UNIVERSITÀ DI PISA

Computer Science Department

Data Mining Report

Group 7

Bots on twitter

Anna Monreale
Francesca Naretto
Lorenzo Mannocci

Tengel Fredheim
Paul M. Magos
Gianluca Morcaldi
Matteo Tolloso

Pisa, January 2023

Introduction

This report aims to explore the contents of the Twitter dataset supplied by course 309AA - Data Mining at the University of Pisa [1]. The work (bots_on_twitter)[1] has focused on clean the data of unwanted noise that may hinder knowledge discovery; apply and compare different clustering and classification techniques for deeper insight and learning; finally, investigate techniques to explain and interpret "black box"-predictions to better understand its results.[17]

Contents

1 Data understanding and Preparation (Task 1) 2
- 1.1 Datasets cleaning . . . 2
- 1.2 Users dataset cleaning . . . 2
- 1.3 Tweets dataset cleaning . . . 2
- 1.4 Merging the datasets and creating indicators . . . 5
- 1.5 Correlation Analysis . . . 6

2 Clustering Analysis (Task 2) 7
- 2.1 K-Means . . . 7
- 2.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN) . . . 9
- 2.3 Hierarchical . . . 11
- 2.4 X-means . . . 13

3 Predictive Analysis (Task 3) 14
- 3.1 Decision Tree Classification . . . 14
- 3.2 Neural network . . . 16
- 3.3 Naive Bayes . . . 18
- 3.4 AdaBoost . . . 19

4 Time Series and Explainable AI (Task 4) 20
- 4.1 Time Series Analysis . . . 20
- 4.2 Explanation Analysis . . . 21
  - 4.2.1 EBM . . . 21
  - 4.2.2 LIME . . . 23
  - 4.2.3 Shap . . . 24

5 Final Summary 25
- 5.1 Data Understanding and Results . . . 25
- 5.2 Clustering Evaluation and Results . . . 25
- 5.3 Predictive Analysis Evaluation and Possible Hindrances . . . 25
- 5.4 Explanation Analysis Evaluation . . . 25
- 5.5 Time Series Analysis Evaluation . . . 26

[1] https://github.com/The-seven-dwarfs/bots_on_twitter

1 Data understanding and Preparation (Task 1)

1.1 Datasets cleaning

This part contains what have been done to clean the datasets and make them suitable for further analysis. In Subsection 1.2 are described the operations performed on the dataset containing users' data, while Subsection 1.3 concentrates on the tweets dataset.

1.2 Users dataset cleaning

The dataset containing the information about the users is composed of 11508 entries and for each entry there are the following columns:

- *id*: the id of the user;
- *lang*: the user's selected language; further analysis on the data showed that this field is described using the IETF language codes [6].
- *bot*: a binary value indicating whether the user is a bot or not.
- *created_at*: the time when the user's profile has been created.
- *statuses_count*: it's the count of tweets made by the user at the moment of data crawling. This number includes the number of retweets, but not replies [16].

Regarding the fields *id*, *lang* and *bot* the cleaning didn't require any special fix: the values were almost all correct and just some minor adjustments were needed, like removing *NaN* values or removing those values which weren't semantically valid (especially for the *lang* field). Concerning the *created_at* field, there were checked all the accounts' creation date to verify the presence of any user whose account has been created before the first tweet ever made on twitter (which is the 21 March 2006, at 9.50 AM) or after the release date of the project (which is the 29 September 2022, at 11 AM).

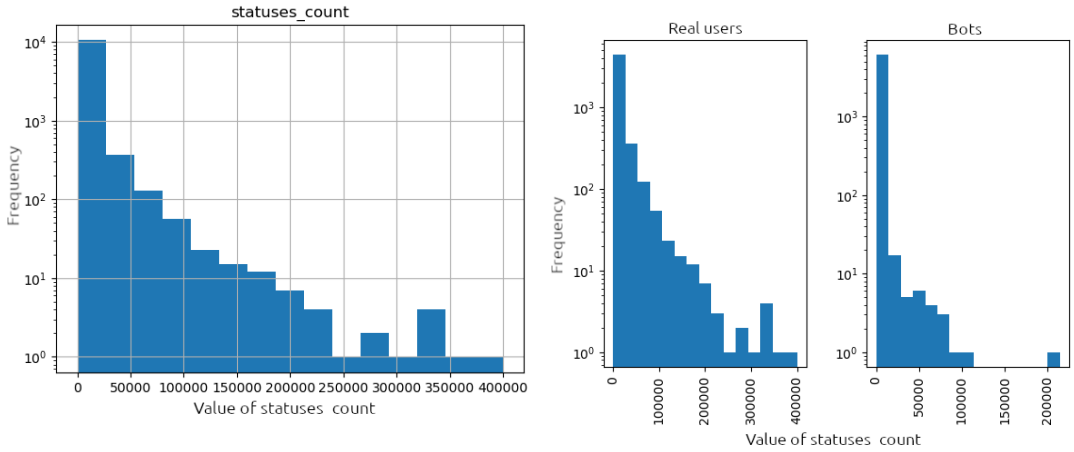
Instead, the analysis of the *statuses_count* of each user has proved more interesting. As can be seen in Figure 1.1a the values follow a power-law distribution, with many (few) users making a low (high) number of tweets; as shown in Figure 1.1b the same behaviour has been observed for the sub populations of bots and real users.

Moreover, since this field is the only numeric field in the dataset, outliers detection was performed on it: by the use of a boxplot the values above the upper whisker were substituted with the median of the distribution. This affected approximately 18% of the data (i.e. 2080 entries) and it didn't alter the values distribution.

1.3 Tweets dataset cleaning

This dataset contains the information about tweets and is formed by approximately 13 million tweets, with each tweet described by the following fields:

- *id*: a unique identifier for the tweet;
- *user_id*: a unique identifier for the user who wrote the tweet;
- *retweet_count*: number of retweets for the tweet in analysis;



(a) Overall (b) Sub populations

Figure 1.1: Frequency distribution for the *statuses_count* field in the users' dataset

- *reply_count*: number of reply for the tweet in analysis;
- *favorite_count*: number of favorites (likes) received by the tweet;
- *num_hashtags*: number of hashtags used in the tweet;
- *num_urls*: number of urls in the tweet;
- *num_mentions*: number of mentions in the tweet;
- *created_at*: when the tweet was created;
- *text*: the text of the tweet.

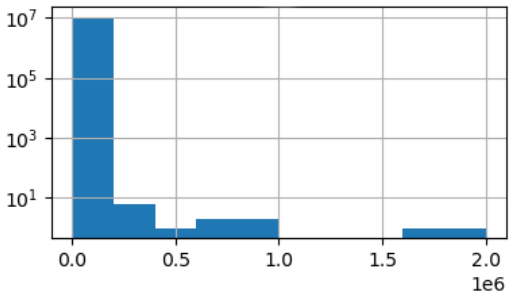
The first thing done on this dataset was keeping only the tweets whose author is saved in the users dataset; in fact, since the aim of this work is to be able to discriminate bots and real users, there wasn't any utility in keeping tweets that were not able to give any information about users. Then, all the duplicates were removed and some minor cleaning operations have been done (i.e. removing entries with NaN values, semantically invalid values and so on.).

Regarding the fields *retweet_count*, *reply_count*, *favorite_count*, *num_hashtags*, *num_urls* and *num_mentions* they represent discrete positive values, so invalid values such as infinity, floating point and negative values were removed. Furthermore, thresholds were imposed on these fields; some of these threshold can be found by looking at the maximum value that a field (i.e. retweet count and favorite count) has ever reached on the Twitter platform, while other values had to be inferred[2]. Details about the entity of the thresholds and how many values were above them are shown in Table 1.

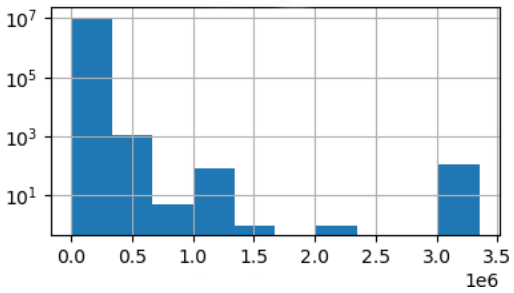
Table 1: Threshold applied on numeric fields

Field	Threshold	Removed values
retweet_count	3738380 [9]	36
favorite_count	7114892 [8]	34
reply_count	4000000	42
num_hashtags	93	279
num_urls	23	469
num_mentions	93	381

Finally, there were studied the distributions of the numeric fields in the dataset. Because of space constraints, only the attributes *reply_count* and *retweet_count* have been reported here; however all the numeric fields in this dataset have shown a very similar behaviour, so the conclusions drawn here can be extended to the other attributes. In Figure 1.2c are shown the distributions for the two attributes, highlighting how even here we have a power-law shaped distribution; in Figure 1.2d the boxplots for these 2 attributes are displayed.

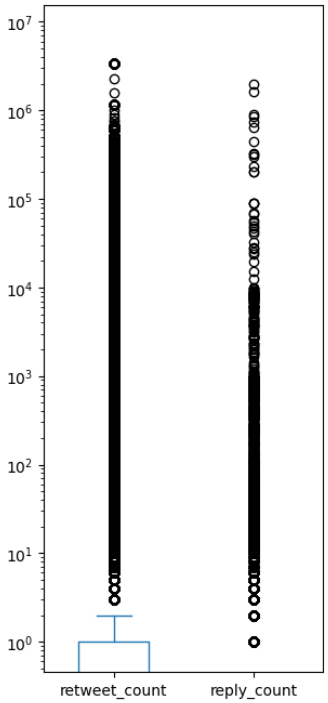


(a) Field *reply_count*



(b) Field *retweet_count*

(c) Distribution of the fields



(d) Boxplots of the fields

Figure 1.2: Frequency distribution for the *statuses_count* field in the users' dataset

[2]For example, the *num_hashtags* has been computed as the maximum possible length of a tweet (280 characters) divided by the minimum possible length of an hashtag (3 characters).

1.4 Merging the datasets and creating indicators

The analysis proceeded by merging the datasets; this has been done by associating each tweet information with the information about its author. This led to the creation of a dataset of around 10 million entries.

