University of Pisa
Data Mining

Analisys of Twitter users behaviour

Irene Pisani[1] and **Andrea Iommi**[2]

[1] *M.Sc. Computer Science, Artificial Intelligence* - 560104 - i.pisani1@studenti.unipi.it
[2] *M.Sc. Computer Science, Artificial Intelligence* - 578212 - a.iommi2@studenti.unipi.it

January, 2023

Abstract

Given a several millions of tweets, crawled from the famous social network, the aim of this project is to perform a complete analysis of the data to identify patterns and relationships. The result obtained, supply a well formed information about the user enrolled on the platform.

1 Data Understanding and Data Preparation

1.1 Datasets

In this project we have explored and exploited the *Users* dataset, which collects information about the user, and the *Tweets* dataset, which contains information about some published tweets. Concerning the datasets types we can state they both store records in form of data matrix. The *Users* dataset is represented by an 11508 by 6 matrix, where there are 11508 users are described through its 6 attributes. Likewise, the *Tweets* dataset can be represented by an 13664696 by 10 matrix, where there are 13664696 rows, one for each tweet, and 10 columns, one for each attribute that describe the tweet.

1.2 *User* dataset insight

The initial analysis of the dataset revealed the presence of null values in the `name` and `statuses_count` columns, the absence of duplicate rows, and incorrect parsing of some columns. Indeed, we we proceeded to correctly casting the column types, to replace null values with consistent ones and to check out the syntactic accuracy of the columns. In this way, we will be able to further explore the attribute's values distributions.

id - `int64` It is a numerical attribute that uniquely identifies each user. This column has 11508 syntactically correct unique values within the closed range of 678033 to 3164941860.

name - `string` It is a continuous attribute that contains the name (or name and surname, or nickname) chosen by the user when has joined on the Twitter platform. We judged this attribute useless for our purposes we proceeded to discard this columns.

real - `boolean` It indicates if a user is a bot (False/0) or a genuine user (True/1); no missing/null values has been detected. 6116 users (53,146% of the users) are labelled as genuine users, while the remaining 5392 users (46,854 %) are labelled as bot; we can state the dataset is well-balanced w.r.t to the bot attributes.

lang - `string` This categorical attribute counts 26 unique values with no null value, but with some erroneous ones. We handled problems such as system error at registration time by replacing the incorrect values with most frequent language (i.e. en) and we substituted value that does not correspond to any existing language acronym with plausible ones. In addition, in order to reduce features carnality, we unified different variants of the same language under their more general acronym and we assigned the label "other" to language having less than 10 users. We ended up with 10 unique values: 87% of the recorded user have registered with English language, 0.0787% with Italian, and 0.028% with Spanish; all the other 7 languages have been chosen by less than 0.005% of the users.

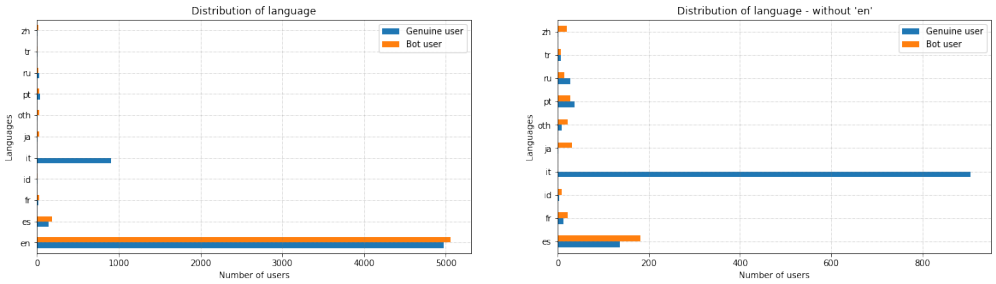


Figure 1: Language distribution for genuine and bot users.

created_at - `datetime` It contains the date and time of each user registration on Twitter. The observation period identified by this attribute starts on the 2012-01-24 and ends on 2020-04-21. We can observe that except for the last year (2020) there is an incremental tendency to subscribe to the platform with a spike on February and March 2019.

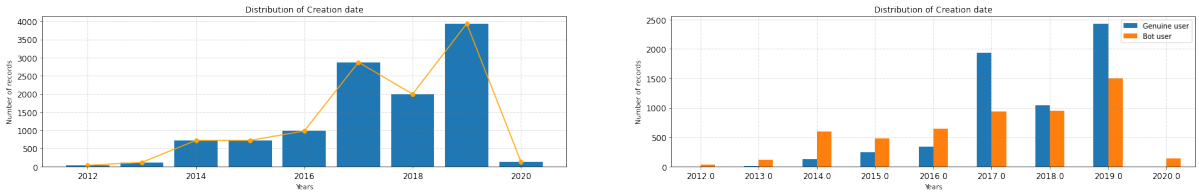


Figure 2: Amount of account registration per year.

statuses_count - `int64` It is a numerical attribute describing the total amount of tweet posted by an user. This column presents a relevant number of missing values (only for bot users) since 399 records report a null value. Moreover, we have found several inconsistent information: when retrieving from *Tweets* dataset the number of posted tweets per user, sometiems the retrieved value does not match the old one stored in *status_count* column. We have enforced consistency on *statuses_count* by replacing the old value with new retrived one when the new one is higher. Next, we improve data quality of this attribute by looking for outliers and removing them; we have removed 0.032% of the users stored in the users dataset. Looking at Table 1 we can observe that the behavior of real users and bot users differs in the number of total tweets published. We can notice also that given the total number of posted tweets, the 11.1% are made by genuine users and the 88,9% are made by bot users.

User	count	mean	std	min	25%	50%	75%	max
Real user	6116	1185.102	5373.551	0	32	53	78	214410
Bot user	4993	11638.219	26719.438	0	61	2041	11585	399555

Table 1: Statistics from `status_count` attribute: difference between genuine and bot users behaviour

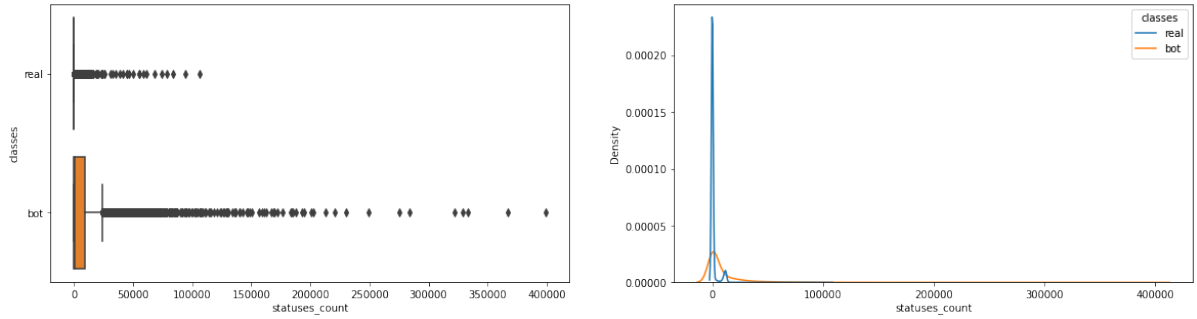


Figure 3: `statuses_count` distributions for genuine and bot users: boxplot and KDE.

1.3 *Tweets* dataset insight

Also in this case we started by analyzing the dataset looking for missing, duplicated and syntactically incorrect values as well as by checking the correct parsing of the column. This study revealed that all the columns are not properly parsed and most of them contain a large percentage of inconsistent values (both null and alphanumeric values[1]).

Attribute	`user_id`	`retweet_count`	`reply_count`	`fav_count`	`num_hash`	`num_urls`	`num_ments`	`text`
Incorrect value	0.031%	0.046%	0.136%	0.136%	0.136%	0.136%	0.0723%	0.0394%

Table 2: Percentage of incorrect values for each attribute in *Tweets* dataset.

1.3.1 Cleaning procedure for *Tweets* dataset

This section aims at illustrating the procedures adopted to improve quality of the *Tweets* dataset by removing errors and replacing missing values. Table 3 shows how many users have been discarded.

(A) Handle null `user_id` We have identified 2 types of situations in which records are stored with null value for the `user_id` attributes. In the first case there are not other rows with the same `created_at` while in the second case there is another row that has the same `created_at` and `text` values. In the former case we delete it because we cannot retrieve any useful information for the user. In the latter, we can state the row with null `user_id` is a just a meaningless duplicated row and we proceed again to delete it.

(B) Handle duplicated row Sometimes there are rows that share the same values across the columns, in this case, since it makes no sense to keep all copies we proceed to delete them.

(C) Handle not-numeric user_id In the `user_id` column we might find a non numeric `user_id`, since we cannot associate it to any user stored in the *Users* dataset we delete these rows.

(D) Columns casting and incorrect value replacements We replaced null value in the *text* column with the empty string and we temporarily marked the erroneous values (other null values or alphanumeric values) with -1, in order to have the possibility to cast the column. `user_id`, `retweet_count`, `reply_count`, `favorite_count`, `num_hashtags`, `num_urls`, `num_mentions`, columns was cast to integer value; `text` column was parsed as string, and `created_at` as date-time values. Then we proceeded to replace all incorrect values previously transformed into -1 with plausible ones. We have thought that the better values we can place is the median on a subset of correct values.

[1] Alphanumeric values were not considered suitable since all these attributes must be numeric attributes

(E) Irrelevant tweets delete The rows with all zero-values and empty text were deleted since they cannot give us any useful information about the users.

(F) Check the real user_id We checked if in the *Tweets* dataset there are tweets posted by user not described in the *Users* dataset (i.e. invalid user). The result shown that there are no mismatch between the users in *Users* dataset and *Tweets* dataset.

(G) Outliers elimination in tweets In *Tweets* dataset we dropped out a total of 33 tweets.

Figure 4: Example of results for outliers detection and removing process: bboxplot for outlier detection in `retweet_count` distributions before and after managing outliers.

Cleaning step	A	B	C	E	F	G
Deleted row	217283	2999450	216730	139985	0	33

Table 3: Number of rows deleted in the each one of the aforementioned cleaning step.

1.3.2 Statistics and distributions for *Tweets* dataset

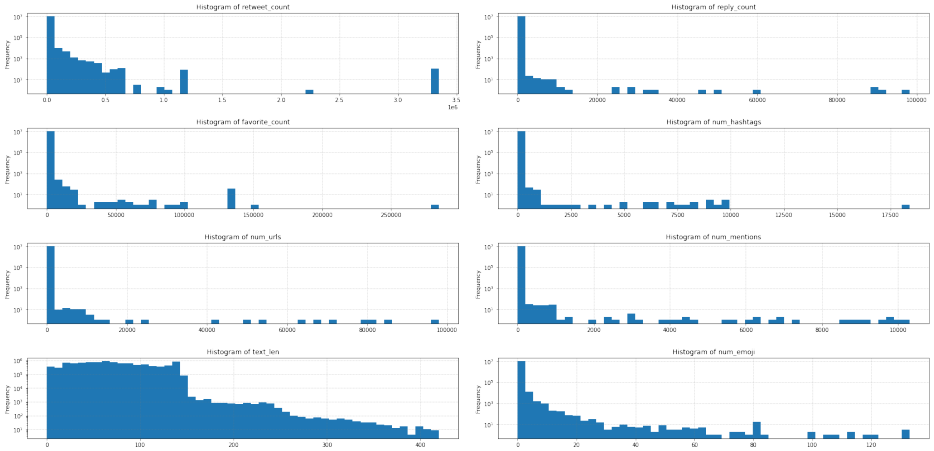


Figure 5: Histograms of *Tweets*'s attribubtes distributions after the cleaning phases.

