University of Pisa
Department of Computer Science
Master Degree in Computer Science - AI curriculum

Bots in Twitter
Data Mining project report

Pistolesi Veronica, Poli Francesca

A.Y. 2022/23

Contents

1 Introduction **1**

1.1 Users.csv . . . 1

1.2 Tweets.csv . . . 1

2 Task 1.1: Data Understanding **1**

2.1 Data cleaning on the original dataset . . . 2

2.1.1 Extreme outliers detection and management . . . 2

2.1.2 Dependencies or correlations between attributes . . . 3

3 Task 1.2: Data Preparation **3**

3.1 Development of new features for users profiling . . . 4

3.2 Outliers and high correlations removal . . . 5

4 Task 2: Clustering analysis **6**

4.1 K-means . . . 6

4.1.1 K-means experiment setup . . . 6

4.2 Density-based: DBSCAN . . . 9

4.2.1 Estimation of best hyperparameters . . . 9

4.2.2 Clusters characterization . . . 11

4.3 Agglomerative Hierarchical Clustering . . . 12

4.3.1 Hierarchical experiment setup . . . 12

4.4 Other clustering techniques from pyclustering library . . . 15

4.4.1 CURE . . . 15

4.4.2 K-medoids . . . 16

4.4.3 G-means . . . 16

5 Task 3: Predictive Analysis **17**

5.1 Preprocessing . . . 17

5.1.1 Feature selection . . . 17

5.1.2 Encoding of categorical features . . . 18

5.2 Classification models . . . 19

5.2.1 Hyperparameter tuning . . . 19

5.2.2 Model evaluation . . . 19

5.3 Final results and observations . . . 19

6 Task 4: Time Series analysis **20**

6.1 Clustering analysis 21

6.2 Classification analysis 24

Abstract

The project consists of a data analysis based on data mining tools and performed using Python. The given original dataset consists of 2 *.csv* files, `tweets.csv` and `users.csv`, from which we extracted the data for the project. The work is broken down in four phases:

1. **Task 1** which includes the two sub-tasks of data understanding and data preparation
2. **Task 2** which is the clustering of our processed data
3. **Task 3** comprising a predictive analysis on the data through classification models
4. **Task 4** in which time series were developed and analysed based off the data by days.

1 Introduction

1.1 Users.csv

The dataset in `users.csv` contains information about Twitter users. Without going into details, we just report the type classification of the attributes:

- Nominal/Categorical: *id*, *name*, *lang*;
- Binary: *bot*;
- Numeric: *created_at*, *statuses_count*.

1.2 Tweets.csv

In `tweets.csv` each row contains information about a single tweet. The following are the types of each attribute:

- Nominal/Categorical: *id*, *user_id*, *text*;
- Numeric: *retweet_count*, *reply_count*, *favorite_count*, *num_hashtags*, *num_urls*, *num_mentions*, *created_at*.

2 Task 1.1: Data Understanding

As a first step, we carefully analysed and observed each individual attribute of the dataset. To do this, we uploaded the original given datasets and created a section for each attribute in `users_data_understanding.ipynb` and `tweets_data_understanding.ipynb` respectively, running semantic and consistency checks with the type of data we expected. For each data type and feature, we defined guidelines to validate its consistency and reliability, not only to assess the overall data quality, but also to detect possible errors and outliers to be dealt with later.

2.1 Data cleaning on the original dataset

The data cleaning phase is clearly based on observations from the inspection of individual attributes. The aim is filtering and improving the data through removal of redundant, duplicate or incorrect data and, when feasible, replacement of extreme outliers and null values with the most suitable data obtained from statistics.

2.1.1 Extreme outliers detection and management

Outliers are observations whose characteristics are significantly different from the rest of the data. Although outliers can be legitimate data values, it can be useful to exclude them from the analysis or manage them in some way. In `tweets.csv`, the presence of extremely large outliers led to completely misleading values for statistic measures: that's why we wanted to cut them during this phase.

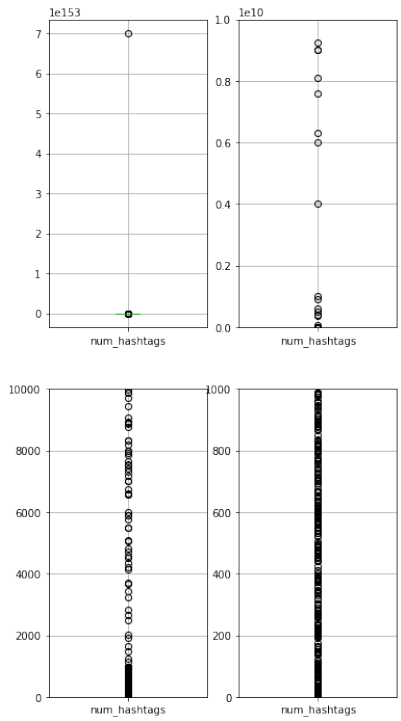


Figure 1: Iterative cutting of Box plots for the *num_hashtags* attribute.

The percentage of outliers detected through the notion of **Interquartile Range (IQR)** fluctuates between approximately 10% and 21% for the majority of attributes, a significant amount especially considering the size of the dataset. On the other hand, the percentage of the extreme ones stands between 0.0001% and 0.005% per cent: to discover them, we plotted iteratively the box plot of the values, in order to cut the sparser data with a graphic approach until the tail at the end of the box became more compact. In this way, we were also able to find an approximate visual threshold for when an outlier becomes an extreme one, making it easier to substitute those destabilising values, using the threshold itself. Let's take the ***num_hashtags*** attribute in `tweets_data_understanding.ipynb` as an example: in Figure 1 the transition from the boxplot with the original distribution to the final one shows a strong change in the compactness of the processed data, which is visible in the closeness of the plots, even though we only cut a very small percentage of the outliers, in this case 0.001%.

In Table 1 are shown the original statistics on the untouched data, while in Table 2 we reported the new statistics after the cut of the extreme outliers only.

mean	$6.75418e+46$
std	$2.17438e+36$
min	0.00
25%	0.00
50%	0.00
75%	0.00
max	$7.0e+46$

Table 1: Original statistic measures for *num_hashtags* attribute in `tweets.csv`.

mean	0.20
std	4.32
min	0.00
25%	0.00
50%	0.00
75%	0.00
max	1000.00

Table 2: Statistic measures for *num_hashtags* attribute, after extreme outliers substitution.

2.1.2 Dependencies or correlations between attributes

Correlation measures the relationship between objects, in order to captures similar behaviours. In particular, Pearson's coefficient is a measure for a linear correlation between two numerical attributes. The heatmap in Figure 2 shows all possible pairwise Pearson's coefficients among the numerical attributes of `tweets.csv`, where the dark cells indicate the absence of linear correlations. The same matrix could not be calculated for the dataset `users.csv`, since it has only one numerical attribute of type `float64` (*statuses_count*).

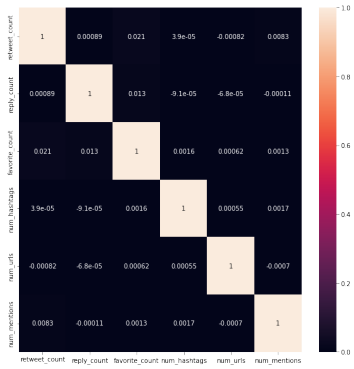


Figure 2: Correlation heatmap of numerical attributes in `tweets.csv`.

3 Task 1.2: Data Preparation

During this phase, the aim was to create new informative features by combining the data extracted and cleaned from the two original files, to enrich the dataset and our data analysis. For this purpose, the data preparation took place on two levels: in the first one, carried out on `data_preparation.ipynb`, the two original datasets were merged and new indicators were created, then inserted together with the other pre-existing attributes into a new dataset, which contains all the information retrieved. This new dataset has been named `users_profile.csv` to emphasise its purpose as a database for profiling Twitter users. In the second level, carried out on `data_preparation_outliers_removal.ipynb`, `users_profile.csv` is cleaned from all the outliers of the basic indicators described below, forming a homogeneous dataset.

3.1 Development of new features for users profiling

We hereby report all the features computed during this phase and present in the final version of the `users_profile.csv`:

- **Basic Indicators**: *retweet_count*, *reply_count*, *favorite_count*, *num_hashtags*, *num_urls* and *num_mentions* are simply the sum of the respective attributes in `tweets.csv` per user.
- **Maximum and standard deviation** of numerical informative attributes *retweet_count* and *favorite_count* are added as numerical features in the dataset.
- **Activity** is a numerical indicator corresponding to the measure of total productivity per user, which is the total sum of all the tweets, retweets and replies of every user. It is calculated by counting the total tweets attributable to that user from `tweets.csv`.
- **Popularity** is a numerical indicator of the ratio between the total number of favorited tweets, replies and retweets per user with its activity (as in the indicator *activity*).
- **Time-related indicators:**
  - **Nightly** is a binary indicator to classify users according to their predisposition, whether one is more active at night or during the day. It is calculated by considering the hourly average of the user's tweets. If the average is between 8 p.m. and 8 a.m., the user is classified as nightly (1), otherwise as diurnal (0).
  - **Most frequent year** is a categorical indicator of the most prolific year of activity per user.
  - **Most frequent hour** is a categorical indicator of the most prolific hour of activity per user.
- **Tweets' richness** is a numerical indicator of the presence of URLs, hashtags and mentions in the user's tweets, retweets and replies, calculated by the ratio between the sum of those elements and the user's activity.
- **Entropy per user** are 8 numerical indicators created computing the entropy of each basic attribute and the length of the tweets for every user in the dataset.
- **Average length of tweets per user** is a numerical indicator based on the mean of the tweets' lengths of every user in the dataset.
- **Days on Twitter** is a numerical indicator of the number of days every account has been on Twitter. For simplicity, we consider all accounts retrieved in the dataset as still active.