Next, new features (indicators) have been created from the existing data, with the aim of making interesting attributes that could be useful for the clustering phase. This process has led to the formulation of the following 26 indicators:

- Account age in days;
- Account tweets number;
- Account highest daily tweet count; that's it, the maximum number of tweets that the user has ever produced in a day;
- Account average number of tweets per day;
- Account average number of tweets per actually day; i.e. the number of tweets divided per the number of days the user has actually produced at least one tweet.
- Day with the maximum number of tweets produced;
- Account entropy per timedeltas; account entropy measured for each day, hour and minute;
- Account average hasthag use per tweet;
- Account average tweet text length;
- Account average number of mentions per tweet;
- Account average number of special characters in text;
- Account number of likes;
- Account average number of likes per tweet;
- Account number of comments;
- Account average number of comments per tweet;
- Account discussion creation score; ratio between number of tweets the user has made and the total number of retweets he got.
- Ratio between number of tweets and number of likes;
- Ratio between number of tweets and number of comments;
- Entropy to be a post different from the others of the same user; this indicator defines the probability to be an original post over the total number of posts of the same user with the same text (i.e. 1 over the number of tweets with the same text).
- Entropy to be a post different from all the post with the same text among all the available tweets in the dataset.

- Mean of account length of inactive time periods;
- Median of account length of inactive time periods;
- Mode of account length of inactive time periods;
- Mode count; how many times the mode computed by the indicator above appears.

So, at the end of this cleaning phase, there is just one merged and cleaned dataset formed by 11109 entries; each entry describes a user by the use of 33 attributes: 7 from the original users dataset and 26 new indicators.

1.5 Correlation Analysis

The analysis then moved on the study of existing correlations among the attributes, as can be seen in Figure 1.3.

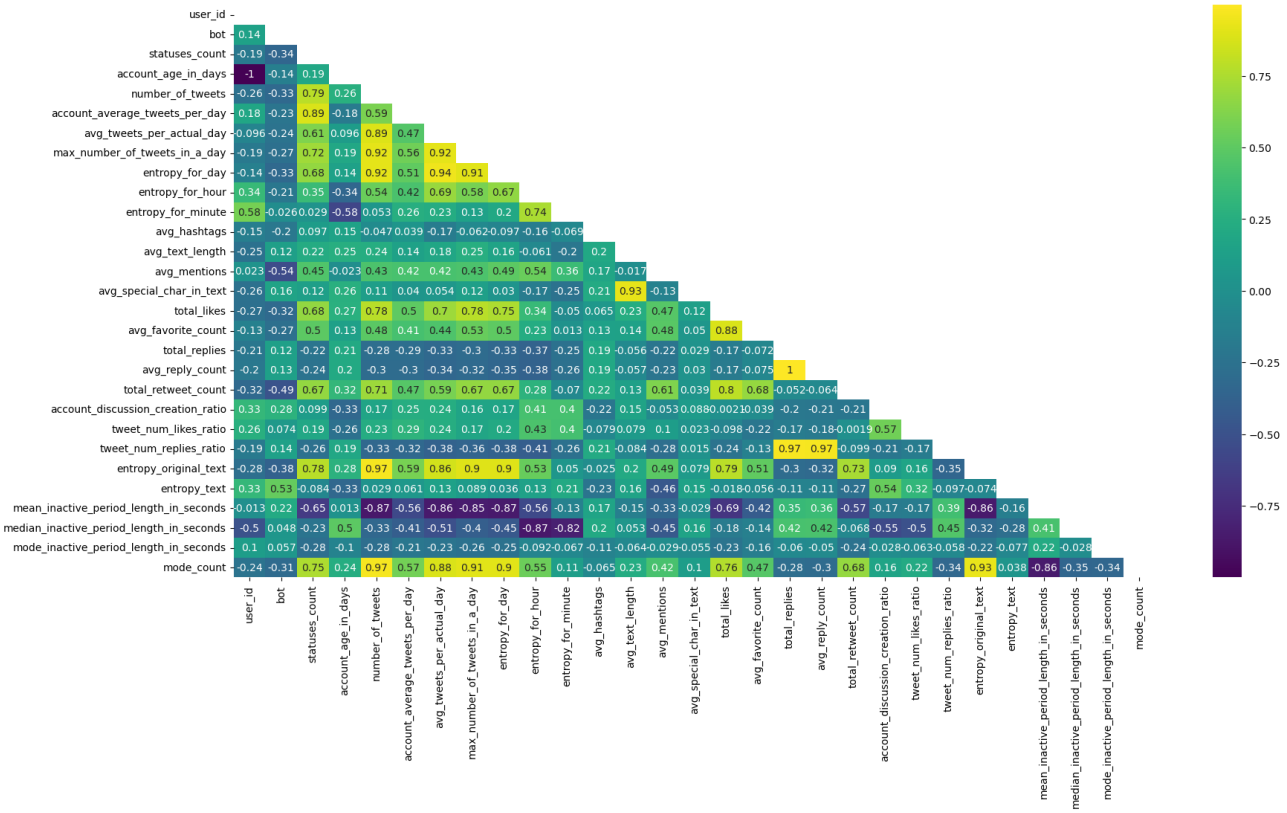


Figure 1.3: Correlation matrix heatmap

Highly correlated attributes have been removed during the clustering phase.

2 Clustering Analysis (Task 2)

2.1 K-Means

In this section we describe the approach and the decisions taken in the process of K-Means clustering of the users in the dataset. The goal of this phase was to find things that we consider interesting.

We first defined a way to filter the attributes that were correlated in a certain way. The first idea was to use the Spearman's rank correlation coefficient to exclude the attributes that had a correlation above 0.7. This led us to results that were not interesting, so we have decided to expand the filter to use 0.5 and 0.3 correlations and eventually 0.1 Pearson's correlation.

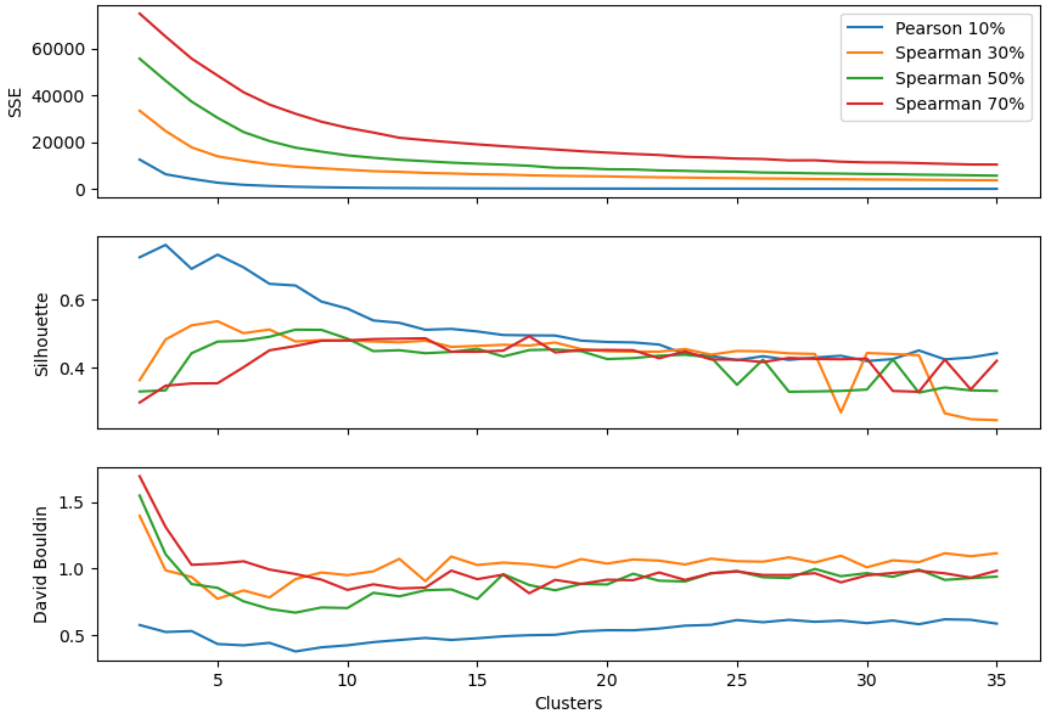
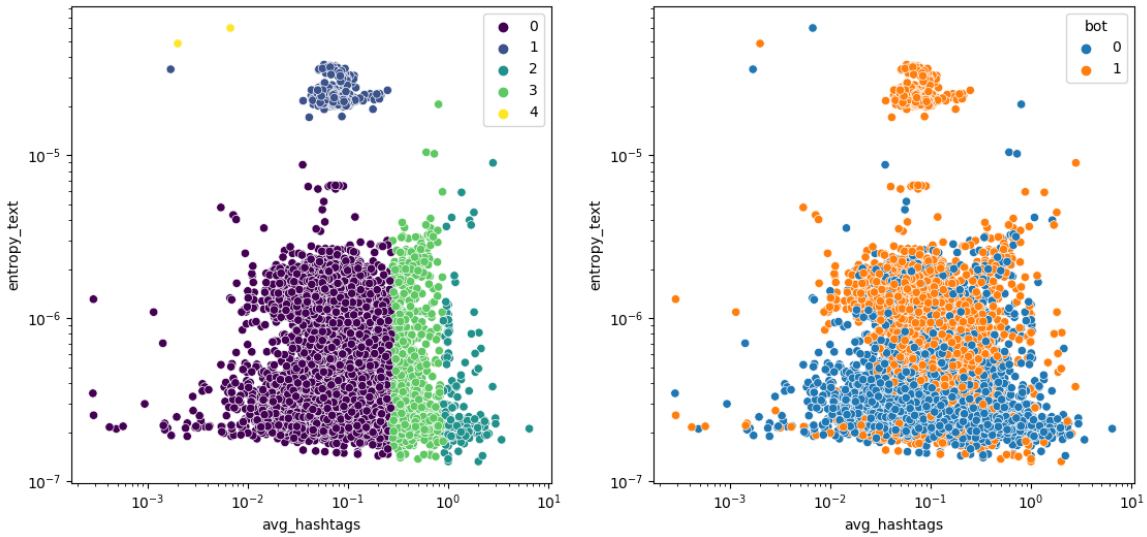


Figure 2.1: Scores of K-Means Elbow Method with k=2, ..., 35

In the Figure 2.1 we have a clear picture of the behaviour obtained by K-Means Clustering with different K (clusters) and we can see that all of the scores are better in the range between 5 and 10. Particularly we can see how the lines in the case of Pearson with 0.1 are the best among the others. This approach of comparing the results of SSE above many executions with different K is called Elbow Method and the goal of it is to find the best K from which the SSE starts to decrease in a linear way. In our case the best K could be between 5 and 10 as we have seen before, but choosing a K greater than 5 will bring us a lower Silhouette score so we will choose a K equal to 5.

