index aims to minimize similarity between clusters, helping us to identify the most effective configuration for data segmentation. We obtained this result :

=== Clustering performance (KMeans) ===
Silhouette Score: 0.1585 (more is better)
Davies-Bouldin Index: 1.8555 (less is better)
Calinski-Harabasz Index: 961.9592 (more is better)

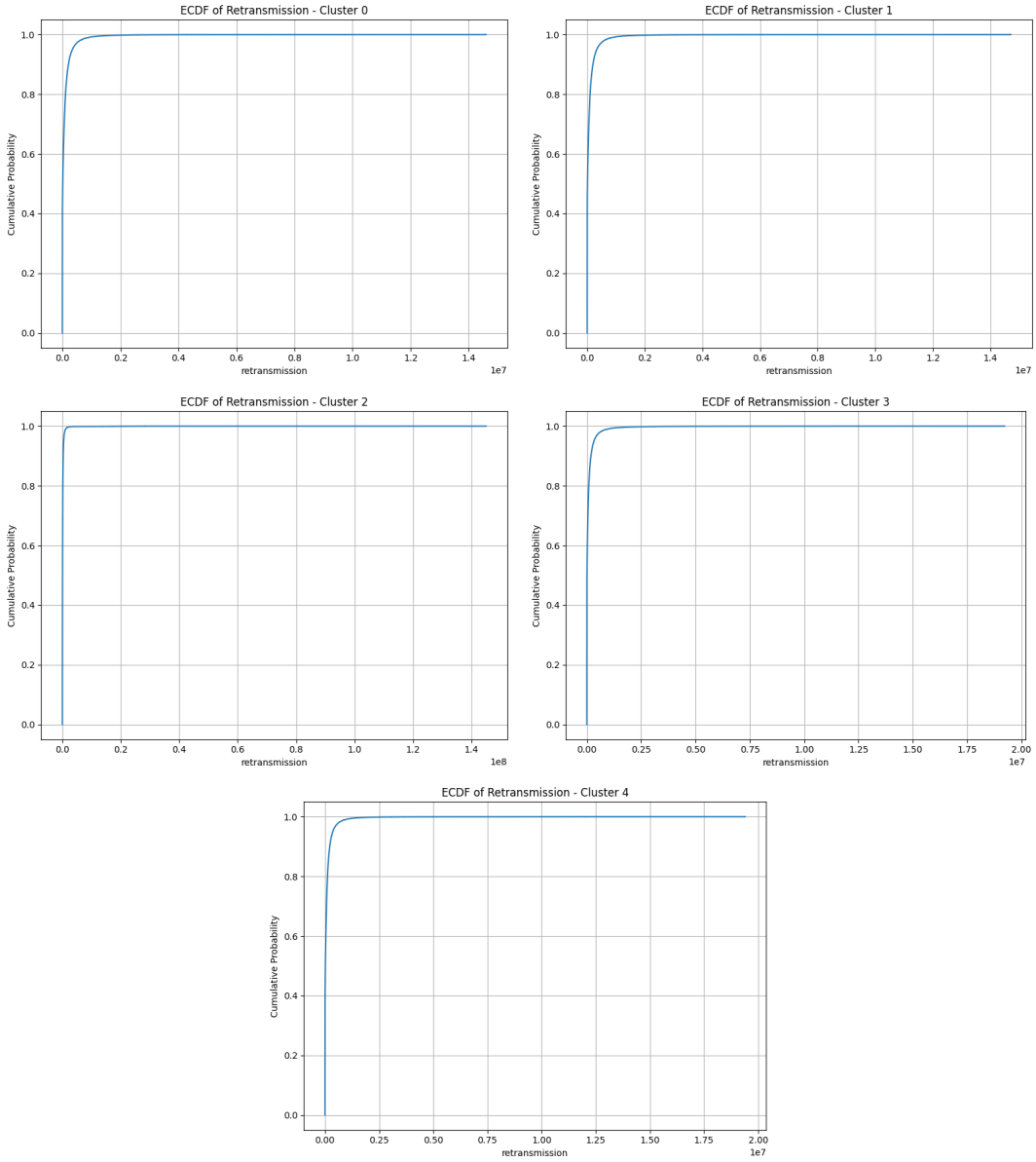
3.1 Confusion matrix

A confusion matrix is a tabular summary used to evaluate the performance of a classification algorithm by comparing its predicted labels against the true labels. We used the confusion matrix because it provides detailed insight into the types of errors the model makes, such as false positives and false negatives, along with true positives and true negatives. This information is crucial as it helps to identify where the classifier might be under-performing and allows for targeted improvements. By analyzing the confusion matrix, one can assess the overall accuracy of the classifier as well as other important performance metrics like precision, recall, and F1-score, which are key to understanding the model's effectiveness in real-world applications.