- **Activity richness** is a numerical indicator computed as the ratio between the indicator *activity* (number of tweets, retweets and replies) and the time spent on Twitter. The higher the value, the richer the user activity.

The analysis of the new datasets was enriched with some statistics concerning the distribution of data within the old and new indicators, in order to ascertain their quality and functionality in view of the tasks for which they will be used.

3.2 Outliers and high correlations removal

In `data_preparation_outliers_removal.ipynb` we imported the `data_preparation.csv` file in order to substitute all the outliers with a chosen value. We focused on the basic indicators (*statuses_count*, *retweet_count*, *reply_count*, *favorite_count*, *num_hashtags*, *num_urls* and *num_mentions*) because, despite the initial cutting of the most extreme outliers, they still reached the highest cardinality by being sums. Using the Interquartile Range we checked if their values were higher than the third quartile (75%) or lower than the first quartile (25%), that we can consider as thresholds: if that's the case, the value is substituted with the closer quartile, in order to give upper and lower limit to our data. After that operation, a new dataset is created, namely `users_profile_noout.csv`, that will be used for the subsequent classification task.

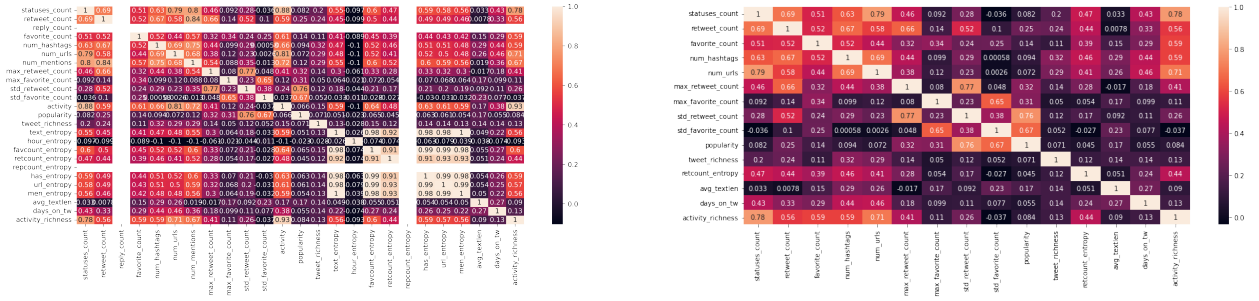
As a last step, the correlations were checked, in particular focusing on those correlations higher than 0.8% that are considered conventionally too high to be informative and can therefore be removed. The heatmap visible in Figure 3a shows clearly the impressive correlations between some features, that will be deleted to create a new cleaned dataset for clustering, `users_profile_clustering_noout.csv`. Another interesting note is the white lines shown in the heatmap, representing particular features that, apart from the outliers, mainly had 0s as values, and will be eliminated because of that.

An important remark, very evident in Figure 3b, is that it appears that the *reply_count* values are all 0s once outliers have been replaced. This happens because there is a very high percentage of outliers, while the only base value is precisely 0.

Here we transcript the features deleted from the dataset for clustering:

- **Almost-all-zero values:** *reply_count*, *repcount_entropy* (entropy of *reply_count*), *hour_entropy* (entropy of *most_frequent_hour*).

- **Many correlations higher than 0.8%:** *activity*, *num_mentions*, *text_entropy* (entropy of the length of the tweets' texts), *favcount_entropy* (entropy of *favorite_count*), *has_entropy* (entropy of *num_hashtags*), *url_entropy* (entropy of *num_urls*), *men_entropy* (entropy of *num_mentions*).



(a) (b)

Figure 3: Correlations in *users_profile.csv* dataset with outliers.

4 Task 2: Clustering analysis

Cluster analysis groups data objects based only on information found in the data which describes the objects and their relationships. In this work we experimented some clustering techniques: **K-means**, **DBSCAN**, **agglomerative hierarchical clustering**, **CURE**, **K-medoids** and **G-means**. At a first step, each method was evaluated with different numerical scores (SEE, the Silhouette coefficient, Davies-Bouldin and Calinski Harabasz). Then, an evaluation based on the characterization of the clusters was carried out: for the categoricals we plotted their distributions in clusters, while for the numericals we looked for the mean, the standard deviation and the minimum and maximum of the values in the clusters. Since these algorithms are applied to objects in a continuous n-dimensional space, we decided to perform them only on the indicators in `users_profile_clustering.csv` and `users_profile_clustering_noout.csv`, as the others could potentially have negatively affected clustering because of their high correlations, or being meaningless for the task due to the huge presence of zeros. In addition, the values of these datasets were subjected to z-score normalization, which standardizes features by removing the mean and scaling to unit variance, to avoid the bias given by the range of different attributes.

4.1 K-means

K-means is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.

4.1.1 K-means experiment setup

Two types of analysis were performed with this algorithm: the first focuses on the selection methods of the optimal K and then explores in detail the algorithm performed with the latter, the second repeats the selection methods and then runs the algorithm on several K-values in order to make a

comparison. Both analyses were run on the dataset with and without outliers. From now on, only the results of the algorithm run on data lacking of outliers will be commented, as they are more trustworthy. This is in line with the fact that the cluster centroids (prototypes) produced when outliers are present, are often less representative than they would otherwise be with an higher SSE.

Methods to select K

As previously stated, the first attempt was to find a subset of potentially optimal K-values for clustering. The method used was the observation of *Elbow* and *Silhouette* score plots and of dendrogram from hierarchical clustering (Ward method). Then the selected were further evaluated by looking at the scores of Sum Squared Error, Silhouette, Davies Bouldin and Calinski Harabasz (see Tab 3). Based on these metrics, the optimal K turned out to be $K = 3$ because it scored top in two out of four measures, while $K = 8$ and $K = 2$ performed better in the first and the last evaluation metrics, respectively.

K	Sum Squared Error	Silhouette	Davies Bouldin	Calinski Harabasz
2	121197.8994	0.4757	1.2414	**4881.7930**
3	103989.8542	**0.4790**	**1.0451**	3796.4753
6	75286.8792	0.4197	1.3222	2974.0305
8	**63338.2631**	0.4087	1.3271	2834.5321

Table 3: Evaluation Metrics to select the best K-value (top scores in bold).

Clusters characterization

To fully understand and analyse the output of the K-means algorithm, we conducted an analysis to investigate the population of the clusters. For the said analysis were used all the K-values between the ones that had been chosen before ($K = \{2, 3, 4, 5, 6, 7, 8\}$).

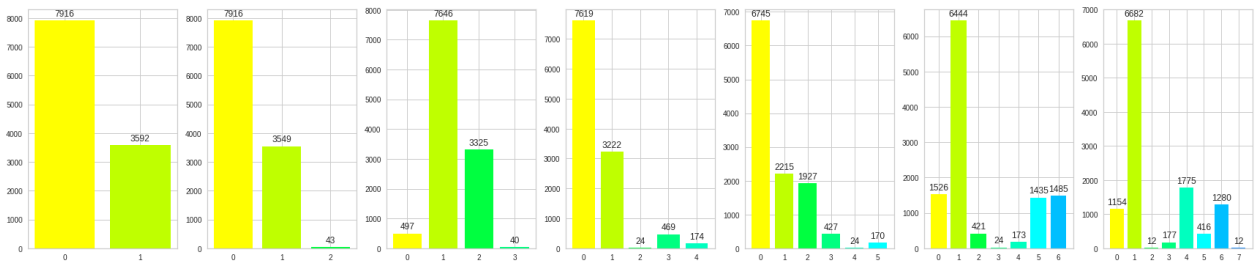
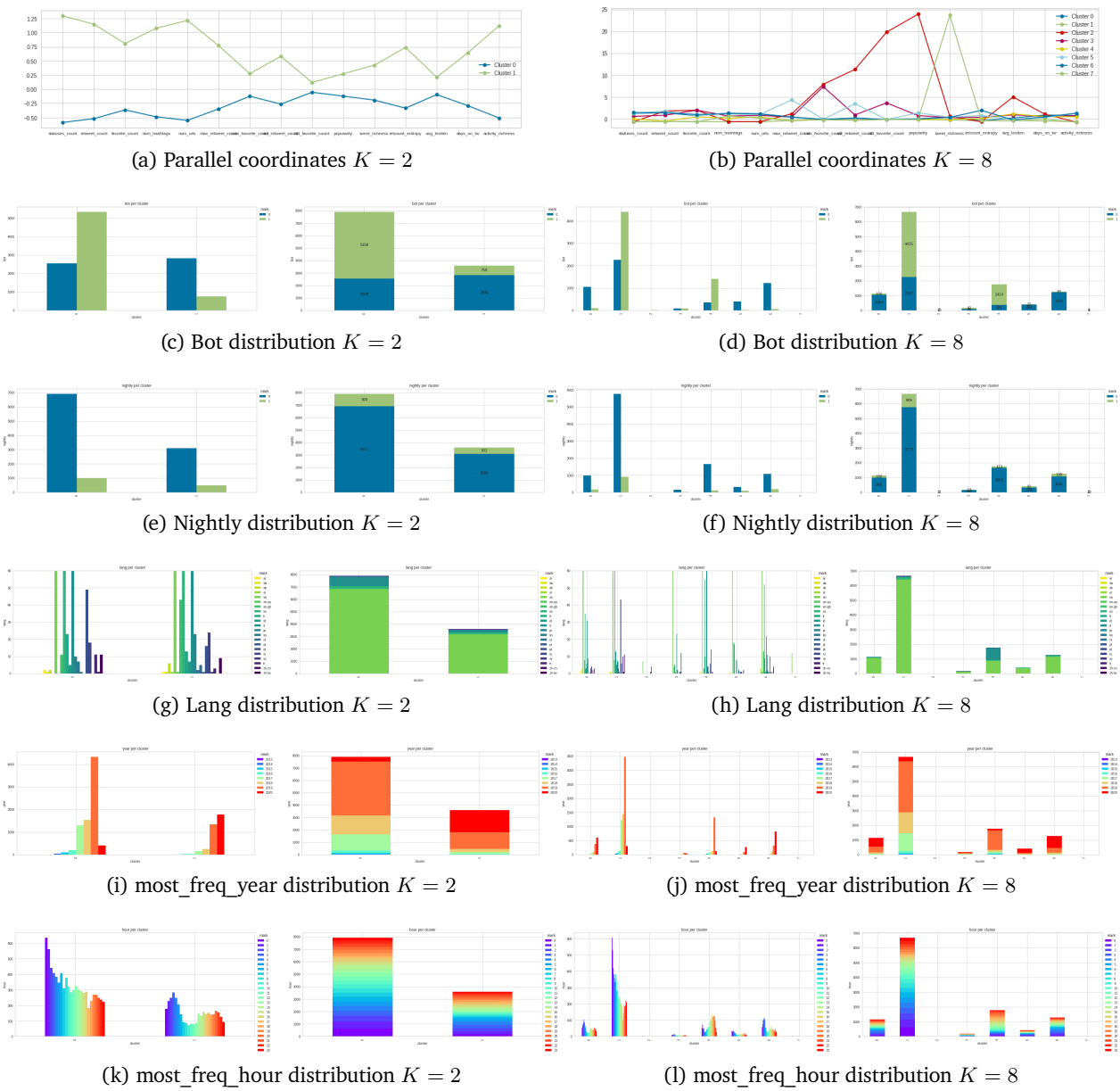


