Data Mining

Team 8

Alberto Anzellotti, Silvio Martinico, Nicola Pitzalis

October 16, 2022

Abstract

In this project, we look at a couple of datasets from Twitter and perform Data Mining tasks on it. One dataset contains information about users, while the other contains information about tweets. The analysis proceeds in four main steps: Data Preprocessing (understanding and preparation), Clustering analysis, Bot Classification and Time Series Analysis.

Contents

1 Data Preprocessing

In order to work on this datasets, we first need to understand it's content, identify its issues and cleaning it from errors and bad data.

Let's begin by understanding the properties and nuances of our datasets.

1.1 Users

1.1.1 Data Understanding

The dataset `users.csv` is the smallest one [`640kb`] and provides the following features about Twitter users:

- `id`: the ID of the user
- `name`: the name of the user
- `lang`: the language of the account
- `bot`: 1=bot, 0=proper user
- `created_at`: account signup date
- `statuses_count`: number of tweets of the user

The first thing we did was to assess if the field `id` is a uniquely identifying field for the given dataset (if each value corresponds to exactly one record or not). In order to do that, we just compared the number of records in the dataset with the number of unique values for the `id` field: the two numbers are equal, indicating that the field is uniquely identifying. Hence, we chose to Index the pandas dataframe with the column `id`.

We are not quite ready to start with the final preparation stage yet, in fact we still need to gain a better understanding of the data and solve some easy-to-fix issues in order to make the data more manageable.

The next field we chose to analyze is `name`. Having an ID which is a primary key, we don't need the names to be unique, neither we have particular requests for them; therefore, after inspecting the field properties we chose to drop it, since any possible meaningful analysis on it would require tools and processing techniques that are out of the scope of this project.

Up next we analyzed the attribute `lang`. Using the command `value_counts` on this field, we could state the following observations:

1. There are two users with missing language value
2. There are some invalid ISO format value
3. There can be multiple variants for each language (e.g.: `en`, `en-gb`, `en-GB`)

The third problem was fixed by associating every variant of a language to its main language, while for the observations 1 and 2 we chose to set a dummy string "wrl". This choice is supported by the fact that the amount of records matching such conditions is very small and the operation will have a negligible effect on the dataset.

After these small adjustments, we gave a look at the languages distribution via a bar plot, where the scale of the y axis was set to logarithmic for a better visualization. In Figure 1, we can see (as we saw from the description of `lang`) that English is the most frequent language, followed by Italian and Spanish, respectively one and two orders of magnitude below. We also observed that,

approximately half of the languages are negligible (below the $0,1\%$ of the users). Now, looking at the histogram in Figure 2 for the `bot` feature, we chose to also analyze the languages distribution for each category separately, surprisingly we saw that while almost all the real users use the English language, the bots feature a non-negligible percentage of Italian ($\approx 15\%$).

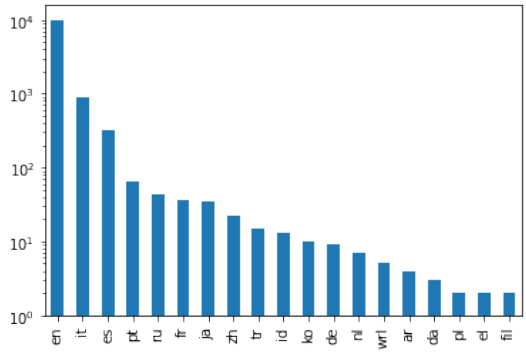


Figure 1: Users per language

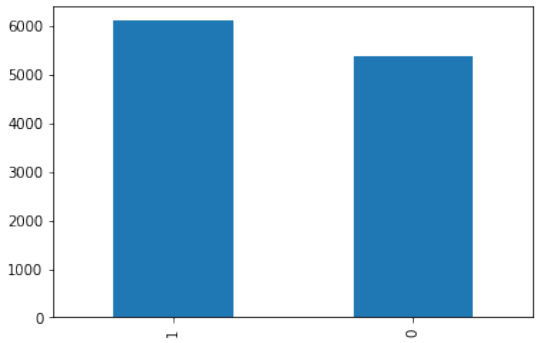


Figure 2: Bots vs non-bot users

The column `bot` has no problems whatsoever and, from the bar plot in Figure 2, we can notice that we have almost an equal number of bot and non-bot users.

For the `statuses_count` field, we observed that there is a small percentage of missing values ($\approx 3.4\%$). So far, we've decided to postpone the nan-handling.

One interesting thing is that, counting the number of statuses for both bots and non-bot users, it turned out that the number of statuses posted by real users is approximately 8 times larger compared to the bots'. We thought of two theories to explain this result: bots' main activity isn't sharing tons of statuses; bots do share tons of statuses, but in short bursts, or for short periods of time getting banned or deleted soon after.

Regarding the `created_at` field, we observed that it's a `datetime` object and so we analyzed it exploiting this property.

First, we looked at the number of accounts created each year.

What the bar chart in Figure 3 highlights is that we have a kind of bimodal distribution, with two peaks in 2017 and 2019. Clearly, this plot does not give us enough information, so we want to increase the granularity. Hence, for each year, we looked at the number of enrollments day by day.

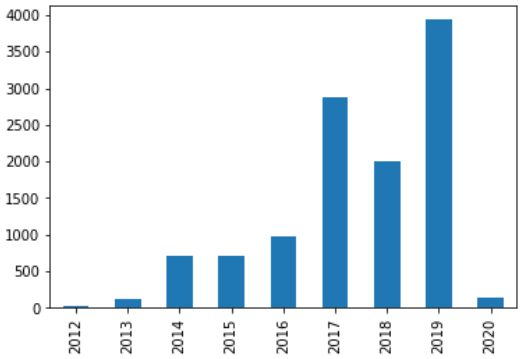


Figure 3: Enrollments per year

In Figure 4 we can observe a peculiar behaviour. We have a small number of accounts created per day but there are some, apparently anomalous, peaks. It is very unlikely for a randomly chosen users dataset to have all the sign-ups concentrated in just a few days, see in particular the years 2017 and 2018. We also looked at the same plot but by month dividing users by `bot` category (Figure 5). We can observe that the anomalous peaks are related to the registration of bots, as we could imagine.

