Data Mining Project Report
Group 13

Diego Arcelli, Malio Li, Paolo Fasano

A.A. 2022/2023

1 Introduction

In this report, we describe all the procedures and operations we applied to the datasets we were given for the Data Minig project. The data we analyzed is composed of two datasets: one contains information about tweets written in the Twitter platform, the other contains information about some Twitter's users. This report is organized in different sections:

- Section 2: Data understanding and preparation
- Section 3: Features extraction
- Section 4: Clustering analysis
- Section 5: Predictive analysis
- Section 6: Explainable AI

In section 2 we provide a detailed description of each attribute in the `users.csv` and `tweets.csv` datasets explaining how we dealt with problems and errors we encountered. In section 3 we are going to list and explain how we extracted some new indicators for the users. We then built a new dataset with these new indicators. In section 4 we perform a clustering analysis using the new features in order to discover groups of similar users. In section 5 we use the dataset to train and compare various classifiers on predicting whether a user is a bot or not. In section 6 we apply XAI techniques to some of our classifiers in order to better understand their decisions, and we also fit two interpretable by-design models.

2 Data understanding

In this section, we provide a detailed description of the two given datasets and the data cleaning and preparation operations we applied to them. The code with all the plots is in the `data_understanding_preparation.ipynb` notebook. In table 1 and table 2, we report a brief description of the attributes of the datasets.

2.1 Semantics of the attributes

Name	Description	Domain
`name`	the name of the user on the platform	string
`user_id`	a unique identifier of the user	integer 19 digit
`statuses_count`	the count of the tweets and retweets made by the user at the moment of data crawling	non-negative integer
`lang`	selected language of the user	language ISO code
`created_at`	the timestamp for the user profile creation	date time
`bot`	identifies if the user is a bot (1) or not (0)	binary

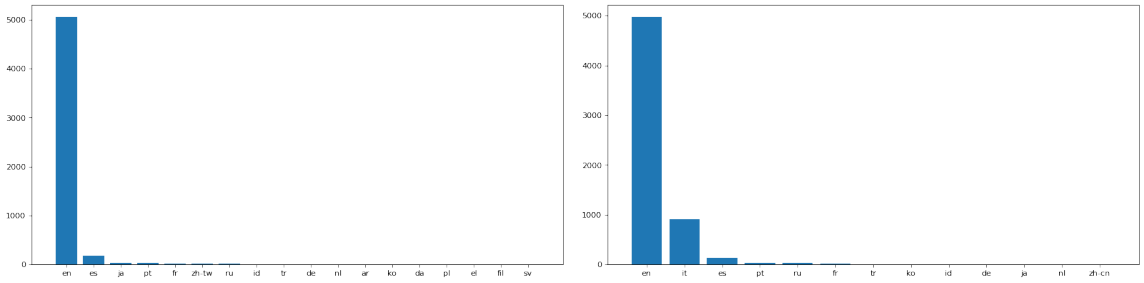
Table 1: Users dataset attributes

Name	Description	Domain
`id`	unique identifier of the tweet	integer 19 digit
`user_id`	unique identifier of the user who wrote the tweet	integer 19 digits
`retweet_count`	number of retweets for the tweet	non-negative integer
`reply_count`	number of replies for the tweet	non-negative integer
`favorite_count`	number of likes received by the tweet	non-negative integer
`num_hashtags`	number of hashtags used in the tweet	non-negative integer
`num_urls`	number of URLs in the tweet	non-negative integer
`num_mentions`	number of mentions in the tweet	non-negative integer
`created_at`	the timestamp of the tweet creation	date time
`text`	the text of the tweet	text

Table 2: Tweets dataset attributes

2.2 User data-set

We started by studying the users dataset. The dataset didn't contain duplicate values, but there was a missing value for the attribute `name`, which we fixed by assigning the string "name" in its place. Then we analyzed the `lang` attribute where we found a few problems: one record had the value "Select Language..." and another an ISO code = "xx-lc" which does not exist. We decided to fix these issues by setting them to "en" since it is the mode of the attribute and, moreover, we checked the tweets of the users corresponding to the wrongs `lang` values, and they were written in English. Another operation we performed was to set the ISO codes of British English `en-gb`, `en-GB` and Australian English `en-AU` to simply `en`, and also the two Chinese codes `zh-tw` and `zh-cn` were set to `zh`. The final distribution of the languages is reported in figure 1, where we can see that English is dominantly the most frequent language for both real and bot users.

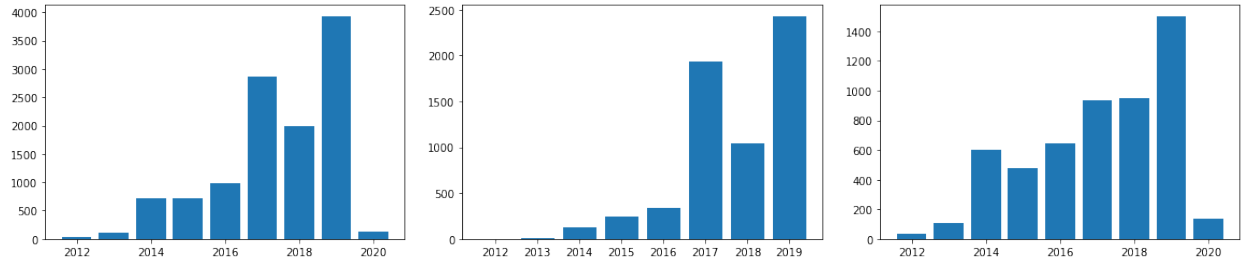


(a) Real users (b) Bot users

Figure 1: Languages distribution

Then we checked the validity of the dates of the `created_at` attribute. All the dates are in the correct format and they are in a valid range from `2012-01-24 01:57:38` to `2020-04-21 07:28:31`. We can check the distribution of the accounts creation date in figure 2, where figure 2-a is the total accounts created per year. We also made other two plots to distinguish between real users (figure 2-b) and bot users (figure 2-c)

Then we analyzed `bot` column which is valid since all the values are either 0 or 1. We can count the records associated with those 2 values and we obtain the plot in figure 3, and we can see that the two classes are quite balanced.



(a) total accounts creation year distribution (b) Bot users accounts creation year distribution (c) real users accounts creation year distribution

Figure 2: accounts creation year distribution

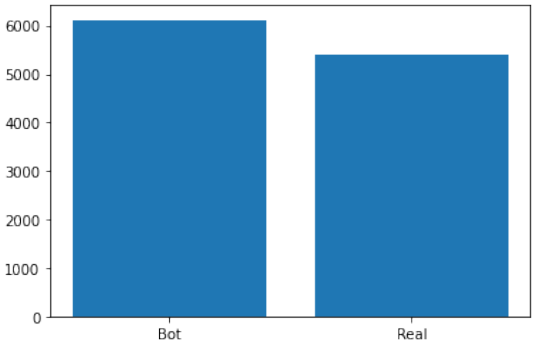
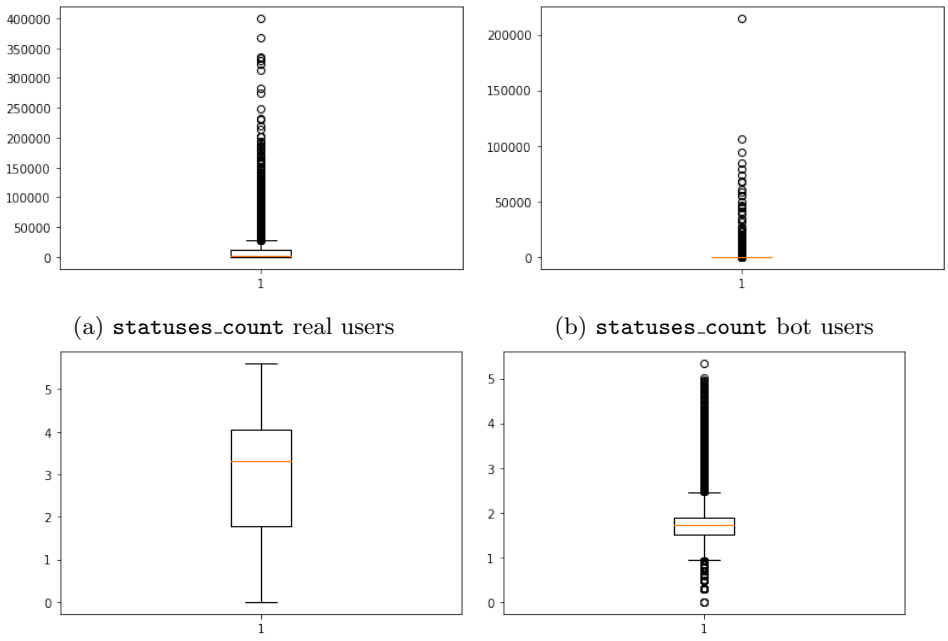


Figure 3: Distribution of real and bot users

After that, we checked the `statuses_count` column which has 399 missing values. We start analyzing the distribution (ignoring the missing values) conditioning on the bot and non-bot users. Since the distributions are extremely left-skewed, we also study them in logarithmic scale, applying the transformation $\log_{10}(x+1)$ to the values of the attributes. The boxplots can be seen in figure 4.



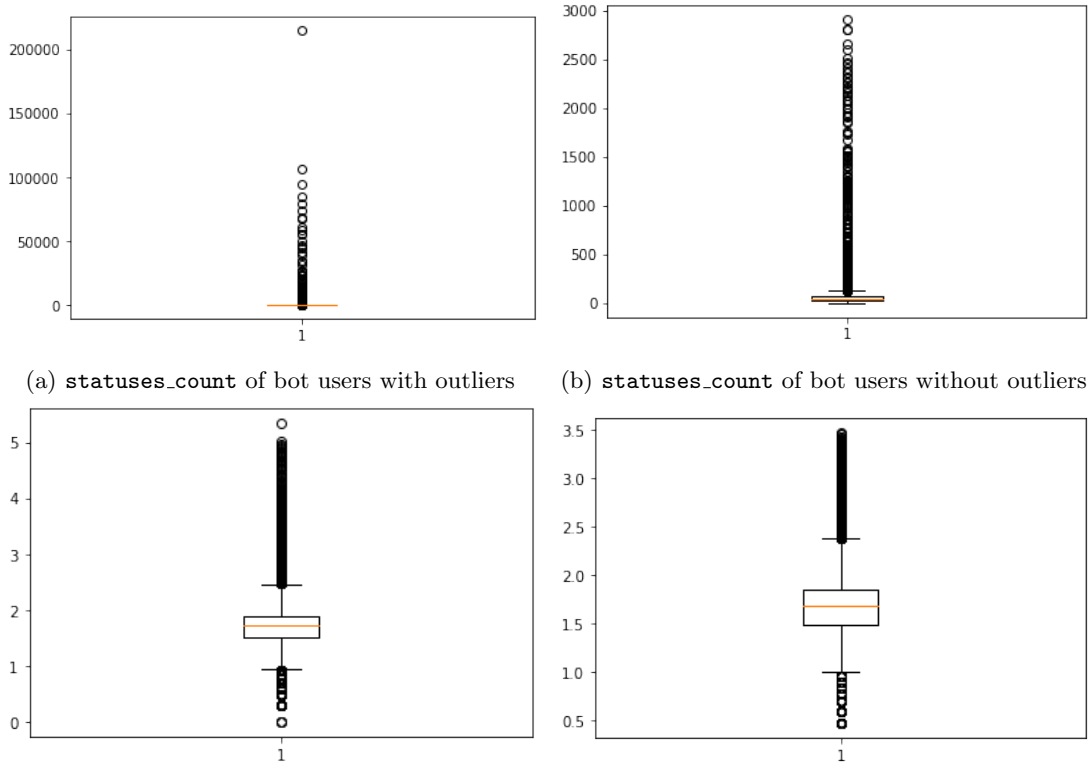
(a) `statuses_count` real users (b) `statuses_count` bot users

(c) Log `statuses_count` real users (d) Log `statuses_count` bot users

Figure 4

For the real users, the log scale boxplot does not detect any outliers, while for the bot users, it detects many

outliers, therefore we correct `statuses_count` only for the bot users. For the correction, we decided to cut from two standard deviations from the mean (computed in log scale) setting to NaN the values which are not in this range. The distribution changes as shown in figure 5.