Figure 4: Clusters dimension for different K-values (`users_profile_clustering_noout.csv`)

Fig. 4 shows the clusters dimension for each K: we can observe that a cluster always seems to include around 7000 users, independently of the K parameter. According to this phenomenon, the least populous cluster in the $K = 2$ scenario is the one that is involved with subsequent clustering. For brevity's sake, only the first ($K = 2$) and the last ($K = 8$) k-values are shown in detail for the

clusters characterization comparison.



(a) Parallel coordinates $K = 2$

(b) Parallel coordinates $K = 8$

(c) Bot distribution $K = 2$

(d) Bot distribution $K = 8$

(e) Nightly distribution $K = 2$

(f) Nightly distribution $K = 8$

(g) Lang distribution $K = 2$

(h) Lang distribution $K = 8$

(i) most_freq_year distribution $K = 2$

(j) most_freq_year distribution $K = 8$

(k) most_freq_hour distribution $K = 2$

(l) most_freq_hour distribution $K = 8$

Figure 5: Attributes contribution to the clusters.

Case $K = 2$. Parallel coordinates (Fig. 5a) seem to reveal that the highest cardinality cluster (0) is populated by the users with the lowest values in all indicators (the less active and popular class of users) while cluster 1 is populated by the users with the highest ones. A more thorough statistical investigation with respect to numerical features supports this finding (Fig. 6).

Bot Distribution (Fig. 5c) shows that cluster 0 is predominantly made up of bots (68%) as opposed to cluster 1 which is more populated by humans (79%). A first interesting aspect is that the number of human users is almost equally distributed between the two clusters: 47% in group 0, 53% in group 1 with respect to their total. Nightly and Lang distributions (Fig. 5e, Fig. 5g) reveal similar behaviours

	statuses_count	retweet_count	favorite_count	num_hashtags	num_urls	max_retweet_count	max_favorite_count	std_retweet_count	std_favorite_count	popularity	tweet_richness	retcount_entropy	avg_textlen	days_on_tw	activity_richness
label_K=2															
0	119.675752	5624.873689	112.308679	24.143886	9.997094	2624.173699	63.906139	274.619316	9.384655	74.363481	0.437556	0.492798	68.709223	1659.205912	0.153487
1	4802.244397	100945.649	565.021123	194.59201	176.334911	112410.5039	557.172606	4874.627936	31.055466	903.266057	1.149177	2.329789	73.804822	2176.320991	1.294792

	statuses_count	retweet_count	favorite_count	num_hashtags	num_urls	max_retweet_count	max_favorite_count	std_retweet_count	std_favorite_count	popularity	tweet_richness	retcount_entropy	avg_textlen	days_on_tw	activity_richness
label_K=2															
0	390.382951	21221.31174	280.279645	46.40753	32.037977	11452.12084	485.922789	1000.302535	56.720086	437.026831	0.633019	0.958825	14.235051	429.981958	0.259638
1	2087.580448	56415.9716	401.480668	114.121482	83.926431	148992.7658	2049.212136	8833.712058	204.955593	3633.390031	1.72915	2.234338	20.450538	621.673665	0.723351

	statuses_count	retweet_count	favorite_count	num_hashtags	num_urls	max_retweet_count	max_favorite_count	std_retweet_count	std_favorite_count	popularity	tweet_richness	retcount_entropy	avg_textlen	days_on_tw	activity_richness
label_K=2															
0	6237.625	138813.625	1031.125	297.5	245	246185	10000	20616.04343	1174.763061	18511.66667	17.033898	5.981757	245	3853	2.360635
1	6237.625	138813.625	1031.125	297.5	245	500000	10000	106034.9826	3722.299247	69042.4	41.833333	6.369161	189	3900	4.148438

	statuses_count	retweet_count	favorite_count	num_hashtags	num_urls	max_retweet_count	max_favorite_count	std_retweet_count	std_favorite_count	popularity	tweet_richness	retcount_entropy	avg_textlen	days_on_tw	activity_richness
label_K=2															
0	0	0	0	0	0	0	0	0	0	0	0	0	3	890	0.000319
1	16	0	0	0	0	0	0	0	0	0	0.002976	0	21.107632	896	0.004533

Figure 6: Clusters characterization ($K = 2$) with respect to numerical features with mean(), std(), max() and min() statistics.

in both clusters (prevalence of daytime users and english tweets), remaining consistent with their individual statistics. In the *most_freq_year* distribution (Fig. 5i), 2019 and 2020 are dominant years in clusters 0 and 1, respectively. With humans making up the majority of cluster 1, this majority is consistent with the widespread use of Twitter at the beginning of the recent pandemic. Additionally, it might imply that the majority of the bots currently in use were made in 2019. The *most_freq_hour* distribution (Fig. 5k) shows that in both of the clusters the most frequent hour of publication is a nightly one, namely midnight for cluster 0 and 3 a.m. for cluster 1 even though both groups' hourly average publication times are during the day.

Case $K = 8$. As previously stated, when $K > 2$, one of the clusters - in this context, cluster 1 - tends to follow the same trends as the previous cluster 0. This leads to the hypothesis that the cluster 1 from $K = 2$ case is the one involved with the subsequent clustering because it is an accurate representation of the active human users on Twitter (indeed, since they are real and highly heterogeneous, they can be clustered in more detail) whereas people in cluster 0 have such a low profile that it seems like they are fake accounts, namely bots.

4.2 Density-based: DBSCAN

Density Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm that uses two parameters: ***MinPts***, the minimum number of points (a threshold) clustered together for a region to be considered dense; and ***eps*** **(ε)**, a distance measure that will be used to locate the points in the neighbourhood of any point. These parameters must be entered manually at the discretion of the programmer, therefore is complex to justify one's choice, if not on the empirical basis of graphs and tests of the algorithm on different values.

4.2.1 Estimation of best hyperparameters

After discretizing and normalizing the data with Standard Scaler and running PCA to reduce the dimensionality at 2, the first challenge to be faced is the estimation of the best ε and the MinPts

parameters. The basic and universally acknowledged approach is the *knee method*, a visual way to look at the behaviour of the distance from a point to its kth nearest neighbour, which we will call the *k-dist*. For points that belong to some cluster, the value of *k-dist* will be small, if k is not larger than the cluster size. On the other hand, the *k-dist* of points not insertible in a cluster will be relatively large. Our choice for the *k* was $k = 2 * dim(dataset) - 1$.

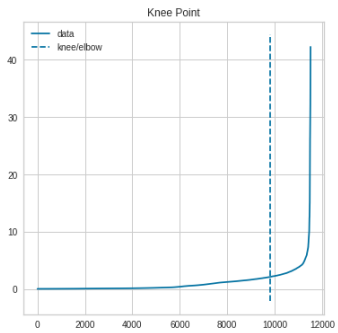


Figure 7: kneed locating the best ε value at 2.11

The aim is to see a visible change at the value of k-dist that corresponds to a suitable value of ε. We look for the point of maximum curvature, the so-called *knee* or *elbow* point. If we select this distance as the ε parameter and take the value of k as the MinPts parameter, then points for which k-dist is less than ε will be labeled as *core points*, while other points will be labeled as *noise* or *border points*. For a more precise estimation of the knee method the *kneed* package is used, resulting in 2 as the best ε value. The result graph is visible in Figure 7.