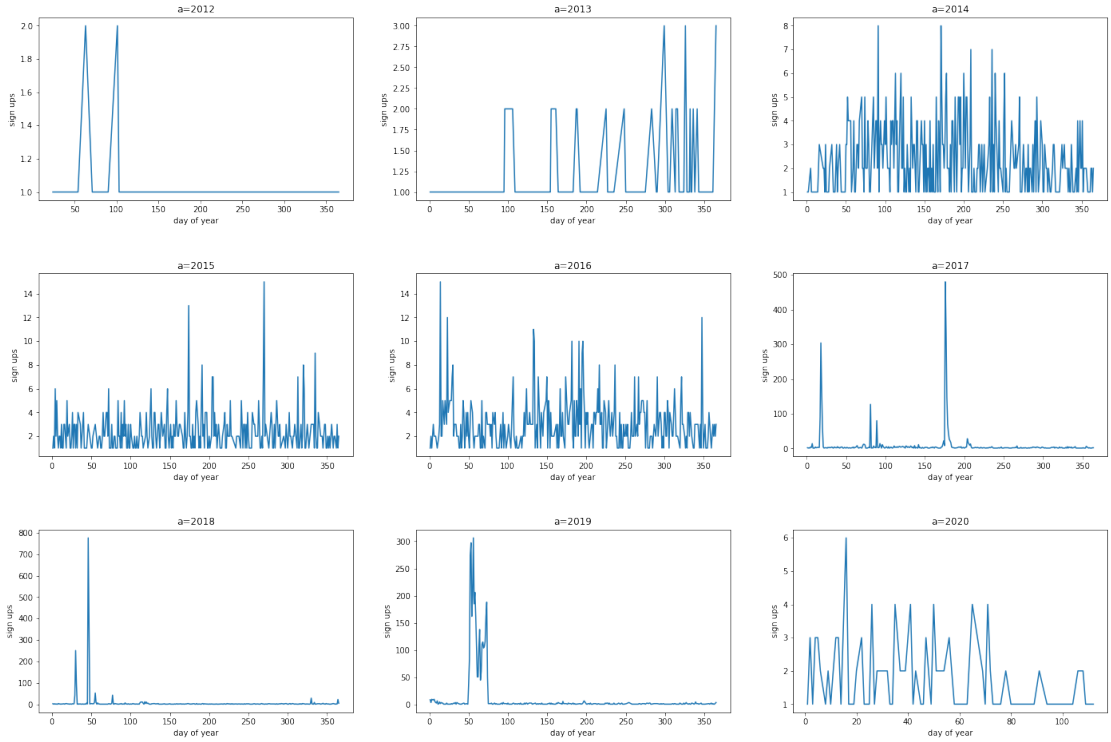


Figure 4: Number of sign-up per day, divided by year

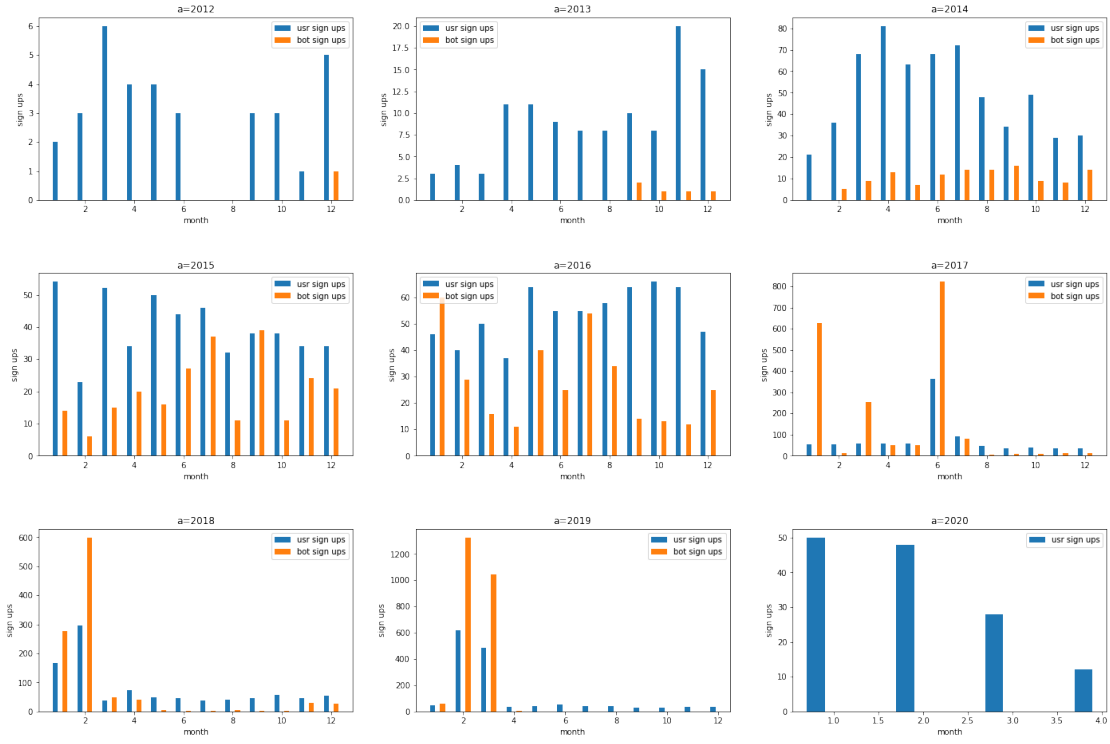


Figure 5: Number of sign-up per day, divided by month, flagged by bot/non-bot

1.1.2 Data cleaning

We found it interesting to check whether there is a correlation between the user's lifespan (metric given by the time, in days, from the creation of the account to the moment of data crawling) and the `statuses_count`. However, we observed no significant correlation ($\approx 0.287\%$); so, we tried to look at this correlation divided by `bot` category, but the result has not changed and the correlations for the two categories is almost the same. At the same time, we looked at different statistics for the field `statuses_count`, like mean, std, median, etc. Also in this case, we observed that the two categories of users does not differ by much. Anyway, we noticed some inexplicably high values, so we decided to look at a boxplot for `statuses_count` in order to detect possible outliers.

Looking at Figure 6, considering also that the scale on the y-axis is logarithmic, we can see too many values outside the maximum limit, and it does seem suspicious, since they're not too big to be considered real outliers. So, we looked at users with `statuses_count` over the 90-th percentile, but, even looking at these rows, we cannot be sure any of them represents real outliers. For instance, 10000 statuses in 3 years of activity might be legit.

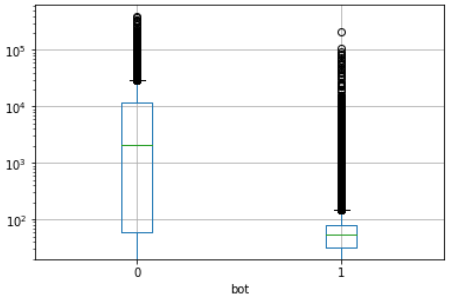


Figure 6: Number of statuses

Clearly, we need something more specific in order to detect outliers. Thus, we defined the metric `statuses_per_day`, which is given by the ratio between `statuses_count` and `usr_lifespan`. Looking at this new metric both for bot and non-bot users, we noticed that bot users have not too many statuses per day (the max is 173, which is not impossible for a bot), while for real users, we noticed some strange things: we found out that the min $= -133$ and max $= 3812$, so we have two problems: the minimum is negative and the maximum is too big.

Analyzing these values, we found that there was just one user with a negative `statuses_per_day`; this might be the value of statuses made by the last user registered (since we took the last date present on the dataset as the crawling date). Checking twice, it turned out that we were right and discovered that this user had 133 statuses in a day, so we can also consider it an outlier. Thus, we decided to drop it.

Regarding the maximum, looking at statistics, we observed that the 75-th percentile of the field `statuses_per_day` corresponds, approximately, to a tweet every hour and a half, which is an high number but not totally impossible; we can not drop users over the 75-th percentile because there are too many of them. Also looking at the points over 90-th percentile, most of them were legit. We found a single value which was an obvious outlier and thus, get rid of it. We've closed the users' dataset preprocessing stage with just 4 record less than the original dataset, going from 11.508 to 11.504.

1.1.2.1 NaNs handling

We want to handle the NaN values in the dataset. Looking at them, we noticed that the only field containing NaN values is `statuses_count`, which had 399 NaNs. What we did was to repair those values, replacing them with the value of the users' lifespans multiplied by the median of `statuses_count` made per day based on `bot` category.

So, we replaced all the NaN values with the number of statuses those users would have made based on the `statuses_per_day`'s median. Note that we have not considered all the outliers values since we removed them from the dataframe.

1.2 Tweets

The dataset `tweets.csv` is the largest of the two [`1.7gb`] containing the following information about tweets:

- `id`: the ID of the tweet
- `user_id`: user ID of the author
- `retweet_count`: number of retweets
- `reply_count`: number of replies
- `favorite_count`: number of likes
- `num_hashtags`: number of hashtags contained
- `num_urls`: number of urls contained
- `num_mentions`: number of mentions contained
- `created_at`: tweet's creation date
- `text`: text of the tweet

We suddenly noticed that, unlike users dataset, we have a lot of null values.

We started making some general assumptions: we saw that there were few null-id records, which also carried close-to-none information, so we deleted them. We also dropped records with null `user_id` (because we needed them to match users information) and exactly duplicated records.

Another thing we immediately wanted to do was to enforce types that we expected from each column, so we replaced not-numerical values in the numerical columns with NaNs and we forced the column `created_at` to be a datetime object.

The column `id` was deleted since its values were not unique and it did not carry useful information.

Even though we checked for duplicated records, there is a subset of columns which has to be unique: the pair (`user_id`, `created_at`); in fact, a user can not make two tweets at the same exact time. For this reason, we discovered that too many records are not unique in this sense (about 2.8 millions, which means 1.4 millions of records to delete, more than 10% of the dataset). We found out that dropping all records beholding at least one NaN-value would also remove any duplicated ones. However, this would mean about 1.6 millions records lost. So, instead of deleting every "NaN-valued" record, we decided to adopt a finer strategy: define a new metric, called `information_score`, which measures how much information a record carries in the numeric fields (it counts the number of NaN numerical fields, from 0 to 6). Thanks to this rating, we kept the most informative records out of all the duplicated ones, also dropping the less informative.

Finally, we deleted the records which have not a match in the user dataset. In fact, we want to study the users, so a tweet is not interesting to us if it has no associated user.

1.2.1 Data Understanding

After a pretty rough initial cleaning phase, we can start the understanding phase.

1.2.1.1 Correlation

First, we looked at the correlation between numerical attributes. What did come out from the correlation matrix in Figure 7, is that there is absolutely no correlation between any pair of attributes. This is quite strange, we will make a deeper analysis.

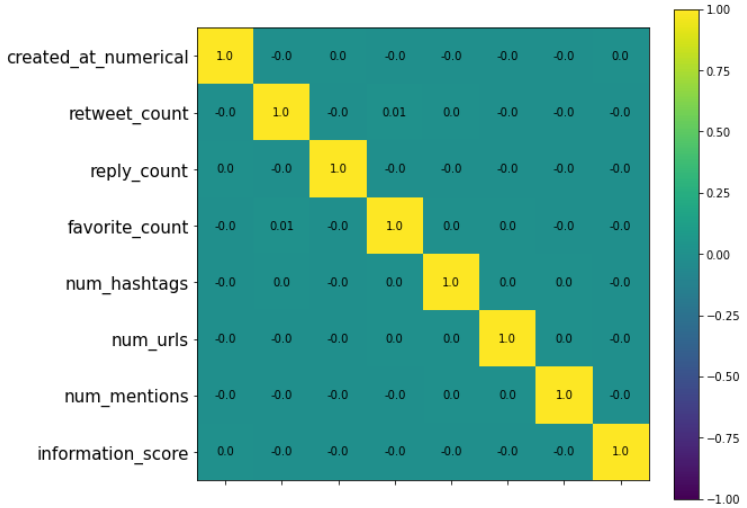


Figure 7: Correlation matrix

1.2.1.2 Date

From the bar plot in Figure 8, we can see something very bizarre: there are a lot of tweets posted both in the far past (decades before the creation of Twitter) and in the far future. This is clearly impossible, so it is for sure the result of some error.

Almost 1% of the tweets have a date which does not respect the date boundaries. It seems like these tweets have been shifted by 60 years in the past and 20 years in the future. This clearly is a strong assumption; however, we can see how the distribution of the past and future tweets resembles that of the proper tweets. For this reason, we actually brought them to the present without changing the data distribution.

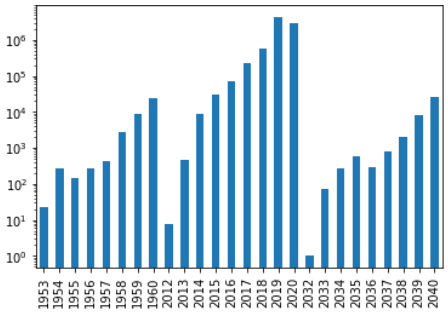


Figure 8: Number of tweets per year

1.2.1.3 Numerical fields

At this point, we checked some basic conditions for the numerical fields.

Starting from the fields concerning tweets' text (`num_mentions`, `num_urls` and `num_hashtags`), knowing that the maximum length of a tweet is 280 characters (140 before November 2017), we checked that each tweet had a reasonable number of urls, mentions or hashtags. We used 5 as minimum length of a url or mention and 2 for hashtags and we considered also the spaces between them. The records which did not respect these bound were few (about a hundred, i.e. $\approx 0.001\%$ of the dataset), so we decided to drop them without problems.

In the same way, we dropped the tweets with too many characters in the field `text` (because they were few, so no statistical significance).

We did something similar with the fields concerning the interactions of a tweet (`reply_count`, `favorite_count` and `retweet_count`): for retweets and favorites we checked that these two counts did not exceed the bound given by the most retweeted/liked tweets on Twitter (respectively 4

millions of retweets and 7.1 millions of likes). Since we have not found such information also for the replies, we analyzed this field more in detail. First, thanks to a boxplot, we saw that there were some (more than a thousand) clearly impossible values (more than 10^{13} replies). Even excluding these values and looking closer to zero, couldn't help us identify the box (due to the fact that all the percentile values were zero). So, we analyzed the values using the pandas function `value_counts()` and we suddenly noticed that more than the 99.8% of the values were 0: the field contains no information. Thus, we checked whether we altered important values during the previous phases and it turned out that this was not the case, so we decided to drop the column.

We checked if the other two interactions columns had the same problem, but they were fine. Nonetheless these two attributes have very long tails, we didn't modified them because there was no valid reason to cut such tails.

At this point, we further investigated the correlation between attributes and we noticed that, cutting out some outliers and restricting to non-zero values, the attributes `favorite_count` and `retweet_count` have a good correlation ($\approx 0.75\%$). All other attributes are not correlated as we observed in the correlation matrix 7. Another interesting (but bizarre) thing we noticed was that less than the 1% of tweets with more than 1000 retweets have at least one favorite. However, we don't have any clues that suggest a possible fix.