Since we have found our best K we proceeded by analyzing the clustering of the users among the attributes with Pearson's correlation below 0.1; Those attributes are *avg_hashtags* and *entropy_text*. Using those attributes and a K equal to 5 led us to some interesting results, in the figure 2.2 we can clearly distinguish a cluster of bot in the higher part of the plots that's recognized as a unique entity by our clustering. By analyzing the distribution of the users in the clusters (Fig. 2.3a), it's clear that we have created two clusters made up of bots only. Since these results were gained by using the average hashtags used by the user and the

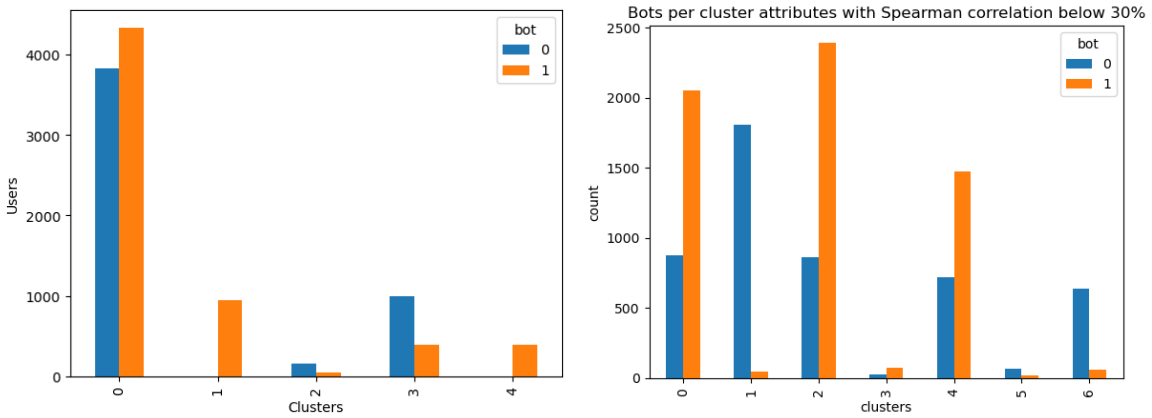


(a) K-Means Clusters (b) Actual Bots

Figure 2.2: Plots of the results obtained by the K-Means Clustering with k=5 on the two attributes: *avg_hashtags* and *entropy_text*

entropy of his tweets text, we believe that this led us to find some of the so called Spammers. The Spammers are bots that creates a large amount of tweets with the same content (e.g. advertising purposes).

In conclusion we can say that at the best of our knowledge this algorithm itself it's not enough to distinguish the bots from the users, but can be used to find some kind of bots as we've seen, this doesn't mean that other K-Means Clustering attempts are not welcome. Furthermore we can see that the same analysis made with Spearman on 30% correlation (Fig. 2.3b) gave us an equally interesting result with two clusters composed almost only of real users.



(a) Pearson (b) Spearman

Figure 2.3: Bar plot with the clusters created

2.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

The advantages of density-based clustering algorithms is that we are able to find clusters of arbitrary shape, as well as finding outliers (noise points) within our dataset, should it be of interest. The algorithm will label each point in our dataset as either a core, border or noise point. Which can be of use during noise removal.

The disadvantages of this clustering algorithm is that it does not discern between clusters of different densities. Additionally is has two parameters which must be fine tuned manually in order to find relevant clusters. This can be hard as it is an exploratory process based on experience. For this reason an extension of DBSCAN called OPTICS with semi-automated parameter tuning and density discernibility was used for result comparison.(Fig. 2.5 & 2.6) OPTICS achieves this dynamic density clustering through ordering and reachability [4].

For DBSCAN The parameters to be tuned are:

1. min_samples: The number of samples in a neighborhood for a point to be considered as a core point. This includes the point itself.

2. eps: The maximum distance between two samples for them to be considered as in the same neighborhood.

Following the exploratory process described in subsection 2.1, where we clustered our data based on the removal of features with different levels of correlation. We continued this process with DBSCAN in order to compare the results of each clustering method more easily. Seeing that our data had value differences in many orders of magnitude we scaled our data using StandardScaler.

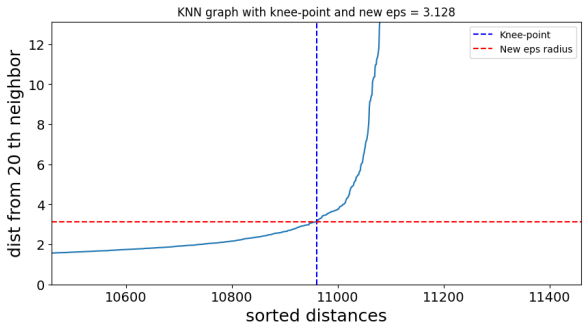


Figure 2.4: Optimal eps value calculation using the python kneed module

We solved the tuning of min_samples by following the rule of thumb provided by Tara Mullin in her article on medium.[14] min_samples = 2*dimensionality[15]

According to Ramah and Sitanggang 2016, the optimal epsilon value for each dimensionailty(number of features) can be found by calculating the knee point at the distance from the K-th nearest neighbour. Where K is the min_samples value.[11] We created a script to find and plot the optimal eps value automatically. (Fig. 2.4)

Once the parameters were calculated, we could create and run a script that would fine tune these parameters automatically and plot the results based on different correlation thresholds.(Fig. 2.5a & 2.5b)

In figure 2.6, noise points are assigned a cluster label of -1. Found clusters are given labels from 0 to the number of clusters found, ranked by cluster size.

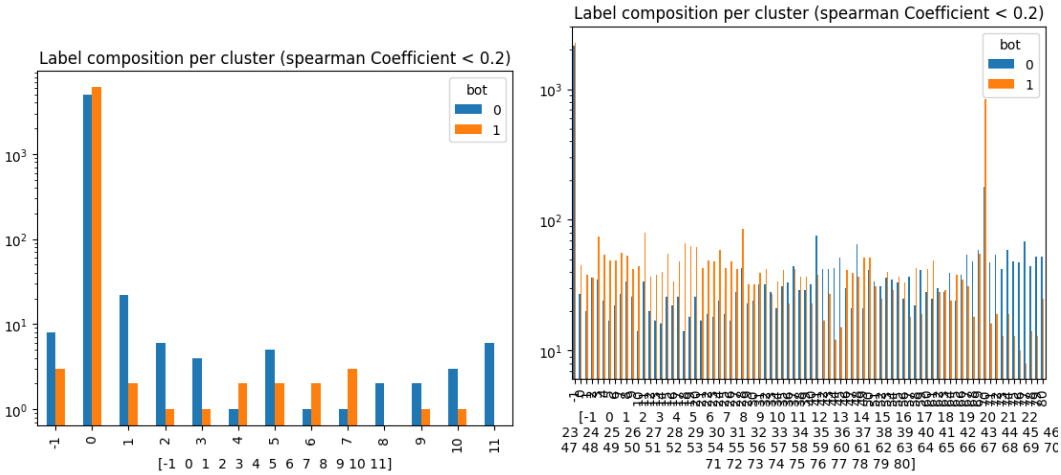
	Correlation Method	Correlation Threshold	new eps	minimum number of samples	number of dimensions	number of clusters	number of noise points	Homogeneity	Completeness	V-measure	Adjusted Rand Index	Adjusted Mutual Information	Silhouette	Davies Bouldin
0.1	pearson	0.1	0.327	4	2	2	29	0.000169	0.005283	0.000328	0.000125	0.000051	0.992574	1.546842
0.2	spearman	0.2	0.108	2	1	12	11	0.003471	0.041218	0.006402	0.001909	0.004689	0.844982	1.657026
0.3	spearman	0.3	1.056	6	3	1	54	0.000053	0.001191	0.000102	-0.000239	-0.000025	0.890416	1.213045
0.5	spearman	0.5	9.281	12	6	1	12	0.000057	0.004602	0.000112	0.000118	-0.000024	0.943064	1.337073
0.7	spearman	0.7	3.128	20	10	1	90	0.000061	0.000891	0.000114	0.000329	-0.000009	0.813939	2.206009
0.9	spearman	0.9	9.148	38	19	1	53	0.000119	0.002698	0.000227	-0.00035	0.000101	0.876943	2.101347

(a) DBSCAN Cluster parameters and scores at different correlation thresholds

	Correlation Method	Correlation Threshold	cluster_method	minimum number of samples	number of dimensions	number of clusters	number of noise points	Homogeneity	Completeness	V-measure	Adjusted Rand Index	Adjusted Mutual Information	Silhouette	Davies Bouldin
0.1	pearson	0.1	xi	4	2	42	3602	0.097463	0.022922	0.037115	0.009229	0.036101	0.288127	2.305912
0.2	spearman	0.2	xi	2	1	81	4442	0.071376	0.015609	0.025616	0.004515	0.023746	0.061292	12.218433
0.3	spearman	0.3	xi	6	3	4	10063	0.073741	0.134739	0.095316	-0.00539	0.095008	-0.41139	1.55467
0.5	spearman	0.5	xi	12	6	3	10094	0.067379	0.128139	0.088318	-0.006482	0.088082	-0.166698	1.25737
0.7	spearman	0.7	xi	20	10	2	9996	0.093496	0.165906	0.119595	-0.005283	0.119447	0.180345	0.797098
0.9	spearman	0.9	xi	38	19	3	6780	0.171897	0.123215	0.143541	0.080917	0.1434	0.192925	1.305517

(b) OPTICS Cluster parameters and scores at different spearman correlation thresholds

Figure 2.5: Comparing our results between our manually tuned clustering parameters and parameters tuned by OPTICS at the same correlation thresholds