To continue the evaluations, we implemented a grid search that takes into consideration the KNN algorithm's results, the estimated number of clusters and the Silhouette Score for the DBSCAN clustering for each selected value of ε and *MinPts*. In Figure 8 the mean noise distances and the number of clusters obtained through the grid search are reported in the heatmaps below. On another note, the choice of the *MinPts* value stems from our research about estimation of hyperparameters: it seems that if the data has more than 2 dimensions, choosing *MinPts* $= 2 \times dim$ (where dim = the number of features of the dataset, in thsi case $2 \times 16 = 32$) is optimal, but considering that we reduced dimension at 2 from PCA the good practice is to maintain a default value of k as *MinPts* $= 4$. That's why we implemented the grid search to test ε values in range $[2 - 5]$ and the *MinPts* values in range $[4 - 32]$. The Table 4 shows the results of the grid search with highest Silhouette Score and good number of clusters, the combinations we then tested the DBSCAN on.

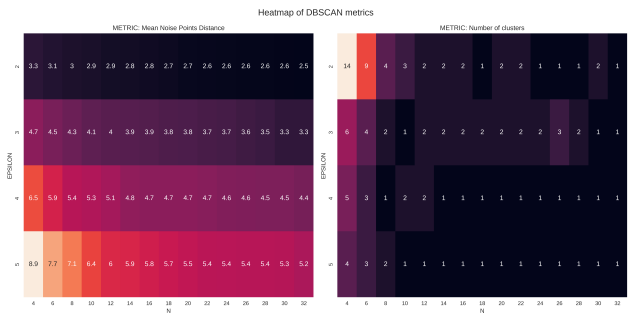


Figure 8: kneed locating the best ε value at 2.11

HYPERPARAMETERS	DISTANCES	NUM OF CLUSTERS	SILHOUETTE
eps = 2 and min_samples = 16	2.79	2	0.576
eps = 3 and min_samples = 8	4.28	2	0.694
eps = 4 and min_samples = 6	5.86	3	0.798
eps = 5 and min_samples = 8	7.09	2	0.839

Table 4: Best results from grid search

Figure 10 is the graphical display of the clusters derived from the application of the DBSCAN with the hyperparameters mentioned above. The highest Silhouette Scores in our quest delineate the most definite clusters, but at the same time, they tend to include a lot of noise points into the clusters, while the DBSCAN algorithm is normally used for outliers detection through them. Because of that, we understand that the clusters with too few noise points are not the most informative and we might consider the ones with lower ε and Silhouette Score (specifically, Figure 10a) as the most accurate for our intentions, which are to analyse and give insight on the clusters' population.

4.2.2 Clusters characterization

The results of those statistics are quite interesting, because they clearly show the criteria to detect the noise points: the clusters with label -1, which is the noise points' cluster, contains all of the users from the dataset with by far the highest maximums and means of data, and as a result obviously the standard deviation of the values is an important one. However, the other clusters of non-noisy data seem to be quite variegated and not strongly characterized by the label *bot* or *non-bot*. In particular, there seems to always be a major central cluster which concentrates the majority of the values, and after some experiments it seems clear that the situation doesn't change, even in choosing lower parameters. An example is in Tab. 5, where we reported the results of the DBSCAN with default hyperparameters, and still there one the central major cluster.

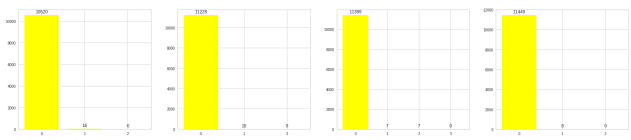
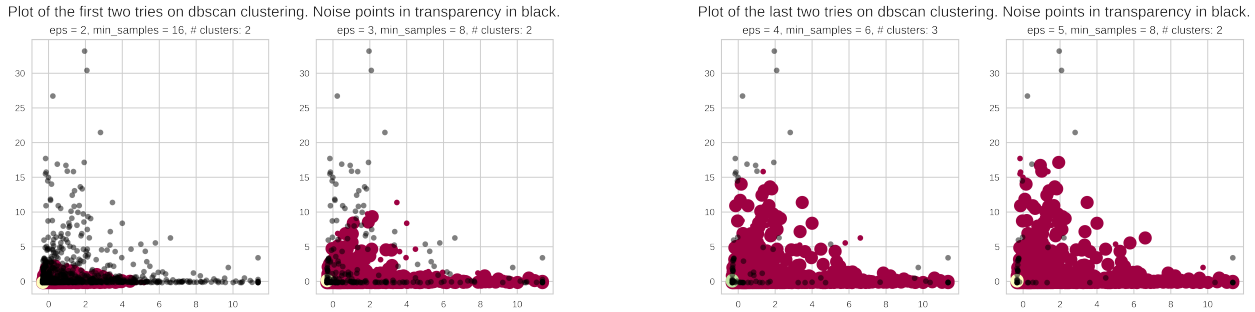


Figure 9: Clusters dimension for different parameters

We might deduce from that peculiarity that however clean, scaled and quite evenly distributed the data can be, the perception of density in a probabilistic space might not be the best method for this specific dataset and work to get precise clusters. However, as we noticed, might be very good to detect outliers and provide a setup for other tasks and data analysis techniques.

CLUSTER LABEL	POPULATION
-1	700
0	10705
1	20
2	7
3	5
4	8
5	6
6	3
7	17
8	22
9	11
10	4

Table 5: Population of DBSCAN with $eps = 2$ and default $min_samples = 4$



(a) (b)

Figure 10: Visuals of the final clusters with higher Silhouette score

4.3 Agglomerative Hierarchical Clustering

As with K-means, Hierarchical clustering approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. In our work we decided to test the agglomerative approach in which methods start with individual points as clusters and gradually combine the two nearest clusters until only one cluster is left.

4.3.1 Hierarchical experiment setup

The Complete, Ward, Single, and Average linkage methods' dendrograms were examined in the first step of this process in order to determine the ideal number of clusters (K) for each of them. The second was to extensively compare all of the methods by testing them all with various K-values. The best Linkage performer's clusters were then characterized. The procedures were carried out on both datasets — those with and those without outliers — as in the K-means analysis.

Methods to select K and Linkage

Two distinct techniques were used to find the best K-values: the default color cut (Fig. 11) and counting the number of vertical lines crossed by the cut made on the half of the largest distance between clusters.

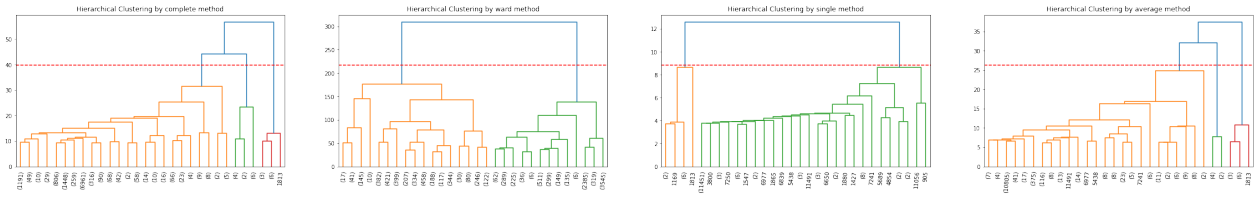


Figure 11: Default dendrograms color cut (dataset without outliers).

The cardinality of the obtained clusters from the algorithm run with the selected K-values for each method is shown in Fig. 12.

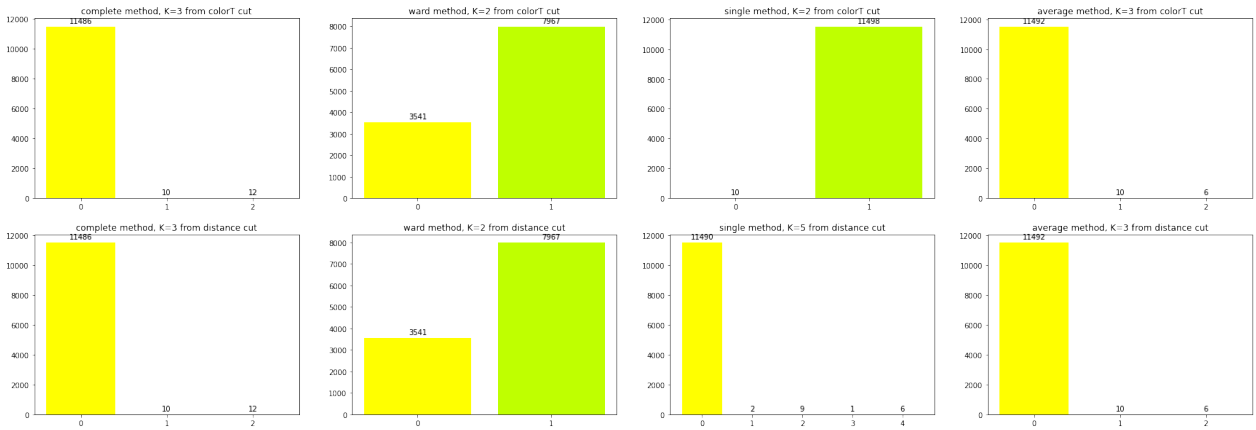


Figure 12: Clusters cardinality, different K-values and Linkages (dataset without outliers).