(a) `statuses_count` of bot users with outliers

(b) `statuses_count` of bot users without outliers

(c) Log `statuses_count` of bot users with outliers

(d) Log `statuses_count` of bot users without outliers

Figure 5

After that, we proceed to fix NaN values of the attribute by substituting the median of the attribute conditioned on the `bot` column (which is 2041 when `bot` is 0 and 53 when `bot` is 1).

2.3 Tweets dataset

For the tweets dataset, as for the users, we start by checking the validity of the different attributes. The first thing we noticed is that there are many duplicate values, in fact after we remove the duplicates the number of records decreases from 13664695 to 11712597. Even after removing the duplicate, we noticed that there are some records that still have the same values for the attributes `user_id`, `created_at` and `text`, therefore we also drop the records which are duplicates for those three attributes. After that, we considered the `id` column, where we noticed there are duplicated values. Moreover, the format of the attribute should be a 19 digits integer, but there are also integers of fewer digits and some values are random strings. Since the `id` column, in the end, is just a unique identifier of each tweet and we're not gonna use this column directly for any type of analysis we decided to just reassign incrementally the IDs so that they are unique.

We then considered the `user_id` columns, even in this case, not all the IDs follow the same format and there are strings among the values. More importantly, the number of unique user IDs in the tweets dataset is 222285 and it is much greater than the number of unique user IDs in the users dataset, which is 11508. This is a problem since, in the end, we wanna merge the two datasets so that each tweet has also the information of the user who wrote it. We also noticed that even if the users that do not appear in the users dataset are the majority, they only wrote a very small part of the total number of tweets (just 210777). In the end, we decided to drop the tweets of the users whose ID do not appear in the user dataset because:

1. Those tweets are just small fractions of the total tweets, therefore we lose a small amount of information
2. We can't use those users in further analysis, since for them we don't have the attributes of the users dataset

After removing those tweets the dataset has 10447963 records.

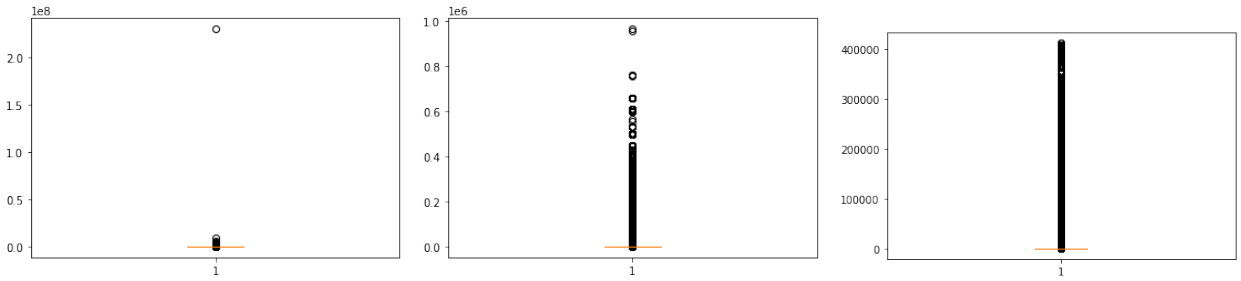
For the `created_at` attribute, all the dates are in a valid format, but there are some dates that are before the creation of the account of the user who wrote the tweet, and other dates that are too far ahead in time, even after the year 2023. In order to fix this problem we decided to assign a specific date to all the 'broken' dates so that we can identify them if needed.

Then we analized the numerical columns of the dataset which are: `reply_count`, `favorite_count`, `reply_count`, `num_urls`, `num_mentions` and `num_hashtags`. They all should be integers, but instead, they're of type Object, and moreover, all of them have many NaN values. In all the columns there are values that violate the domain like strings and negative numbers, therefore for each column, we set all the values which are not positive integers to NaN. After that, we checked the minimum and maximum values for every column. The minimums are 0 for all the attributes, the maximums are reported in the second column of table 3.

Attribute	Max (with outliers)	Max (without outliers)
`favorite_count`	8955855	10424
`retweet_count`	229531973	411723
`reply_count`	44585999	984
`num_urls`	58593401	7
`num_mentions`	501355	17
`num_hashtags`	658316	24

Table 3: Maximums before and after outliers correction, minimums are all 0

After that, we dealt with the outliers of those attributes. All the distributions are extremely left-skewed even if we apply to them a logarithmic transformation, since there is a big dominance of 0s in the attributes. This makes it impossible to use the boxplot for detecting outliers, since the inter-quantile range is 0 for most of them, and therefore also the whiskers of the boxplot are 0, and this would cause to consider any value greater than 0 as an outlier. Since from the boxplots it is clear that in each attribute there are values which are an order of magnitudes higher than all the other values, we decided to apply the following strategy for each attribute: we consider the boxplot of the attribute and we find an appropriate order of magnitude for the values of the attribute, and we exclude the values greater than that order of magnitude. At this point, we visualize the boxplot again and we fix a threshold for which after removing the outliers above that threshold in the resulting boxplot the outliers will be "dense". Here in the report, we show an example for the `retweet_count` attribute in figure 6.



(a) `retweet_count` boxplot

(b) `retweet_count` boxplot after removing values above 10^6

(c) `retweet_count` boxplot after removing outliers

Figure 6: Outliers correction for `retweet_count` attribute

We repeat the procedure for every one of the six numerical attributes, by setting the values above the thresholds to NaN. After adjusting the outliers the maximums of the attributes changes as shown in the third column of table 3. After detecting the outliers we correct the NaN values of the attributes, substituting them with the median value of the attribute conditioning on the `user_id` column. After that, we also computed the correlation matrix of the numerical attributes, but there are no correlating attributes.

Once we terminated the analysis of the two datasets, we merged them with an inner join operation on the attributes `user_id` for the tweets dataset and `id` for the users dataset, so that in the merged dataset for each tweet we have the information of the user who wrote the tweet. The name of the attributes are the same except for the `created_at` attribute, since there was an attribute with that name in both datasets, therefore we called the column of the creation date of the tweet `tweet_date` and the date of the creation of the user account `user_date`.

3 Features extraction

3.1 Defining new features

Now that we have the new merged dataset, we can use it to extract new features, in order to define a better profile of the users. This has been done in the `features_extraction.ipynb` notebook. The features we defined are reported in table 4.

Name	Description
num_tweets	how many tweets the user have in the tweets dataset
tweets_per_day	how many tweets the user sends per day
avg_len	average length of tweets the user wrote
avg_spc_len	average number of special characters in tweets the user wrote
retweet_count_avg	average retweet count of the user's tweets
reply_count_avg	average reply count of the user's tweets
favorite_count_avg	average favorite count of the user's tweets
num_hashtags_avg	average number of hashtags of the user's tweets
num_urls_avg	average number of URLs of the user's tweets
num_mentions_avg	average number of mentions of the user's tweets
retweet_count_std	retweet count standard deviation of user's tweets
reply_count_std	reply count standard deviation of user's tweets
favorite_count_std	favorite count standard deviation of user's tweets
num_hashtags_std	number of hashtags standard deviation of user's tweets
num_urls_std	number of URLs standard deviation of user's tweets
num_mentions_std	number of mentions standard deviation of user's tweets
retweet_count_total	total number of retweets of user's tweet
reply_count_total	total number of replies of user's tweet
favorite_count_total	total number of favorites of user's tweet
num_hashtags_total	total number of hashtags of user's tweet
num_urls_total	total number of URLs of user's tweet
num_mentions_total	total number of mentions of user's tweet
retweet_count_entropy	retweet count entropy of user's tweets
reply_count_entropy	reply count entropy of user's tweets
favorite_count_entropy	favorite count entropy of user's tweets
num_hashtags_entropy	number of hashtags entropy of user's tweets
num_urls_entropy	number of URLs entropy of user's tweets
num_mentions_entropy	number of mentions entropy of user's tweets
retweet_count_per_day	daily retweets for user's tweets
reply_count_per_day	daily replies of user's tweets
favorite_count_per_day	daily favorites of user's tweets
num_hashtags_per_day	daily number of hashtags of user's tweets
num_urls_per_day	daily number of URLs of user's tweets
num_mentions_per_day	daily number of mentions of user's tweets

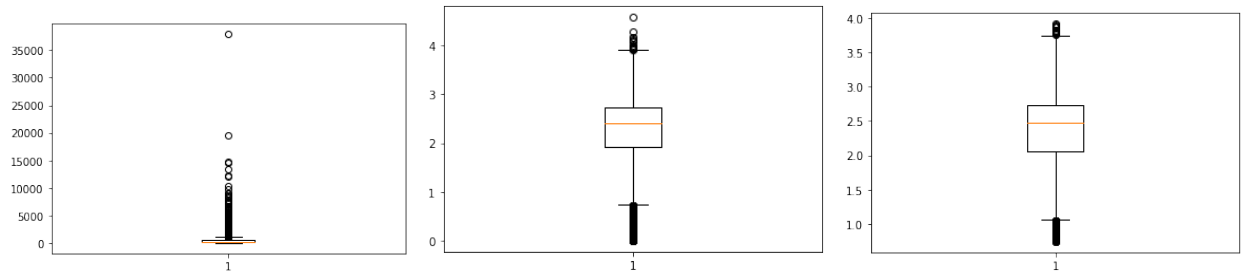
Table 4: Extracted features from the merged dataset

For `avg_spc_len` we considered as special character any character which is not an alphabet letter nor a number. All the per-day stats are computed by taking the sum of the considered attribute over all the tweets of the user, and then the sum is divided by the number of days passed from when the user created his account to the date of the latest tweet in the dataset (which is the 2020-05-03). To compute the standard deviation we divided by n and not by $n-1$ (where n is the number of tweets of the user) in order to avoid the division by 0 cases if the user has only one tweet.