1.2.2 NaNs handling

The only field left with NaN values at this point was `Text`. Since the records with NaN text were just $\approx 0.39\%$ of the dataset, we decided to drop them.

1.3 Merging the datasets

After the exploration and the preprocessing phases, we analyzed various metrics regarding the tweets grouped by user and then we merged the two datasets adding these new metrics in order to obtain a more complete user profile for the next phases of the analysis.

We started counting the number of tweets for each user. In Figure 9 and Figure 10 we can see that the distribution is, as one can expect, decreasing and strongly biased towards zero (as highlighted by the median in the boxplot).

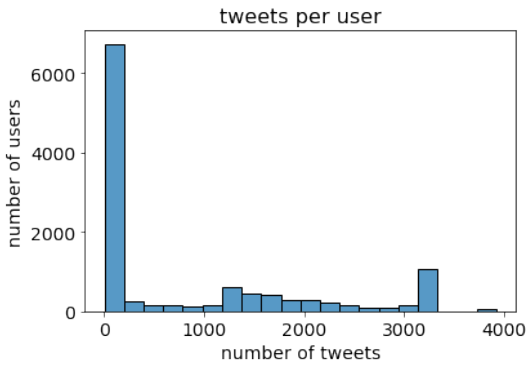


Figure 9: Number of tweets per user

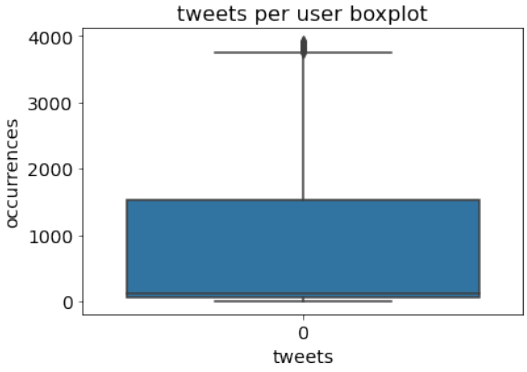


Figure 10: Boxplot of tweets per user

In the same way we looked at each user's number of retweets and favorites. In Figure 11 and Figure 12 we can see the distribution of the two metrics. In the process, we noticed some users that looked like outliers; though their values were not absurd, so we kept them based on the fact that it is possible for a small subset of users to have lots of interactions.

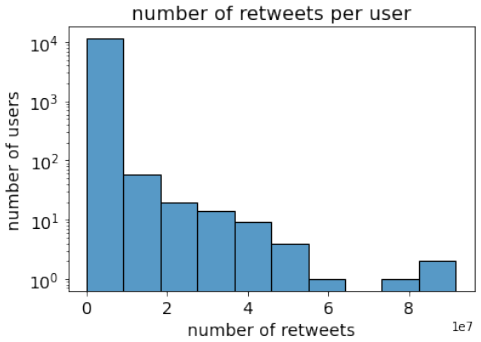


Figure 11: Number of retweets per user

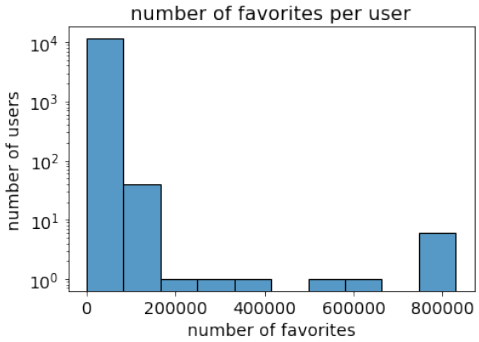


Figure 12: Number of favorites per user

After that, we analyzed few statistics on `retweet_count` and `favorite_count` for the tweets of each user. In particular, we focused on mean and std, for each user, of these two fields.

We had a look at the relations between number of tweets, favorites and retweets of the users. Grouping by user these fields, highlighted some outliers which don't allows us to fully understand these relations. Two predominant patterns emerged:

- Users with more retweets than favorites (most of favorites are close to zero)
- Users with more favorites than retweets (most of retweets are close to zero)

For this reason, even when these two attributes are grouped by user, there isn't a strong correlation.

The next thing we did was to divide the day into four parts: morning, afternoon, evening and night. Then, we grouped again the tweets by user and looked at the distribution in the various moments of the day. There is not a remarkable difference between the four moments. Finally, in Figure 13, we can see, for each year, the histogram representing the number of tweets of users in the four moments of the day. We noticed that every year seems to be pretty similar to each other. Afternoon is the most active time, while morning is the least active one. Furthermore, the distribution per user remarks that of not-grouped tweets.

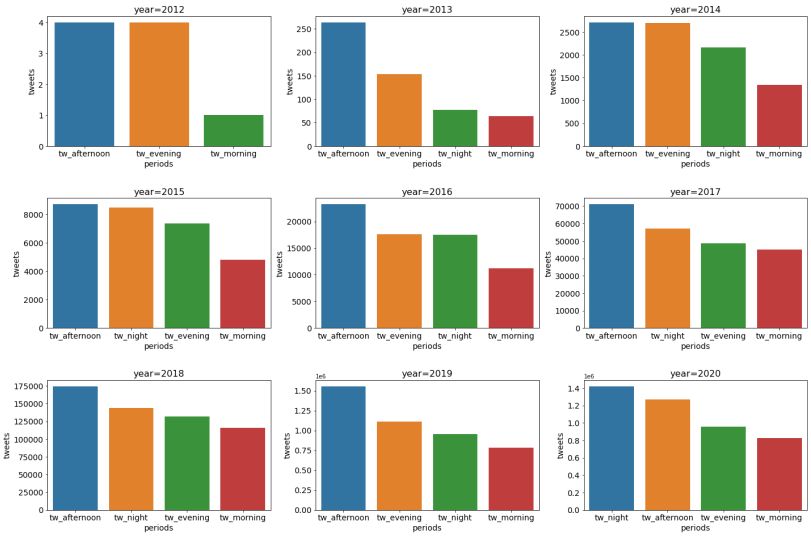


Figure 13: Histograms of users' number of tweets for each year, divided by moment of the day

Other metrics we added are: number of special character in tweets per user, activity period (time, in days, from the sign-up to the last tweet of a user), average monthly number of tweets per user (and we discovered that, on average, a user have between 0 and 5 tweets per month), number of tweets in the first month per user and some entropies for each user (entropy of `retweet_count`, entropy of `favorite_count`, entropy of `num_hashtags`, etc).

After defining all of these metrics and merging the datasets, we looked at the distributions of these metrics through histograms and boxplots and we checked if there were correlations. The most relevant correlations we have found were that between the number of tweets of a user in the first month and the average monthly number of tweets of a user ($\approx 0.6\%$) and that between the number of tweets of a user and its activity period ($\approx 0.63\%$). We show the scatterplots which highlight these correlations in Figure 14 and in Figure 15.

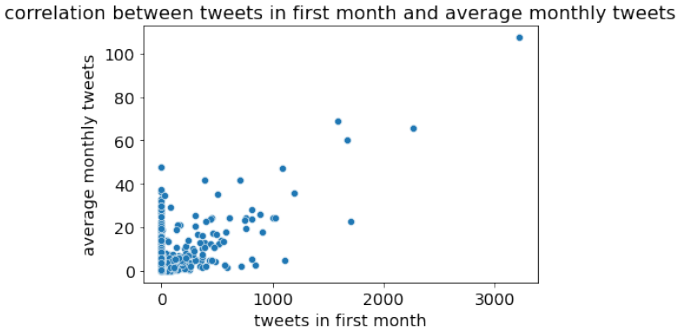


Figure 14

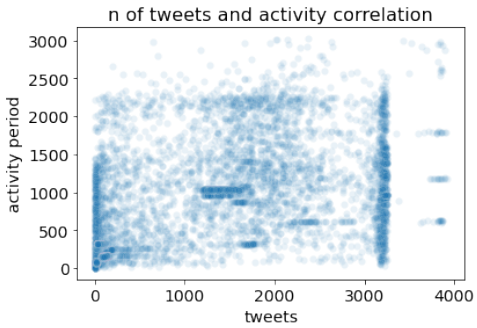


Figure 15

At the end of this phase, we had a new dataset with a total of 46 columns, given by the original columns of the users dataset plus the features we defined putting together various metrics from both the original datasets. In the next phase we will select only some of these features in order to reduce the dimensionality and facilitate clustering and classification.

2 Clustering Analysis

Three main clustering algorithms were investigated: the centroids-based algorithm *K-means* (and its variant *X*-means), the density-based clustering algorithm *DBSCAN* and the *Hierarchical Clustering*.

2.1 K-means

First of all, we selected a subset of features.

2.1.1 Features selection

In order to choose which numerical features to use for running *K*-means, we mainly looked at two things: the entropy of the columns (we want to maximize the information a column gives us) and the correlations between them (we want to avoid highly correlated columns).

In Figure 16 we can see the columns sorted by their entropy. We cut the tail, removing the last 11 columns ($\approx 25\%$ of the columns). This way, we also removed categorical columns not suited for K-means.

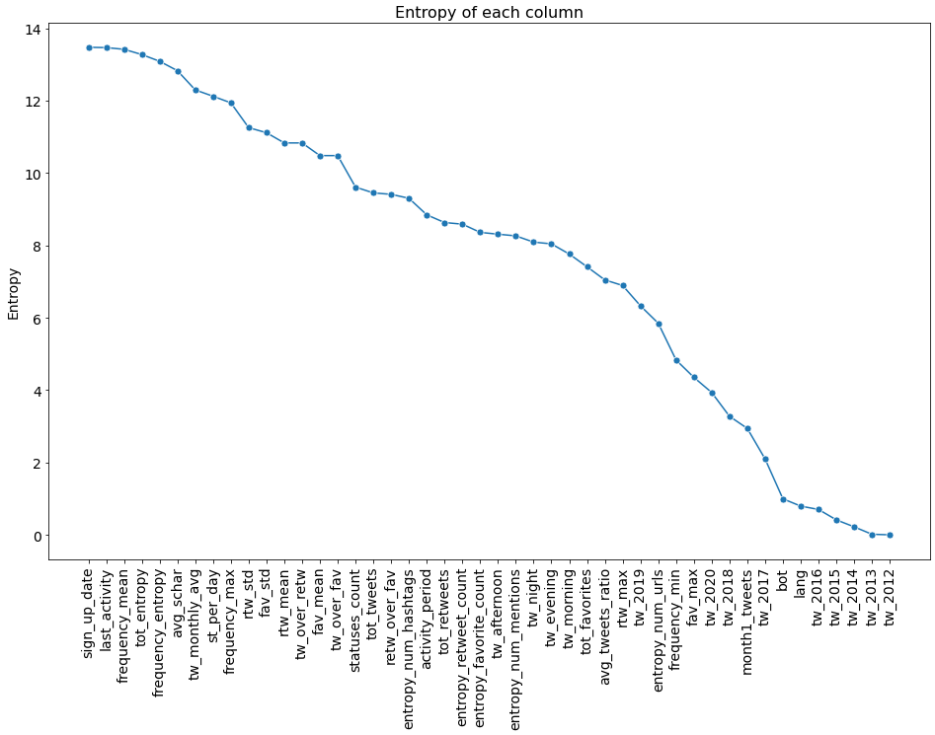


Figure 16: Entropies of columns sorted in decreasing order

The remaining columns were selected by keeping, between the most correlated ones, only those with the biggest entropy. At the end of this process, we were left with 20 columns (less then half of the original 46 columns).