(a) Manually Tuned

(b) Optics tuned

Figure 2.6: Visually comparing manually tuned best results and cluster sizes with optics clustering methods at similar correlation threshold

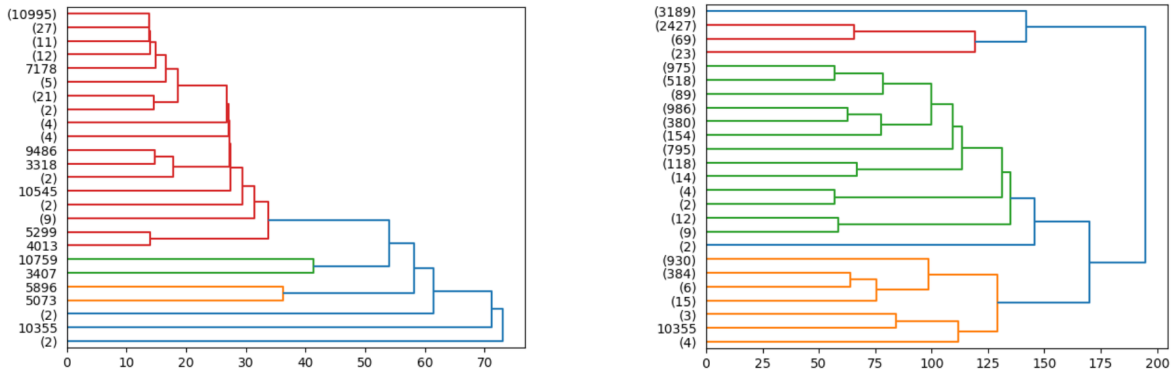
Observing figures 2.5 and 2.6 we can clearly tell that DBSCAN is able to seperate some clusters from the main user behaviour (cluster 0), but having a harder time separating samples into multiple clusters compared to OPTICS. We can also observe that OPTICS is able to find more clusters in general. DBSCAN is also placing most users into the same cluster. The reasons for these results could be:

1. Inadequate DBSCAN parameter tuning
2. DBSCAN is unable to separate behavioural clusters of differing densities
3. Bots are well designed, able to convincingly mimic human behaviour, thus creating clusters with a high degree of overlap between the classes.

2.3 Hierarchical

In this part we tried to analyze our user dataset with the just created attributes exploiting the features of hierarchical clustering and trying to retrieving the best flat partition.

We performed the agglomerative hierarchical clustering with all possible variants of the algorithm, i.e. all possible inter-cluster distances using all the attributes with Pearson correlation below 0.7 (and without the categorical ones). We will show that the majority of the inter-cluster measures (as well as the correlation coefficients different from our choice) give us very poor clustering results. The main problem is that both with a high or low number of clusters the cardinality of them is highly unbalanced, with usually one cluster containing 95-99% of points, you can see an example on figure 2.7a with average distance. The best inter-cluster metric we found is Ward's distance, shown in figure 2.7b.



(a) Average distance

(b) Ward's distance

Figure 2.7: Dendrograms with 25 leaf clusters, if the label is in parentheses it is the cardinality of that cluster, otherwise it is a single point

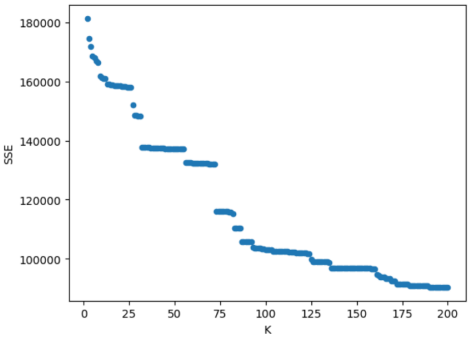


Figure 2.8: SSE of the hierarchical clustering using Ward's distance related to the number of clusters K.

Ward's distance has been shown to be able to create balanced clusters of users that deserve to be analyzed, while it's impossible to perform further analysis with the other metrics, since as you can see from figure 2.7a, almost all points are in one cluster. We can also see that the distances between couples of clusters is higher with Ward compared with other metrics, sign that it is better in separate users.

The main advantage of hierarchical clustering is that we do not have to assume any particular number of clusters because any desired number can be obtained by cutting the dendrogram. We want to try to establish the best number of clusters taking into account the SSE and the meaning of the resulting clusters. In figure 2.8 we evaluated the SSE of all the cutting of the dendrogram from 0 to 200 clusters.

The general rule is to choose K where the slope of the curve approaches zero, that in this case is both after 25 and after 100 clusters.

We think that K near to 100 is not a good choice for two reason. The first one is semantic, in fact if either the objective of the analysis is to identify bots or groups of people with similar

behaviour even with perfect balanced cluster we would have only 100 observation in each cluster, that is a too low number and could be easy influenced by noise and outliers. The second reason is indeed that the clusters are still highly unbalanced, as discussed above.

cluster name	size	bot percentage
0	646	0.63
1	10	0.70
2	5940	0.66
3	35	0.00
4	76	0.65
5	2	1.00
6	34	0.64
7	22	0.50
8	122	0.65
9	42	0.66
10	7	0.85
11	37	0.13
12	579	0.01
13	9	0.00
14	221	0.01
15	60	0.00
16	11	0.00
17	20	0.00
18	3	1.00
19	1	0.00
20	2	0.00
21	12	0.00
22	11	0.90
23	6	1.0
24	6	0.00
25	171	0.01
26	2178	0.67
27	839	0.03
28	6	0.66
29	1	0.00

Table 2: Size and bot percentage of the flat partition with 30 cluster of the hierarchical clustering with Ward's distance.

After some test about the composition of the flat partition we noticed that the shoulder of the function between 25 and 75 is the perfect compromise between cluster cardinality balancing and SSE score.

In table 2 you can see the partition with K=30, and for each cluster the percentage of bot users inside. We think that this table is really interesting because it's able to identify big groups of users belonging almost completely to only one category. In particular, cluster number 12, 14, 25, 27 are very big and are composed by basically only humans. As opposite, it seems much more difficult to identify clusters with only bots.

In conclusion, the result of this section tells us that the bots are able to hide among human users in a way that hierarchical clustering alone is not able to detect, in fact the two biggest clusters (2 and 26) are composed by 66-67% of bots. Despite that, clustering could be combined with other tools like classification models that as we will show in the next chapter have been shown, as opposite, to be slightly better to identify bots with respect to human users .

2.4 X-means

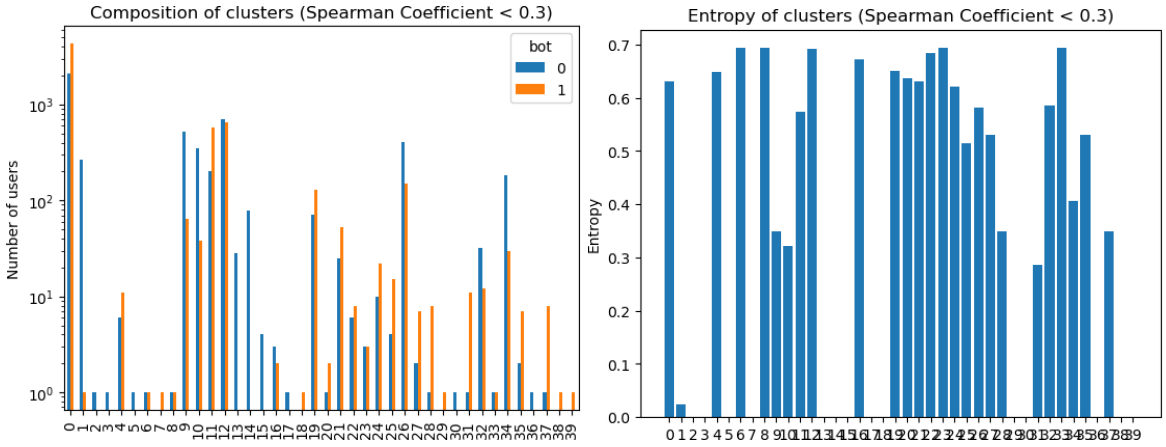
As additional clustering algorithm, the X-means algorithm has been chosen. Note how this algorithm tries to automatically determine the number of clusters using the BIC score, so it isn't required to find the optimal number of clusters.

For this reason the main work here has been done in the pre-processing phase. The data was scaled and only the features with a pairwise correlation below a given threshold has been kept; to do so, both Spearman and Pearson have been used, each one with different threshold values. The results of each run are shown in Table 3.

Correlation		Score		
Method	Threshold	Silhouette	WCE	Davies Bouldin
Spearman	0.3	0.477	5752.899	0.511
	0.5	0.348	23322.037	0.858
	0.7	0.340	27159.453	0.935
	0.9	0.414	51879.036	0.808
Pearson	0.3	0.509	5858.451	0.621
	0.5	0.323	24710.919	0.84
	0.7	0.329	27888.894	0.824
	0.9	0.415	48277.229	0.911

Table 3: Results of X-means runs

By looking at the outcomes (especially at Silhouette and Davies-Bouldin scores), a threshold of 0.3 produced good results with both Spearman and Pearson. In Figure 2.9 are shown some plots related to the clusters obtain with this particular threshold; in Figure 2.9a it's possible to see how the algorithm has been able to find some relatively big clusters containing almost all bots or real users.



(a) Clusters size and compositions

(b) Clusters entropy

Figure 2.9: Clusters found by X-means keeping only attributes with Spearman coefficient below 0.3

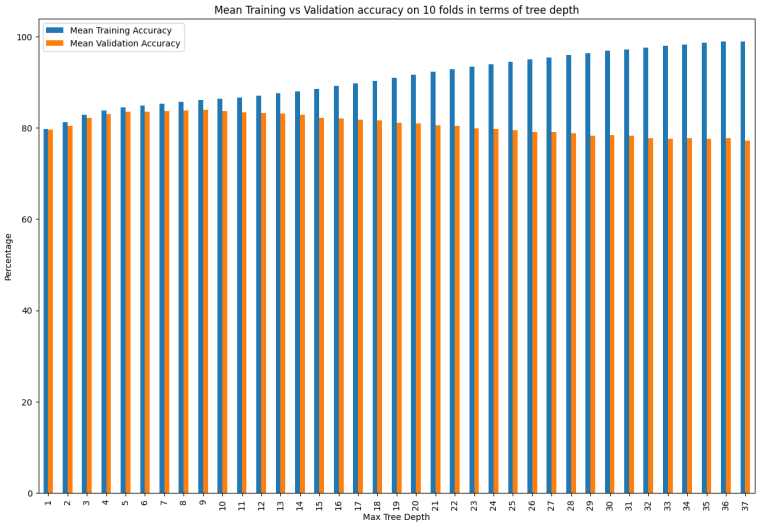
3 Predictive Analysis (Task 3)

3.1 Decision Tree Classification

In this subchapter we will be exploring parameter tuning, classification performance and interpretability of the decision tree classification model. We will be exploring how well this classification model is able to predict our labeled dataset of bots and humans.