Usually, empty columns (all zeros) in a confusion matrix occur when the model never outputs that particular label for the given test set. In our example:
Training only on Week_1 – If "Par" and "Zur" labels (edge-nodes) never appeared (or were under-represented) in Week-1's training data, the model may not have learned to predict them at all.
Validation on Week_4 – The model processes new data but apparently never assigns a flow to "Par" or "Zur." So, those columns remain zero in the matrix (no predicted samples for those classes).
In short, because the model was trained on Week_1 (which might not contain or contain very few "Par"/"Zur" instances), it doesn't predict those labels at validation time on Week_4, resulting in zero counts in those columns. As for Week_6, we can see that Bruxelles is replacing London. However, Bruxelles is not present in the training set, despite that the confusion matrix still reports values. This is because the cluster algorithm, in this case Kmeans, failing to assign some samples to either Milan, Frank and Amst consequently assigns them to the remaining label which is Brux.

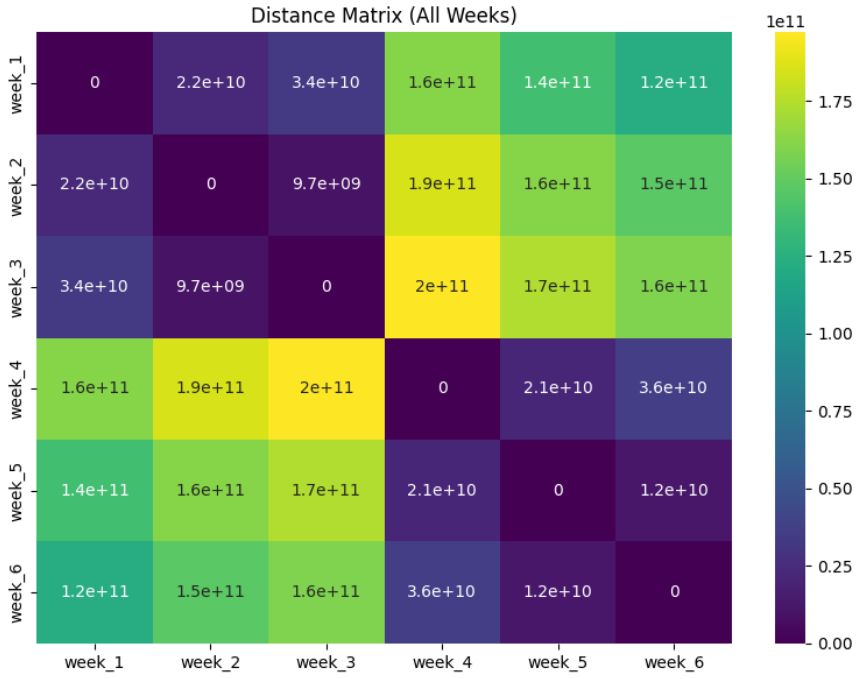
3.2 Coarse analysis of the clusters



By examining the ECDF plots of the retransmission variable across the five clusters (Clusters 0 to 4), it becomes evident that most flows in all clusters exhibit very low retransmission values, as indicated by the steep rise of each ECDF curve near zero. Nonetheless, key differences arise in the tail portions of the distributions. Cluster 2 contains the highest retransmission levels, reaching approximately 1.5×10^8, thus indicating extremely high retransmission in a subset of flows. Clusters 3 and 4 demonstrate elevated retransmission values, with maximums around 2×10^7, placing them midway between the less critical clusters (0 and 1) and the highly critical Cluster 2. Clusters 0 and 1 remain below 1.4×10^7 and therefore include flows with noticeably fewer retransmissions overall. As a consequence, interventions aimed at mitigating excessive retransmissions would benefit from focusing first on Cluster 2, where outlier values are most pronounced, followed by Clusters 3 and 4, while Clusters 0 and 1 appear comparatively stable with lower retransmission counts.