2.1.2 K-means with Standard Scaler

Many columns are very skewed, for this reason we standardized them via the Standard Scaler.

First, we ran K-means with a range of K values from 2 to 15, so we could analyze the trend of *SSE* (Figure 17), *silhouette* (Figure 18) and *separation* (Figure 19).

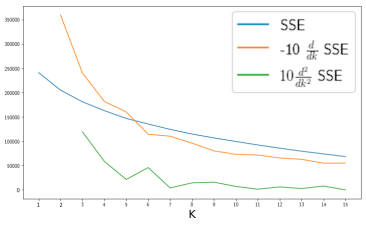


Figure 17: SSE and deriv.

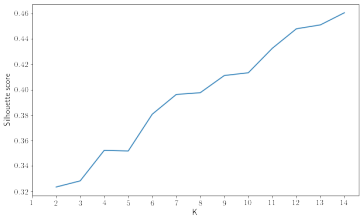


Figure 18: Silhouettes

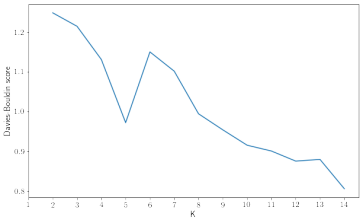


Figure 19: Davies-Bouldin

These plots don't highlight particularly interesting K values. The SSE curve is almost flat, no elbow-points. We can instead observe some bad values to certainly discard, e.g. $K = 5$ (the silhouette decrease with respect to the previous step) and $K = 6$ (the DB score increases significantly).

A good value seemed to be $K = 9$, because the silhouette has a good increment and it's the first value where the DB score goes below the minimum reached in $K = 5$.

Looking at the cluster we noticed that some of them were really small, so we decided to assign the points of these small clusters to the cluster of the nearest centroid. This resulted in the elimination of 3 clusters.

Looking at Figure 20, we selected the best feature for visualizing the data. Below, we can see two of the most relevant plots: in Figure 21 a 2D plot, while in Figure 22 a 3D plot. We can see a quite good separation. However, clusters are not globular and well spaced out due to the nature of the distributions of our data.

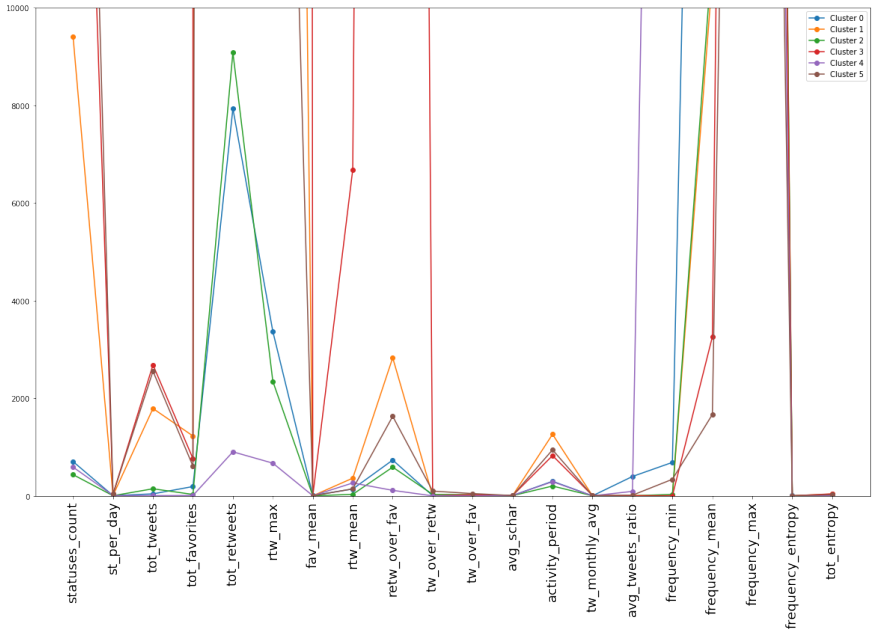


Figure 20: Centroids

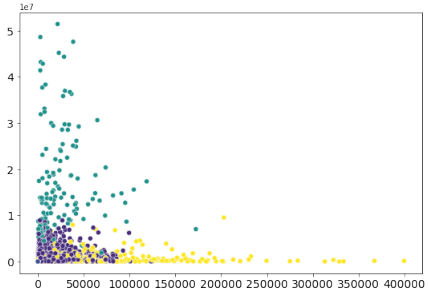


Figure 21: X = statuses count, Y = tot retweets

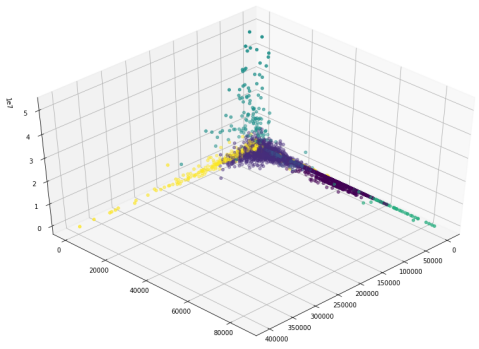


Figure 22: X = statuses count, Y = frequency mean, Z = tot retweets

After that, we tried two other K values: 14 and 4. For $K = 14$ the results were almost the same, since we had a lot of small clusters and the meaningful clusters were very similar to those found for $K = 9$. For $K = 4$ we found 3 meaningful clusters which, graphically, appear again similar to the clusters found with $K = 9$, even if some of them were clearly merged.

2.1.3 K-means with Min-Max Scaler

Interesting and quite different results were obtained changing the kind of preprocessing of the data: in this case the data were normalized using the Min-Max Scaler.

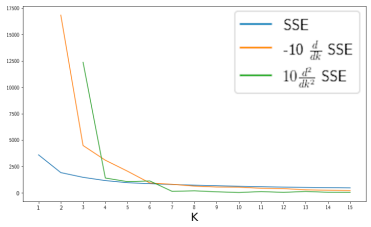


Figure 23: SSE

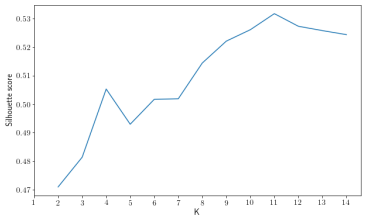


Figure 24: Silhouette

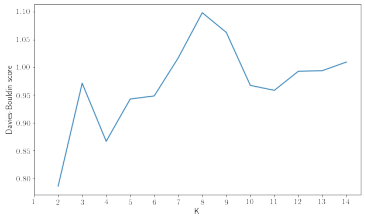


Figure 25: Davies-Bouldin

This time we have undoubtedly found a very good K candidate. In fact, in $K = 4$ there is a prominent (local) minimum for the second derivative of the SSE curve (Figure 23), a max (local) peak for the silhouette score (Figure 24) and a minimum (local) peak for the Davies-Bouldin score (Figure 25).

Unlike the case of the Standard Scaler, this time the clusters were well balanced, so we kept them all. We tried, like before, to individuate by eye the best features for the visualization of the clusters; however, the groups was not very clear, so we defined a new metric which helped us in the process. This metric measures the quantity of information a columns carries dividing its std by its mean. We can see the result of this clustering in Figure 26, where the axis are given by the columns `tot_tweets`, `retw_over_fav` and `frequency_mean`.

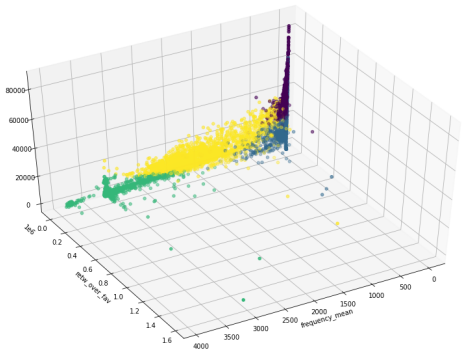


Figure 26

For each of the clusterings we found, we analyzed the distribution of the user's features in relation to the various clusters.

2.2 DBSCAN

First off we have to find the distance from each point to the nearest ones. Thanks to that, we can find the optimal distance value ε to give to the DBSCAN algorithm in order to run the clustering. For this purpose, we looked at plots in Figure 27 and Figure 28 and used the `KneeLocator` function.

Unfortunately, the results are very bad. No significant clustering was found, almost all the points were in one single big cluster.