1.4 New features extraction

At this stage of the project, we exploited the *Tweets* dataset for extracting new features able to describe users through additional information. For each user we computed the following indicators (i.e., features):

- `Ayears`: Average number of published tweets per year
- `Amonths`: Average number of published tweets per month
- `Adays`: Average number of published tweets per day
- `Achars`: Average number of alpha-numeric characters used in published tweets
- `Schars`: Standard deviation of the number of alpha-numeric characters used in published tweets
- `Aemoji`: Average number of emoji inserted in the text of a tweet
- `Alike`: Average number of likes received per tweet
- `Slike`: Standard deviation of number of likes received per tweet
- `Aretwee`: Average number of retweet received per tweet

- `Sretwee`: Standard deviation of the number of retweet received per tweet
- `Areply`: Average number of reply received per tweet
- `Sreply`: Standard deviation of the number of reply received per tweet
- `Ahash`: Average number of hashtag inserted in a tweet
- `Shash`: Standard deviation of the number of hashtags inserted in a tweet
- `Thash`: Total number (a summation) of hashtags inserted in all tweet
- `Aurls`: Average number of URLs inserted in a tweet
- `Aments`: Average number of mentions inserted in tweet
- `Sments`: Standard deviation of the number of mentions inserted in tweet
- `Fyear`: Year in which the user post the most
- `Fmonth`: Month in which the user post the most
- `Fday`: Week days in which the user post the most
- **pop_sco**: It is a popularity score of the user that takes into account the total number of like, reply and retweet received to all its tweets.
$$pop\_score = \frac{tot\_likes + tot\_reply \ + tot\_reweet}{tot\_tweets}$$
- **itr_sco**: It is a interactivity score of the user that takes into account the number of hashtags, urls and mentions inserted in all its tweets.
$$itr\_score = \frac{tot\_hashtags + tot\_mentions \ + tot\_urls}{tot\_tweets}$$
- `Echars`: Shannon entropy of *Achars*
- `Elike`: Shannon entropy of *Alike*
- `Epop_scr`: Shannon entropy of *pop_sco*
- `Eint_scr`: Shannon entropy of *itr_sco*
- `Einervals`: Shannon entropy of the interval

Some additional KPI was computed later for classification purpose and were not employed in the clustering task:

- `Tlike`: Total number (a summation) of likes received for all tweet
- `Tretwee`: Total number (a summation) of retweet received per tweet
- `Treply`: Total number (a summation) of reply received per tweet
- `Surls`: Standard deviation of the number of URLs inserted in a tweet
- `Turls`: Total number (a summation) of URLs inserted in a tweet
- `Tments`: Total number (a summation) of mentions inserted in tweet

1.5 Final *User* dataset for clustering

All the features listed in Section 1.4, with the exception of those defined later for the classification task, were added to the original *Users* dataset to perform, again, the outlier detection. 445 outliers were removed, thus 96% of users were retained.

Then we tried to figure out any highly correlated features in order to remove them and obtain a not-correlated dataset for clustering process. By setting a maximum correlation threshold to 0.80 we identified 4 pairs of highly correlated features:

- (`Aretwee`, `popsco`) with correlation value equal to 0.9991;
- (`Ayears`, `Amonths`) with correlation value equal to 0.9166;
- (`Amonths`, `Adays`) with correlation value equal to 0.8901;
- (`Aurls`, `Eitr_sco`) with correlation value equal to 0.8325.

We proceeded to remove from the dataset `Aretwee` and `Aurls`, `Amonth` attributes.

We ended up with a no-correlated features dataset containing a total amount of 11063 sample (i.e. unique users) described through 30 attributes (categorical attributes and numerical attributes) whose distributions, for the new ones, are reported in Figure 6 and 7.

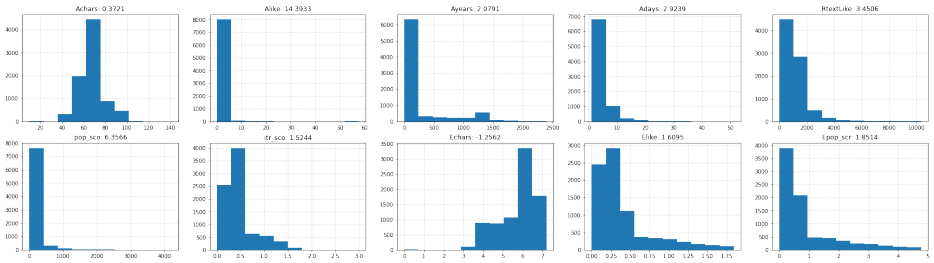


Figure 6: Values distributions for new-generated numerical attribute.

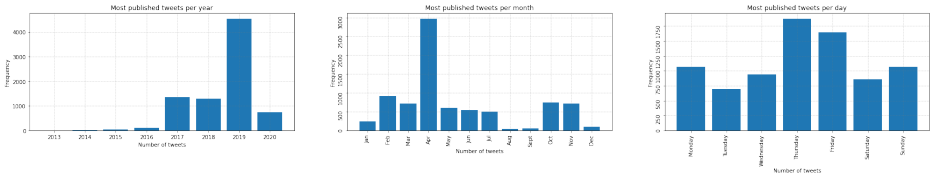


Figure 7: Values distributions for new-generated categorical attribute.

2 Clustering

This section aims at illustrating the methodology used to group together users with similar kind on behaviour. We have to mention that while the numerical features of the final *User* dataset has been directly implicated in the clustering process, the categorical ones were only exploited for the clusters characterization.

As a preliminary step we standardized our dataset with *z-score* standard scale and we proceeded by looking for natural clusters in our scaled dataset by exploiting different clustering algorithm.

We start by taking into account *DBSCAN* and *Kmeans*. Since early experiments shows that *DBSCAN* algorithm seems to not working well in this context we decided to exploits this algorithm in 2-step clustering technique proposed by [1]. The main concept involves using *DBSCAN* to detect some marginal outliers or noisy points in the dataset; after removing them, *K-Means* algorithm is used to extract more meaningful and significant clusters, thus improving the overall final partitions. We were able to adopt this alternative approach since the removal of outliers in data preparation was performed with a very gentle and minimalist approach.

Next, coming back using the full *User* dataset, we analyzed the clusters detected with hierarchical clustering approaches, *Xmeans* and *SOM* (Self Organizing Map).

2.1 DBSCAN Clustering

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is density-based approach that proved to be useful in identifying non-globular shaped clusters. The algorithm estimates the density around each point by counting the number of points in a ϵ (epsilon) circle and it applies a thresholds called **minPts** (minimum points) to identify core, border and noisy points. In a second step, the central points are grouped together into a cluster if they are "reachable by density". This algorithm requires only these 2 aforementioned parameters.

2.1.1 Model selection

Since we are in an *unsupervised field*, there is no direct manner to assess our parameters, like in a supervised learning with the loss. So we have relied on two heuristic methods that help us defining a promising range

of values for performing grid search.

- **Epsilon**: we used the Knee method to figure out optimal values for ϵ. The method consists of computing the k-nearest neighbor distances in a matrix of points. These k-distances are plotted in an ascending order for detecting the presence of a knee. A knee corresponds to a threshold where a sharp change occurs along the k-distance curve and it suggests the optimal ϵ value.
- **minPts**: we looked range that covers the double of $|D| + 1$, where D is a dimension of dataset, to be sure to including promising values.

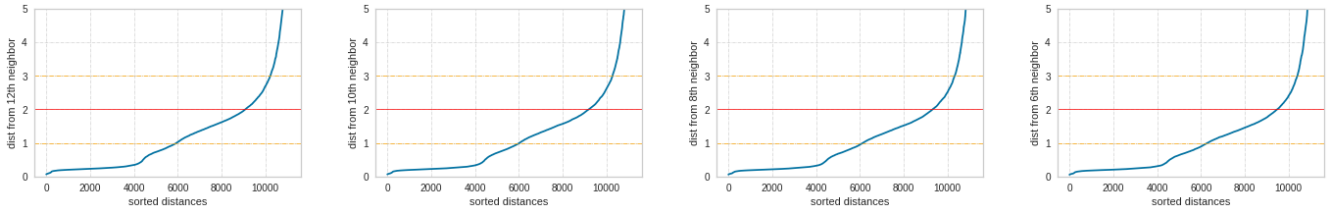


Figure 8: Knee method for finding optimal DSCAN parameter values.

Grid search. Based on charts 8, we have discovered that the promising values ϵ were around 2, so we have searched the optimal value in a range between 1 and 3. After launching grid search reported in Table 4 we found out that the best solution is ϵ equal to 1.7 and minPts equal to 12. With this hyper-parameter configuration we obtained 3 clusters + 1 Noise cluster.

Parameter	Start	End	Stride
epsilon	1	3	0.1
minPts	10	30	2

Table 4: Hyper-parameter space explored with grid search.

2.1.2 Clusters interpretation and characterization

Generally we can state that DBSCAN has recognized a one big cluster and two small ones. In the cluster 0 (the bigger one) and 1 we are not able to identify a particular behaviour or pattern within all generated charts since points belonging to these group appear too sparse; instead, an high degree of cohesion is observable for cluster 2. For this we decided to report below and in-depth characterization of this cluster.

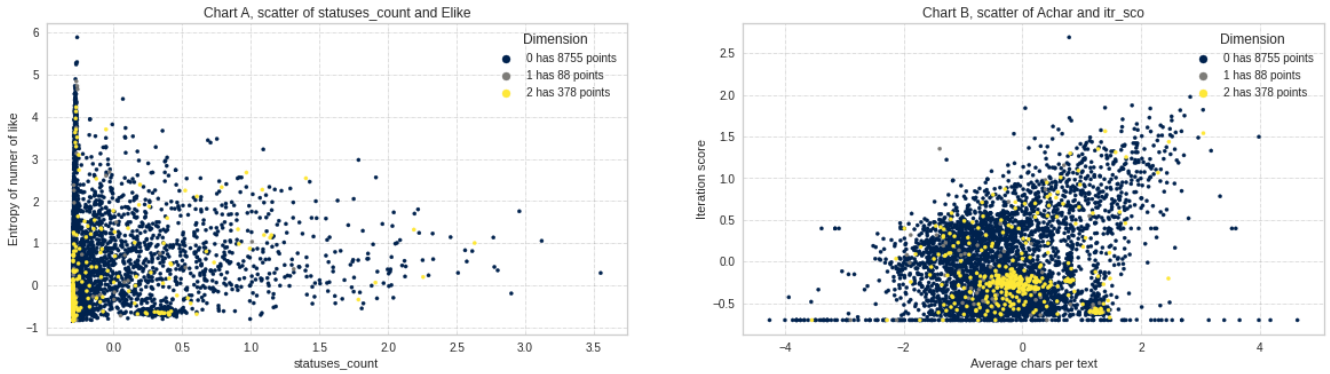


Figure 9: subset of overall results of DBSCAN

In order to give a more relevant interpretation for the main outcomes we selected two charts that seem to be interesting and we omitted the noise cluster in order to have a cleaner vision of other groups.