3.2 Data understanding on new features

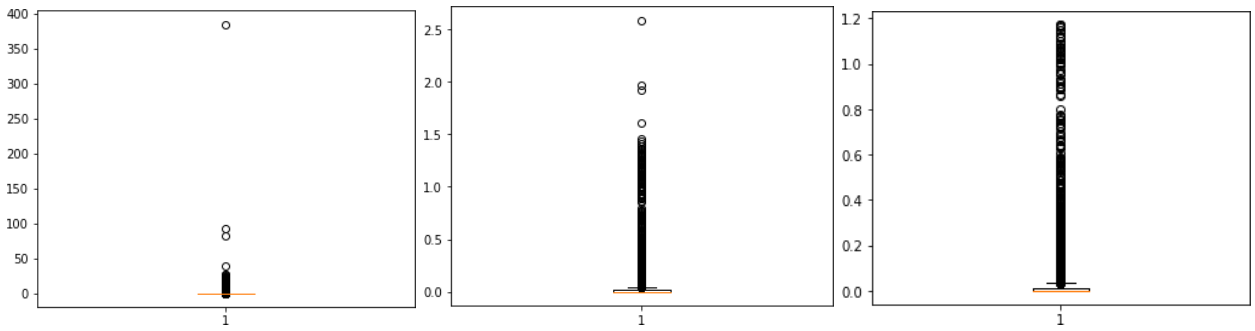
After extracting these new features, we repeat the data understanding and preparation analysis on the new features, in the file `DU_extracted_features.ipynb`. For each feature we computed all the relevant stats (mean, median, standard

deviation, MAD, and others) and then we studied their distribution using the histograms and the boxplots. We studied each feature conditioning it with respect to the `bot` column. The main focus of this part was the correction of the outliers. As for the numerical attributes of the tweets dataset, many of the new attributes are extremely left-skewed. Therefore we decided to apply the following strategy for each attribute: we check the histogram of the attribute: if it is not too left skewed we use the whiskers of the boxplot to detect outliers. Otherwise, we apply a log transformation to the attribute. If after applying the log transformation now the distribution of the attribute is not too left-skewed, then we use the whiskers of the boxplot of the log-transformed attribute to detect the outliers, but if even after the log transformation the distribution remains too skewed, then we consider the log boxplot of the attribute and we try to find a threshold for the cut of outliers such that after removing values greater than the threshold in the resulting boxplot the outliers will be dense. Here we show some examples for two attributes in figure 7 and 8.



(a) `retweet_count_avg` boxplot (b) `retweet_count_avg` log boxplot (c) `retweet_count_avg` log boxplot after removing outliers using boxplot whiskers

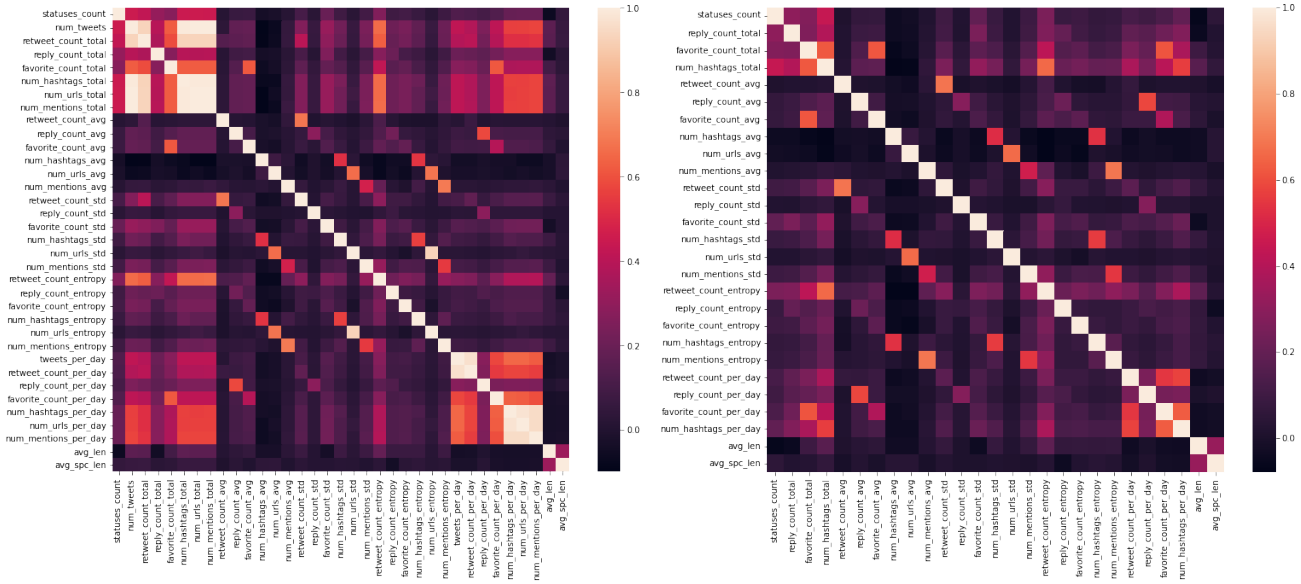
Figure 7: Outliers correction using boxplot for `retweet_count_avg`



(a) `reply_count_std` boxplot (b) `reply_count_std` lo gboxplot (c) `reply_count_std` log boxplot after removing outliers

Figure 8: Outliers correction using the density criterion for `reply_count_std`

The points which are detected as outliers are corrected as we did for the previous datasets, hence by setting them to the median (conditioned on the `bot` column) of the attribute. Finally, we plot the correlation matrix of the new features in figure 9-a. As we can see from the picture some of the attributes are highly correlated.



(a) Features correlation matrix

(b) Correlation matrix of uncorrelated features

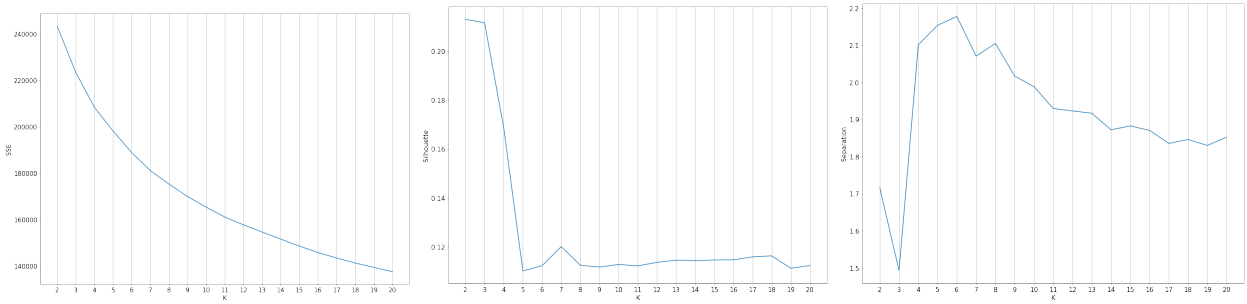
Figure 9: Correlation matrix of extracted features

4 Clustering analysis

For the clustering analysis, we applied K-means, DBSCAN, and the agglomerative hierarchical clustering. We also tried to use Self Organized Map (SOM) and X-means from the `pyclustering` library. We decided to remove some features according to the correlation matrix shown above (figure 9-a): if two features have a correlation index greater or equal to 0.8 we drop one of them. In this way, we can reduce the dimension of the space for the clustering analysis to mitigate the problem given by the curse of dimensionality. The selected features are those showed in figure 9-b. For the preprocessing in order to handle left-skewed distributions, we decided to apply a logarithmic transformation to attributes whose distribution is left-skewed (by studying the histogram of each individual attribute). After that, we normalized each attribute by applying standard normalization, so that the different scales of the attributes do not affect the clustering algorithms.

4.1 K-means

For the K-means to find the best value for the number of clusters k we ran the algorithm many times varying the values of k from 2 to 20 (fixing the maximum number of iterations to 1000 and the number of initializations to 100). For each value of k, we recorded the measure for SSE, silhouette, and separation. The results are shown in figure 10.



(a) SSE

(b) Silhouette

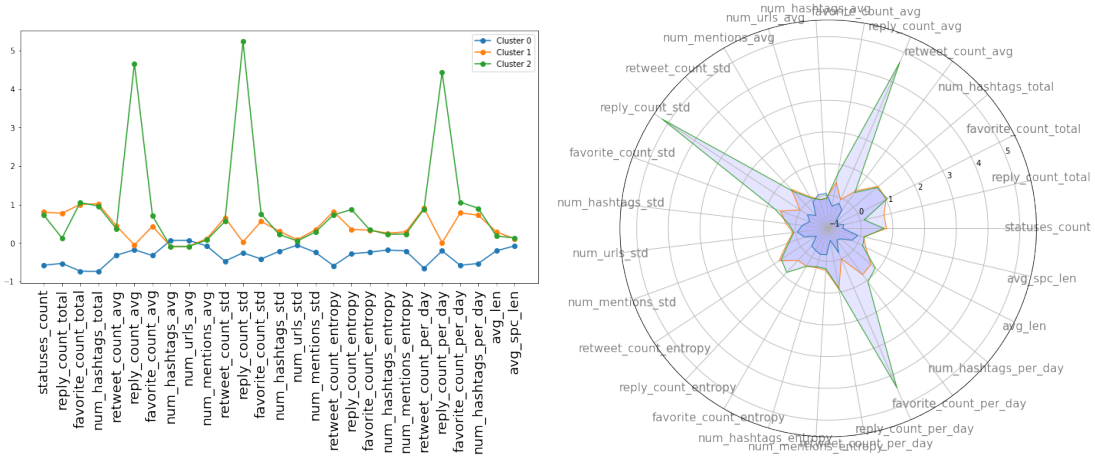
(c) Separation

Figure 10: Clustering validation metrics for K-means for different values of k

From the SSE plot, we can see that there is no clear elbow in the curve, the curvature that resembles the elbow on the plot is more or less between 5 and 8. If we look at the plots of silhouette and separation in the range 5-8, we can see that both metrics don't change much, the silhouette has a small peak in 7, while the separation has a peak in

6 after which it starts to decrease. In general, both the silhouette and the separation get really worse after the value 3, which is the best one for both metrics, even though SSE is still a bit high in 3. Given these considerations, we decided to select $k = 3$, since even if the SSE is high and it is not precisely in the "elbow", as we just explained, the separation and the silhouette get significantly worse after 3.

After executing the algorithm we proceed to study the centroids of the three clusters (figure 11). Additionally, we also computed for each cluster the median value of each attribute for the users in that cluster, so that we can have a clearer understanding of the type of user which is represented by the cluster, this can be seen in tables 5 and 6, where the last row contains the median values of the attributes of the whole dataset, in order to compare the median values of the dataset with the median values of each cluster.