We tried then to play with the value of ε but this gave no relevant result. Lower epsilon values gave some graphically visible cluster, however it was just a consequence of the fact that almost all the

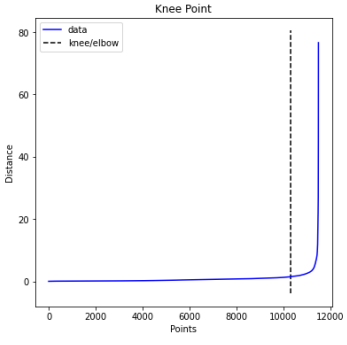


Figure 27: Std Scaler

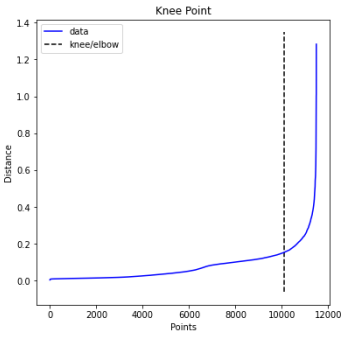


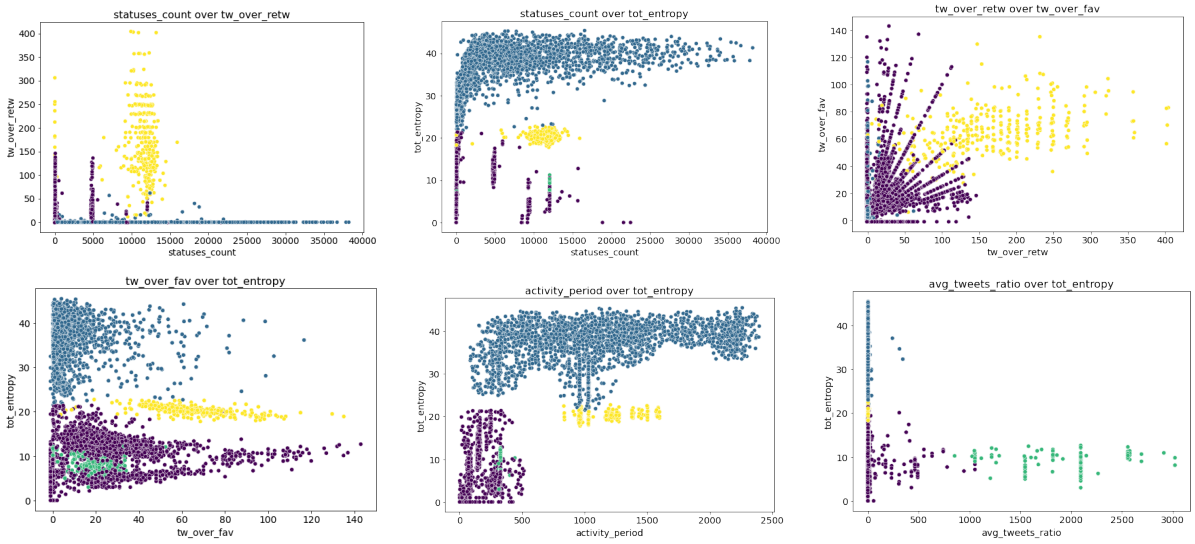
Figure 28: MinMax Scaler

points were classified as noise so the algorithm found some clusters in the few very close points.

2.2.1 Further dimensionality reduction

In order to obtain a valid clustering, we tried to reduce drastically the dimensionality. We kept only a minimal set of features composed by the following columns: `statuses_count`, `retw_over_fav`, `tw_over_retw`, `tw_over_fav`, `activity_period`, `avg_tweets_ratio` and `tot_entropy`.

The results improved a bit, especially with the Min-Max Scaler. The algorithm found two main clusters and two smaller ones. About 2000 points were identified as noise. We show the most interesting plots below:



Even if these results could be considered way better than the original ones, it is not very interesting since we have just two real clusters (thousands of points), while the other two are very small (245 and 347 points). Furthermore $\approx 18.5\%$ of the points were classified as noise. We can conclude that density-based clustering is not very suitable for our dataset.

2.3 Hierarchical Clustering

With Hierarchical Clustering we made a deeper exploration of the possible approaches for determining the features to use. The two main methods are:

- *Statistical approach*: compute statistical metrics and observe data through plots, then choose which dimensions happen to be more interesting for clustering.
- *PCA*: Powerful but we lose interpretability.

In the statistical approach, for selecting the most relevant features, we defined a *quality score* given by a weighted sum of three metrics: *spread* (std/mean), *entropy* and *total independency* ($1 - S/C$, where S is the averaged sum of the correlations of a column with every other columns and C is the max of the correlations of that column, so that $S/C \leq 1$). Once computed, we took the 10 features with the highest quality score.

Regarding the PCA, we just applied it and took the first 10 principal components. We also took a smaller subset of features composed by the first 4 principal components.

After the choice of the clustering features, we also considered three different types of normalization: Standard Scaler, Min-Max Scaler and log. We applied them in many various combinations to each of the features subsets identified before to generate the clustering datasets. Finally, we individuated the best combinations through plots observations.

We explored the Agglomerative Hierarchical Clustering algorithm, with 4 different linkages:

- Min: less divisive, good for concave shapes
- Max: complete linkage, more divisive
- Average: globular biased, less susceptible to noise
- Ward: less susceptible to noise and outliers, biased towards globular clusters

At this point, for each of the four possible linkages, we tried all of the 18 possible combinations of normalizations and PCA (4 and 10 principal components) and we visualized their dendograms. We can see an example in Figure 29.

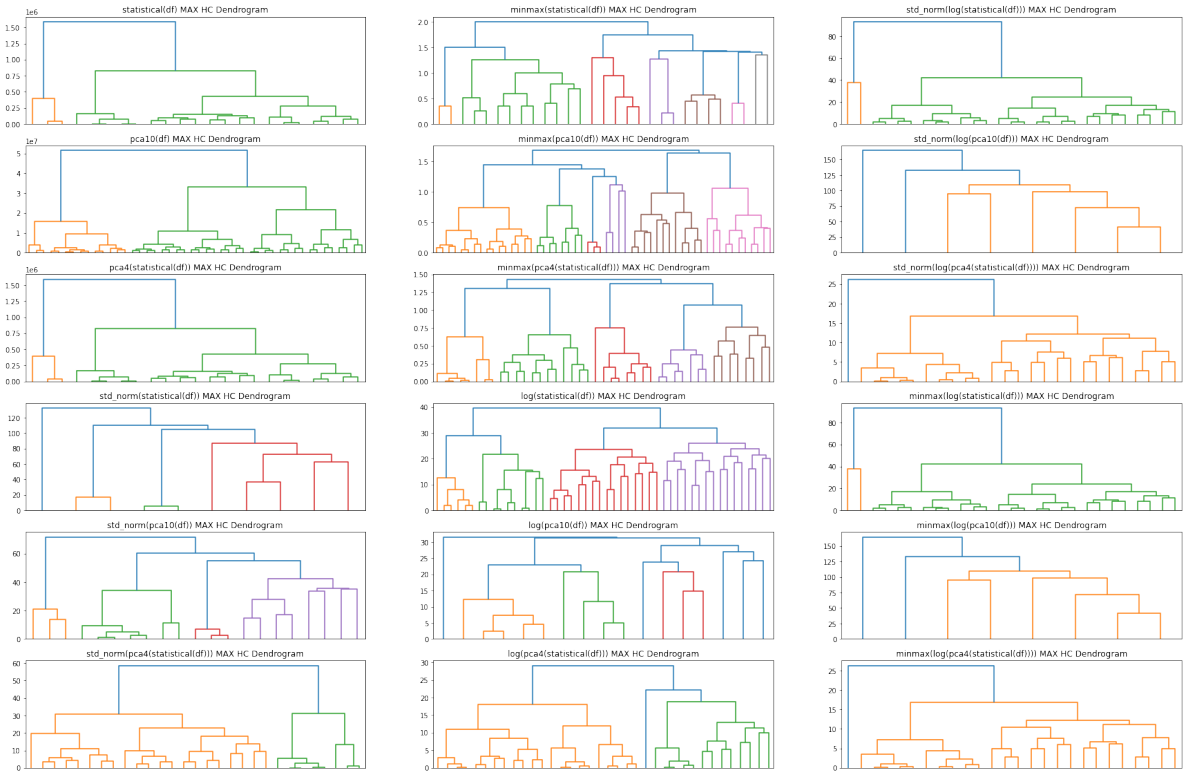


Figure 29: Max linkage

From these plots, we selected the combinations giving the best dendrograms (by symmetry and separation) and then, we went more in-depth by choosing the best dimensions to visualize and the number of clusters to identify, looking at the scatter plots like that in Figure 30.

For the chosen features we plot the dendrograms and the clusters plot, where the ellipses are drown around clusters at different separation levels. Blue ellipses are at 2 clusters, green at 4 clusters and orange at $n\_clusters$. We can see some results in figures from Figure 31 to Figure 36.

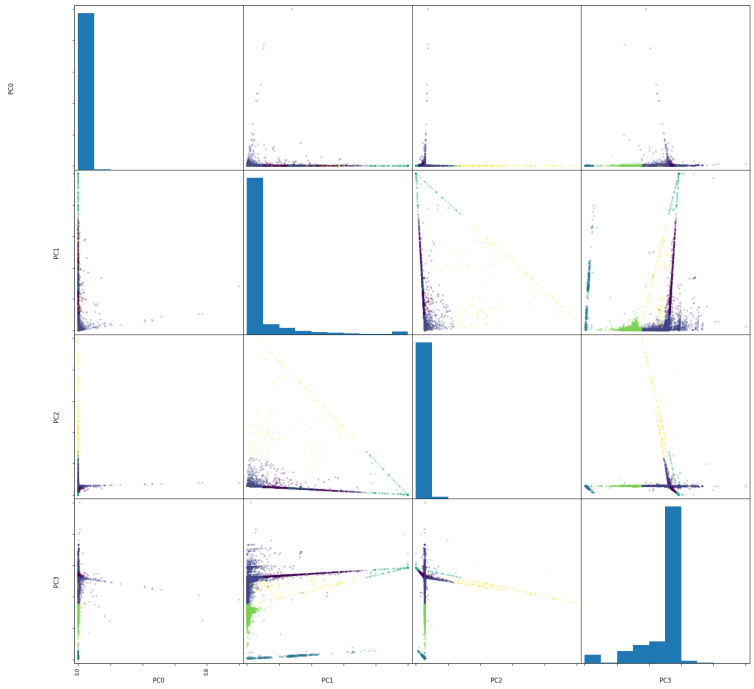


Figure 30: PCA 4

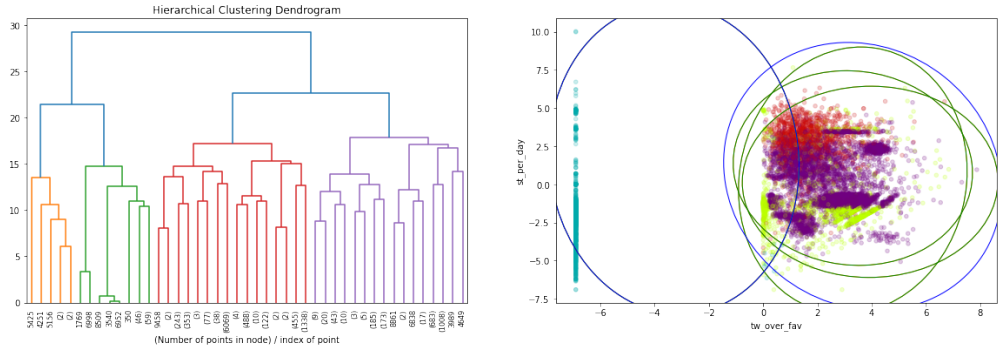


Figure 31: log normalization, Statistical, Avg linkage, X=tw/fav, Y=st per day, 4 clusters