On the left-side of Figure 9 we have reported a scatter-plot of `Elike` and `statuses_count`. Cluster 2 is mainly characterized by users having a low number of published tweets and relatively low value of `Elike`.

On the right-side of Figure 9 we have showed a scatter-plot of `Achars` and `Eint_sco`. Also in this way we can observe that the cluster 2 is concentrated at 60 chars per tweets and has low interactivity score. This situation could be plausible typically for the genuine users since they not insert a lot of URLs, mentions and hashtag and their texts length is roughly the same.

The previous interpretation is supported by the genuine/bot user distribution in the cluster showed in Figure 10; in fact, we can notice several things:

- Cluster 2 is composed only by genuine users, this confirm that we have found a typical behavior of some genuine user.
- In all clusters except of Noise there is a genuine users predominance.
- Bot users are mostly grouped into cluster 0, the greatest one.

This mix of users and the previous interpretation lead us to say that not meaningful patterns can be detect with this algorithm.

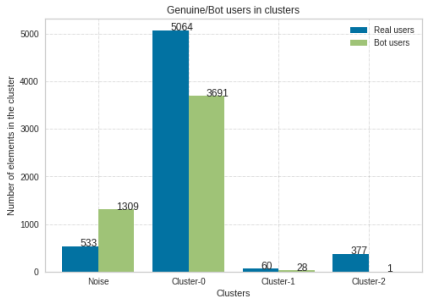


Figure 10: Histogram of Bot/Genuine users into clusters

2.1.3 Clusters evaluation

Davies-Bouldini	Silhouette	Calinski-Harabasz
2.087	0.2064	505.5334

Table 5: DSCAN clustering evaluation via predefined metrics.

All the metrics score reported in Table 5 confirms us the low quality performance achived with DBSCAN algorithm. A Davies-Bouldini score value equal to 2.087 confirms the fact that the obtained cluster are not well separated; this is also remarked by the low value of silhouette score, indicating also a low degree of cohesion within clusters. The Calinski-Harabasz, which, evaluate the dispersion of the data, highligthed how the detected clusters are not well formed and overlapped.

Overall conclusion. After performing the DBSCAN algorithm several times with different configuration we discover some configuration that allow a result interpretation; anyway, we conclude that the DBSCAN algorithm is not suitable and performing on our dataset. This limitations could be due to the fact that DBSCAN algorithm is often not able to recognize different cluster with different densities; this weakness clearly emerges from the scatter plot available on the notebook where is possible to observe how our data-points are placed in a variable density space.

2.2 K-Means clustering

After identifying noisy data with DBSCAN we proceed to remove them from the *User* dataset and we start looking for more understandable and well defined clusters using *K-Means* algorithm with *k-means++* initialization method.

2.2.1 Optimal number of clusters

We adopted 3 different solutions in attempting to identify the optimal number of clusters k.

- **Elbow method with SSE.** By plotting the SSE as a function on the number of clusters, this method aims to identify the optimal value of k by searching elbow or knee within the curve. However, since we would like to analyze the clustering results also for some sub-optimal k values, we perform the elbow method by considering incremental range of k values. In this way, by not discarding sub-optimal k values, we obtained a list of candidate k values (one candidate value for each considered range). Obtained candidate k values: 8,9.
- **Silhouette method.** By plotting the Silhouette score as a function on the number of clusters, this method aims to identify the optimal value of K by searching for the maximum value in the curve. Also in this case we incremental range of k values. Obtained candidate k values: 3, 13.
- **Ward linkage method.** By performing hierarchical clustering with ward linkage method we were able to consider the optimal value of k suggested by this algorithm. Obtained candidate k value: 2.

We ended up with a list of several candidate k values and we evaluated the clusters obtained for each one of these k values using different metrics to gather some insight about clusters cohesion, separation, definition and dispersion. As show in Table 6, this evaluation step allow us to select the optimal value of k based on the clustering evaluation metrics.

K value	SSE	Davies-Bouldini	Silhouette	Calinski-Harabasz
2	69969.311	1.515	0.364	**3181.567**
3	59030.091	1.309	0.404	2739.499
8	29097.497	**1.013**	0.445	2940.946
9	26952.299	1.049	0.448	2869.494
13	**19922.963**	1.101	**0.471**	2857.561

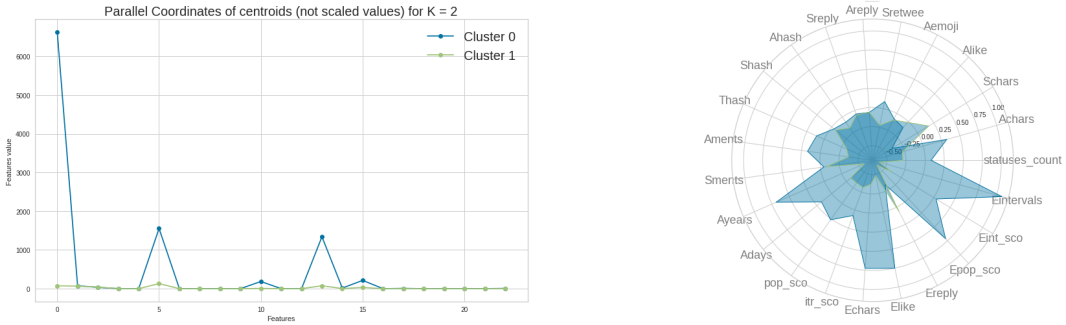
Table 6: KMeans clustering evaluation with several metrics.

Since all these metrics did not agree in the optimal number of clusters we decided to take 2 as optimal k value following Calinski-Harabasz best score. This choice was enhanced by the fact that when computing these scores on the not normalized version of our dataset all 4 metrics agreed in indicating 2 as the optimal k value.

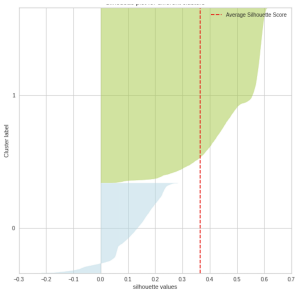
2.2.2 Clusters analysis and characterization

In order to fully understand the obtained cluster with K-Means algorithm for k equal to 2, we try to figure out additional information regarding clusters composition and population.

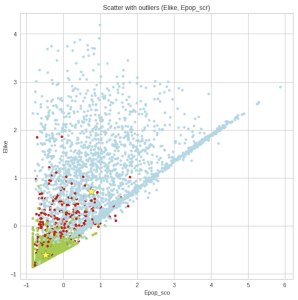
We start by analyzing the coordinates of cluster centroids in order to identify the most influential attributes. Figure 11a shows that features `statuses_count`, `Sretwee`, `Ayears` seem to be the most influential ones since the in the not-scaled features space the coordinates mostly differ in their values. Furthermore, considering the radar plot showing coordinate value in scaled features space we can observe a regular trend: for most features the coordinate of centroid 0 has significant higher value then coordinates of centroid 1.



(a) Coordinate of clusters centroids (not scaled features space on the right, scaled features space on the left.



(b) Silhouette analysis for each cluster.



(c) Cluster visualization in features space (`Elike`, `Epop_sco`) highlighting point with negative silhouette score.

Figure 11: Insight about the resulting *K-Means* clustering solution

Looking at Figure 11 we can asses that the resulting 2 cluster differs in dimensions as well as in cohesion and sparsity degree. While the cluster 1 seems to be a globular and dense cluster where all point are close to each other, cluster 0 seems to be more sparse. We can also observed that 2 cluster are overlapping each other (thus leading to low degree of separation) and that points with negative value of silhouette can be treated as noisy points. The data-points scatter plot have confirmed and explained the evaluation results obtained in cluster evaluation via predefined metrics.[2]

A supplementary characterization step has been carried out w.r.t. the categorical attribute in order to verify if the clustering process follows one of these categorical features. From the performed analysis, not reported here in details but fully available in the notebook, we can assess that both clusters are relatively well-balanced with respect to the `real` and `lang` categorical features and they don't seem to follow anyone of this partitions. At the same way, by studying the distribution of user in each cluster w.r.t. `Fyear`, `Fday`, `Fmonth` attributes we cannot derive any significant clusters property related to these specific categorical attributes.

Other interesting information was derived by looking at the distributions and some statistics of each cluster. More in detail, for each feature we explored its values distribution within each clusters, then clusters were visualized in 2 dimensions considering each possible features pairs; finally for each cluster useful features statistics were computed. We report the results obtained for some salient features such as `Eint_sco`, `Epop_sco`, `Eintervals`, `Elike` and `statuses_count` in Figure 12 and in Table 7.

Looking to the partitioning of the points in the feature space, the features distributions and statistic we can observe how the two generated clusters divides the data-points into two distinct groups following a

[2]This conclusion has been drawn by looking at visualization in 2 dimension of the correlation between `Elike` and `Epop_sco` but analogues ones came out from looking at different numerical features.

new separation principle. By investigating this new separation principle more deeply we can deduce the following observations.

The cluster 1, the more dense and cohesive, contains those points described by low feature values: low popularity score, low interactivity score, low number of tweets posted in certain period etc. The cluster 0, the most scattered and least dense, contains those points described by high feature values: high popularity score, high interactivity score, high number of tweets posted in certain period etc.