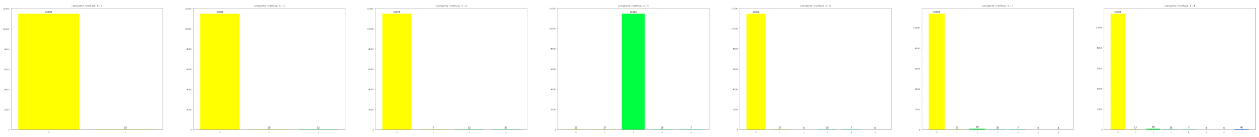
Next, we calculated the CPCC (Cophenetic Correlation Coefficient) to determine the best linkage method for our data. The values are shown in Tab. 6.

Linkage	CPCC
Complete	0.8111
Ward	0.6189
Single	0.9035
Average	**0.9530**

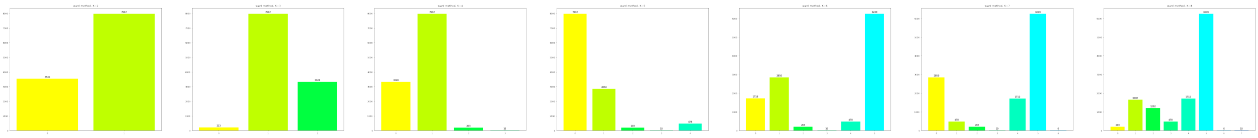
Table 6: CPCC on data without outliers (top score in bold).

Based on the cophenetic correlations of hierarchical clustering methods presented, Average Linkage provided better grouping, as it presented the highest cophenetic correlation (0.95). At this point, we noticed a small contrast: the methods with the highest CPCC appear to perform worse even with their ideal K values (Fig. 12), whereas the approach with the lowest one (Ward) appears to cluster

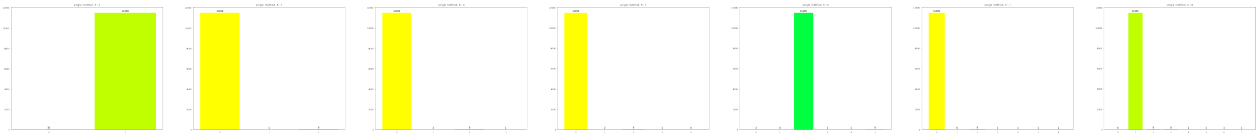
our data more successfully. From some studies[1] we know that the use of cophenetic correlations and related measures of agreement may be rather misleading if these measures are considered as indicators of presence of clusters. For this reason we decide to further explore all the different methods with multiple K-values (Fig. 13).



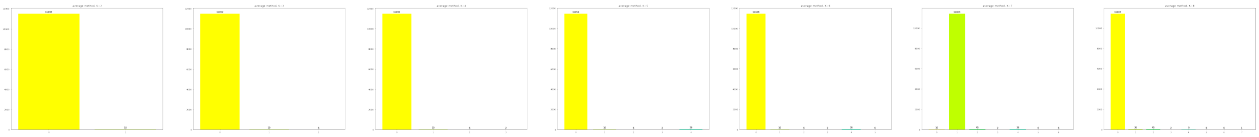
(a) Complete Link



(b) Ward's method



(c) Single Link



(d) Group Average

Figure 13: Clusters cardinality and methods comparison (dataset without outliers).

The clustering produced by Ward's method is different from those produced by the other ones. Centroid methods calculate the proximity between two clusters by calculating the distance between the centroids of clusters. This technique may seems similar to K-means, but it is its hierarchical analogue. Centroid methods also have a characteristic — often considered bad — that is not possessed by the other hierarchical clustering techniques: the possibility of inversions. More precisely, two clusters that are merged can be more similar (less distant) than the pair of clusters that were merged in a previous step while for the other methods, the distance between merged clusters monotonically increases (or is, at worst, non-increasing) as we proceed from singleton clusters to one all-inclusive cluster. So, in this case, the poor performances of Complete, Single and Group Average method depend on their definition of distance between cluster which turned out to be suboptimal for our data indipendently of the lack of outliers.

Coming back to the selection of the best K, this time specifically for the Ward's method, all the K-values were further evaluated by looking at the scores of Silhouette, Davies Bouldin and Calinski

[1] The limited value of cophenetic correlation as a clustering criterion

Harabasz (see Tab 7). Based on these metrics, the optimal K turned out to be $K = 2$ because it scored top in all of the measures.

K	Silhouette	Davies Bouldin	Calinski Harabasz
2	**0.8832**	**0.1911**	**998.9345**
3	0.8646	0.2117	747.0007
4	0.8239	0.2423	531.9668
5	0.7459	0.4427	600.2110
6	0.7359	0.4622	509.4631
7	0.6484	0.5619	514.6789
8	0.6482	0.5271	442.5919

Table 7: Evaluation Metrics to select the best K-value (top scores in bold).

Clusters characterization

Case $K = 2$. We can do the same considerations done for the K-means clusters characterization case $K = 2$ since the clustering result and its chacaterization are very close to that ones (Fig. 5) if not slightly more robust (90% of bot are in the most populated cluster instead of 68%).

Regarding the current analyses done on the dataset with outliers, we got the same trend in the results, but with more lack of precision as we expected.

4.4 Other clustering techniques from pyclustering library

In addition to the three previous clustering techniques, we explore the opportunity to use alternative ones from the pyclustering library.

4.4.1 CURE

The CURE (Clustering Using REpresentatives) algorithm is an agglomerative hierarchical clustering method. We decided to test it because it attempts to eliminate the disadvantages of the centroid approach and all-points approaches by presenting a hybrid of the two. CURE can identify non-spherical shaped clusters and wide variances in size with the help of well scattered representative points and centroid shrinking. It can handle large databases by combining random sampling and partitioning method and it is robust to outliers. This algorithm was run on both datasets (with and without outliers) by using different number of clusters. Fig 14 shows the cardinality of the obtained clusters.

Unfortunately, despite the good foundation, the algorithm did not work well. The primary drawback of current agglomerative hierarchical systems is that merging decisions are based on static modeling

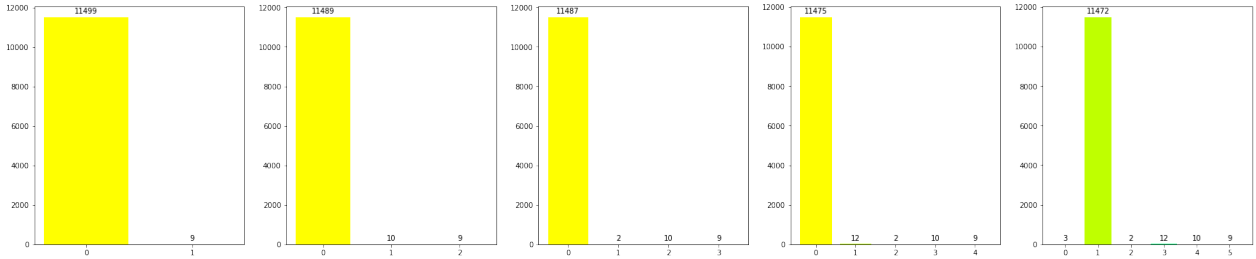


Figure 14: CURE algorithm clusters cardinality (dataset with outliers).

of the clusters to be merged. In other words, these schemes may fail to consider the special traits of each cluster, which may lead to erroneous merging decisions.

4.4.2 K-medoids

The clustering method K-medoids is prototype-based. This approach can be used with a wide range of data since it simply needs a proximity measure for a pair of objects and defines a prototype in terms of a medoid, which is the most representative point for a set of points. A medoid must be an actual data point by definition, in contrast to how rarely a centroid corresponds to one. We chose to test it because, despite being more expensive than K-means, it is less susceptible to outliers and does not have the same restriction to data for which there is a concept of a center (centroid). Fig. 15 shows the clusters curdinality obtained by the algorithm run with different numbers of clusters.

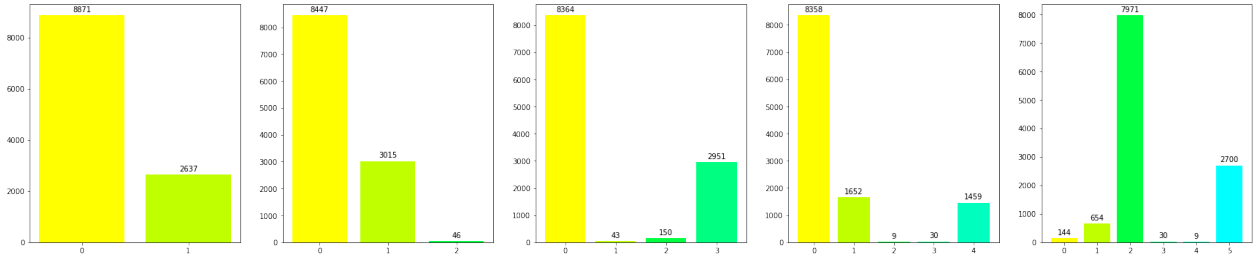


Figure 15: K-medoids algorithm clusters cardinality (dataset with outliers).

The results for $K = 2$ maintain the same trend as the $K = 2$ case of K-means and the Ward's method of agglomerative hierarchical clustering but with an even more defined presence of bots in the most populated cluster: 95% (68% and 90% in K-means and Ward's method, respectively).