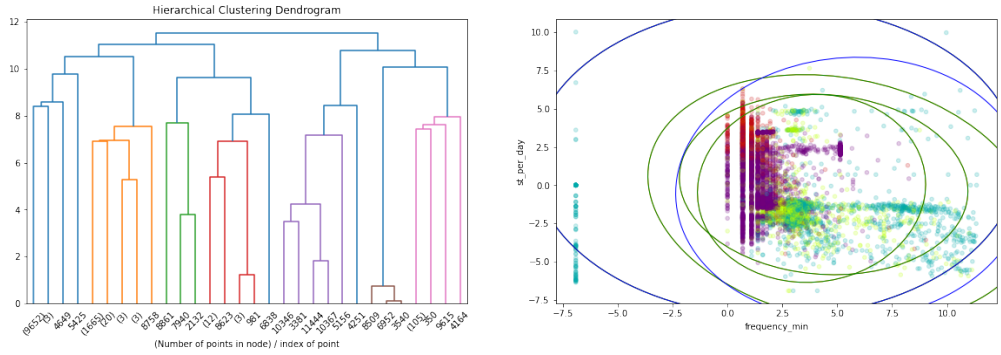


Figure 32: log normalization, Statistical, Min linkage, X=freq min, Y=st per day, 4 clusters

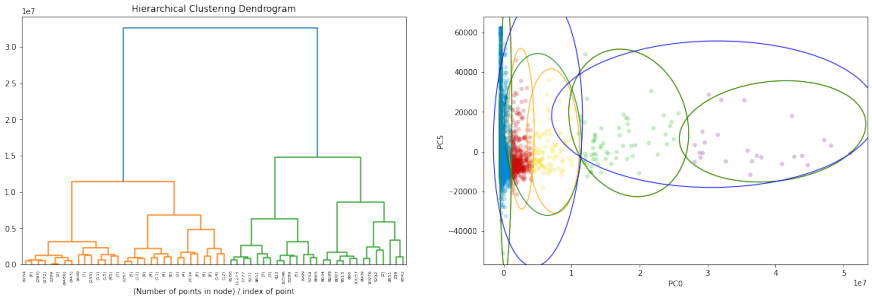


Figure 33: No normalization, PCA10, Avg linkage, X=PC0, Y=PC5, 6 clusters

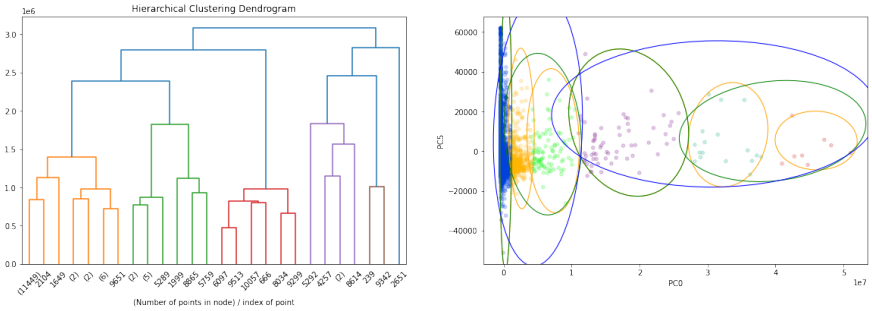


Figure 34: No normalization, PCA10, Min linkage, X=PC0, Y=PC5, 6 clusters

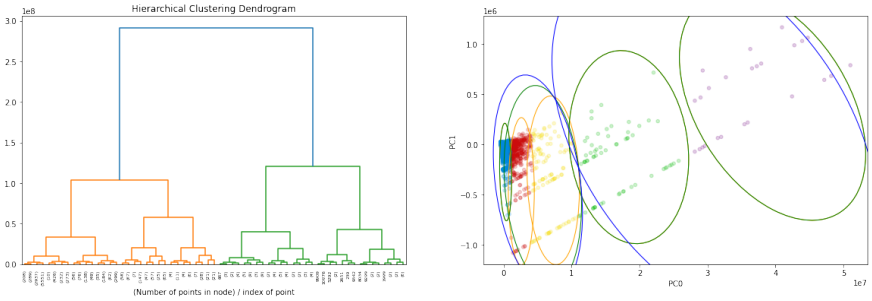


Figure 35: No normalization, PCA10, Ward linkage, X=PC0, Y=PC1, 5 clusters

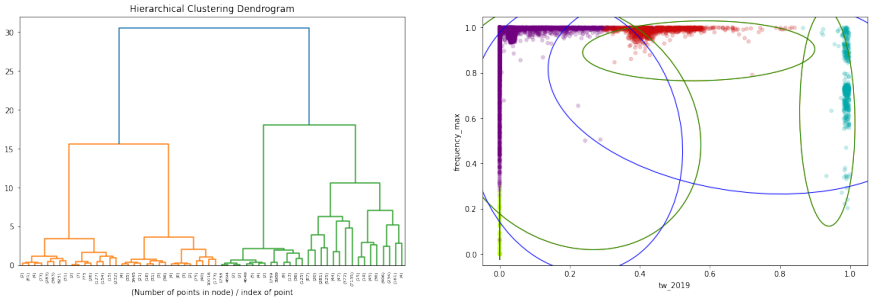


Figure 36: MinMax normalization, Statistical, Ward linkage, X=tw 2019, Y=freq max, 4 clusters

2.4 X-Means

The last clustering algorithm we ran is the X-Means algorithm from the python `pyclustering` library.

X-Means works by alternatively applying two operations: The K-Means algorithm to optimally detect the clusters for a chosen value of K, and cluster splitting to optimize the value of K according to the Bayesian Information Criterion. So, since we got quite good results (but surely improvable) with K-Means, we decided to try this algorithm.

The results are, unfortunately, not very good. The algorithm found 20 clusters, which is a very high number. However, some of these clusters are basically empty (a few dozen of points or less) and there is a cluster that alone contains more than 77% of the points, as we can see in Figure 37.

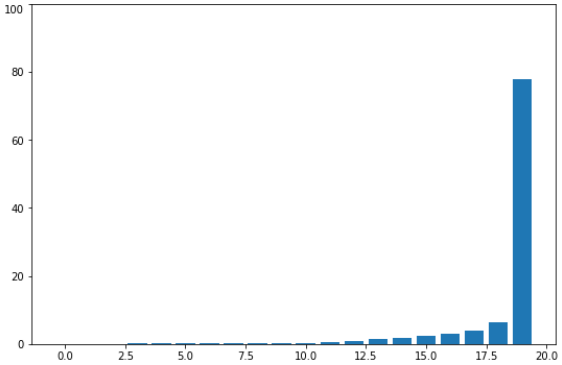


Figure 37: % of points in each cluster

We can conclude that this algorithm does not work well with our data. This may be because it tries to optimize the number of clusters based only on some numerical values associated with the clusters (e.g. entropy, silhouette) but has no knowledge of the distribution of our data, which is difficult to deal with. So, to optimize the numerical characteristics of the clusters, it ends up grouping almost all points into one big cluster and putting all other points into some smaller ones.

3 Classification

We divide the classification algorithms into two categories: those which are based on distance metrics and those which are not; we make this distinction because in non-metric algorithms categorical attributes can be included by discretizing them. In our case, since we want to make classification by the `bot` column, the only categorical attribute left is `lang`.

The classification was done by splitting the dataset into *training set* (70%) and *test set* (30%).

3.1 Algorithms not based on distance metrics

3.1.1 Decision Trees

The first classification algorithm we ran was the Decision Tree.

We started setting the criterion which measures the quality of a split as the default 'gini', while the max depth was set to 10. The other parameters are `splitter`='best', `min_samples_split`= 3 and `min_samples_leaf`= 4.

In Figure 38 we can see the resulting decision tree.

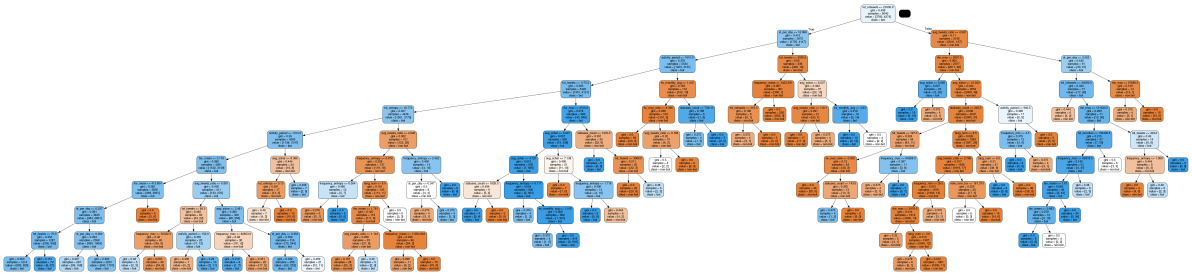


Figure 38

Once created, we assessed its prediction capability both on training set (TR) and test set (TS). First, we can see some metrics about the classification in Table 1 and Table 2.

Accuracy TR	Accuracy TS	Precision TR	Recall TR	F1 score TR
0.858	0.842	0.881	0.858	0.854

Table 1

	precision	recall	f1-score	support
non-bot	0.95	0.70	0.81	1615
bot	0.79	0.97	0.87	1832

Table 2: Metrics on test set for each class

After that, we made a further evaluation of the classification via validation and confusion matrix. The confusion matrix in Figure 39 highlights how almost the totality of prediction errors was done on non-bot users (which were predicted as bots).
Even if these results are very good, we tried many possible improvements.

First, we tried to make classification on the full dataset (46 columns), in order to understand if our features selection is good. The accuracy reached on full dataset is 0.858 on training set and 0.865 on test set, i.e. the same on training set and just 2% more than before on test set. Clearly, having more than double the features for just a 2% improvement isn't worth it.

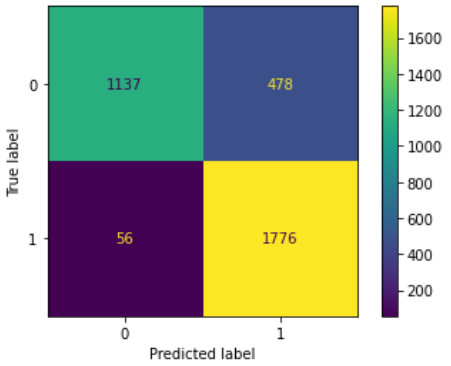


Figure 39

On the other hand, we tried to reduce drastically the number of columns (from 20 to 8) as we did with the DBSCAN. The performances were very similar, with an accuracy on test set of 0.835%. Thus, reducing the features could be a good move, since we lost just 1% of accuracy by taking less than half of the features. However, we continue to work with our set of 20 features because other classifiers could need them. This detail will be further analyzed.