Stats	statuses_count	Epop_sco	Eint_sco	Eintervals
Mean	2773.364	2.041	1.216	6.755
Min	509.959	-0.044	0.351	2.557
50%	1462.371	1.929	1.299	7.032
Max	23710.039	6.689	2.770	9.526

(a) Statistic for cluster 0

Stats	statuses_count	Epop_sco	Eint_sco	Eintervals
Mean	531.769	0.273	0.856	2.994
Min	506.541	-0.065	0.087	-0.152
50%	521.924	0.282	0.953	2.203
Max	8298.814	2.188	1.930	9.033

(b) Statistics for cluster 1

Table 7: Explore salient features statistic for each clusters

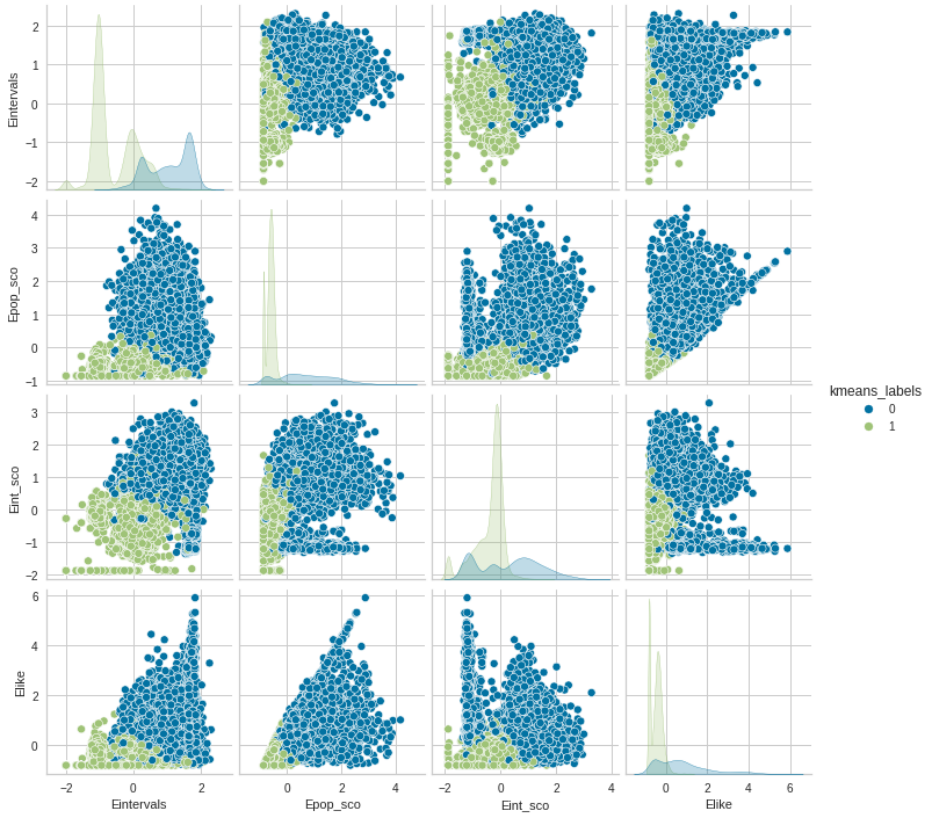


Figure 12: add caption

2.2.3 Final interpretations for K-means with K=2

Thus, we can say that the cluster process assigns a certain point to a cluster based on a score that, for convention, we call social score. The user's social score depends on his behavior on the platform: the more active, popular and interactive the user is on the platform the higher his social score will be. Some additional observations led us to the following final considerations. The clustering process divides users into 2 groups by looking at the magnitude (high or low) of the social score obtained.

We can observe that a larger portion of users are assigned to the cluster that identifies a lower social score, and only a small portion of users are assigned to the cluster defined by an higher social score. This phenomenon leads toward an unbalanced bi-partition of the data that nevertheless reflects what really happens within social networks where few users are distinguished by their high level of popularity (how many like, reply retweet a users received for each post), activity (how frequent the users post a tweet) and interactivity (how many characters, mentions, emoji, URLs the user insert in its tweet).

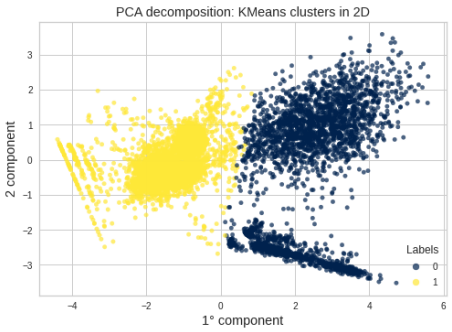


Figure 13: 2-step clustering results - visualization with PCA.

Overall conclusions about 2 step-clustering. In our case, despite a low separation of this data groups in different features space, we are still able to properly visualize points partitions in the reduced features space with PCA. As show in Figure 13 deleting noise detected with DBSCAN allow us to extract 2 significant clusters which are almost linearly separable when they are reduced to 2 dimensions. This outcomes also gives us significant and meaningful information about the social score/social power of the user on the Twitter platform. Moreover, the good convergence property of these clustering algorithms allow us to arrive at this sub-optimal solution in relatively short time.

2.3 Hierarchical clustering

Hierarchical clustering approach has been used for building clusters by merging them successively in an agglomerative (or bottom-up) way starting from each observation in its own cluster and ending in a hierarchy of clusters. We decided to further explore the following most common linkage method:[3]

- **Ward linkage**: it minimizes the sum of squared differences within all clusters;
- **Average linkage**: it minimizes the average distances between all observations of clusters pairs;
- **Complete linkage**: it minimizes the maximum distance between observations of clusters pairs;
- **Single linkage**: it minimizes the distance between the closest observations of clusters pairs.

2.3.1 Hierarchical clustering with default `cut-threshold`

It is possible to render the linkage matrix encoding the hierarchical clustering as a dendrogram and exploit it for deriving the optimal number of clusters: the number of resulting clusters depends on a *cut-threshold*(by changing the height of the cut we can change the number of resulting clusters). Scipy library offers a standard rule for computing the best *cut-threshold* in the hierarchy and allow to render the dendograms in such a way that each resulting clusters under the *cut-threshold* is highlighted in different color. We decide to exploits this default parameter to obtain some preliminary insight about the best *cut-threshold* and the resulting main clusters.

[3]In all the aforementioned methods the points distance was computed as euclidean distance.

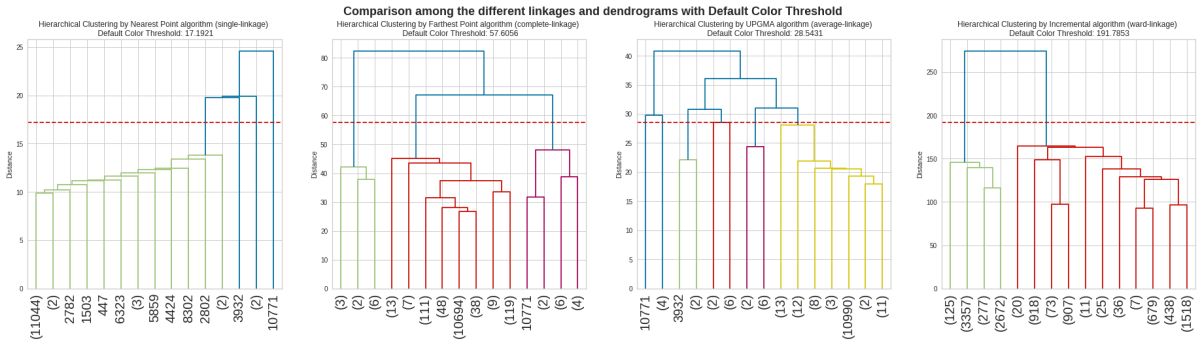


Figure 14: Dendrograms resulting using different Linkage method (single, complete, average, ward), with corresponding optimal cut suggested by a Scipy default *cut-threshold*.

Figure 14 shows how these method differs in the outcomes. The single, complete and average linkage method, detecting respectively 2, 3 and 6 clusters, ending up grouping most of the observations in a single huge cluster. Only the 2-clusters partition obtained with the ward methods seems to generate a more well-balance and meaningful points partitions.

2.3.2 Resulting clusters evaluation

Linkage	cut-threshold	N. clusters	Davies-Bouldin	Silhouette	Calinski-Harabasz	Cophentic
single	17.192	2	0.064	0.905647	146.295	0.889
complete	57.605	3	0.775	0.845	585.918	0.841
average	28.543	6	0.615	0.827	289.504	0.925
ward	191.785	2	1.830	0.281	1913.841	0.373368

Table 8: Evaluation of hierarchical clustering outcomes.

The higher quality performance reached with WARD method is also evidenced by the higher values of the Calinski-Harabasz evaluation metrics reported in the Table 8. Although the magnitudes of the other metrics seem to favor different hierarchical clustering approaches (this is still due to the presence of one huge cluster) the visualization of the clusters reduced to 2 dimension via PCA decomposition led us to conclude that the best solution achieved through a hierarchical approach is the one proposed by the ward linkage algorithm. In fact, considering Figure 15, we can easily asses that complete, single and average linkage method converge more or less the same solutions where the greater part of the point are assigned to a single bigger cluster; just an irrelevant amount of point that can be treated as noisy points are assigned to the other ones.

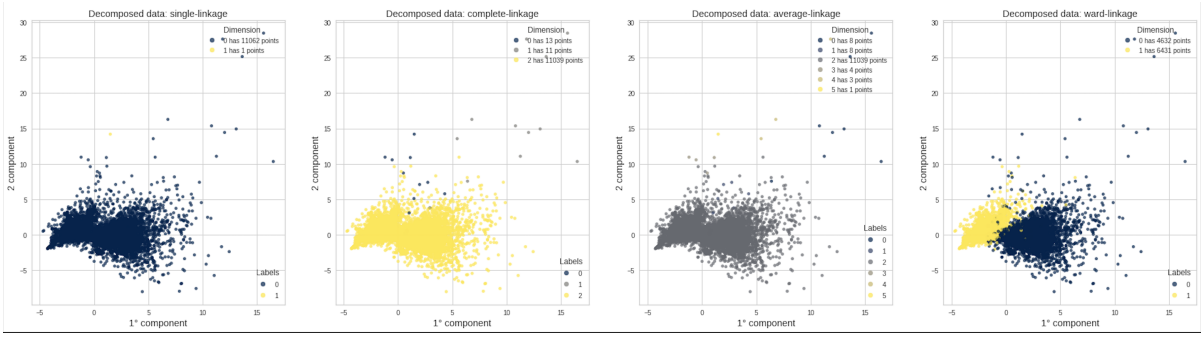


Figure 15: Visualization features space of the generated clusters.

2.3.3 Clusters characterization for WARD Linkage outcomes

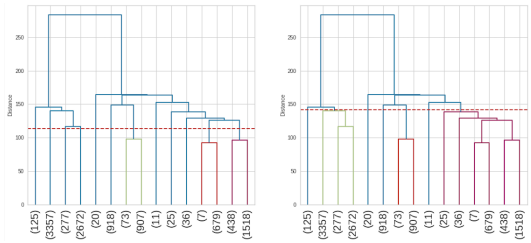
It goes without saying that WARD linkage is the only worthwhile solution for performing a results characterization step. By further exploring feature's statistics and distributions for each clusters we have noted that the outcomes further remarks those one already reached with *K-Means*. In fact, although this 2 obtained clusters have a more balanced populations with respect to *K-Means* they still seems to partition the users based, more or less, on a *social score*: the bigger cluster contains user described by higher features values (thus, high social score) and the other one contains users characterized by lower features values (thus, low social score). [4] This trend is not as evident as in *K-Means* but is still clearly observable.