Before finding the optimal parameters for the Decision Tree bot classification model, we performed some operations on the dataset:

1. Removing all categorical attributes, except for the "lang" column.
2. Transform the categories in the "lang" column to numeric representatives.
3. Decided to keep our slightly biased dataset (55% bots, 45% humans)"un-balanced" in order to perform training and validation on more samples.



Tree depths	9	5	3
Accuracy	83.7%	83.4%	82.2%
Precision	78.5%	78.1%	77.2%
Recall	86.8%	86.6%	85.5%
F1	86.8%	86.6%	85.5%

Table 4: Detailed 10-fold mean test set scores on select tree depths; Best performing tree depth of 9

Figure 3.1: Over-training with increase in model complexity

We performed tests in order to find the optimal values for the following parameters:

1. **Scaler choice:** In order to find evidence of over/under training due to noise we compared the performance of StandardScaler vs RobustScaler. Finding no real performance gain from using the RobustScaler we opted for the StandardScaler for this model.

Scaler	StandardScaler
Max tree depth	37
Criterion	Gini Coefficient
Optimal max tree depth	9
Most interpretable tree depth	3

Table 5: Best hyperparameters for bot classification

2. **Maximum tree depth:** We found our perfomance curve to flatten at a tree depth of 37. (Fig. 3.2)
3. **Criterion selection:** In order to find which information gain criterion created the best performing model, we compared the performance of the Gini and Entropy coefficients. Finding that the Gini performed equally or better at most tree depths of relevance. (Fig. 3.2)

4. **Optimal tree depth:** Found the best performing tree depth at a depth of nine. With the highest scores and lowest over-training values. (Fig. 3.1)

5. **Optimal tree depth for interpretability:** In order to find the best ratio between performance and interpretability, tree visualizations at different high performing tree depths were made. Example in Figure 3.3 from a tree depth of 3, with a slight performance drop to our optimal tree depth of 9, we receive the added benefit of easier interpretation of results. (Table. 4)

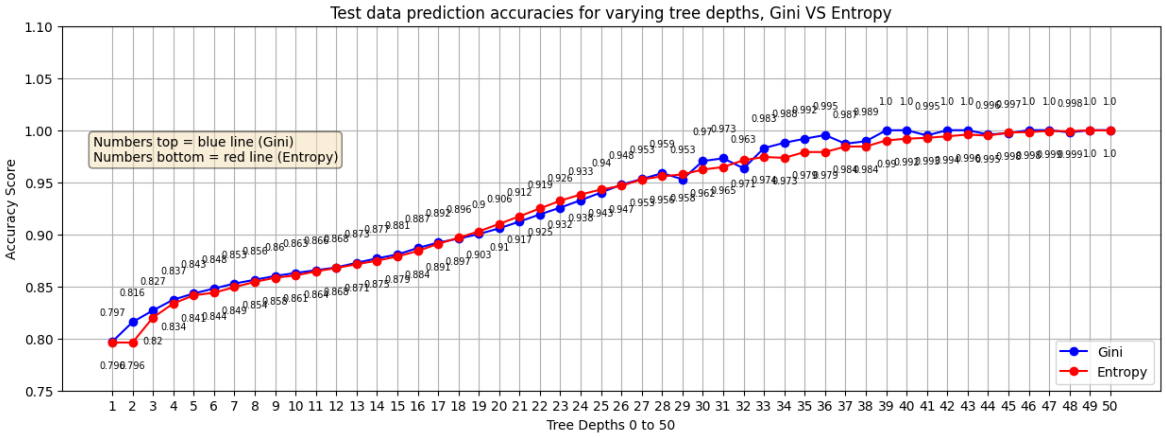


Figure 3.2: Performance comparison of the Gini and Entropy coefficients

We chose not to prune our tree as performance and computational cost was adequate for our use case. We split our training and test sets using K-fold cross validation with 10 folds.

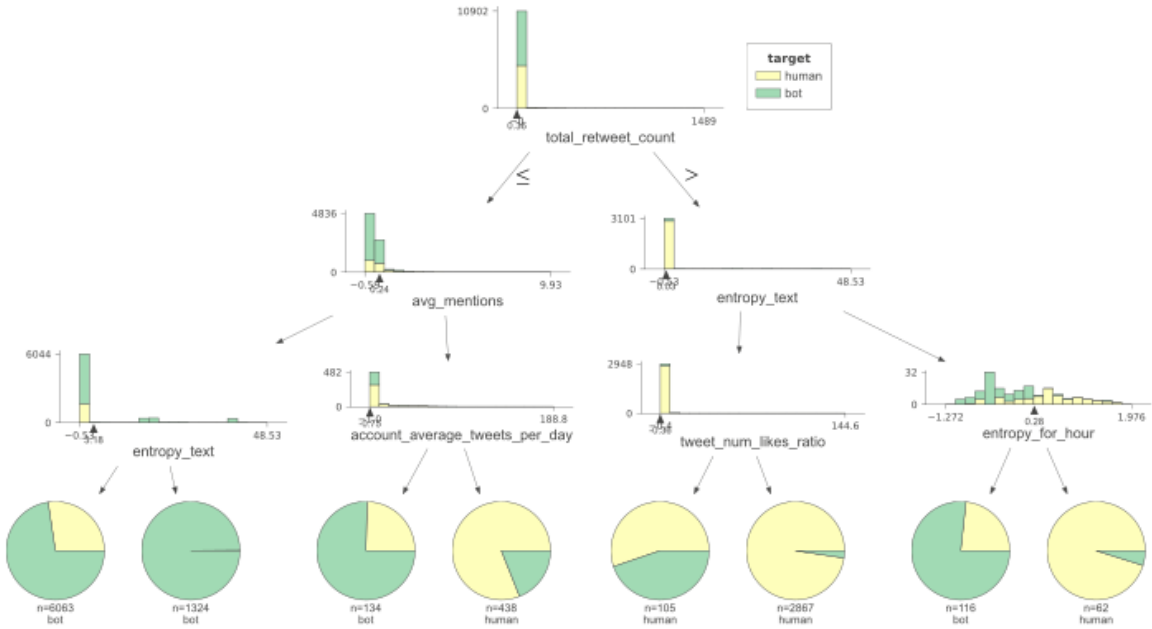


Figure 3.3: Decision tree with a depth of 3; visualized in DtreeViz

3.2 Neural network

Before training the neural network for the bot classification, we performed some operations on the dataset:

1. Removing the categorical attributes. The alternative is transform them with a one-hot encoding but this produces a very high dimensional space in case of date attributes; in case of the language this could be done but did not give any improvement.

2. Shuffle the data in order to avoid bias.

3. Split train, validation and test set. Train and validation data are used during the training and grid search of the best model while test data are used only once, after the choice of the best model, in order to have an estimation of the model behaviour on completely unseen data. The proportions of the split are 70%-20%-10%.

4. A min-max scaler is applied independently on train, validation and test set, so that the model doesn't know which is the maximum value in the test set and the final evaluation will not be biased.

We performed a random grid search with 200 trials, each of them repeated two times in order to decrease the variance. Since the grid search is a computational intensive task, we made some assumptions about the structure of the network: since we already have processed features (and not row data), and there is no sign of a hierarchy of features to exploit with the inductive bias of a deep neural network (as for the image classification), is reasonable to adopt a shallow model, with only few hidden layers [5, 7].

The hyperparameter are:

- Number of layers: from 1 to 5
- Number of units in each layer: 32, 64, 128, 256
- Learning rate: 0.05, 0.1, 0.2
- Weight decay: 1e-5, 1e-4, 1e-3
- Epochs: 250, 500
- Batch size: 1024, 2048

The activation function used for the hidden layers is the relu, while in the last neuron that is the sigmoid, the loss is the binary crossentropy and the learning algorithm is the stochastic gradient descendent.

At the end of the grid search, the result is that the best model is the one in table 6, with a training accuracy of 85% and a validation accuracy of 84%.

Now we can use the test set for the final evaluation, and the accuracy on it is 83%. The figure 3.4 is the confusion matrix on the test set, and in the table 7 there are other useful metrics.

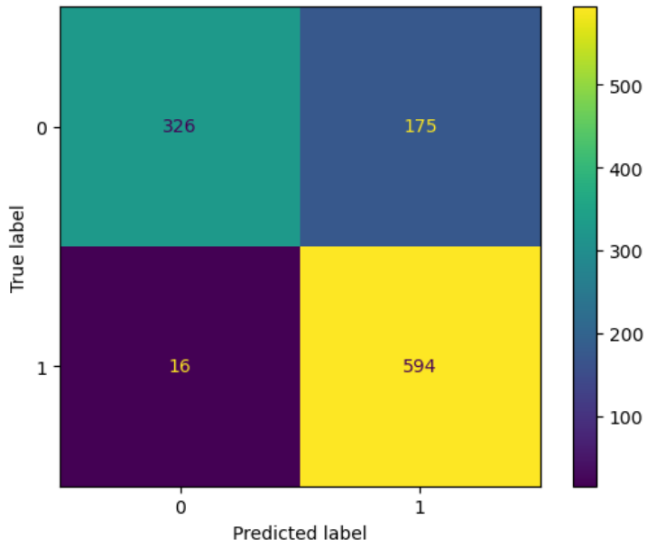


Figure 3.4: Confusion matrix on the test set

Accuracy	83%
Precision	77%
Recall	97%
F1	86%

Table 7: Detailed score on the test set

number of layers	3
units layer 1	128
units layer 2	128
units layer 3	256
learning rate	0.2
weight decay	0.00001
epochs	400
batch size	1024

Table 6: Best hyper-parameters for bot classification

It's really interesting to compare the results of the neural network with the ones in the hierarchical clustering. The neural network is very good in finding the positive samples (recall 97%), but sometimes confuses humans for a bot (precision 77%), while with the hierarchical clustering (section 2.3) was easier to identify the behaviour of human users.