Lastly, we tried to play with parameters. Most of the parameters were almost totally irrelevant. One thing we noticed was that we can take a shallower tree. In fact, deeper trees got a worse accuracy, while with a max depth of 5 we reached almost the same performances we got with a

max depth of 10. In particular, we obtained an accuracy of 0.843 on training set and 0.837 on the test set.

Summing up, if we want to minimize the number of used resources, we can obtain a very good classification with a minimal dataset of 8 features and a max depth 5 tree.

3.1.2 Random Forest and AdaBoost

Since Random Forest is an Ensemble method based on Decision Tree, we expect this method to work at least as good as the latter.

In practice, we had no advantages with ensemble, the best reached accuracy was the same we reached with decision trees.

We tried different parameters combinations. One interesting thing we tried was to assign different weights to the two classes: since the non-bot users were classified with a very high precision while bot users were often misclassified, we tried to give a bigger weight to bot class. However, we saw a very little improvement, still not better than decision trees.

We ran AdaBoost with Decision Tree as base classifier because, among non-metric classifiers, it was the best one. However, it seemed not capable of improving the performances achieved with the basic Decision Tree.

3.1.3 Naive Bayes Classifier

Another non-metric classifier we used is the Naive Bayes Classifier, i.e. a Bayes classifier with the naive assumption of independence.

We started with a Gaussian Naive Bayes classifier; however, it gave us not very good performances as we can see in Table 3.

	precision	recall	F1-score	support
non-bot	0.98	0.41	0.58	1615
bot	0.66	0.99	0.79	1832
accuracy			0.72	3447
macro avg	0.82	0.70	0.68	3447
weighted avg	0.81	0.72	0.69	3447

Table 3: Gaussian NB

Bayes classifier needs to assume the distribution of the likelihood of the features. With Gaussian Naive Bayes the results are not really good. Thus, we tried to standardize the data. We repeated the classification after having applied the Standard Scaler and, as reported in Table 4, the results improved noticeably.
Still, results are worse than those obtained with the Decision Tree. Hence, we tried with the Complement Naive Bayes, which is an adaptation of the standard Multinomial Naive Bayes algorithm that is particularly suited for imbalanced data sets, so that it might fit our case best. However, MultinomailNB assumes positive data, while we have some negative data in the ratios. In order to get around this problem, we normalized the data with the Min-Max Scaler, but results were bad.

Finally, we tried to reduce dimensionality as we already did with Decision Tree, but the results got a little worse.

	precision	recall	F1-score	support
non-bot	0.96	0.52	0.68	1615
bot	0.70	0.98	0.82	1832
accuracy			0.77	3447
macro avg	0.83	0.75	0.75	3447
weighted avg	0.82	0.77	0.75	3447

Table 4: Gaussian NB with Standard Scaler

We have seen that, in general, Naive Bayes classifier has poor performances. Maybe, assuming independence in our data is too strong of an assumption. Also, assuming a distribution of this kind (Gaussian or Multinomial) having distributions with extreme long tails is a strong assumption too. So, we can conclude that Naive Bayes does not work very well for our data.

3.2 Distance-based algorithm

3.2.1 K-NN

The classification started with the standard distance metric of the `KNeighborsClassifier` from `sklearn`, which is the Minkowski distance. The weights were uniform, i.e. each neighbor have the same impact in the choice of the class.

We searched for the best number of neighbors, starting from 10. After we tried some values, we found that the best value is $K = 20$. With this setting we get an accuracy of 0.821.

We expect better performance with a weighted distance: giving the same weight to each neighbor point of the point we want to classify independently on the distance, could lead to a less accurate classification. In fact, changing the value of the argument `weights` from 'uniform' to 'distance' led to a little (but quite irrelevant) improvement on the accuracy on test set: 0.823.

Changing the metric from Minkowski to Euclidean did not changed anything.

Reducing the columns have slightly degraded the accuracy.

So, the best accuracy we got with K-NN is 0.823, which is a good classification accuracy, even if it is slightly smaller than that of Decision Tree.

3.2.2 Support Vector Machine (SVM)

We ran SVM both with Standard Scaler and Min-Max Scaler and also without normalization. The best results were obtained with the first one, with an accuracy of 0.8402 on the training set and 0.8416 on test set. This is one of the best results among all the classifiers we ran.

We explored some possibilit5es of improving the performance of the SVM classifier.

First of all we tuned the regularization level, reducing it (by increasing C from 1 to 2) because, having such skewed data with a too strict regularization could not be optimal. Accuracy is improved by a negligible amount and we get 0.8427 on the training set and 0.8422 on the test set.

One thing we noticed was that the prediction performances on the two classes are different, as shown in Table 5, so we tried to give different weights to the two classes. However, we noticed no improvements.
In all the tests the `gamma` parameter regarding kernel coefficients, was set to 'auto', because we observed worst performances setting it to 'scale'.

	precision	recall	f1-score	support
non-bot	0.97	0.69	0.80	1615
bot	0.78	0.98	0.87	1832
accuracy			0.84	3447
macro avg	0.87	0.83	0.84	3447
weighted avg	0.87	0.84	0.84	3447

Table 5: SVM

Finally, we ran the algorithm with a minimal subset of features but the performances degraded a bit. So, we can conclude with a result of 0.8422 of accuracy on test set, which is very good.

3.2.3 Neural Network (MLP)

The Neural Network classifier `MLPClassifier` from `sklearn` was initially trained trying different parameters combination looking for the best results. We observed that the best performing parameters have not shown a smooth decreasing learning curve, but since thesy performed well on an independent test set we excluded overfitting. The best result achieved 0.8 accuracy. In order to efficiently explore the hyperparameters space in a more objectively way, we implemented a *Genetic Algorithm* to iteratively look for the best model. We tried this algorithm for both a broad hyperparameters space and a more focused space close to the best solution we found by hand. Note that using a grid search with these many parameters together with the given dataset and 20 features would not be feasible because of computational constraints.

The set we explored includes the following hyperparameters: hidden layers (numbers of layers and sizes), learning rate (value and schedule), activation function, solver, batch size and maximum number of iterations. The validation was performed via the `StratifiedKFold` from `sklearn`.

We can see the results of the search in the two spaces (wide and narrow respectively) in Tables 6 and 7.

gen	0	1	2	3	4	5
nevals	5	10	10	10	10	10
fitness	0.638	0.721	0.723	0.770	0.774	0.774
fitness_std	0.116	0.094	0.095	0.003	0.0006	0.0006
fitness_max	0.773	0.773	0.773	0.773	0.775	0.775
fitness_min	0.489	0.531	0.531	0.765	0.773	0.773

Table 6: Wide space search

Best model parameters: activation='logistic', batch= 240, hidden l.= $(50, 50, 20)$, learn. rate init= 0.0077, max it.= 1195

After having searched different spaces we see that we achieve comparable results with respect to other classifiers. Therefore, we can state that the neural network approach can achieve up to 0.8 accuracy score on average. The best result was obtained in generation 3 of estimator 3 narrow space exploration, achieving an accuracy of 0.826. This result is also not far from what was achieved by manually exploring the hyperparmeteres space.

gen	0	1	2	3	4	5	6	7	8
nevals	16	32	32	32	32	32	32	32	32
fitness	0.591	0.749	0.803	0.812	0.814	0.804	0.800	0.803	0.808
fitness_std	0.129	0.070	0.024	0.011	0.012	0.029	0.024	0.023	0.013
fitness_max	0.825	0.817	0.822	0.826	0.826	0.826	0.822	0.822	0.822
fitness_min	0.468	0.552	0.727	0.780	0.780	0.730	0.726	0.729	0.781

Table 7: Narrow space search

Best model parameters: batch= 110, hidden l.= $(50, 50)$, learn. rate='invscaling', learn. rate init= 0.0032, max it.= 1064

3.3 Classification results

In Table 8 we summed up the accuracy of all the classifiers we ran on our dataset.

DecisionTree	RandomForest	AdaBoost	NaiveBayes	20-NN	MLP	SVM
0.85	0.84	0.84	0.77	0.82	0.83	0.84

Table 8

4 Time Series Analysis

After having defined:

$$AcceptanceScore \coloneqq retweet\_count + reply\_count + favorite\_count \ ,$$
$$Diffusion Score \coloneqq num\_hashtags + num\_mentions + num\_urls \ ,$$

we created, for each user, the time series of the *success scores*, defined as:

$$SuccessScore \ \coloneqq \ \frac{AcceptanceScore}{Diffusion Score + 0.1} \ ,$$

where each timestamp corresponds to a day of the year 2019 (more precisely, it's a day where there is at least a tweet) and its value is given by the success score in that day for that user.

Note that, if a user has more than one tweet in a specific day, we took the sum of the fields of such tweets in order to compute the scores.

4.1 Clustering

We explored three different kinds of clustering. Let's start with the clustering based on the whole series.

4.1.1 Shape-based clustering

After an initial try without any scaler, where we got a very high inertia and clusters differentiated some extremely high peaks of the success score, we applied both the Min-Max Scaler and the Mean-Variance Scaler, separately.

4.1.1.1 Min-Max Scaler

We analyzed clusters' metrics (SSE, Silhouette Score and Davies-Bouldin Score) for a sequence of K values from 1 to 11 in order to identify the best ones.

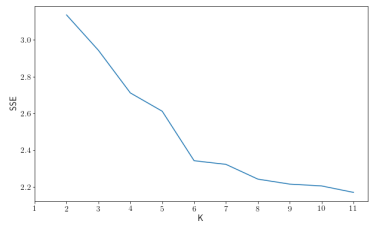


Figure 40: SSE

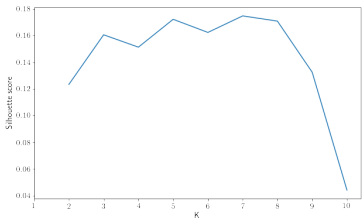


Figure 41: Silhouette

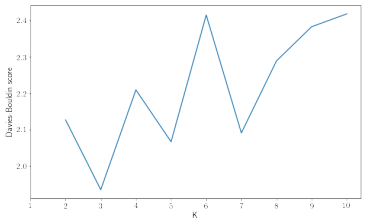


Figure 42: Davies-Bouldin

Among the best K values we identified there is $K = 3$ because it's the global minimum point for the Davies-Bouldin Score and it has a good Silhouette Score. The best values for SSE and Silhouette Score are instead achieved for $K = 6$ and $K = 7$. In Figure 43 we can see the centroids series for $K = 3$.

We can see that the three series are not completely different from each other but there are time intervals in which they are quite similar and other (often small) time intervals in which they differ more.