(a) Parallel coordinates

(b) Radar plot

Figure 11: K-means centroids analysis

cluster id	num users	statuses count	retweet count avg	reply count avg	favorite count avg	reweet count per day	reply count per day	favorite count per day	bots frac
0	6648	46	249.9	0.0	0.37	19.4	0.0	0.05	0.66
1	4553	2296	469.9	0.0019	0.53	738.6	0.0032	1.05	0.35%
2	307	2261	450.5	0.141	0.65	777.1	0.223	1.34	0.36%
All	11508	64	372.0	0.0	0.45	114.99	0.0	0.13	0.53%

Table 5: K-means clusters median attributes values

cluster id	num users	tweets per day	avg len	num hashtags avg	num mentions avg	num urls avg	num hashtags per day	num mentions per day	num urls per day
0	6648	0.18	63.64	0.157	0.484	0.148	0.0182	0.076	0.02
1	4553	1.47	69.89	0.158	0.489	0.148	0.229	0.705	0.215
2	307	1.61	66.51	0.158	0.486	0.148	0.243	0.752	0.229
All	11508	0.28	64.60	0.158	0.487	0.148	0.0462	0.139	0.0418

Table 6: K-means clusters median attributes values

4.2 DBSCAN

For the DBSCAN to find the best values for the minimum distance ϵ and the minimum number of neighbors for a core point k, we plot the graph of the distance to the k-th less distant point for each record of the dataset. We tried for different values of k ranging from 2 to 1000. In the below picture, you can see an example for $k = 50$.

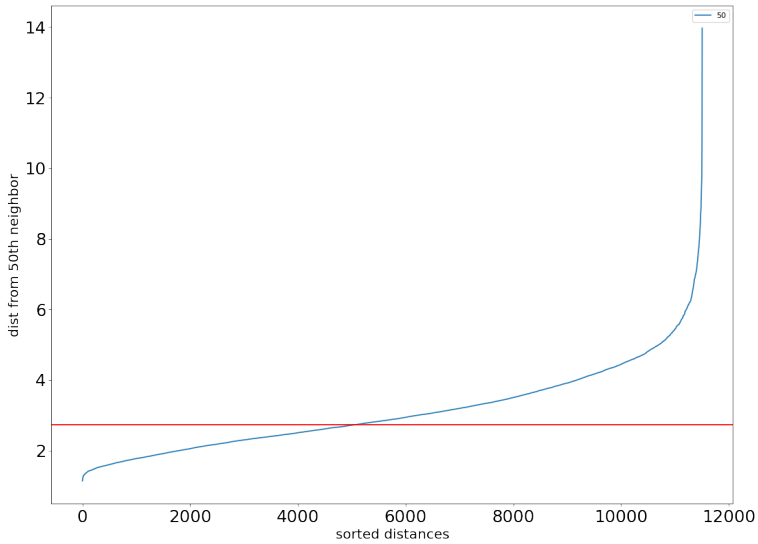
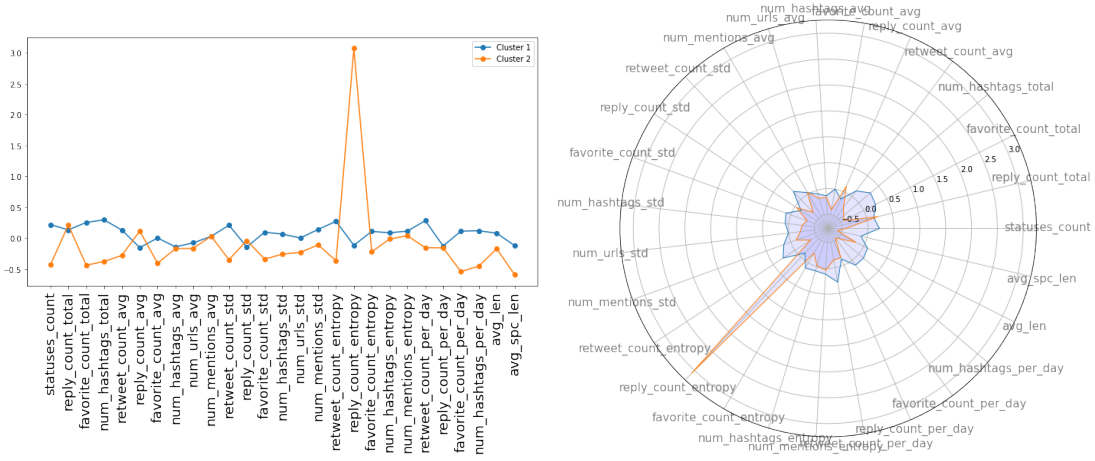


Figure 12: Distances from the 50-th closest neighbor for each data point and vertical line $y = 2.73$

We tried to find for the selected k the best value of ϵ, by choosing a value for which most of the points of the dataset are at least at distance ϵ from their k-th closest point. In most cases, the algorithm found only a small noise cluster and a huge core cluster that contains all the points which are not noise. In some cases, we were able to find a third very small core cluster. Here we present the results for the values of $k = 50$ and $\epsilon = 2.73$ where three clusters were found. Even if we know that DBSCAN is a density based algorithm, and therefore the resulting clusters might not have a globular shape, we compute anyway the centers of the core clusters (excluding the noise cluster) and we report in figure 13 their values for the attributes, in order to compare the results of DBSCAN with the ones of the other algorithms we applied. We also compute again the table of the median values of the attributes which is shown in tables 7 and 8.



(a) Parallel coordinates plot

(b) Radar plot

Figure 13: DBSCAN core clusters centers

cluster id	num users	statuses count	retweet count avg	reply count avg	favorite count avg	reweet count per day	reply count per day	favorite count per day	bots frac
0 (noise)	3603	41	177.1	0.0	0.44	11.6	0.0	0.02	0.70%
1	7783	90	385.1	0.0	0.46	307.9	0.0	0.32	0.46%
2	122	61.5	217.1	0.009	0.36	60.2	0.002	0.10	0.42%
All	11508	64	372.0	0.0	0.45	114.99	0.0	0.13	0.53%

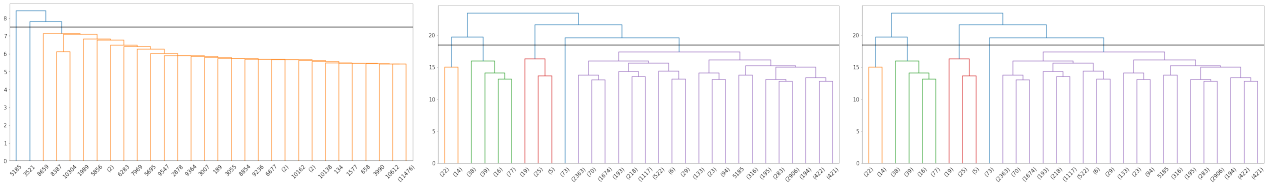
Table 7: DBSCAN clusters median attributes values

cluster id	num users	tweets per day	avg len	num hashtags avg	num mentions avg	num urls avg	num hashtags per day	num mentions per day	num urls per day
0 (noise)	3603	0.04	63.5	0.154	0.484	0.146	0.009	0.02	0.0
1	7783	0.64	65.7	0.155	0.487	0.146	0.1	0.3	0.09
2	122	0.27	64.1	0.150	0.491	0.139	0.04	0.131	0.04
All	11508	0.28	64.60	0.158	0.487	0.148	0.0462	0.139	0.04

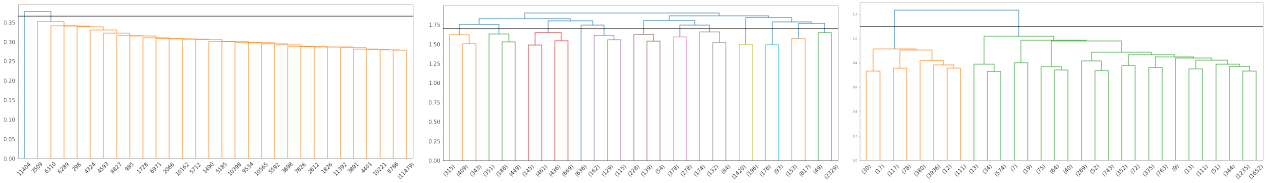
Table 8: DBSCAN clusters median attributes values

4.3 Hierarchical

The third algorithm we tested is the agglomerative hierarchical clustering. We tested different linkage methods namely complete, average, single, and ward. For the distance metric, we tested both the euclidean and the cosine. For each combination of distance metrics and linking criteria, we studied the corresponding dendrogram. All the dendrograms are shown in figure 14 and 16-a. For each dendrogram, we also make a cut to decide the number of clusters. After establishing the cut, and therefore the number of clusters, we executed the clustering algorithm to have an idea of the results by measuring SSE, silhouette, and the separation metrics and by counting the number of points in each cluster. The metrics are reported in table 9. In most cases, there was one cluster that contained almost all the points of the dataset, and other extremely tiny clusters. In the end, to perform a more detailed analysis we decided to use the clustering result obtained using euclidean distance and ward linkage since, as figure 16-a shows, from its dendrogram we can clearly visualize three well-distinguished clusters, while in the other cases it is not so clear. Moreover, as table 9 shows, even if the euclidean-ward is not the best combination for any metric, is a good trade-off if we consider all the three metrics together. In figure 17 we also report the radar and the parallel coordinates plots of the centers of the clusters, which we computed manually since the algorithm does not return them.

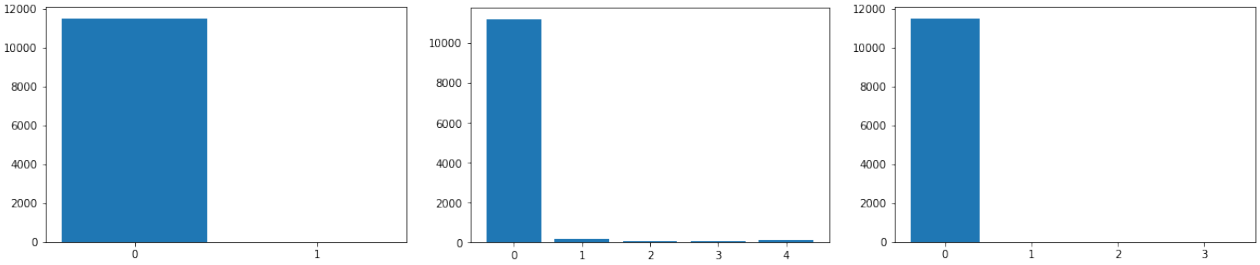


(a) Euclidean distance, single linkage (b) Euclidean distance, complete linkage (c) Euclidean distance, average linkage

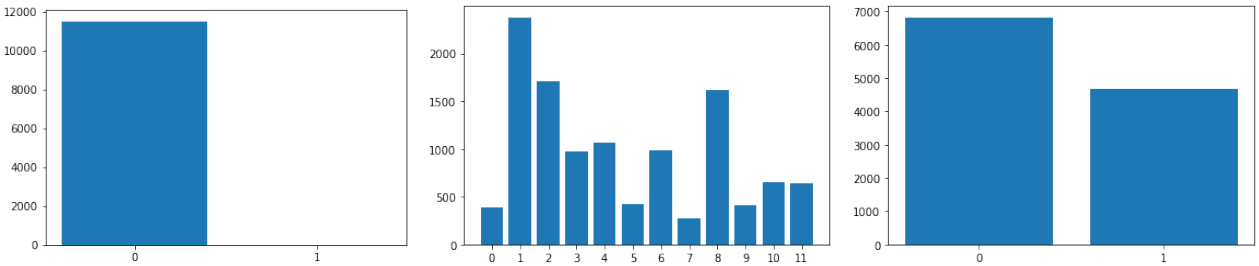


(d) Cosine distance, single linkage (e) Cosine distance, complete linkage (f) Cosine distance, average linkage