2.3.4 Explore clustering result by changing the cut-threshold

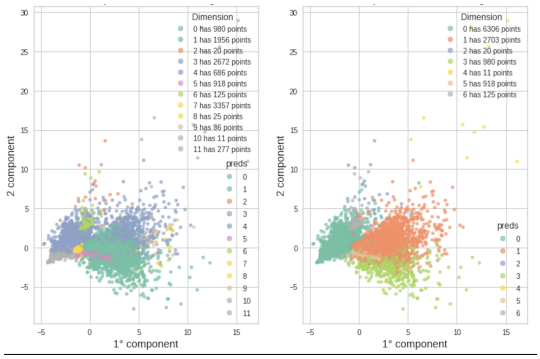
Since sometimes using the default *cut-threshold* for stop merging together clusters could led to sub-optimal clustering solutions, we have also try to figure out how algorithms results change by changing the *cut-threshold* values. Starting from Scipy standard rule :

$cut\_threshdold = coefficient * Linkage\_Matrix[:, 2]$
$coefficient = 0.7$

we consider the cut-threshold value obtained by varying coefficient value in the range between [0.4, 0.9]. We choose this range since with coefficient value lower than 0.4 we have observed all observation in its own cluster and for coefficient value greater then 0.9 we would have a single cluster grouping all observations together.
Even changing the *cut-threshold* value we did not observe any significant improvements with single, average and complete method: they still group in a bigger cluster the greater part of the sample, and only some single observations are grouped in some underpopulated noisy clusters.
Some interesting results are visible with ward method using 0.5 and 0.4 as coefficient values with *cut-threshold* values equal to 141.818 and 113.454. In the first case we obtain 6 clusters: 4 cluster are really more populated, well-defined and cohesive, while the remaining 2 cluster seems to group together some marginal outliers. They are overlapping each other, thus leading to lower separation degree. In the second case 11 clusters are detected. It's clear the most of them are non significant clusters but although the bad separation/cohesion we can still visualize them in a clear way.



(a) KDE, ECDF and boxplots for `Eint_sco` feature.



(b) KDE, ECDF and boxplots `Areply` feature.

Figure 16: Exploratory analysis

This analysis marked how Scipy standard rule have returned us the best *cut-threshold* for our clustering purpose.

2.4 Visual comparison of *Xmeans* and *SOM* clustering results

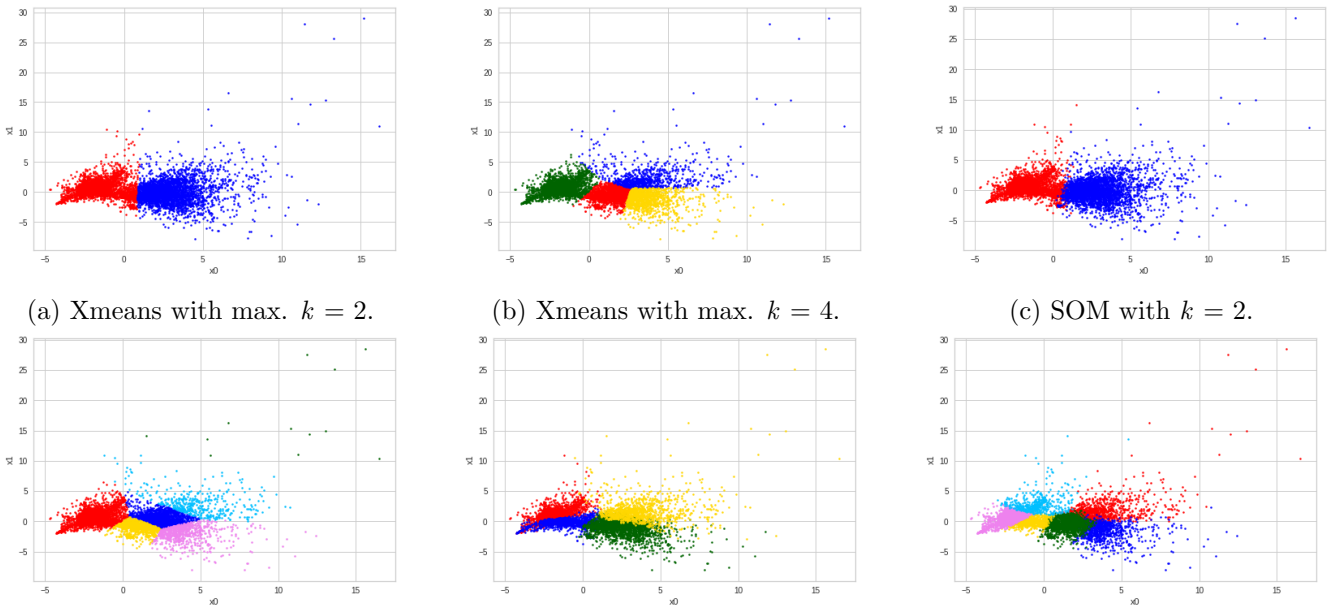
We decided to investigate the performance of *Xmeans* algorithm since it is able to improve *K-Means* by iterative making local decisions about which subset of the current centroids should be split to attain

[4] Charts and statistics that allow us to reach these conclusion are available on the notebook.

a better fit. Instead,*SOM* is a neural approach that was chosen since its implementation proposed for clustering task can be seen as the neural correspective of *K-Means*.

Given these assumptions, we presumed that these algorithms were the most suitable for performing a visual comparison of their results; in this section we are going to illustrate the clusters detected with these algorithms as the maximum number of desired clusters increases.

Figure 17 shows that for a number of cluster k equal to 2 the obtained results are very similar one to each other and they further reminds WARD linkage results; a bit more more overlapping is visible in the case of *SOM*s. For $k = 4$ and $k = 6$ the identified clusters widely differ, suggesting that the separation principle used is not the same. More precise and linear boundaries are visible for *Xmeans*. It can also be seen that, unlike *SOM*, with $k = 6$ the *Xmeans* identify a cluster containing only what we can call marginal outliers.



(a) Xmeans with max. $k = 2$. (b) Xmeans with max. $k = 4$. (c) SOM with $k = 2$.

(d) Xmeans with max. $k = 6$. (e) SOM with $k = 4$. (f) SOM with $k = 6$.

Figure 17: Visual comparison of additional clustering results using PCA with 2 components.

3 Predictive analysis

Addressing the problem of predicting whether a user is a genuine user or a bot one requires having an informative user profile as well as leveraging a classification model suitable for this downstream task. This section aims to illustrate the methodology employed for satisfying these two requirements and enabling competitive inference performance on never-before-seen users.

3.1 User profiling for classification

Although an user profile had already been defined during clustering, the resulting clusters did not partition users into 2 groups corresponding to bots and real users; this led us to reconsider the validity of this user profile for the current classification task and to figure out which features are most representative for the user's kind of nature.

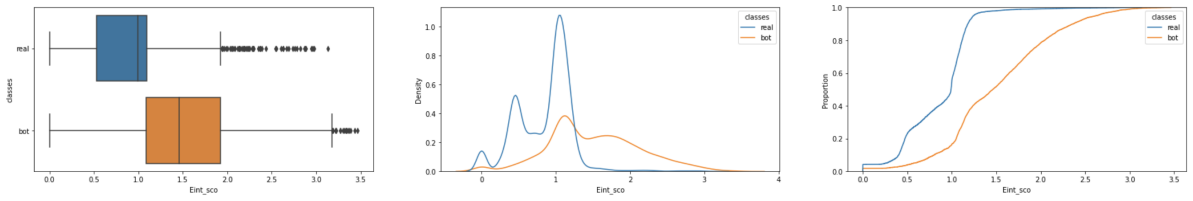
To ensure a new worthwhile user profile to be derived, all the attribute generated in Sec. 1.4 has been

consider an user profile's feature. Each feature was subjected to an aggressive outlier removing [5]. Any correlated features has been removed to prevent them from affecting estimators performance.
1271 samples and 10 features [6] were removed from the *Users* dataset for making the corresponding distributions as dense as possible. Once these new features were added to original *Users* dataset to express additional information about them, the obtained User profile account for 27 features - 23 new defined ones, and 4 already contained in the original *User* dataset.

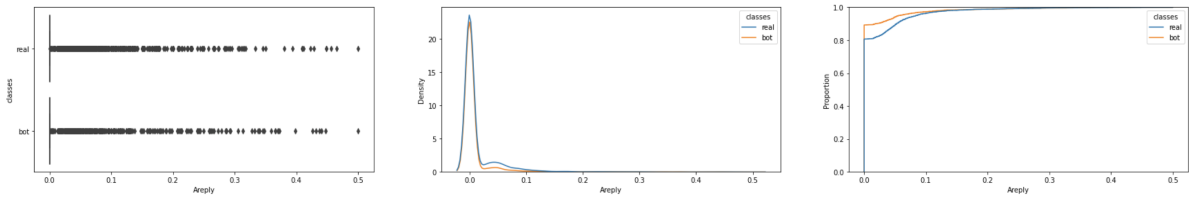
3.1.1 Features selection

Multiple features selection approaches were employed and combined together to find out whether these features incorporate meaningful knowledge to discriminate real users from bots; in other words, we aims at identify the most informative features for this classification purpose and discard any meaningless ones. For both numeric and categorical features the method used to assess their informativeness is based on 2 steps: an initial exploratory analysis aimed at capturing the differences between feature values for genuine and bot users through statistics, charts, and distributions, then some well-known feature selection techniques were exploited to confirms exploratory analysis results.

Exploratory analysis This exploratory analysis want to derive preliminary hypotheses regarding features discriminative capability by exploiting some graphical tools like Kernel Density Estimation (KDE), Empirical Cumulative Density Function (ECDF), and boxplot of the distributions. We assume that a feature is more informative the more different are the KDE, ECDF and the boxplots computed for bot and genuine users. Figure 18 shows the different behavior of an informative feature, `Eint_sco`, from a not informative one, `Areply`.



(a) KDE, ECDF and boxplots for `Eint_sco` feature.



(b) KDE, ECDF and boxplots `Areply` feature.

Figure 18: Exploratory analysis

`Eint_sco`, the entropy of interactivity score, seems to be an appealing feature since the the overall boxplot in Fig. 18a suggest us that the median value is higher for algorithmically-driven accounts rather than human-based ones. Moreover, the KDE shows the differences in the distribution between bot and human accounts as well as give us the information on different kind of interactivity score entropy among the bot category. While the KDE for the bot account goes up monotonically until it reaches two maxima around ... and ... the KDE for the genuine account shows a more spiked distribution: with 3 incremental peak

[5] Here, outliers were handled with a more extreme approach, since many machine learning estimators are highly sensitive to noisy data, and even marginal outliers could corrupt the suitability of the produced decision boundary.