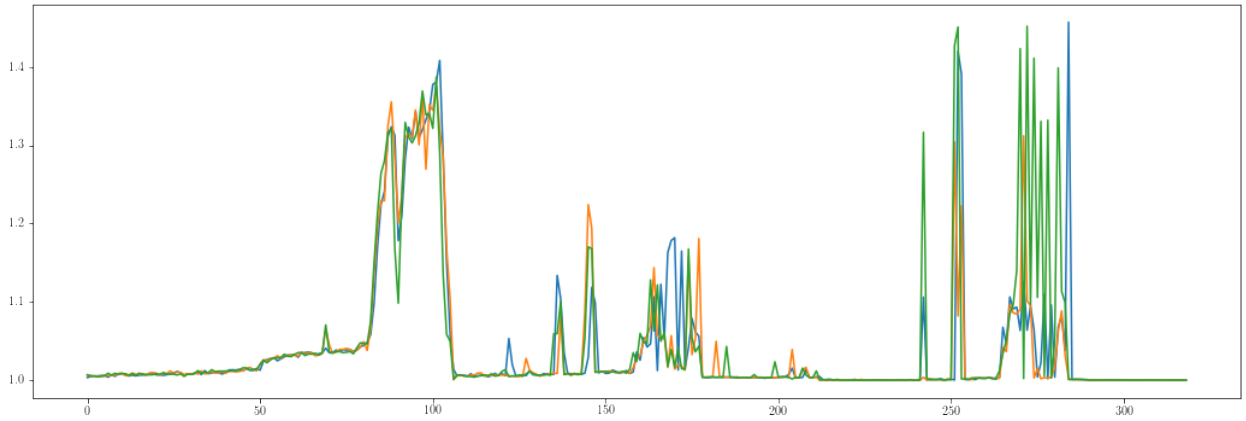


Figure 43: Centroids for $K = 3$

4.1.1.2 Mean-Variance Scaler

We did the same thing as before with Mean-Variance Scaler and we can see the results of the clusters metrics in Figure 44, Figure 45 and Figure 46.

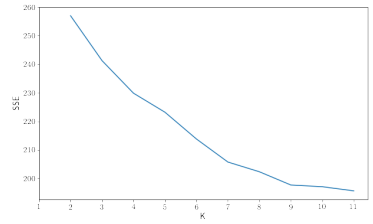


Figure 44: SSE

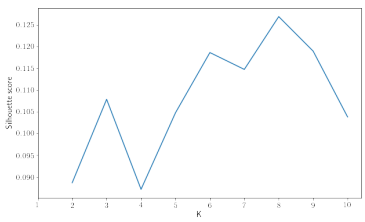


Figure 45: Silhouette

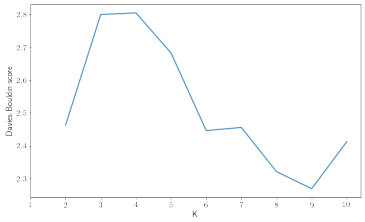


Figure 46: Davies-Bouldin

We observed that the overall results given by the metrics is slightly worse than that we obtained with Min-Max Scaler. We show the centroids for $K = 6$ in Figure 47

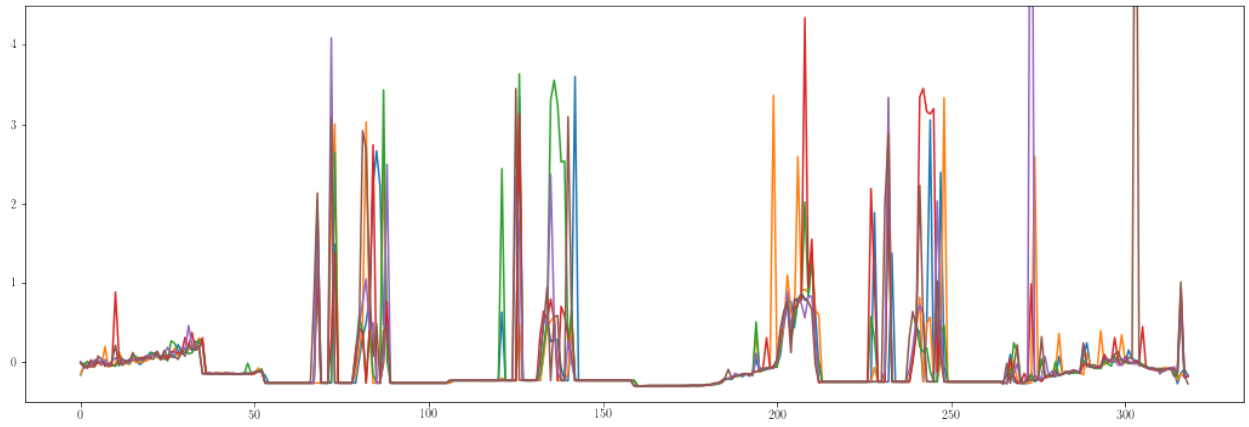


Figure 47: Centroids for $K = 6$

4.1.2 Features-based clustering

For the features-based clustering we selected as features various statistics like mean, std, variance, percentiles, etc.

The results without a scaler were not very clear, so we applied the Min-Max Scaler. The resulting centroids are reported in Figure 48. Through the analysis of clusters' features we selected as best K value $K = 4$.

As we can see, the centroids seems to be more differentiated with respect to the centroids we obtained in the shape-base clustering. Thus, the features we selected allow the algorithm to better discriminate the time series of our dataset.

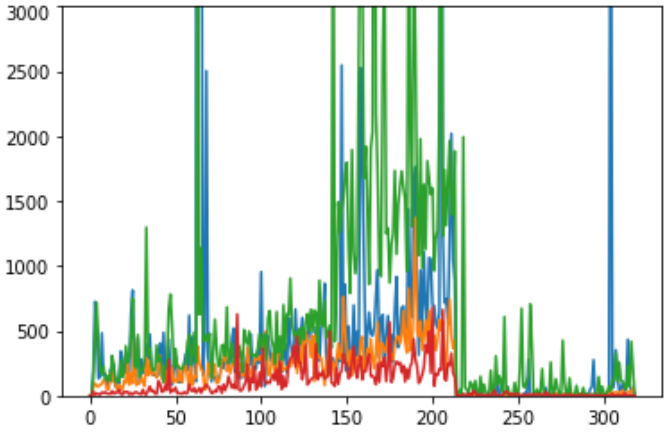


Figure 48: Centroids for $K = 4$

4.1.3 Compression-based clustering

Compression-based clustering consists of compressing the time series and then grouping their compressed versions.

We used two different kind of compression:

- The first method transforms a time series into a string which characters correspond to single timestamp values. Once we have compressed the series, we defined a distance between two strings S_1 and S_2 in the following way:

$$dist(S_1, S_2) := \frac{|zip(S_1 \oplus S_2)|}{|zip(S_1)| + |zip(S_2)|},$$

  where $S_1 \oplus S_2$ represents the concatenation of the two strings.
  In this case we used DBSCAN for clustering the compressed time series.

- The second method is the *Piecewise Aggregate Approximation*, which minimizes dimensionality by the mean values of equal sized frames. Then, the distance between the compressed series is computed via a vector distance. In our case, we used Euclidean distance.
  We computed the clusters through K-Means.

Both methods performed poorly: DBSCAN just found one cluster, while K-Means performed best with $K = 3$ but the clusters were very unbalanced (6686, 2 and 98 points respectively).

4.2 Classification via Shapelets

To classify through shapelets, we used two different methods of two well-known library (`tslearn` and `pyts`).

`pyts` method, `ShapeletTransform`, is the one who actually gave us best results, using both K-NN and RandomForest classifiers.

Since computing shapelets is computationally expensive, we used a small number of shapelets, just 5, and set the window length to 319, which is the actual total length of the dataset's time-series.

K-NN classifier performed best with $K = 28$, with an accuracy value of 0.848, while the RandomForest closed the test predictions with an accuracy of 0.830.

Since using the ShapeletTransform method while taking a full length window disrupts the purpose of shapelets themselves, we tried once more setting the window size to 150, with 5 total shapelets and a window step of 5. We couldn't go too far with the number of shapelets due to the high computational cost. Because of this very reason, we ran the last training using the 10% of the total dataset as training set (having 678 time series). Classification's results (using RandomForest classifier) had accuracy value of 0.775. On the other hand, `tslearn` method `LearningShapelets`, supported by a neural netweork from `keras` resulted in an accuracy value of 0.718.

We can see the shapelets we found through `LearningShapelets` with window size 127 and 6 shapelets in Figure 49.

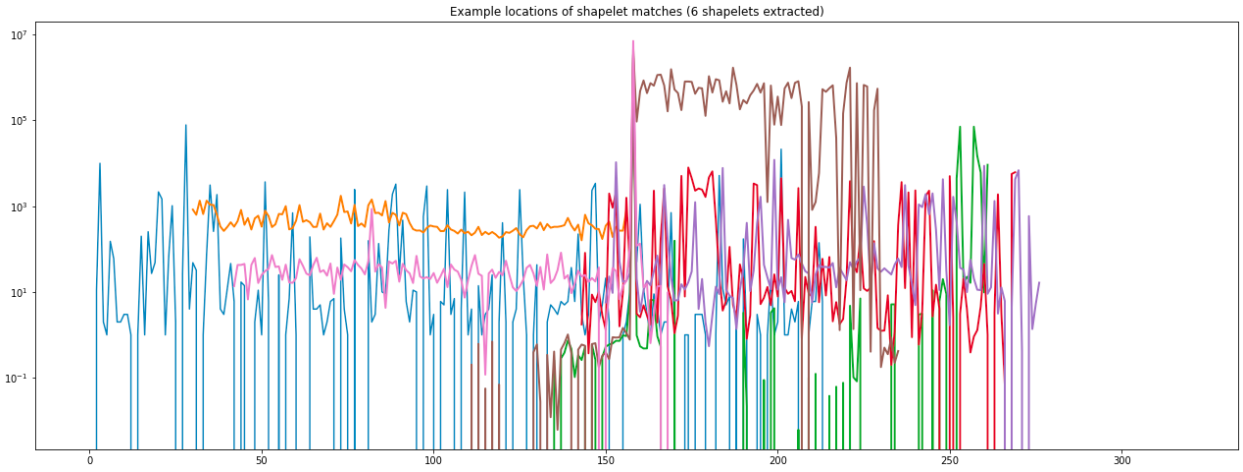


Figure 49

We resume the results of time series classification in Table 9.

	ShapletTransform			LearningShapelets
	28-NN	RandomForest		NeuralNetwork
Window Size	319	150	319	127
N. shapelets	5	5	5	6
Accuracy	0.85	0.78	0.83	0.72

Table 9