4.4.3 G-means

Since the G-means algorithm does not require a priori cluster selection as does K-means, we chose to test it. In fact, this method applies a hierarchical strategy to determine the number of clusters by performing a statistical test to determine whether to divide a K-means center into two. Splitting the clusters continues until the data has a Gaussian distribution. We conducted this procedure on our

data and obtained too many clusters (around 80), which led us to hypothesize that it was not the best approach in this case study. Nevertheless, although knowing that a wrong choice for the K parameter would result in inaccurate findings, we still chose to test it by forcing the number of clusters in order to compare it with other techniques. Fig. 16 shows a weaker clustering performance than K-means, confirming our hypothesis.

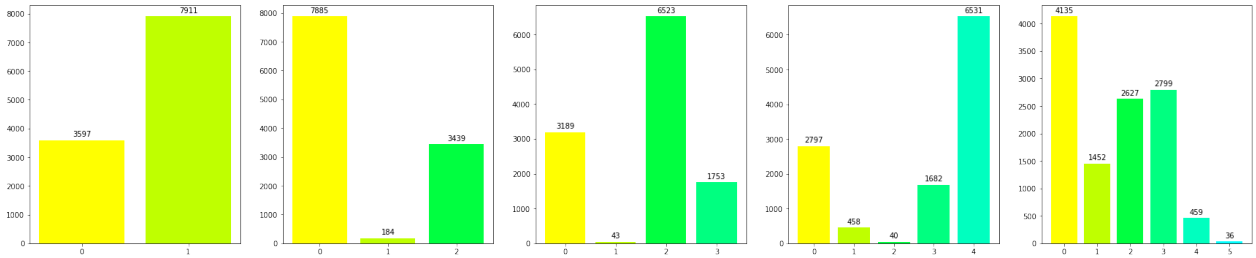


Figure 16: G-means algorithm clusters cardinality (dataset without outliers).

5 Task 3: Predictive Analysis

The aim of this task was to test different classification models with the *bot* feature as target label, in order to train them to predict and recognise if a user is a bot or not. We developed four different notebooks for different approached adopted for the predictive analysis, whose names are structured as follows: `classification_FeatureSelector_Encoder.ipynb`.

5.1 Preprocessing

Since the `datetime` datatype should be considered as a categorical and encoded, to facilitate the classification task we decided to select only the year from the date of creation of the Twitter account (the *user_created_at* column) and create a new indicator from it. After that, we proceed in distinguishing the features' types for the encoding phase, namely categorical, numerical and binary features, creating three different dataframes. For the numericals, we exploit the clustering dataset which was already cleaned and purged of high correlations. We normalize the numerical dataframe with the Standard Scaler and we go on to the first step, which is the feature selection.

5.1.1 Feature selection

This is one of the most delicate and essential tasks for classification: the choice of the right features can make the accuracy of the classifiers rise up highly, so we must select the most informative ones that also have the lowest correlation between them. To do so, for what concerns the **numerical**

dataframe, we tested two different methods: one is the **SelectFromModel** method with Random Forest Classifier from ScikitLearn, used only in the `classification_SFM_SKB_OneHot.ipynb` notebook, and the other is the **Random Forest Regressor** as a selector, used in all the other notebooks.

SelectFromModel: it's a ScikitLearn meta-transformer for selecting features based on importance weights and a base estimator from which the transformer is built. Selecting features by using tree-derived feature importance is a very straightforward, fast and generally accurate way of selecting good features. However, there's a bias: Random Forests and decision trees, in general, give preference to features with high cardinality. Moreover, correlated features will be given equal or similar importance, but overall reduced importance compared to the same tree built without correlated counterparts. In situations like this, it is better to select features recursively. That's why we used a Random Forest Regressor selector in `classification_SFM_SKB_OneHot.ipynb`, `classification_RFR_SKB_OneHot_NOOUT.ipynb` and `classification_RFR_SKB_Binary.ipynb`.

Random Forest Regressor: each tree can calculate the importance of a feature according to its ability to increase the pureness of the leaves. The higher the increment in leaves purity, the higher the importance of the feature. Once we have the importance of each feature, we perform feature selection using Recursive Feature Elimination: the idea is to fit the model, then remove the less relevant feature on the basis of the value of some performance metric, and we chose the permutation of the feature importance. That's because the permutation based importance can be used to overcome drawbacks of the default feature importance computed with mean impurity decrease: this method will randomly shuffle each feature and compute the change in the model's performance. The features which impact the performance the most are the most important one.

For the **categoricals**' dataframe we chose only the SelectKBest method, a ScikitLearn class that calculates a score to help keeping only the most important features in your dataset, returning the best features according to the k highest scores.

5.1.2 Encoding of categorical features

A necessary step is the encoding of categorical data, in order to be used for classification while models only accept numerical values. There we tested again two methods: the largely known One Hot Encoding and the Binary (only in `classification_RFR_SKB_Binary.ipynb`). Binary encoding creates fewer columns than one-hot encoding. It is more memory efficient. It also reduces the chances of dimensionality problems with higher cardinality. For ordinal data, most values that were close to each other when in ordinal form will share many of the same values in the new columns. Binary encoding can be thought of as a hybrid of one-hot and hashing encoders: creates fewer features than one-hot, while preserving some uniqueness of values in the column.

5.2 Classification models

We tested nine models of different types for classification: **Decision Trees**, **AdaBoost**, **AdaBoost with Decision Tree as base**, **Random Forest**, **Gaussian Naive Bayes**, **KNN (K-Nearest Neighbors)**, **C-Support Vector Classification**, **Multi Layer Perceptron** and **Stochastic Gradient Descent**.

To train our models, we split the processed dataset in training set (made up from 80% of the original dataset) and test set (the remaining 20% of the dataset).

5.2.1 Hyperparameter tuning

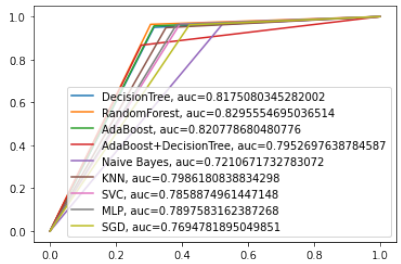
To enhance the classifiers' results, we run a hyperparameter tuning tailored for every classifier, using ScikitLearn functions. For most models we used the RandomizedCV search, while for the ones with less parameters to test we went with GridSearchCV, by which all parameter values are tried out. This choice stems from the good trade-off with computational time and weight. RandomizedSearchCV, in fact, implements a "fit" and a "score" method. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings. Only a fixed number of parameter settings is sampled from the specified distributions, which we mostly set at 100 with the *n_iter* parameter of the RandomizedSearchCV function, while the *n_jobs* is mostly set at -1 to let the function run until reaching a good accuracy result with no limits. To further improve the hyperparameter search, we also added a Stratified K-Folds cross-validator. To select the best parameters for the tuning, in order to avoid long runs on useless values, an exhaustive search was made for the best values attributable to the model depending on the amount of data and features in the dataset.

5.2.2 Model evaluation

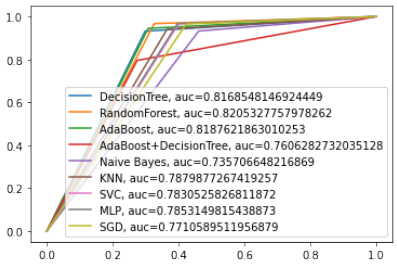
After having fitted the model, it is important to understand how it is performing. To evaluate the goodness of the model, we computed several metrics, such as *F1, accuracy, precision and recall*, and also run a cross-validation on fit and score time. Also, we computed a confusion matrix and scatter plots with true labels and predicted labels for every model's results.

5.3 Final results and observations

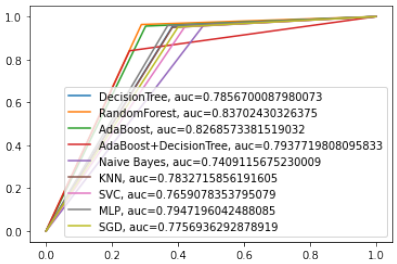
We tested those combinations of numerical feature selectors and encoders: Random Forest Regressor with OneHot encoder, Random Forest Regressor with Binary encoder, SelectFromModel with OneHot encoder, and all of those combinations were run with both the dataset with outliers and the one without. There are variable results between the combinations of methods we used, going from around



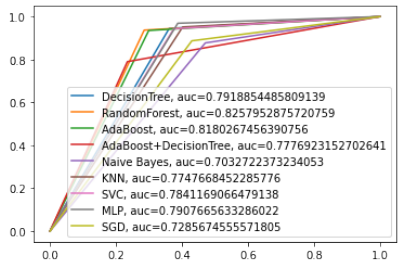
(a) RFR_SKB_Binary



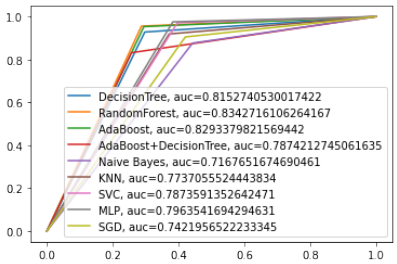
(b) RFR_SKB_OneHot



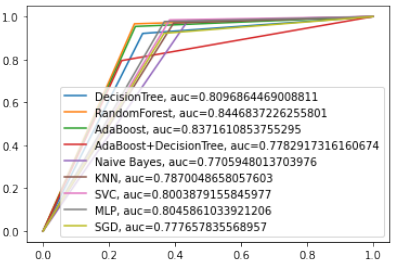
(c) SFM_SKB_OneHot



(d) RFR_SKB_Binary_NOOUT



(e) RFR_SKB_OneHot_NOOUT



(f) SFM_SKB_OneHot_NOOUT

Figure 17: Classification results.