We tried to improve the performance of the neural network, but we found and intrinsic difficult in this classification task, which is also evident with other classification models, in particular it is difficult to find a general pattern that classify a user as a bot. A bigger neural network is actually able to separate humans and bots in the training set with a 100% accuracy, but then the performances are really bad on unseen data, indication that the classificator has not found a general pattern but only memorized the examples, the results of our test are in figure 3.5.

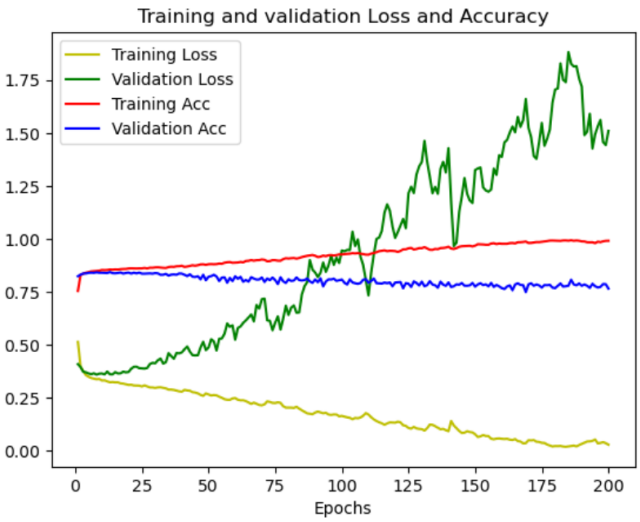


Figure 3.5: An example of 100% accuracy but in overfitting.

3.3 Naive Bayes

The test with Naive Bayes classifiers has been done using the following probabilistic models:

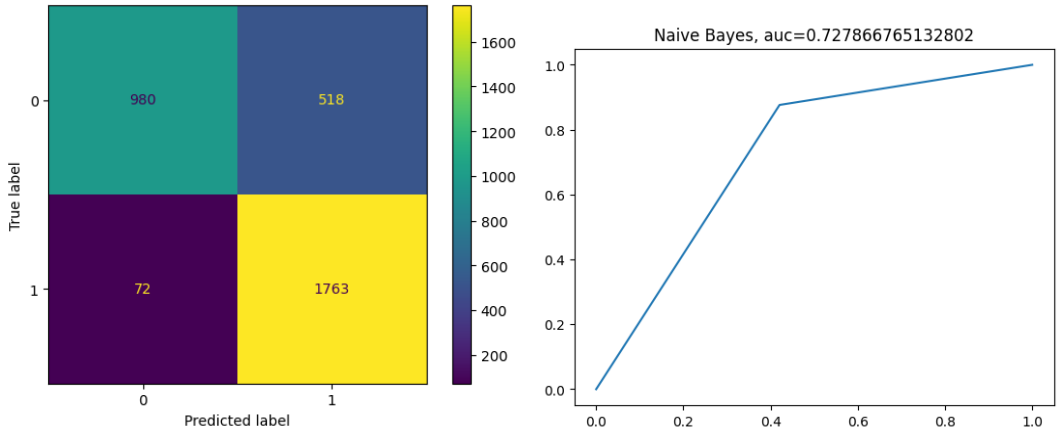
- **Gaussian**: this model works under the assumption that the probability distributions of the (continuous) features follow a normal (Gaussian) distribution [12];
- **Multinomial**: suitable for classification with discrete features [10];
- **Complement**: a variation of the Multinomial Naive Bayes [13] ;

Note how each classifier requires strong hypothesis about the probabilistic distribution and the the nature of the features; however, because of the high heterogeneity of the features of the dataset, it wasn't possible to elaborate a suitable pre-processing phase.

The three models listed above have been fitted with the scaled data, alongside an ensemble method applying the majority rule; the ensemble has proved superiori w.r.t. the single models, producing the scores described in Table 8 and in Figure 3.6.

	precision	recall	f1-score	support
Real users	0.93	0.65	0.77	1498
Bots	0.77	0.96	0.86	1835
accuracy			0.82	3333
macro avg	0.85	0.81	0.81	3333
weighted avg	0.84	0.82	0.82	3333

Table 8: Scores obtained by a hard voting schema of Naive Bayes models



(a) Confusion matrix

(b) ROC Curve

Figure 3.6: Plots of the results obtained by the voting schema for the Naive Bayes

3.4 AdaBoost

In this section we will describe the results obtained by using AdaBoost to classify the users type. AdaBoost is a classification meta-algorithm that actually boosts weaker algorithms [3]. The first part of this classification attempt consisted into mapping the language of the users with one-hot encoding and to clean the users dataset by removing the attributes not needed, all information that were not required at all for the classification (e.g. *user_id*).

After that we split the data into a training set and a test set, created a general kind of *AdaBoostClassifier* and made a fit with the training data to test the algorithm. At first the prediction made on the test set by this classifier were good enough, being similar to the ones of the Neural Network in terms of accuracy (Section 3.2).

This led us to try many combinations to create a better classifier, trying many settings:

1. Set the base estimator DecisionTree (DT) a max depth equal to 5 and 2
2. Normalize the data with MinMaxScaler and DT max depth 2
3. Normalize the data with the Mean and DT max depth 2

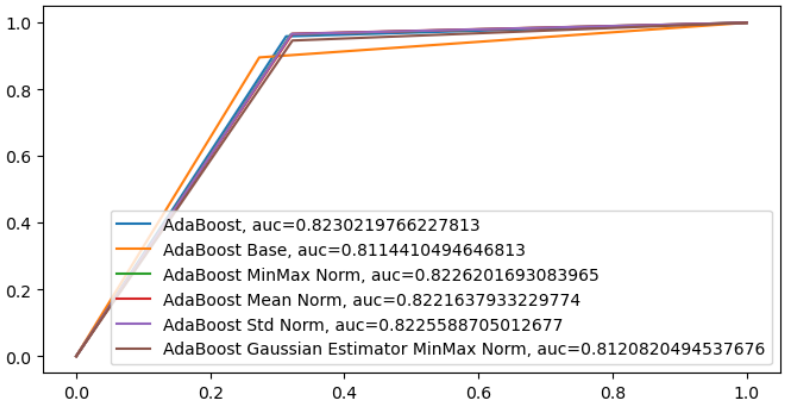


Figure 3.7: Adaboost tests results in accuracy

4. Normalize the data with the Standard Deviation and DT max depth 2
5. Change the base estimator to a Gaussian Naive Bayes and use the Min Max Scaler

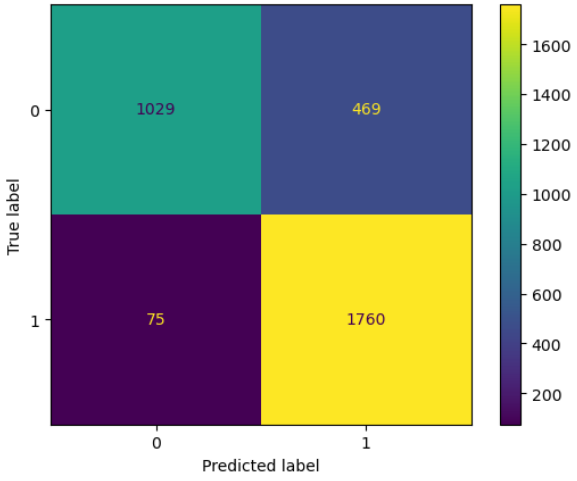


Figure 3.8: AdaBoost label prediction comparison

None of those improved the prediction accuracy (≈82), since we had slighy worse values. We can see the accuracy of the various test in the Figure 3.7, the algorithm implemented in Python, in our case, works better as it is out of the box.

The results of the AdaBoost Classifier on the test set are shown in Figure 3.8, there is a lot of difference between the false negatives and the false positives. In conclusion we can say that this attempt was a good one since we had a good result in term of recognition of bots, but there is still a high rate of false positives and we want to lower that value.

4 Time Series and Explainable AI (Task 4)

4.1 Time Series Analysis

This section describes the various tests and approaches, chosen, to group the users through uni-variate time-series analysis. The value for each time point it's the SuccessScore, a value obtained through the ratio between the AcceptanceScore (the sum of a user's retweet count, reply count and favorite count) and the DiffusionScore+0.1 (the sum of a user's total number of hashtags, urls and mentions in the tweets).

In total there are 3 main tests made with these values: one using the TSLearn library, one using the same type of Neural Network of Section 3.2 (this is not described since it doesn't use shapelets) and the last try is made by usign the SkTime library. The analysis is made taking the values for each day of the year 2019 for any user, in case a user did not have a value for a certain day we considered the value as -1.

This filter left us with only 6580 users compared to the initial 11109 dataset. Lastly all the time-series were transformed according to the Noise removal method seen in the laboratory.

Clustering At first, a clustering attempt is made by using kmeans with euclidean distance. This led us to find 2 little clusters, over 7 of them, made by only bots similar to the K-Means attempt in Section 2.1.

TSLearn ShapeletModel is part of the TSLearn library, shown in the laboratory part of the course. Since it's shown we tried to use it for time-series classification, using different shapelets sizes. The attempt was done by using 4 shapelets size with a 1% length of the original timeseries length as a start point: 36: 6, 72: 6, 108: 6, 144: 6. The model was trained over 200 epochs, with a batch size of 256, a weight decay of 0.01 and optimizer was *Adam*. The best accuracy obtained by this algorithm, as we can see in Table 9, is 0,82.