Figure 14: Dendrograms which have been discarded

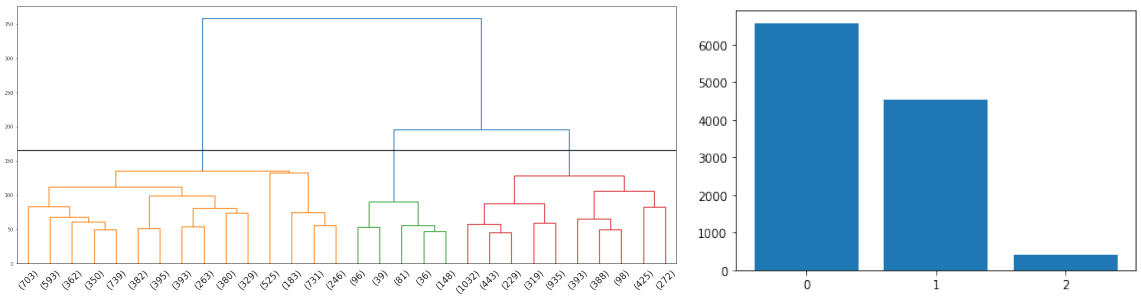


(a) Euclidean distance, single linkage (b) Euclidean distance, complete linkage (c) Euclidean distance, average linkage



(d) Cosine distance, single linkage (e) Cosine distance, complete linkage (f) Cosine distance, average linkage

Figure 15: Clusters size

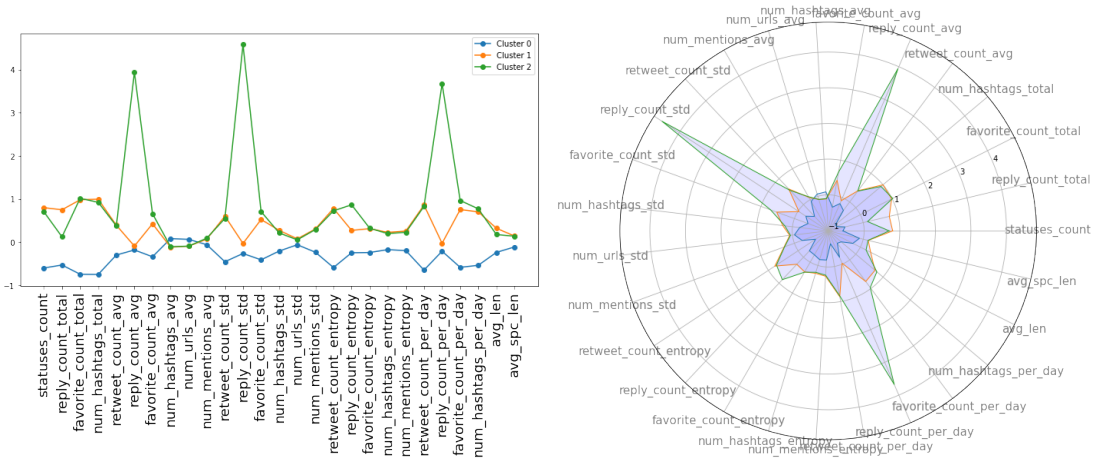


(a) Dendrogram (b) Clusters size

Figure 16: Dendrogram and clusters size for euclidean distance and ward linkage

	Euclidean distance				Cosine similarity		
	Single	Complete	Average	Ward	Single	Complete	Average
SSE	310578.88	278474.27	309046.23	227391.32	310684.70	**198843.93**	246347.18
Silhouette	**0.45**	0.34	0.44	0.20	0.08	0.01	0.20
Separation	**0.42**	1.29	0.59	1.58	0.86	2.74	1.75
Clusters	2	5	4	3	2	12	2

Table 9: Metrics of the clustering results



(a) Parallel coordinates plot

(b) Radar plot

Figure 17: Hierarchical clustering centers

cluster id	num users	statuses count	retweet count avg	reply count avg	favorite count avg	reweet count per day	reply count per day	favorite count per day	bots frac
0	6574	46	156.1	0	0.37	19.9	0.0	0.05	0.68%
1	4534	2177.5	460.5	0.002	0.52	720.8	0.003	1.03	0.33%
2	400	2159	453.8	0.149	0.68	750	0.242	1.31	0.36%
All	11508	64	342.1	0.0	0.45	115.0	0.0	0.13	0.53%

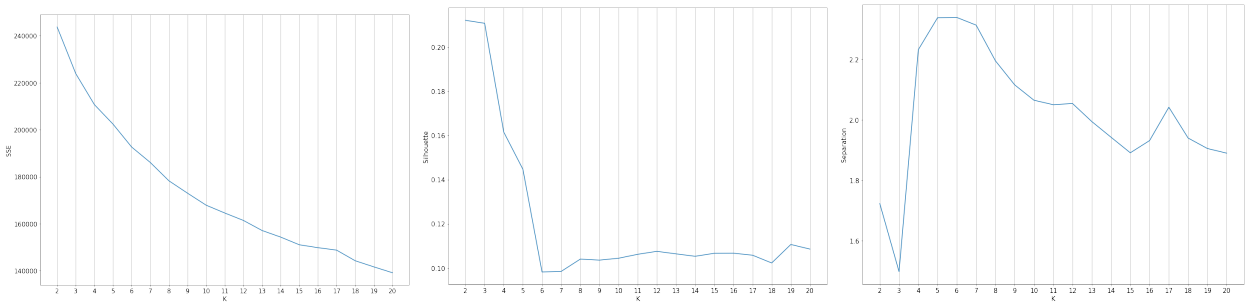
Table 10: Hierarchical clusters median attributes values

cluster id	num users	tweets per day	avg len	num hashtags avg	num mentions avg	num urls avg	num hashtags per day	num mentions per day	num urls per day
0	6574	0.19	63.53	0.143	0.481	0.143	0.02	0.08	0.02
1	4553	1.45	76.52	0.158	0.488	0.147	0.23	0.70	0.21
2	400	1.53	69.99	0.157	0.487	0.148	0.24	0.76	0.23
All	11508	0.28	64.60	0.158	0.487	0.148	0.0462	0.139	0.0418

Table 11: Hierarchical clusters median attributes values

4.4 Self Organized Map

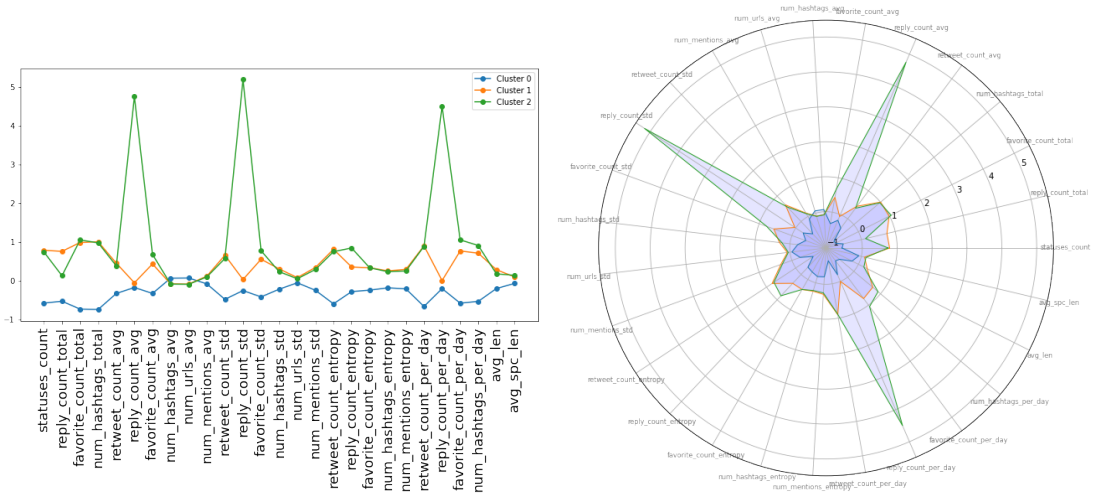
For the Self Organized Map (SOM) model, which is a specific type of neural network, the only hyper-parameter to tune is the number of clusters k. In order to find the best value of k we followed the same methodology used for the K-means and the plots of the metrics are shown in figure 18.



(a) SSE (b) Silhouette (c) Separation

Figure 18: Clustering validation metrics for SOM for different values of k

We can see that the obtained plots are extremely similar to the ones of K-means, therefore for the reasons we explained before, again we select $k = 3$ for the number of clusters.



(a) Parallel coordinates plot hierarchical (b) Radar plot hierarchical

Figure 19: SOM clusters

cluster id	num users	statuses count	retweet count avg	reply count avg	favorite count avg	reweet count per day	reply count per day	favorite count per day	bots frac
0	6600	46	151.2	0.0	0.365	19.0	0.0	0.05	0.66%
1	4606	2184.5	471.8	0.002	0.530	732.3	0.003	1.05	0.36%
2	302	2377.5	453.8	0.19	0.66	772.1	0.336	1.35	0.35%
All	11508	64	342.1	0.0	0.45	115.0	0.0	0.13	0.53%

Table 12: SOM clusters median attributes values

cluster id	num users	tweets per day	avg len	num hashtags avg	num mentions avg	num urls avg	num hashtags per day	num mentions per day	num urls per day
0	6600	0.18	63.6	0.143	0.478	0.143	0.02	0.07	0.02
1	4606	1.45	74.0	0.158	0.489	0.147	0.23	0.71	0.22
2	302	1.61	69.4	0.158	0.486	0.148	0.25	0.80	0.24
All	11508	0.28	64.60	0.158	0.487	0.148	0.0462	0.139	0.0418

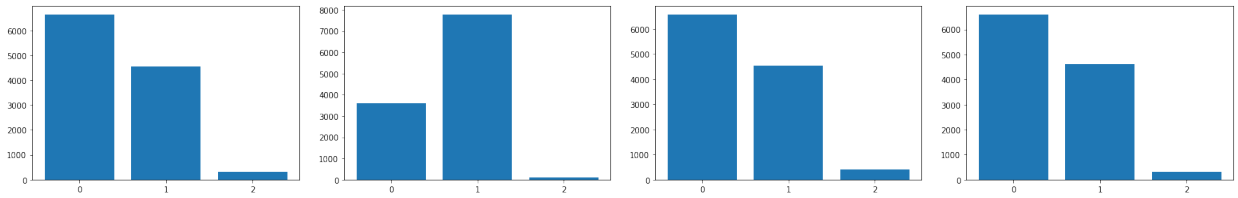
Table 13: SOM clusters median attributes values

4.5 X-means

We also (unsuccessfully) experimented with X-means, which uses K-means while trying to find automatically the best number of clusters. We performed different executions, varying the number of initial clusters, but unfortunately in any case the number of clusters was between 60-80, which made it basically impossible to perform an analysis. We report anyway the values of the metrics obtained in table 14.