[6] RlikeReply, Elike, Alike, Tretwee, pop_sco, Treply, Aments, Ayears, Amonths, itr_sco

respectively in 0 0 0 - the highest maximum is very much higher than the human one. It is becoming evident that human are typically more specialized different activities and not as diverse as bot users. On the other hand an example of a feature that is almost indistinguishable between the two categories could be the average number of reply a user receive for a tweet. Figure 18b shows how similar the two ECDF are, making it a weak information in telling apart the two categories.It is true, indeed, that in this group are coexisting different kinds of bot accounts.
Going further in investigating all other features behaviour, we came up expecting `Epop_psco`, `Eint_sco`, `Turls`, `Tments`, `Aurls`, `Thash`, `Ahash`, `Aretweet`, `Sreweet`, `Tlike`, `Achar`, `statuses_count` to be indicative features.
Categorical features, instead, are explored studying unique values occurrence for both human and not human based accounts; we ended up supposing `Fyear` is a discriminative feature while `Fday` is not.

Select best categorical features using ChiSquare and Mutual Information metrics Univariate feature selection is a common method that works by selecting the best features based on univariate statistical test. Firstly, we apply Ordinal Encoder on each categorical features columns for transforming them into numeric values [7] and we split the dataset in training set and test set. Then both chi-squared and mutual information metrics measure the dependency between features and target class. This score can be used to select the best features by fixing a customized threshold (or simply by keeping the first n-ranked ones).
It this way we have been able to weed out the features that are the most likely to be independent from the target class and therefore irrelevant for classification. Although the two metrics value are not in accordance in judging the most important categorical features (i.e.the features with the highest statistical dependency score with the target class), they agree in judging `Fday` a less informative ones. For this reason, we decided to keep all but `Fday` features (i.e. only `Fmonth`, `Fyear`, `lang` and `created_at(year)` was kept).

Select best numeric features based on Random Forest features importance Two common method for selecting the most relevant features take into account the feature importance score computed using a forest of tree.
In the first strategy a random forest classifier was fitted to the dataset and used to compute feature importance as the mean and standard deviation of accumulation of the impurity decrease within each tree. The highest is the mean decrease in impurity (MDI) the more important is the features; the problem is that impurity-based feature importance can be misleading for high cardinality features (i.e. many unique values).
An alternative strategy, the permutation feature importance, overcomes these limitations since it does not have a bias toward high-cardinality features but the computation for full permutation importance is more costly. Here, features are repeatedly shuffled and the model refitted to estimate the importance of it. Moreover, permutation importance is more likely than MDI to fully omit a feature.

With this two method we have obtained two different features ranking: although the relative importance vary, the same features are detected as most important using both strategy. However, in order to avoid overly reductive selection, we proceed to discard only those features that both method agree in judging less important. We came up with 12 most important features: `statuses_count`, `Achars`, `Schars`, `Aretwee`, `Sretwee`, `Sments`, `Tments`, `Adays`, `Echars`, `Epop_sco`, `Eint_sco` and `Eintervals`.

[7] One Hot Encoder (OHE) is not suitable for this type of analysis since he create a new column for each possible values: this means that by exploiting the OHE we obtain the informative score of each category instead of the informative score of the whole features.

3.2 Classification: models and results

To identify the optimal classifier for predicting user's target class, a comparative analysis of multiple models was carried out in order to study their behavior and figure out their weaknesses and strengths in this binary classification task.

K-Nearest Neighbor (KNN), *Gaussian Naive Bayes* (GNB), *Decision Tree* (DT), *Support Vector Machine* (SVM), *Feed Forward Neural Network* (NN) and *Multi-Layer Perceptron* (MLP) are all the considered estimators; anyway, an additional effort was addressed for discovering if better performance can be achieved through ensemble methods such as *Random Forest* (RF), *AdaBoost* (AB), *Bagging* (BAG) and *Voting classifier* (VC).

3.2.1 Impose balancing constraint on *Users* dataset

Before getting down in the classification task some statistics regarding the dataset dimensions and populations were collected. The dataset, containing a total amount of 10237 samples, appears to be skewed toward one of the two classes since 5580 samples are representing real users and 4657 bot users. Having one class with a population of 923 samples more than the other one could lead to biased classification model training. To avoid the different classes' populations from affecting the predictive model performance, a balancing constraint between classes was imposed on the dataset. The random sub-sampling technique was exploited for reducing the number of genuine users equal to the number of bot and thus obtaining a perfectly well-balanced dataset. After that, the dataset was finally shuffled to avoid introducing any bias related to samples ordering.

3.2.2 Evaluation schema

The same rigorous methods for model selection and assessment has been applied for each estimators.
The whole user dataset has been split in 2 parts: 66% of the dataset was used as *Design set* in the model selection phase and the other 33% was used as *Internal Test set* in model assessment phase.
The model selection was executed through k-fold cross-validation technique with k = 3; for each fold the Design set has been split into 4 part: 3/4 were used as training set (TR set), while the remaining 1/4 is used as validation set (VL set). A *Randomized Grid Search* was used to explore the hyper-parameters space and identify the most promising hyper-parameter configuration among the possible ones; the accuracy score was used for selecting the best configuration.
Then, model assessment is carried out through Hold-Out technique: the best model resulting from model selection was retrained on the whole *Design set* and its performance are evaluated on the basis of the results produced over the *Internal Test set* that was cut out from the whole dataset in the beginning.

3.2.3 Pre-processing impact on estimator performance

Studying how model behave as the dataset preprocessing change, allow us to identify the best preprocessing technique for each considered machine learning estimator.
It is worth noting that all categorical features were always preprocessed with *One Hot Encoder* (OHE); only the raw numerical feature vectors were transformed into a more suitable representation for estimators using several techniques. *Z-score/Standard Scaler* (SS), *MinMax Scaler* (MMS), *MaxAbs Scaler* (MAS), *Robust Scaler* (RS), *Quantile Transformation* (QT), *Power Transformation* (PT) and *Logarithmic transformation* (LT) are all the employed preprocessing procedures. [8]

[8] In this context, the *Ordinal Encoder* (OE) was not considered for categorical features because we know that it could introduce a bias between feature values due to its imposition of an order within the values. Similarly, no discretization technique was considered for numerical features, since partitioning continuous values into discrete intervals turned out to be too complex. Several dimensionality reduction techniques, such as *PCA* or *SVD*, were also discarded from the candidate

In addition, we would like to point out how an in-depth analysis of the preprocessing impact was carried out only for simpler models such as GNB, KNN, DT, NN, and SVM. We supposed that complex models such as MLP, or ensemble models such as RF, AB, BAG, or even VC, might benefit the most from the same preprocessing chosen for simpler models. This assumption may be too naïve and may have led us to a sub-optimal choice of preprocessing technique but was necessary in order not to make this analysis too time-consuming.

Identify most suitable performance for each estimators. With GNB, DT and NN no significant changes in performance were observed during model selection when varying the preprocessing applied on the dataset; looking at the results obtained from different model selection processes, it is possible to state that especially KNN and SVM are more sensitive to different types of preprocessing.

- Regarding GNB, with 0.805 $\pm$ 0.0003 and 0.804 $\pm$ 0.0009 of accuracy on the TR and VL sets, the performance seems to be insensitive to different types of preprocessing. However, we can still observe that smoothing parameter values lower than 1e-4 are only associated with LT and RS.
- The optimal DT use MMS preprocessing, Entropy as splitting criterion, looking at features while doing best split; the maximum depth of the tree is equal to 10, minimum number of samples for a split and a minimum number of samples for aleaf are equal to, respectively, 8 and17.
- The NN estimators achieved the highest performance when it is trained for 300 epochs with Adam optimizer on a dataset processed with PT technique; this best model need one hidden layer with 100 neurons and logistic activation function, constant learning rate initialized at 0.01 and regularization parameter equal to 0.01.
- Highest KNN performance were obtained using SS preprocessing, with number of neighbor equal to 23, distance weights and Manhattan distance as proximity metric. Euclidean distance and uniform weights were only chosen using respectively with preprocessing.
- SVM estimators achieved the highest performance when polynomial o RBF kernel were exploited with C value close to 1. The best SVM model came up using QT preprocessing, polynomial kernel, 0.9 as C value and scale gamma parameter.

However, for those models having a much larger parameter space (such as DT, SVM and NN) deriving how higher or lower hyper-parameters values are related to specific preprocessing turns out to be too difficult and would require a dedicated study.

Overall best preprocessing. Once the best preprocessing for each classifier has been identified, we want to figure out which technique would best fit most models. By identifying the overall best preprocessing procedure we will be able to perform a sensible comparative evaluation of the performance of considered models. We have followed a specific procedure for choosing the best preprocessing technique: given a certain preprocessing, for both training an validation, we take the average accuracy between the values achieved by each model, this step was repeated for each preprocessing and the results are shown in the Table 9. We can observe that in general in the training phase we get higher accuracy score using Standard Scaler, while in validation we get higher average accuracy using Quantile Transformation. Here, since it is common procedure to follow the ranking provided for VL set we decided to choose as the overall best preprocessing the Quantile Transformation.

preprocessing techniques since feature elimination stage has already led to a significant reduction in the number of features.

Accuracy	Model	SS	MMS	MAS	RS	QT	PT	LT
TR	GNB	**0.806**	0.805	0.805	0.805	**0.806**	0.805	**0.806**
	KNN	**1.000**	0.900	**1.000**	**1.000**	0.849	0.849	0.847
	DT	0.897	**0.917**	0.912	0.900	0.899	0.897	0.900
	SVM	0.830	0.814	0.814	0.824	**0.897**	0.880	0.842
	NN	**0.933**	0.896	0.888	0.925	0.907	0.924	0.901
VL	GNB	0.805	0.804	0.804	0.804	**0.806**	0.805	0.805
	KNN	**0.846**	0.839	0.802	0.791	0.819	0.817	0.817
	DT	0.866	**0.867**	0.865	0.865	0.865	0.865	0.864
	SVM	0.822	0.813	0.813	0.818	**0.881**	0.870	0.838
	NN	0.874	0.863	0.857	0.885	0.883	**0.887**	0.880
Mean TR accuracy		0.893	0.859	0.883	0.891	**0.871**	**0.871**	0.859
Mean VL accuracy		0.842	0.837	0.828	0.833	**0.851**	0.849	0.841

Table 9: Model selection performance

3.2.4 Comparative analysis between estimator performance

This section propose a comparative analysis of fitting and suitability capabilities of the models when they are trained, validated and tested on the dataset pre-processed with the previous winner technique: Quantile Transformation.[9] During model assessments we get some final insight about the performance of each model, and they were ranked based on the higher accuracy score achieved on TS set, as show in Table 10. Some additional tools was also employed for a more complete comparison:

- common evaluation metrics like Precision, Recall and F1-measure;
- confusion matrix to detect possible predominance of misclassified users;
- plot in 2-dimensions of the decision obtained after training (thanks to PCA);
- ROC curves and ROC auc score.