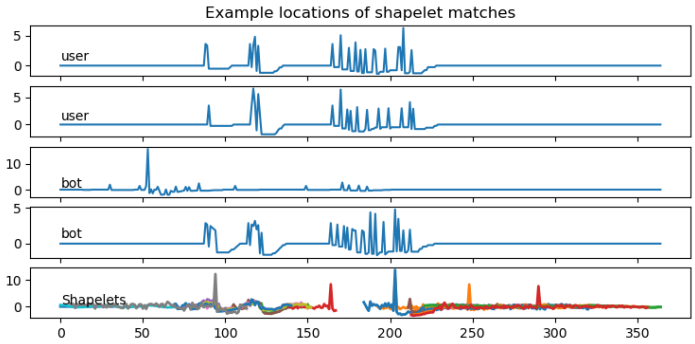


Figure 4.1: Shapelets matches comparison to two users and two bots

Rocket SkTime library is oriented to time series analysis and it includes many algorithms dedicated to the analysis of time series through the learning of shapelets. In our case we made just a mild attempt with a classifier from this library: RocketClassifier. Rocket is a simple linear classifier that uses random convolutional kernels [2].

	Prec	Rec	F1	Acc
TSLearn	0.82	0.82	0.82	0.82
NN	0.85	0.84	0.83	0.84
Rocket	0.88	0.87	0.87	0.87

Table 9: Comparison table for the three approaches

The algorithm obtained an accuracy of 0.87 after a very short time, thus obtaining the primacy among all the approaches tested on the analysis of the time series. Furthermore, we tested the classifier on all the dataset (not only 2019), obtaining the same accuracy with a false negative number of almost 0.

4.2 Explanation Analysis

4.2.1 EBM

Explainable Boosting Machine (EBM) is a tree-based model. In some task it has the same accuracy as state of the art model but it is completely interpretable.

The equation (1) explains how the EBM works. Each T is a decision tree with a small depth, k is the number of feature, so for each of them r trees are trained, added up and weighed by the coefficient a. In this way an high flexibility can be achieved as well as an excellent explainability since there are only additions of decision trees which are then weighted by a linear model.

$$\begin{aligned} f(u) = a_0 + & \\ & a_1(T_1^{(1)}(x_1) + \ldots + T_r^{(1)}(x_1)) + \\ & \vdots \\ & a_k(T_1^{(k)}(x_k) + \ldots + T_r^{(k)}(x_k)) \end{aligned} \tag{1}$$

The EMB is trained in round robin fashion: at each iteration k trees are trained (one for each feature, each tree one can look only at its relative feature to predict the residuals of the previous tree.

From the figure 4.2 and table 10 you can check that the performance of EBM are similar to other classification model much more difficult to explain.

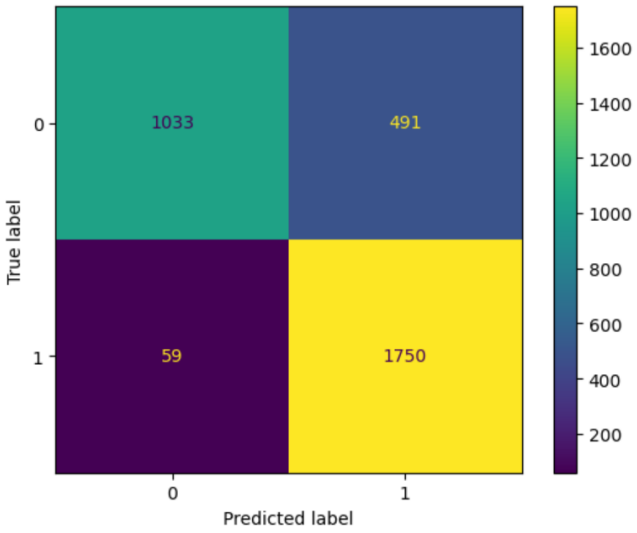


Figure 4.2: Confusion matrix on the test set

Accuracy	83%
Precision	78%
Recall	97%
F1	86%

Table 10: Detailed score on the test set

In the figure 4.3 there are the 15 most important features found by EBM in our classification task.

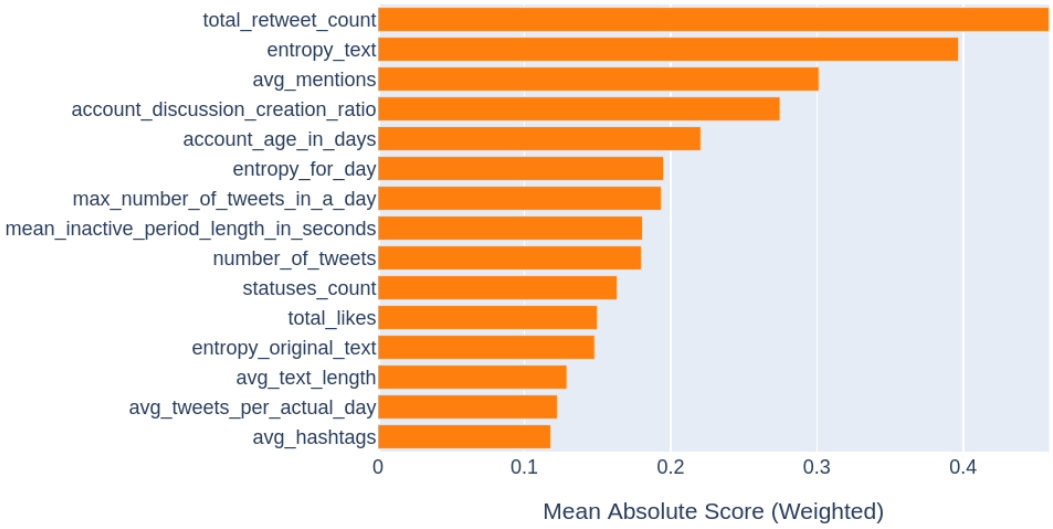


Figure 4.3: Global explanation of the model.

The number of total retweets seems to be the most important indicator to distinguish between humans and bots. Let's take a closer look to how the model uses this value in figure 4.4.

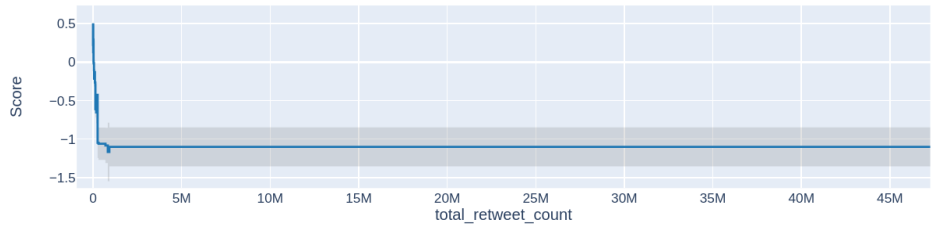


Figure 4.4: Influence of total_retweet_count.

The slope of the graph is really high from 0 to 300k and is basically flat after this value, with a negative contribute, i.e. it is more likely the user is human if it has more than 300k retweet. This probably happens because real users are able to detect bots and they don't retweet their tweets.

The attribute "entropy_text" turned out to be really useful for classification, probably because it measures an uncommon behaviour for a human user, i.e. how often an exactly equal tweet is published. You can see it clearly in figure 4.5.

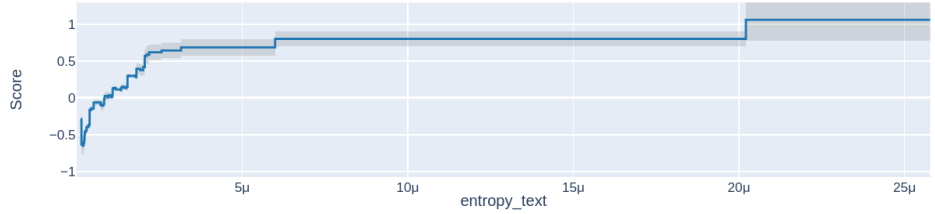


Figure 4.5: Influence of entropy_text.

The figure 4.6 tells us that usually a user that frequently tags another user is a human. We can only make hypotheses on the explanation of this phenomenon: one could be that if a bot often tags a human user, then could be reported and then removed from the platform.

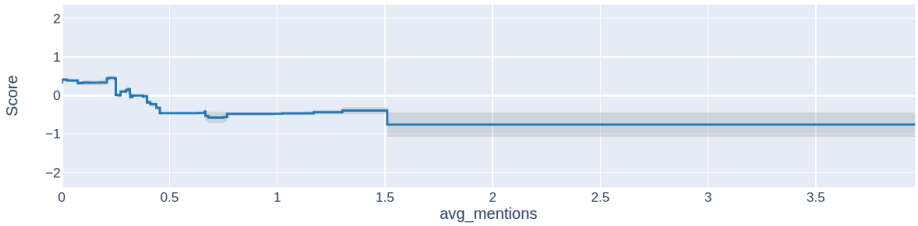


Figure 4.6: Influence of average_mentions.

4.2.2 LIME

LIME is a method for explain prediction of machine learning models. As its name says it works for any model and it can explain a single prediction i.e. a local approximation of the function leaned by the original model, training a surrogate linear model around the point we want to explain.

With LIME we explained the decisions made by our neural network. The results are very encouraging because results are on average in agreement with the feature importance extract with EBM and Shap (on decision tree), even if for some points, the classification logic doesn't follow the average rule, as we will show.

The example in figure 4.7 is really interesting for two reason: the first is that the "total_retweet_count" that on average is the most important feature in EBM and Shap, for this specific area of the neural network function seems to be not important for the classification, the second reason is that the low "entropy_text" of this bot user leads the classifier to be a bit confused and shifts the weight toward "human", though slightly. So this bot is probably not among those that repeat the same tweet over and over again and is able to hide better among human users.

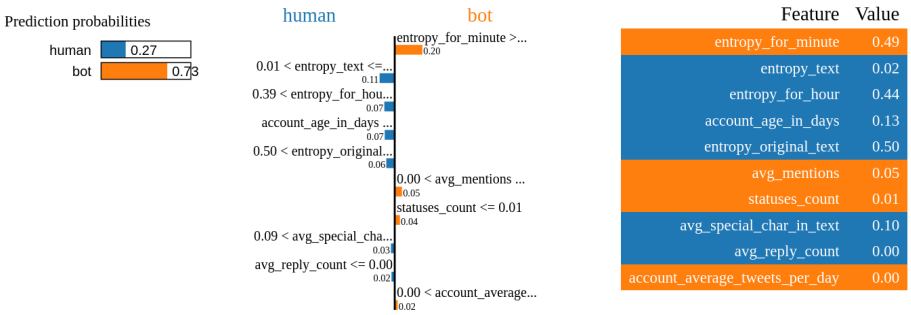


Figure 4.7: Influence of attributes in neural network prediction for a single bot instance.