4 SECTION 4 – CLUSTERING EVOLUTION

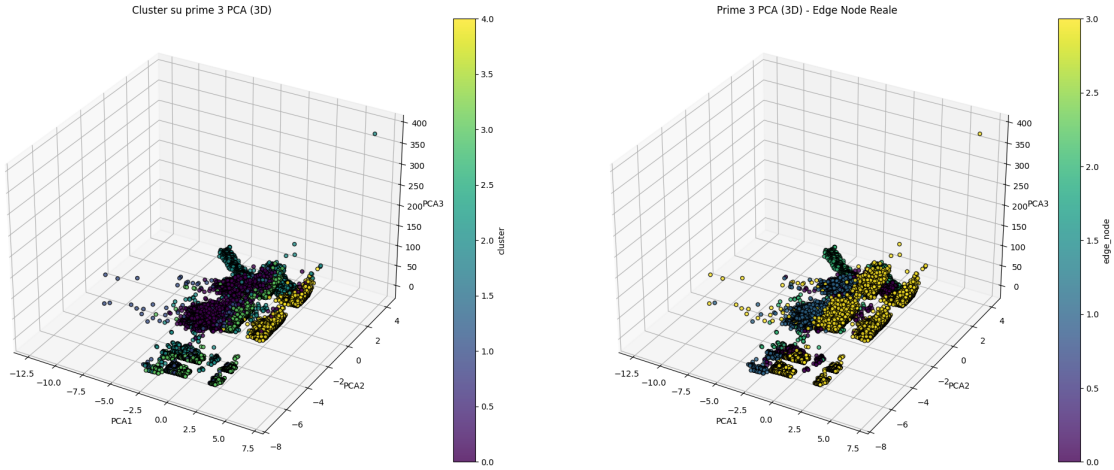
To compute the distance between two clustering results, say, C1 from week 1 and C2 from week 2, we follow a systematic and bidirectional approach. First, for each cluster in C1, we identify the most similar cluster in C2 by calculating a suitable distance metric (for example, Euclidean or Cosine distance). This process produces a distance value for each cluster in C1. Then, we repeated this procedure in the reverse direction: for every cluster in C2, we found its most similar cluster in C1 and computed the corresponding distance. By summing all these distance values, we obtained an overall estimate of the dissimilarity between the two clustering results.
In addition, we created a correlation table to better visualize and analyze the relationships between the clusters. This table was constructed by aligning clusters from different weeks along both axes, with clusters from each week compared against one another. Each cell in the table represents the distance metric calculated between the corresponding pair of clusters. This structured approach helps in identifying which clusters across different weeks are most similar and provides a clear summary of how clustering outcomes evolve over time. Overall, integrating bidirectional distance computations with a correlation table provides a thorough and detailed evaluation of the similarity between the clustering results.



The week which results the most similar one from the baseline (Week_1) is the Week_2 (2.2e10), whereas Week_4 (1.6e11) is the most different one.
From the matrix it can easily be seen that the first three weeks (1-2-3) and the last 3 (4-5-6) are very “close” to each other, while these two groups are far apart. This could be caused by the appearance of new edge-nodes “pushing them apart” in a Euclidean space. This closeness and distance can safely be found in the confusion matrix that enhances their differences. In fact there are 2 new edge-node between week 3 and week 4.

4.1 Cluster images

Cluster images are shown below:



The two plots both show data points in a three-dimensional space defined by the first three principal components (PCA1, PCA2, and PCA3). The main difference is how the points are colored. In the first plot, the colors represent clusters found by the K-Means algorithm: there is a group of purple/blue points toward the lower center, a mostly yellow cluster on the right side, and some green/cyan points spread around the middle or slightly off to the side. In the second plot, the same PCA space is used, but each point is colored by its “edge_node” value (ranging roughly from 0.0 to 3.0). Here, a large cluster of yellow points covers much of the right side of the graph, while the purple/blue and green/cyan points are more scattered and overlapping in the center.

When comparing the two plots, some areas line up quite well. For example, K-Means detects a major yellow cluster on the right side that mostly matches the yellow region in the “edge_node” plot. However, in the center, K-Means creates two clusters (purple/blue and green/cyan), whereas the “edge_node” distribution is more mixed. This indicates that the K-Means clusters do not perfectly align with the actual “edge_node” categories. In some cases, the algorithm’s clusters include points with different “edge_node” values, and vice versa.

To sum up, K-Means has done a reasonable job of grouping the data into clear segments within this reduced PCA space. Still, because these clusters do not fully match the “edge_node” labels, additional steps, like using more principal components, refining cluster parameters, or adding extra features, could help create a closer match between the computed clusters and the real-world attribute.