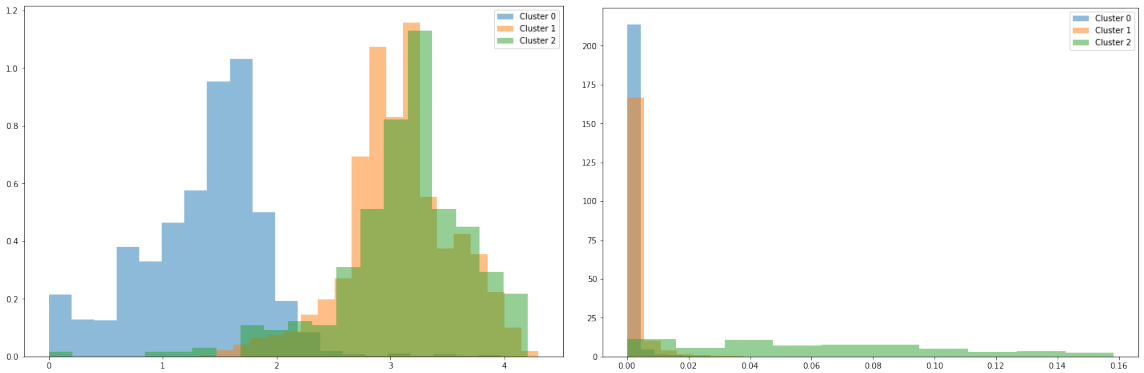
4.6 Comparison and interpretation

The results given by K-means, SOM and the hierarchical clustering are extremely similar, they found three clusters which are almost identical and this can be seen from the parallel coordinates and radar plots of the three algorithms and also from the tables of the median attributes values of the clusters. In fact also the value of the validation metrics (table 14) are almost identical. Therefore for the interpretation of the clusters we refer only to the K-means, but the same analysis can also be repeated for the other two algorithms. The cluster 0, which have more or less 6500 elements, have median values for the attribute which are very similar to the median values of the whole dataset, while cluster 1 and cluster 2 have median values which are higher than those of clusters 0. In particular if we consider the totals and the averages for `retwet_count`, `favorite_count` and `reply_count`, we notice that they are much higher for cluster 1 and 2, therefore we can say that clusters 1 and 2 represent users whose tweets receive more like, retweet and replies than those of users of cluster 0. This can be seen for the attribute `favorite_count_total` in figure 21-a, where we plot the histogram of the attribute for the three clusters, and we can see that users of clusters 1 and 2 received more likes for their tweets, than users of cluster 0. If we compare cluster 1 and 2, which have respectively 4500 and 300 users, we can notice they are very similar for almost any attribute except for those relative to the replies, in particular `reply_count_avg`, `reply_count_per_day` and `reply_count_std`, which are much higher in cluster 2, and this is evident if we look at the radar and parallel coordinates plots, and also from figure 21-b, where we show the histograms of the three clusters for `reply_count_avg`, and we see that there are many users in cluster 2 which have an average number of replies for their tweets which is much higher than the average number of replies for users from cluster 0 and 1. Therefore we can say that the difference between users of cluster 1 and 2, is that users of clusters 2 tend to have more replies on average for their tweets. If we instead consider the averages for `num_hashtags`, `num_mentions` and `num_urls` we see they are basically equal for all the clusters, therefore we can say that from this point of view the clusters do not differ.



(a) K-means (b) DBSCAN (0 is noise) (c) Hierarchical (d) SOM

Figure 20: Clusters size

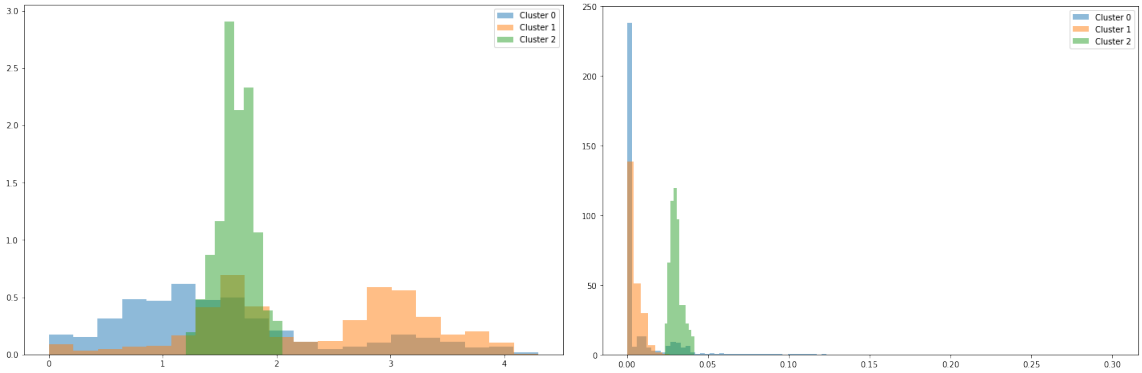


(a) `favorite_count_total` (b) `reply_count_avg`

Figure 21: K-means clusters distributions

The DBSCAN like the other three algorithms finds 3 clusters, but the composition of the clusters, in this case, is

much different. The noise cluster contains around 3500 samples, the big core cluster 7700, and the small core cluster 120. Again we can try to interpret the clusters by looking at the centers (figure 13) and the median values of the attributes (tables 7 and 8). We didn't compute the centers for the noise cluster, since the points of this cluster might be scattered around the whole space, therefore it doesn't make sense to consider the center. The only interesting fact that we report for the noise cluster is that 70% of the users in this cluster are bots, while in the whole dataset the percentage of bot users is 53%. Looking at the centers and at the median values, we can notice that the main difference between the two clusters, is that the bigger one has larger values for almost all of the attributes, except those related to replies: `reply_count_total`, `reply_count_avg`, `reply_count_std` and in particular `reply_count_entropy` which is much higher for the user of the smaller cluster, which is also shown in figure 22-b. Therefore in this case we can say that the algorithm found two clusters where the users of the bigger one receive more likes and retweets (like the histograms of `favorite_count_total` in figure 22-a shows), while the users of the second small cluster receiver more replies for their tweets. If we look at the value of three metrics for DBSCAN in table 14, we can see that is the worst model for all the metrics, therefore we can say that the density based approach in our case produced worse results than the other three algorithms.



(a) `favorite_count_total` (b) `reply_count_entropy`

Figure 22: DSCAB clusters distributions

Algorithm	SSE	Silhouette	Separation
K-means	223477.00	**0.21**	**1.49**
DBSCAN	298609.60	0.13	3.43
Hierarchical	227391.32	0.20	1.58
SOM	223901.60	**0.21**	1.50
X-means	**102724.27**	0.07	1.89

Table 14: Clustering metrics

5 Predictive Analysis

For the predictive analysis, we considered the same dataset that we used for clustering, therefore we remove correlating features and we apply a logarithmic transformation to left-skewed features. We split the dataset into a training set and a test set, which have respectively 80% and 20% of the total data. The division is made in a way that the balancing of the `bot` attribute on both datasets is the same as the original dataset (which is 53% for 1 and 47% for 0). We also reasoned on the possibility of encoding the attributes `lang` and `created_at`:

- `lang` could be encoded using one hot encoding. We were reluctant on using it, since as we saw, the distribution of the feature is almost totally dominated by the value `en` for both real and bot users, therefore we thought that the attribute isn't very informative. In the end we gave it a try and as expected it basically had no impact on the performances, therefore in the results that we present later `lang` has not been used.
- `created_at` could be encoded by converting the dates to their timestamps, or computing the number of days passed from the account creation date and a reference date. In the end we decided to discard the attribute since if we use it to train the model, and in the future we want to use the same model to classify new users, then the `created_at` field of these new accounts will vary on rage which is different from the one that we used for the training, causing the model we trained to be inadequate for those new accounts.

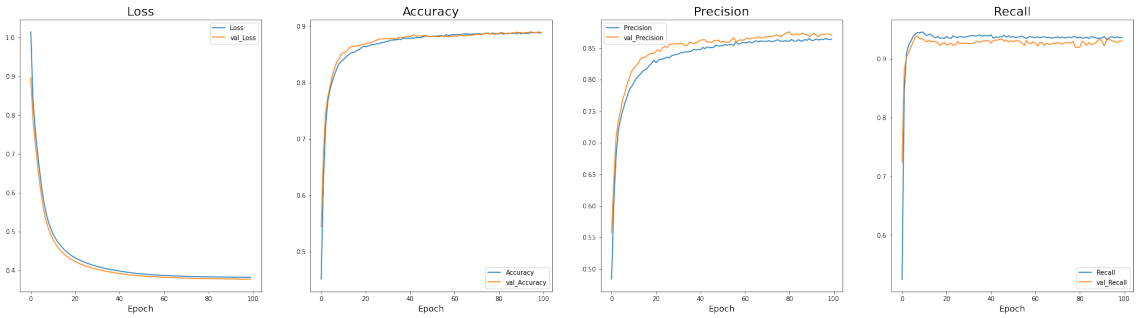


Figure 23: Neural network metrics over the epochs

For the data normalization, we applied standard normalization on the training set, and then we used the means and the standard deviations computed for the features of the training set to standardize also the data of the test set.

All the models that we implemented can be seen in the first column of table 15. For each model, we performed a grid search on all the hyper-parameters required to train the model on the validation set, except for the random forest where we applied a random search since, given the number of configurations and the time required to fit a single model, a complete gird search would have required too much time. The validation schema that we used is 5-fold cross-validation, therefore for each model, we select the combination of hyper-parameters that maximize the average validation accuracy on the 5 folds. After finding the best combination of hyper-parameter we execute a final re-training of the model on the whole training set, and then we measure the performances of the model on both the training and test set measuring accuracy, precision, recall, and the F1 score. The results for the training and the test sets are reported respectively in tables 15 and 16.

For the SVM we considered four possible kernels: linear, sigmoid, polynomial and radial basis function. For each kernel, we tested different values of the C hyper-parameter, which is used for the regularization of the model. Additionally, for the polynomial kernel, we tested different values for the degree of the polynomial and for the radial basis function we tested different values for the γ hyper-parameter. From the grid search, it turned out that the best model was the one using the RBF kernel with $\gamma = 0.01$ and $C = 100$.

For the k-nn we use as the distance the euclidean one, and we test in the gird search different values of k from 1 to 50. The best value we found for k is 17.

For the naive Bayes model, we use the Gaussian one, since most of our features are continuous.

For the decision tree we performed a grid search on the criterion to measure the quality of a split (Gini or entropy), the maximum depth of the tree (`max_depth`), the minimum number of samples for a leaf node (`min_samples_leaf`), the minimum number of samples to split an internal node (`min_samples_split`) and the number of features to consider for a split (`max_features`). The best values were entropy for the splitting criterion, `max_depth` = 20, `min_samples_leaf` = 2, `min_samples_split` = 2 and `max_features` = None (which means that we do not limit the number of features to consider for a split).