With regard to GNB we can state how this extremely simple model is not robust enough to compete with the others: it generates a decision boundary with an oval/elliptical shape that causes a high number of misclassified data and consequently an auc score of 0.78. All these factors help us conclude how GNB is way too simple for being a suitable classifier on this complex dataset.

		Design Set (retrain)			Internal Test Set		
Rank	Model	Acc	Pre	Rec	Acc	Pre	Rec
1°	MLP	0.911	0.908	0.932	0.917	0.9090	0.939
2°	VC	0.914	0.882	0.971	0.915	0.888	0.966
3°	AB	0.914	0.889	0.963	0.914	0.890	0.960
4°	RF	0.899	0.880	0.943	0.902	0.885	0.944
5°	BAG	0.901	0.877	0.950	0.902	0.881	0.949
6°	NN	0.900	0.893	0.928	0.902	0.896	0.928
7°	SVM	0.894	0.859	0.963	0.895	0.864	0.956
8°	DT	0.889	0.871	0.934	0.886	0.868	0.930
9°	KNN	0.829	0.781	0.955	0.830	0.780	0.959
10°	GNB	0.806	0.752	0.962	0.812	0.756	0.965

Table 10: Model selection and model assessment: performance evaluation

A similar behavior occurs when looking at the results of the KNNs; they too show how the model seems to be sub optimal for this task; in fact, storing TR instances to exploit them in the prediction phase lowers the generalization capabilities of the model for new samples from the TS ensemble leading to a high number of misclassified samples. With regard to KNNs, however, we must say that this model suffered the most from the final choice of preprocessing. In fact, we had pointed out in the previous section that this model performed much better in combination with a dataset preprocessing such as Standard Scaler or MinMax Scaler.

Although the DTs were also among the models most affected by the chosen preprocessing, they were still able to distinguish themselves by the high interpretability of decision rule results and the good final

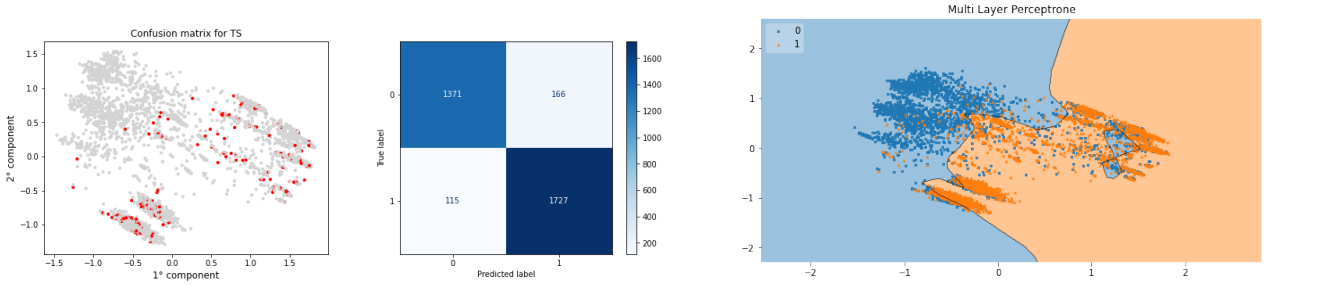
[9] all categorical features were of course always and only treated with the HOE

performance obtained capable of approaching that of SVM and NN. The most obvious difference lies in the decision boundary obtained: while with SVM and NN we find a smoother and more curvilinear function, the decision trees present an angular and more jagged decision surface.

Also for BAG, RF, and AB estimator ensembles, as well as for DTs, the decision surface exhibits extremely edgy and jagged geometric boundaries, in this case, however, the models learn better to discriminate a user between the 2 classes is the number of misclassified samples decreases significantly in both Design Set (i.e, in the final retrain) and TS set.

The 1st and 2nd places are filled by MLP and VC. The idea behind the Voting Classifier is to combine conceptually different machine learning classifiers and their predictions for majority rule voting to infer the final target class. Such a classifier can be useful for a set of equally well performing models in order to weep out their individual weaknesses. The jagged decision boundary, sometimes angular and sometimes more blunt, allows it to achieve some of the best inference performance proposed in this project. The decision boundary learned with MLP closely resembles that of NN with the difference of being a bit more accurate, thus allowing more competitive performance. With MLP the higher number of hidden layers allowed us to implicitly extract patterns from the input data at different levels of hierarchy, and exploit them for an efficient prediction.

Just for clarity we have shown in Figure the confusion matrix, decision boundary and roc curve of the best MLP model emerging from this analysis. Charts showing confusion matrices on TR and TS, decision boundary and roc curve of the other models are available on the notebook to attest to the conclusions exposed in this section.



(a) Misclassified points: scatter and confusion matrix on TS set.

(b) Decision boundary obtained by training MLP with TR points scatter.

Figure 19: Insight of Multi-Layer Perceptron performance

Looking at the confusion matrices for both TR and TS set it becomes clear that all estimators make most of the errors in classifying bot users that behave like human-based accounts. We suppose that this could be solved by exploiting some well-known class weighting strategies to give more importance (an higher weight) to the more critical class.

Overall conclusions for predictive analysis While exploring features informativeness we have observed that genuine user often have a more spiked distribution. This suggest us that a binary classification may be is too general to capture all the different specialized behaviors inside the genuine users category. In the future, further development could be directed toward finding subgroups of real users with the intention of carrying out more precise multi-class classification.

The results emerging from the models ranking and their comparative analysis were confirmed by the ROC curves and the associated AUC score. Since almost all of the models' performances are greater than or equal to 89% accuracy we can finally claim that we are able to optimally classify this 2 king of users.

At the end of this predictive analysis, we can state that competitive inference performance has been achieved: the accurate feature definition and feature selection techniques adopted have generated a meaningful user profile. Although this task can be considered particularly difficult because of the two classes are positioned in two contiguous groups with a high rate of overlapping, by studying in depth the preprocessing techniques of the dataset and hyper-parameter space of the model with a rigorous method it was possible to define all the required step useful for obtaining efficient binary classification.

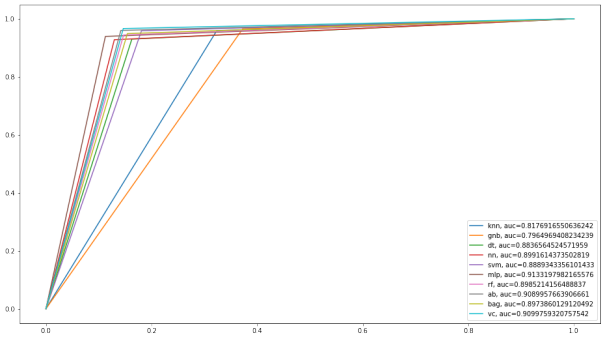


Figure 20: ROC curves on TS set.

4 Time series Analysis

The addressed task consist of creating and analyzing time series derived from *Tweets* dataset, in particular two mainly sub-task are deployed: Time series clustering analysis and Shapelet classification. Since the *Tweets* dataset contains several inconsistent values inside its attribute, we chose to take into account tweets preprocessed during data preparation task, more precisely, after data cleaning sub-task.

4.1 Generation of Time series

First of all we need to generate the time series (TS), in order to do that we selected all tweets in the dataset that were published in 2019. Then for each user, that has published in this period, we create a TS that contains, for every day, a **Success score**. The success score is defined as:

$$Success\_score = \frac{Acceptance}{Diffusion + 0.1}$$

where:

- $Acceptance = retweet\_count + reply\_count + favorite\_count.$
- $Diffusion = num\_hashtags + num\_urls + num\_mentions.$

The Acceptance is considered to understand how much the social network enjoys the shared content, while the Diffusion describes the effort put on promoting the tweet by the content creator. It may happen that a user on a certain day did not post any tweets, in this case the success score takes a default value, -1.

4.2 Segmentation

We thought that was worth to prepare two kind of TS for exploring and comparing them in the aforementioned mentioned sub-task.

For clustering TS we chose to divide them into weeks, mainly for facilitate the results analysis. In fact, when we have to study a general TS behavior a fine granularity is not very informative, while segmentation allows clustering techniques to work better thanks to dimensionality reduction.

For the classification purpose with shapelet, the full TS without "segmentation" reduction was exploited. We applied a *Moving Average technique* as preprocessing technique on the full TS in order to prevent them from being affected from noise.

We ended up with 2 set of TS: one containing roughly 365 days and the other one containing 52 weeks. Analyzing more in-depth the generated TS we discovered that there are no activity after 319-th day,

thus for the full TS were removed the last 46 days. Concerning the segmented TS the last 6 weeks were discarded.

4.3 Clustering

In the following subsection we are going to propose different kind of clustering configurations, in particular we have compared the *K-Means* with the Euclidean and Dynamic Time Wrapping (DTW) distance applying *Z-score* and *Min Max Scaler* normalization. In all these case, 3 is detected as the optimal number of clusters.

4.3.1 K-means with Z-score

The charts below show all TS associate to a particular cluster. Each TS is composed of 46 points (each value corresponding to the mean of week' success score). The color defines if the TS belongs to a real or to bot user (Blue = real user, Green = Bot), while the red line identifies cluster centroid. There is also a purple line which corresponds roughly to one month.

We ended up with two types of result displayed in Figure 22: on the left it was applied the K-means with the Euclidean distance, on the right with the Dynamical Time Wrapping.