In the figure 4.8 you can see that the attribute used are different again, indicating that the learned function of the neural network is really complex. The zero entropy text is a clear marker of a human user, while the total retweet count is present this time, but is slightly misleading.

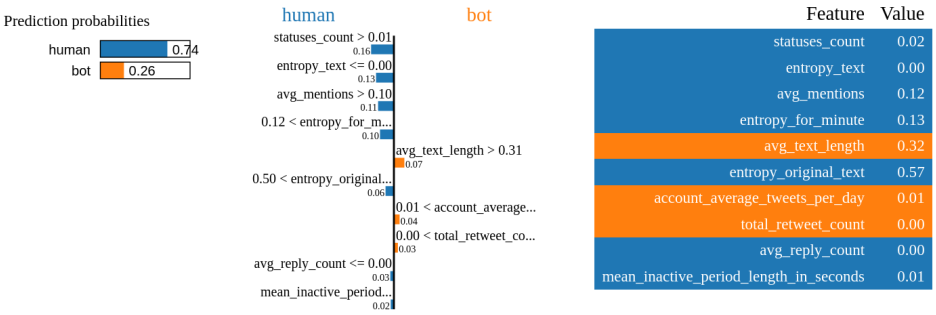


Figure 4.8: Influence of attributes in neural network prediction for a single human instance.

4.2.3 Shap

In this section, to have further confirmation that the classifiers used gave importance to the same features, we applied Shap to the decision tree. SHAP is a game theoretic approach to explain the output of any machine learning model. The average impact on the model output of the feature is shown in figure 4.9.

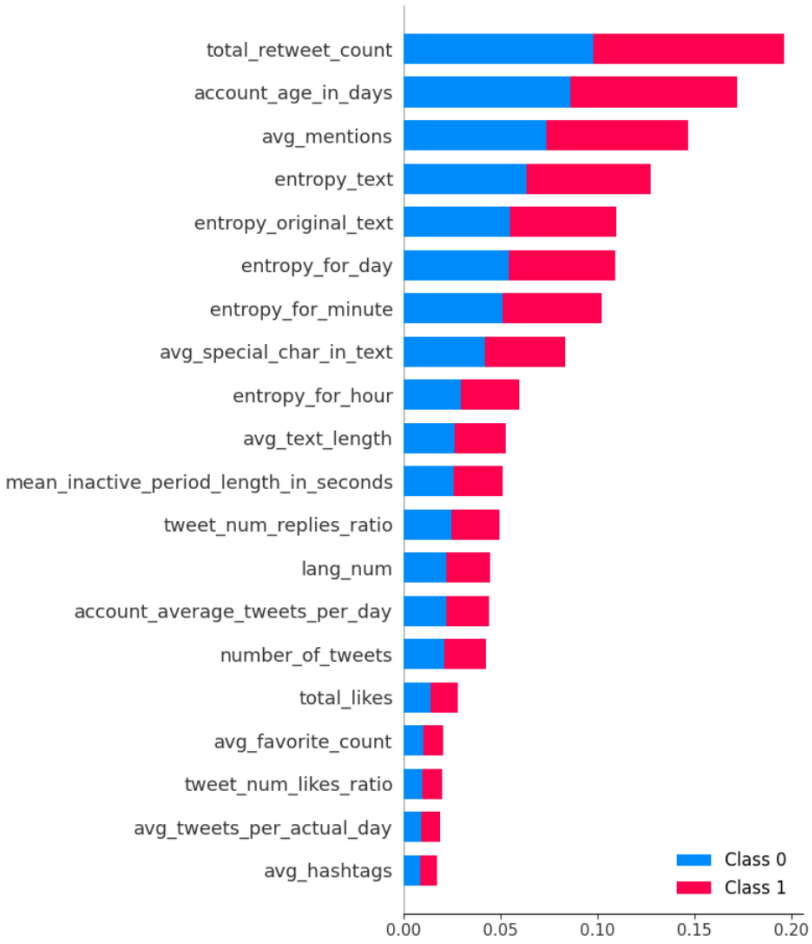


Figure 4.9: Global average influence of attributes in decision tree prediction.

The results are in agreement with the ones obtained with EBM and on average with the LIME explainer on the neural network.

5 Final Summary

5.1 Data Understanding and Results

The data understanding part has been performed by doing basic operations such as handling missing values, duplicates and correcting erroneous values. Knowing the context we were operating on (i.e. Twitter platform) allowed us to leverage context specific information (i.e. the thresholds we introduced in Subsection 1.3) to detect errors and outliers.

Moreover, making the indicators has proved more useful that we initially tought, since (especially for the k-means clustering) some of the indicators (i.e. *avg_hashtags* and *entropy_text*) has allowed us to get interesting results (such as the "Spammers" cluster shown in Figure 2.2).

5.2 Clustering Evaluation and Results

Hierarchical clustering using Ward's distance was our best informing clustering method (table: 2). Its ability to separate multiple one-labeled clusters would deserve to be investigated further.

The semi-automatic clustering methods of X-means and OPTICS also had great results on indicators of low correlation. Sadly X-means had trouble separating the largest cluster into smaller ones, and OPTICS labelled a high degree of samples as noise points(32%<), even on indicators of low correlation.

Finally, one of the most interesting result has been obtained by the K-means algorithm, which as been able to find a particular subgroup (the so called "Spammers") of high retweeters inside the bots population.

5.3 Predictive Analysis Evaluation and Possible Hindrances

The predictive analysis shown to have an intrinsic difficulty since all model struggle to obtain 90% accuracy. It is also of interest to note the bias of all models toward false positives, i.e. a lower precision with respect to recall.

Our hypothesis on possible hindrances that may have effected our predictive performance:

- Inadequate parameter tuning may make the prediction model too simple or too complex increasing likelihood of over/under training.
- Inadequate noise or unimportant attribute removal, creating high variance or dimensionality within the dataset, resulting in over-training.(Fig: 3.1)
- As stated before, there is some imbalance in the dataset (i.e. 55% bots and 45% real users); we can state that this is probably a cause of some lack of precision of our models. However, because of our lack of experience, we don't know if reason completely justifies the results, or if there are other implicit causes that we missed.

5.4 Explanation Analysis Evaluation

The explanation analysis gave us consistent results among different classification model and different algorithms to explain them. A lot of discussion could be done in order to explain why each attribute is important for the classification in addition to the things already said,

without even considering the combinations of two or more features. We found EBM a really fast, powerful and reliable model, able to give really coherent explanation both globally and locally (not shown in the report). On the other hand we found that explain a neural network with a local approximation (LIME) doesn't provide a good comprehensive view because the feature importance could be really different from one instance to the other.

5.5 Time Series Analysis Evaluation

In the time series analysis we had good results exploiting the bot variable through shapelets learning. We can see in the Figure 4.1 that the users and the bots can have similar time series behaviour, this is probably one of the reasons why we have far more false positives than false negatives. However, our best performing time series model was Rocket.(Table: 9) A linear classification algorithm based on convolutional kernels that has achieved state-of-the-art accuracy on known benchmarks[2], and now on our dataset. Comparatively to other state-of-the-art models it is a model of low complexity, which might lead to less over-training. Low model complexity may be the reason for its increased performance. Something that would be interesting to explore in the future.

References

[1] *Data Mining (309AA) - 9 ECTS AY 2022/2023*. URL: http://didawiki.cli.di.unipi.it/doku.php/magistraleinformatica/dmi/start. (15.09.2022).

[2] Angus Dempster, François Petitjean, and Geoffrey I Webb. "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels". In: *Data Mining and Knowledge Discovery* 34.5 (Sept. 2020), pp. 1454–1495.

[3] Yoav Freund and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". English (US). In: *Computational Learning Theory - 2nd European Conference, EuroCOLT 1995, Proceedings*. Ed. by Paul Vitanyi. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2nd European Conference on Computational Learning Theory, EuroCOLT 1995 ; Conference date: 13-03-1995 Through 15-03-1995. Germany: Springer Verlag, Jan. 1995, pp. 23–37. ISBN: 9783540591191. DOI: 10.1007/3-540-59119-2_166.

[4] geeksforgeeks.org. *ML | OPTICS Clustering Explanation*. URL: https://www.geeksforgeeks.org/ml-optics-clustering-explanation/. 14.07.2019.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] *IETF language tag*. URL: https://en.wikipedia.org/wiki/IETF_language_tag. (22.12.2022).

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[8] *List of most-liked tweets*. URL: https://en.wikipedia.org/wiki/List_of_most-liked_tweets. (24.12.2022).

[9] *List of most-retweeted tweets*. URL: https://en.wikipedia.org/wiki/List_of_most-retweeted_tweets. (24.12.2022).

[10] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 0521865719.

[11] Imas Sukaesih Sitanggang Nadia Rahmah. *Determination of Optimal Epsilon (Eps) Value on DBSCAN Algorithm to Clustering Data on Peatland Hotspots in Sumatra*. URL: https://iopscience.iop.org/article/10.1088/1755-1315/31/1/012012/pdf. 2016.

[12] Sebastian Raschka. "Naive Bayes and Text Classification I - Introduction and Theory". In: *CoRR* abs/1410.5329 (2014). arXiv: 1410.5329. URL: http://arxiv.org/abs/1410.5329.

[13] Jason D Rennie et al. "Tackling the poor assumptions of naive bayes text classifiers". In: *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003, pp. 616–623.

[14] Jörg Sander et al. “Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications”. In: *Data Mining and Knowledge Discovery* 2.2 (June 1998), pp. 169–194. ISSN: 1573-756X. DOI: 10.1023/A:1009745219419. URL: https://doi.org/10.1023/A:1009745219419.

[15] Jörg Sander et al. “Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications”. In: *Data Mining and Knowledge Discovery* 2 (1998), pp. 169–194.

[16] *User object*. URL: https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/user. (22.12.2022).

[17] *What Is a Black Box Model? Definition, Uses, and Examples*. URL: https://www.investopedia.com/terms/b/blackbox.asp. (06.03.2022).