For the random forest, we performed a random search for over 100 possible configurations. The hyper-parameters considered were the same as the decision tree, as additionally also the number of estimators and whether we use bootstrap sampling or the entire dataset to build each estimator. The best values were 50 for the number of estimators, Gini for the splitting criterion, false for the bootstrap (which means that we fit each estimator using the whole dataset), `max_depth` = 36, `min_samples_leaf` = 4, `min_samples_split` = 2 and `max_features` = 'auto' (which means that we use the default behavior which is considering all the features for the split).

For the AdaBoost ensemble method we used as the base estimator a weak decision tree and we performed the grid search on the number of estimators and on the learning rate. The best values were 0.1 for the learning rate and 1000 for the number of estimators.

We adopted the TensorFlow library for the neural network classifier. We first did some preliminary tests to figure out the architecture of the network. The final network has two hidden layers, in particular: the input layer has 27 units, the first hidden layer has 8 units, the second hidden layer has 16 units, and the output layer has only one unit. For the two hidden layers, we used the ReLU activation function, while for the output layer, we used the sigmoid activation function. The weights initializer is the Glorot uniform, the biases are initialized to zero, the regularizer is the L2 and we used Adam as optimizer. Once we fixed the architecture we did a grid search on the learning rate and the best value was 0.0002. For the training, we used the binary cross entropy loss function, the number of epochs was fixed to 100 and the batch size is 4.

Model	Accuracy	Precision	Recall	F1 score
SVM	0.95	0.95	0.95	0.95
K-nn	0.91	0.90	0.92	0.91
Naive Bayes	0.70	0.71	0.74	0.73
Decision Tree	0.96	0.94	0.99	0.96
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**
AdaBoost	0.97	0.97	0.98	0.97
Neural Network	0.89	0.86	0.94	0.90

Table 15: Performances of the model on the train set

Model	Accuracy	Precision	Recall	F1 score
SVM	0.93	0.92	0.94	0.93
K-nn	0.89	0.89	0.92	0.90
Naive Bayes	0.70	0.71	0.73	0.72
Decision Tree	0.93	0.90	0.97	0.94
Random Forest	0.96	0.94	**0.98**	0.96
AdaBoost	**0.97**	**0.96**	**0.98**	**0.97**
Neural Network	0.88	0.85	0.93	0.89

Table 16: Performances of the model on the test set

Table 16 shows that the worst model is the naive Bayes, which has values around 0.7 for all the metrics. In contrast, all the other models reach values at least around 0.9 on the test set, especially the decision tree and the SVM perform slightly better than the K-nn and the neural network. The tree ensemble methods increase even more the performances of decision tree, reaching values for the metrics on the test set above 0.95.

6 Explainability

6.1 Interpretable by design models

6.1.1 Explainable Boosting Machine

The first explainable by design method we implemented was the Explainable Boosting Machine (EBM) from the `interpretml` library. We fit the model again by using 20% of the dataset for the testing. The performances of the model are reported in table 17, which are in line with the performances that we obtain for the tree ensemble methods in the predictive analysis. Then we compute the global features importance (figure 24) from which we can see that the most important attribute is `statues_count`. We can also visualize how much each feature contributes to local predictions, as shown for one sample in figure 25, where we can see that for that prediction the features that contribute more are the ones that are also globally the most important ones, even if they might change according to the sample since in this case, they are local explanations.

Dataset	Accuracy	Precision	Recall	F1 score
Train	0.99	0.99	1.00	0.99
Test	0.97	0.96	0.98	0.97

Table 17: Performances of EBM on the training and test set

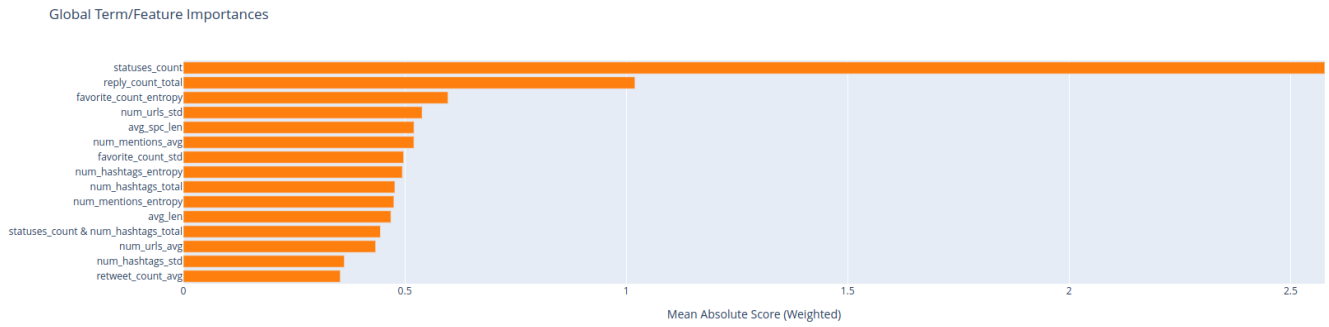


Figure 24: EBM global feature importance

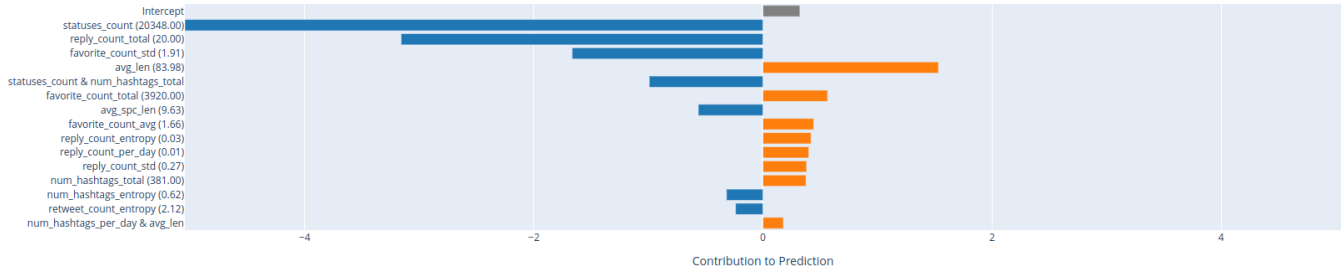


Figure 25: EBM features contribution for the prediction of one sample

6.1.2 TabNet

The second model we tested is TabNet, which is a neural network that uses attention layers to determine which features to reason from at each decision step. There are some hyper-parameters to tune, since the training process of just one single model is quite slow, we didn't apply a grid serach, but we tested manually different configurations of hyper-parameters, until we found a good one. The performances of the model are shown in table 18. To understand the relevance that the model gives to each feature we can plot the aggregated attention mask (figure 26) and the average value of the attention for each feature (figure 26). From the plots is clear that for the model the most important feature is `num_hashtags_std`, followed by `favorite_count_entropy`.

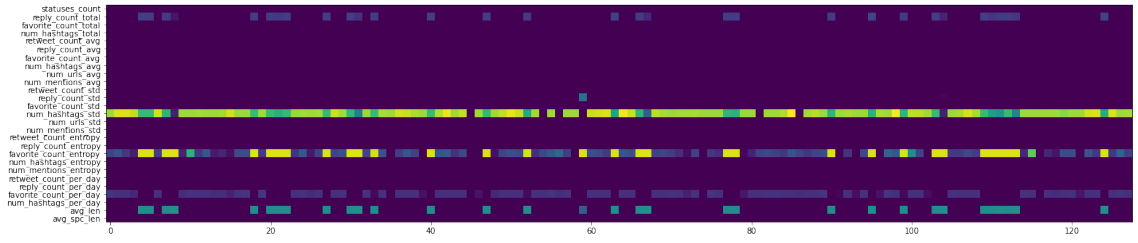


Figure 26: Tabnet aggregated attention mask

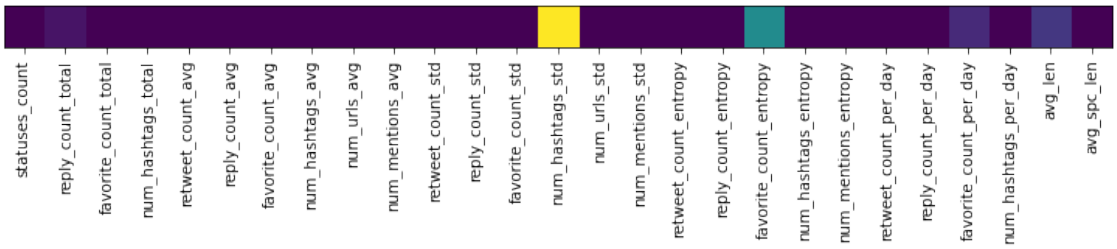


Figure 27: Tabnet average attention per feature

Dataset	Accuracy	Precision	Recall	F1 score
Train	0.88	0.82	0.98	0.89
Test	0.89	0.84	0.98	0.91

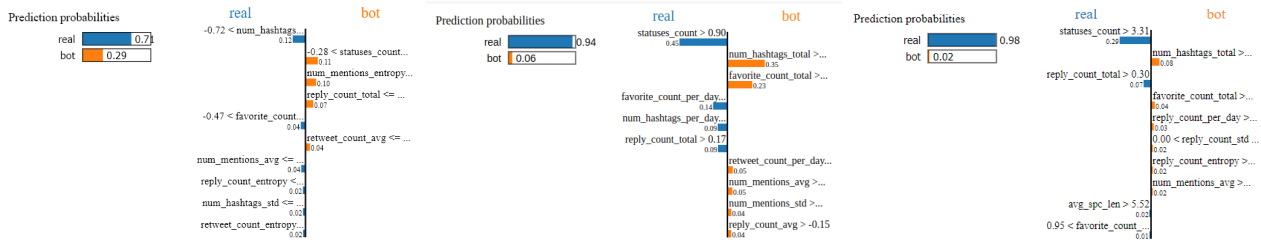
Table 18: Performances of TabNet on the training and test set

6.2 Post training methods

After having tested EBM and TabNet, we applied two post-training XAI techniques, LIME and SHAP, to some of the not interpretable classifiers that we used for the predictive analysis, namely the SVM, the neural network and the random forest.

6.2.1 LIME

We applied the LIME algorithm, which is a local XAI method, therefore we tested it on some of the samples of our dataset to get the contribution of each feature to the prediction of those specific samples, using 5000 as the number of neighbors to generate for the local approximation of the model done by LIME. In figure 28 we can see the feature importance that the method returned for one sample, noticing that `statuses_count`, `num_hashtags_total` and `reply_count_total` are considered for all the models as the features which contribute the most to the result of the prediction, while the contribution of other features changes according to the model. We have to take into account that since LIME is a local XAI method, therefore results may change if we use it on different samples, moreover the local approximation also depends on the random generation of the neighbors around the sample to explain, which might affect the results.