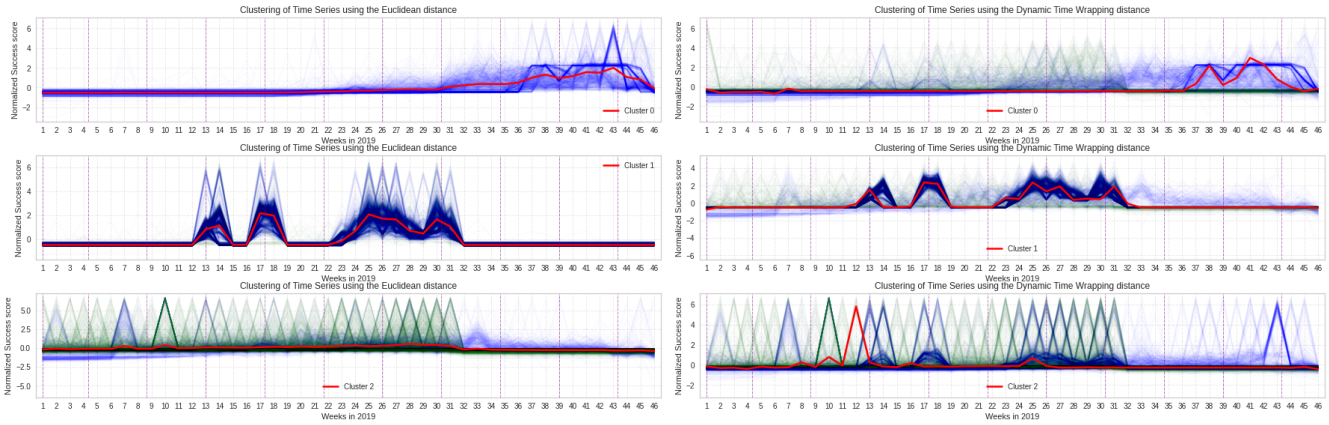


Figure 21: Time Series K-means for Euclidean (left) and DTW (right) with Z-score

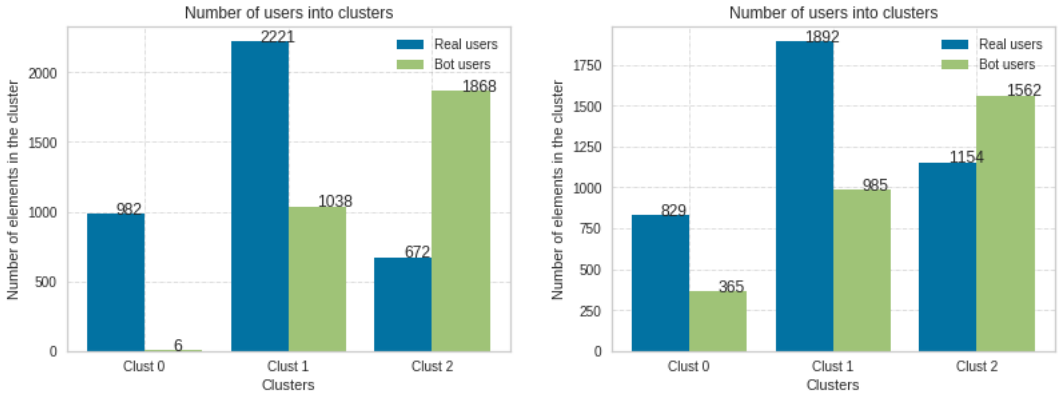


Figure 22: Number of users into cluster for Euclidean (left) and DTW (right)

Cluster interpretation and characterization Concerning **Euclidean distance** results we can observe that cluster 0 is mostly populated by a real user. In this first group we have major activity, approximately, from September to November and very low activity in the rest of the year. The most relevant tweets have been published in October. We can basically state that his cluster has captured the real users that have published relevant tweets during the autumn-winter seasons. In Cluster 1, there is a mix of real and bot users, even if there is a predominance of real users. Different from the previous chart, the cluster covers the spring-summer seasons. This time we have no constancy, in fact, we can notice that there is a short period with low or null activity, this means that any of users have published relevant tweets. In this last group, Cluster 2, we have a bot predominance. Different from the previous charts we have no correlation with the seasons because there are some "spikes" that lays on all weeks. The major activity is given for the first half of the year. The bot users have the particularity of having the patterns repeating.

Regarding to **Dynamic time wrapping distance**, more or less we obtained the same result. However we can summarize the differences in the following observations:

- DTW (6.858) has low interita than Euclidean (24.773) this means the quality of the clustering is better.
- The Euclidean distance perform result cleaner than DTW, we can see there there are more noise in the DTW.
- Euclidean distance was able to separate better the bot and real users that DTW.

Conclusion The algorithm found tree different kind of behavior where the main difference is given from the activity during the seasons. The cluster 0 and 1 (in the euclidean distance) has captured the different behavior of human-based and algorithm-driven accounts.

4.3.2 K-means with MinMax

With *Min Max* normalization the obtained result is almost the same, however the seasonality behavior is no more evident as with *Z-score*. Concerning to intertia also in this way the DTW produced better cluster but it was not able to separate well the real users from and bot ones.

4.4 Shapelet classification

In this phase we learned the shapelet, defined as "subsequences that are in some sense maximally representative of a class". Informally, if we assume a binary classification setting, a shapelet is discriminating if it is present in most series of one class and absent in series of the other class. In order to achieve competitive performance we tried out several configurations using different algorithms, different normalization procedure and different kind of TS.

LearningShapelet provided by tslearn:

1. weekly TS with Z-score
2. weekly TS with MinMax
3. full TS with Z-score
4. full TS with MinMax

LearningShapelet provided by pyts

5. weekly TS with Z-score
6. weekly TS with MinMax

4.4.1 LearningShapelet provided by tslearn and pyts

We tested various configurations using the classical grid search. The dataset was split into Training (80%) and Test(20%) after applied the shuffling on its samples. Some parameters as Learning rate or batch_size

are fixed to reduce the total amount of test and reduce the time cost of these operations.

Considerations The best accuracy score achieved across all configurations is equal to **0.7746**, obtained from the full TS with z-score and the algorithm provided by tslearn (table C).

We can point out basically two things: the first one is that for this kind of task the z-score perform better respect to minmax indeed we can see that the score achieved in z-score are higher with respect to others normalization technique.

The second is that the best parameters for weekly and full TS are exactly the same, but a full TS is more suitable for this classification task. Concerning the values of parameters we can notice that the higher scores are associated mainly to a few and long shapelet with the low number of epochs. We also tried to learn the shapelet with a different library. Since second library (pyts) is much slower with respect to the previous one, the number of configurations tried was bounded. In any case the result achieved is more or less the same, in fact as already said in the previous grid search, the z-score reach better result with respect to minmax.

Itr	Shap.len	N.Shap.	Score
200	0.5	2	0.6708
50	0.5	2	0.6686
500	0.5	2	0.6627
50	0.05	2	0.6524
100	0.5	2	0.6509
500	0.2	3	0.6502
200	0.05	4	0.6487
50	0.1	1	0.6480

(a) weekly TS with z-score (tslearn)

Itr	Shap.len	N.Shap.	Score
50	0.5	1	0.6627
50	0.4	1	0.6067
50	0.5	2	0.6060
50	0.05	1	0.6016
200	0.05	1	0.6001
50	0.05	2	0.5972
200	0.05	2	0.5949
50	0.1	4	0.5942

(b) weekly TS with minmax (tslearn)

Itr	Shap.len	N.Shap.	Score
200	0.5	2	0.7764
100	0.5	2	0.7356
200	0.3	3	0.7319
200	0.4	2	0.7290
200	0.2	4	0.7091
50	0.5	3	0.7002
200	0.05	4	0.6966
50	0.2	4	0.6944

(c) full TS with z-score (tslearn)

Itr	Shap.len	N.Shap.	Score
100	0.05	4	0.6207
200	0.2	3	0.6163
50	0.05	4	0.6156
50	0.05	3	0.6141
50	0.1	1	0.6134
200	0.2	2	0.6126
100	0.05	3	0.6119
200	0.05	1	0.6119

(d) full TS with mimax (tslearn)

Itr	Shap.len	N.Shap.	Score
100	0.2	0.3	0.6958
100	0.2	0.2	0.6936
100	0.1	0.3	0.6877
200	0.1	0.3	0.6877
100	0.1	0.2	0.6671
200	0.1	0.2	0.6671
100	0.3	0.1	0.6627
100	0.2	0.1	0.6575

(e) weekly TS with z-score (tslearn)

Itr	Shap.len	N.Shap.	Score
100	0.1	0.3	0.6458
200	0.1	0.3	0.6458
500	0.1	0.3	0.6458
100	0.1	0.2	0.6200
200	0.1	0.2	0.6200
500	0.1	0.2	0.6200
100	0.1	0.1	0.5655
100	0.2	0.1	0.5655

(f) weekly TS with minmax (tslearn)

Table 11: Overall configurations tried and their results

4.5 Learned shapelets

The algorithm found in total 8 shapelets, the first 5 have a length equal to 158 days, the last 3 have a length equal to 316 days and cover the entire length of all TS. The longest shapelets seems to be more unstable respect to the smaller one. We can notice that all shapelet are very different from each other and this leads to cover possible aspect/characteristic that identify wheter the TS is associated to a bot or real user. In the chart below we can see the shapelet location in respect to an example of TS (for each TS, the shapelet assumes different position).

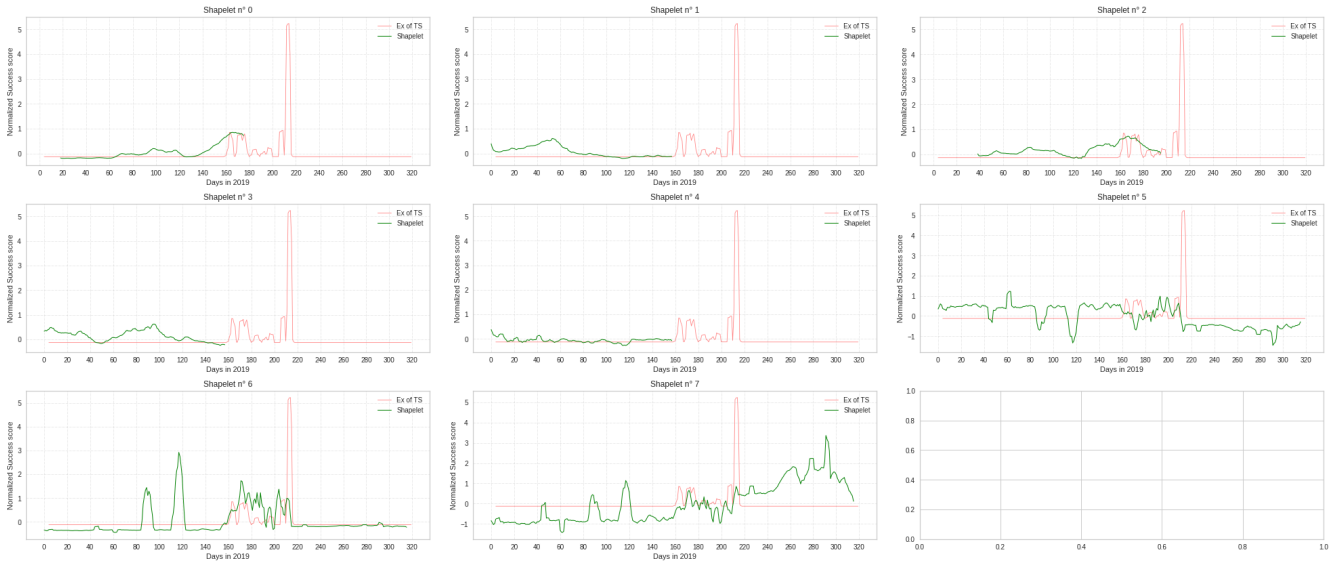


Figure 23: Shapelets learned

5 Final conclusions

By thoroughly analyzing the characteristics of the users in the data preparation and understanding phase, we were able to derive some data-insight that allowed us to effectively perform user clustering and classification. Thanks to these two tasks we were able to derive implicitly, in the case of classification, and explicitly, during clustering, new patterns within our data. Most future developments could be directed toward improving the quality of the timeseries in order to achieve higher performance in extracting shapelets.

References

[1] Xue Liu, Yong Ding, Hao Tang, and Feng Xiao. A data mining-based framework for the identification of daily electricity usage patterns and anomaly detection in building electricity consumption data. *Energy and Buildings*, 231:110601, 2021.