75% to 85% on the fit of the model for the task and the type of preprocessing used. It seems that the best combination overall is the one in which we used the SelectFromModel and OneHot encoder on the dataset without outliers. Other good performances come from the combination of the Random Forest Regressor and the Binary encoder. Those outputs suggest that probably the SelectFromModel method used together with a Binary encoder for categoricals would obtain great results.

The "winning" classifiers seem to be the tree-based ones (*Random Forest* and *Decision Tree*) and the AdaBoost classifier, with an accuracy up to 84% from the Random Forest in 17f. We might infer from that results that the best models for this task are the most flexible ones, specifically the ones based on behaviour analysis and decision trees, rather than probability and statistics. We also need to take into consideration the fact that our data is human-generated, meaning it is not ruled by fixed laws or rules that make it easily predictable.

6 Task 4: Time Series analysis

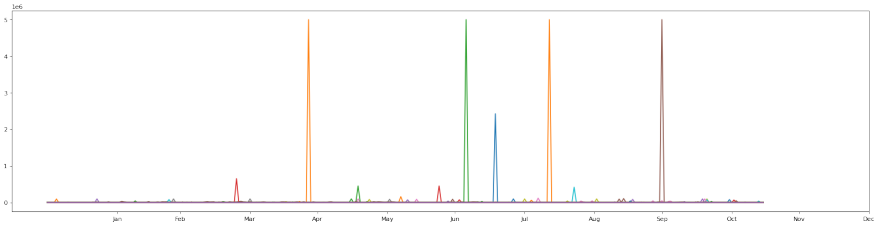
In our case study, the Time Series are the Twitter's timelines of the users. For each day of 2019 in our `tweets.csv` dataset, we computed the SuccessScore:

$$SuccessScore = \frac{AcceptanceScore}{DiffusionScore + 0.1}$$

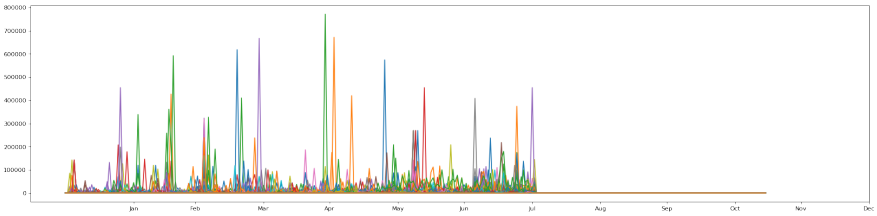
where

- AcceptanceScore = *retweet_count* + *reply_count* + *favorite_count*
- DiffusionScore = *num_hashtags* + *num_mentions* + *num_urls*

First of all we did an exploration of the time series. Fig. 18 shows a comparison between *Bot* and *Human* category.



(a) Bot category time series



(b) Human category time series

Figure 18: Time series comparison.

A first thing to note is that, for the year 2019, `tweets.csv` contains only tweets until November 15th so we decided to set our time series length at 319 (the number of days between $01-01-2019$ and $11-15-2019$) instead of 365 (the total number of days of 2019) since the information would have been the same, and so meaningless for both *Human* and *Bot* category. Moreover, *Human* SuccessScore tends to be grater than zero almost exclusively before the first days of August. Our analysis continued by trying to group similar users by applying classification and clustering analysis to the created time series. The time series used in both analyses were subjected to various preprocessing steps, including no preprocessing, post-denoising (rolling with window size={7,14,30}), and post-denoising combined with *TimeSeriesMinMax* normalization.

6.1 Clustering analysis

We choose shape-based and feature-based clustering strategies to cluster comparable users. In the former, *TimeSeriesKmeans* from *tslearn* was tested using *Euclidean* and *DTW* (Dynamic Time Warping) metrics, while in the latter, the K-means algorithm was applied to a total of 10 features that were extracted from the time series: mean, standard deviation, variance, median, $10th$ percentile, $25th$ percentile, $50th$ percentile, $75th$ percentile, $90th$ percentile and interquartile range. All these clustering techniques were run by setting the number of clusters at 2 to test if they were able to

cluster *Bot* and *Human* categories separately.

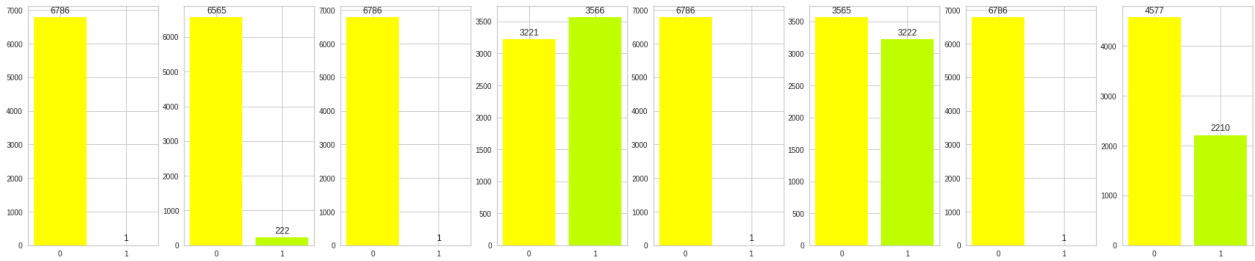
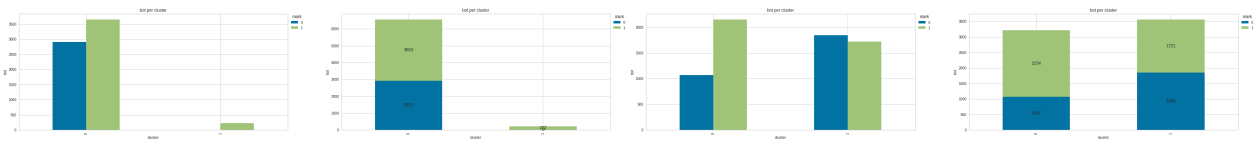


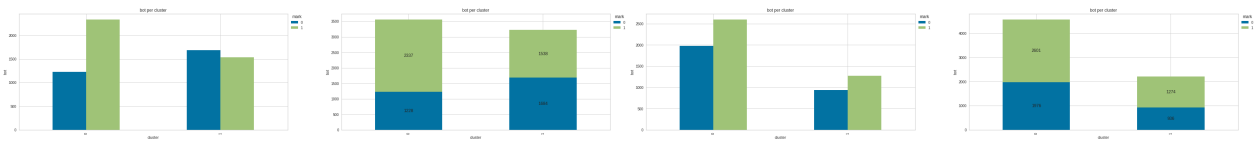
Figure 19: Clusters cardinality for the Euclidean metric in TimeSeriesKmeans. Left to right: original time series, normalized original time series, denoised time series with a window size equal to 7, normalized denoised time series with a window size equal to 7, denoised time series with a window size equal to 14, normalized denoised time series with a window size equal to 14, denoised time series with a window size equal to 30, normalized denoised time series with a window size equal to 30).

Fig. 19 show the clusters cardinality for the Euclidean metric in TimeSeriesKmeans. The first interesting thing that we note is that tests 1, 3, 5 and 7 (from left to right), the ones with non-normalized time series, failed. Fig. 20 instead shows the characterization of the clusters with respect to the Bot distribution in the best-performing tests (2, 4, 6, 8). Test 20a is the one that obtained the most unbalanced clustering but, at the same time, is the only case in which one of the two clusters (1) consists of a single category, in this case Bot. These users are daytime, write in English, and have 2019 as *most_freq_year*. Their mean values are higher in *statuses_count*, *num_urls*, *text_entropy*, *avg_textlen*, *days_on_tw* and *activity_richness*.



(a) Bot distribution (normalized original time series)

(b) Bot distribution (normalized denoised time series w=7)



(c) Bot distribution (normalized denoised time series w=14)

(d) Bot distribution (normalized denoised time series w=30)

Figure 20: clusters characterization with respect to the Bot distribution in the best-performing tests (2, 4, 6, 8).

Fig. 21 shows the clusters cardinality for the DTW metric. This time TimeSeriesKmeans was run only with the normalized time series cases. As in the Euclidean metric scenario, Bot distributions are reported (Fig. 22).