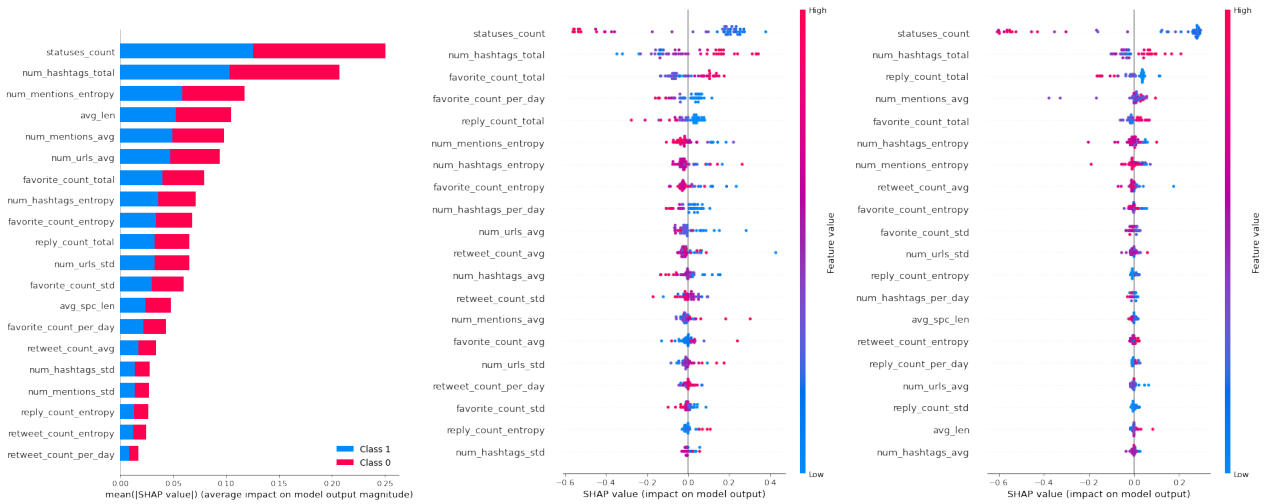
(a) Neural network (b) SVM (c) Random forest

Figure 28: LIME features contribution to the predition of one sample

6.2.2 SHAP

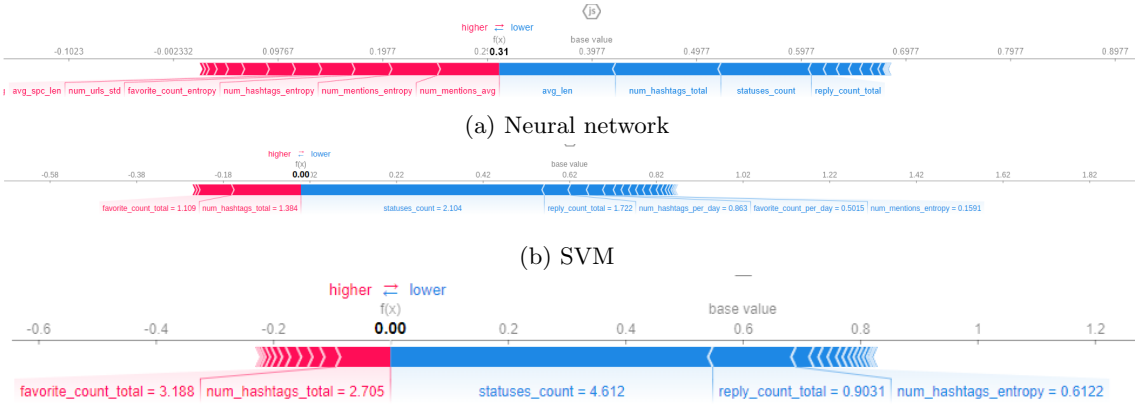
We applied SHAP using the kernel explainer for the SVM and the neural network, while for the random forest, we used the tree explainer. In any case, we fit the explainer using 50 samples from our dataset, due to the training process being really slow. From the summary plots (29) we can see that for all three models, the two features which contribute the most to the predictions are `statuses_count` and `num_hashtags_total`, while the others features have different importances according to the model. For these two features, we also show the dependency plot (figure 32) for the case of the SVM. For `statuses_count` we can see that when the values of the attribute are low, the SHAP values are high, which means that the model is pushed towards predicting 1 (which is bot in our case), while as the values of `statuses_count` increase, the SHAP values decrease, which means that the model is pushed towards predicting 0 (real user). Conversely, for `num_hashtags_total` low values of the attribute push the model towards predicting the user as real, and higher values of the feature push the model towards predicting the user as a bot. This is also confirmed by the force plot for `statuses_count` and `num_hashtags_total` (figure 31) where on the x axis we have the values of the feature and on the y axis we have the impact that the value of the feature has on the output of the model. Again we can see that for `statuses_count` as its values increase, the model is pushed towards predicting the user as real, while for low values it pushes to predict the user as a bot. For `num_hashtags_total` we have the opposite situation.

In figure 30 we can see the force plots for a local prediction. The blue features are the ones that push to model to output 0 (real users), while the red features push the model to output 1. For example, in the case of the SVM and the random forest, we can see that the high value of `statuses_count` pushes the model to predict 0 (so to predict the user as real).



(a) Neural network (b) SVM (c) Random forest

Figure 29: SHAP summary plot, the neural network plot is a bar plot because of the output of the model is continuous

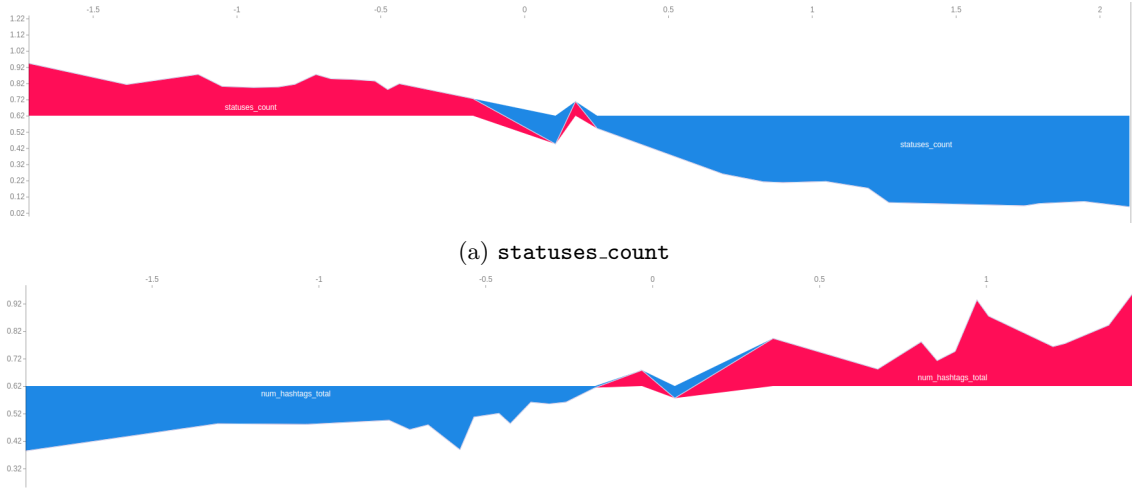


(a) Neural network

(b) SVM

(c) Random forest

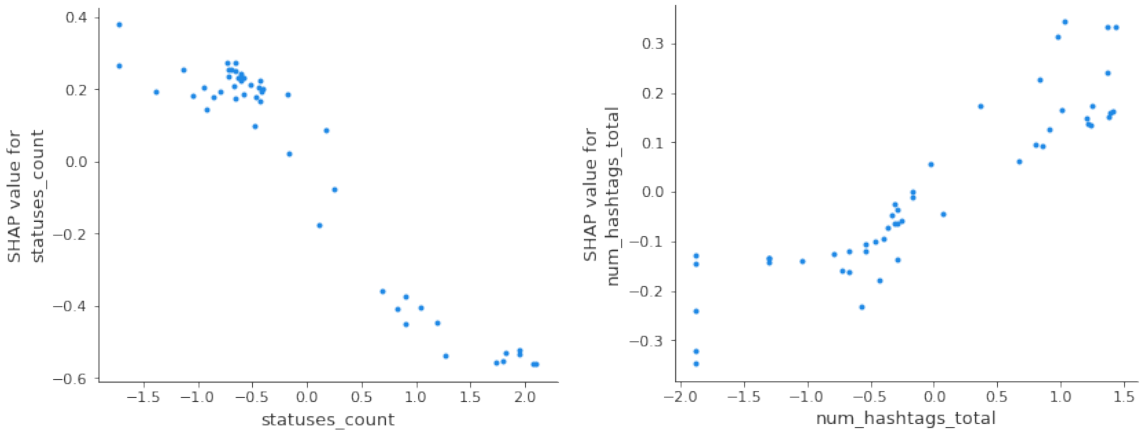
Figure 30: SHAP force plot for a single sample



(a) `statuses_count`

(b) `num_hashtags_total`

Figure 31: SVM shap force plot for `statuses_count` and `num_hashtags_total`



(a) `statuses_count` (b) `num_hashtags_total`

Figure 32: SVM SHAP dependency plots for `statuses_count` and `num_hashtags_total`

6.3 Results

For what concerns the interpretable by-design models, we can see that their performances are in line with the corresponding models that we used for the classification analysis (tree ensembles for EBM and neural networks for TabNet), therefore interpretability didn't affect performances in our case. Anyway, the models come to different conclusions, since according to EBM global features importance the most important feature is `statuses_count`, followed by `reply_count_total`, which are both not considered by TabNet. According to TabNet attention mask, the most relevant feature is `num_hashtags_std` which for EBM is only the 14th most important feature. The only feature that both models consider important is `favorite_count_entropy` which is the 3rd most important feature for EBM and the 2nd for TabNet (even though its importance is far from what both models consider to be the most relevant feature).

The post-hoc methods, agrees that `statuses_count` is important for discriminating between the two classes, but differently from the interpretable by design model they also consider important `num_hashtags_total`.

In table 19 we compare the results obtain from LIME and SHAP, showing for both methods the average and the standard deviation of the faithfulness over 100 samples, and also for how many samples the monotonicity has been evaluated to true. The SVM has quite good faithfulness for LIME, and pretty low for SHAP even if in both cases the standard deviations are high, which means that for some samples the faithfulness is good and for others it is bad. The monotonicity instead is never evaluated to be true. For the random forest instead, the average faithfulness is low for LIME and a bit better for SHAP but still quite low, and even in this case the standard deviations are high. For the neural network, the average LIME faithfulness is high, but on SHAP is still low. Moreover, for the nerual newtork, the standard deviations are much lower and the monotonicity gets evaluated to be true for almost half of the samples for LIME and for all the samples in SHAP.

	LIME faithfulness avg	SHAP faithfulness avg	LIME faithfulness std	SHAP faithfulness std	LIME monotonicity	SHAP monotonicity
SVM	0.4750	0.1772	0.3004	0.3745	0/100	0/100
Random forest	0.1920	**0.2490**	0.3918	0.3440	2/100	0/100
Neural network	**0.6337**	0.1597	**0.1648**	**0**	**43/100**	**100/100**

Table 19: LIME and SHAP evaluation metrics