Here 22d cluster 1 consists almost entirely of Bot category. From the characterization of this cluster

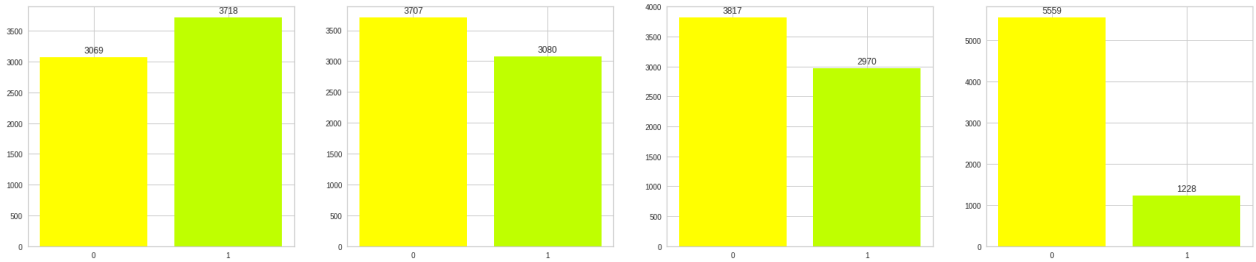
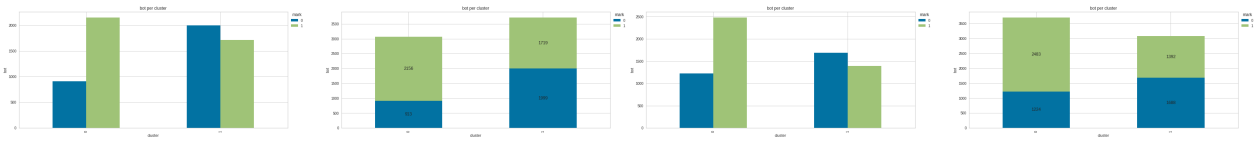
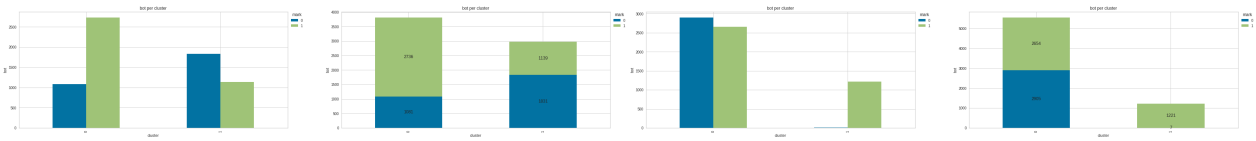


Figure 21: Clusters cardinality for the DTW metric in TimeSeriesKmeans. Left to right: normalized original time series, normalized denoised time series with a window size equal to 7, normalized denoised time series with a window size equal to 14, normalized denoised time series with a window size equal to 30).



(a) Bot distribution (normalized original time series) (b) Bot distribution (normalized denoised time series w=7)



(c) Bot distribution (normalized denoised time series w=14)(d) Bot distribution (normalized denoised time series w=30)

Figure 22: clusters characterization with respect to the Bot distribution, DTW metric.

with respect to categorical features, we can infer that: 7 Bots are nightly, most of them uses Italian language, their *most_freq_year* continues to be 2019 while their *most_freq_year* is varied with a slight majority of 11 p.m. and midnight. Moreover, their mean values are higher in *statuses_count*, *favorite_count*, *num_hashtags*, *num_urls*, *max_favorite_count*, *std_favorite_count*, *text_entropy*, *avg_textlen*, *days_on_tw* and *activity_richness*.

Finally, we report the results of the feature-based clustering.

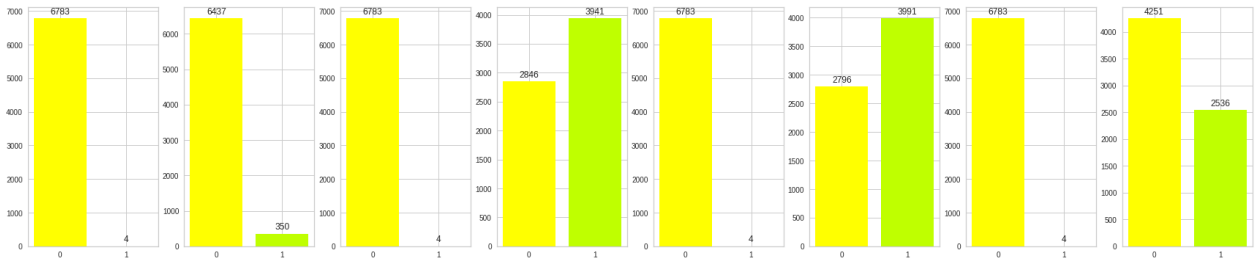
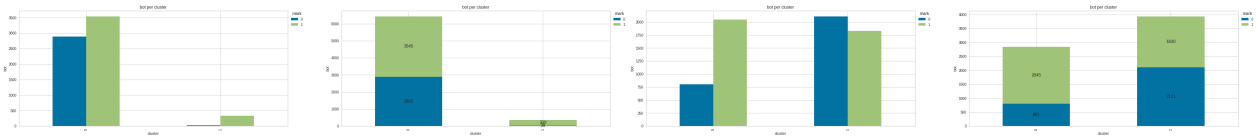


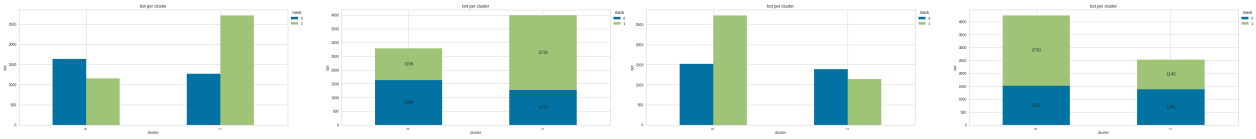
Figure 23: Clusters cardinality for the Euclidean metric in K-means, feature-based clustering. Left to right: original time series, normalized original time series, denoised time series with a window size equal to 7, normalized denoised time series with a window size equal to 7, denoised time series with a window size equal to 14, normalized denoised time series with a window size equal to 14, denoised time series with a window size equal to 30, normalized denoised time series with a window size equal to 30).

Fig. 23 shows the clusters cardinality obtained by K-means (Euclidean distance) ran on the time series features extracted. Comparably to the Euclidean distance in TimeSeriesKmeans, the bar plots reveal that normalization of the time series turns out to be a key step in their clustering. Furthermore, looking at the Bot distributions (Fig. 24) we also notice that the Euclidean distance in the $K = 2$ cases seems to be less optimal than DTW and that in feature-based clustering, the algorithm succeeds in clustering the two categories with more struggle.



(a) Bot distribution (normalized original time series)

(b) Bot distribution (normalized denoised time series w=7)



(c) Bot distribution (normalized denoised time series w=14)

(d) Bot distribution (normalized denoised time series w=30)

Figure 24: clusters characterization with respect to the Bot distribution, DTW metric.

6.2 Classification analysis

For this task, we tried to classify time series by exploiting their shapelets and using *LearningShapelets* model. As before, shapelets were extracted from time series subject to different types of preprocessing. The results can be found in Table 8.

Time series preprocessing	Accuracy	Shapelets
Original time series	**69%**	Fig. 25a
Normalized original time series	68%	Fig. 25b
Denoised time series rolling $w = 7$	43%	Fig. 25c
Normalized denoised time series rolling $w = 7$	68%	Fig. 25d
Denoised time series rolling $w = 14$	60%	Fig. 25e
Normalized denoised time series rolling $w = 14$	**69%**	Fig. 25f
Denoised time series rolling $w = 30$	67%	Fig. 25g
Normalized denoised time series rolling $w = 30$	67%	Fig. 25h

Table 8: Shapelets classification accuracy (top scores in bold).

Time series were previously splitted to form train and test sets (80% and 20%) but we mantained their length invariated (319). Like in the clustering analysis, the results reveal that normalization is a good practice. The most surprising and unexpected finding was that the time series that had not undergone preprocessing had the best performance. The best option for denoising among the tested window sizes (7, 14, and 30) is $w = 14$, which is equal to the number of days in 2 weeks. Certainly,

the performance of the utilized model might be improved with additional preprocessing and a wider search for parameters.

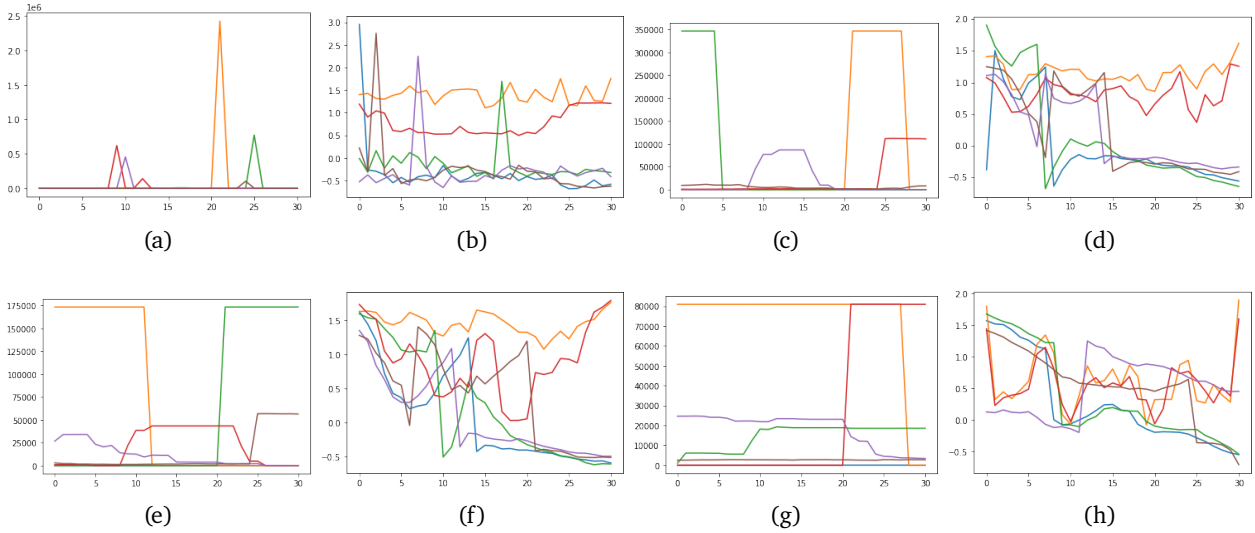


Figure 25: Extracted shapelets.