DM - Project report

Group 12
Leonardo Vona
545042

January 8, 2023

1 Data understanding and preparation

1.1 Tweets data set

The tweets data set contains over 13 million tweets. Each tweet has associated 9 attributes.

1.1.1 Characterization of the features

user_id The user_id is a categorical (String) feature.

The tweets that does not have an associated user_id (200.000 entries) have been removed from the data set because they are not useful for the scope of the project, which is to create a profile of the users.

For the same reason, the tweets containing a non numeric user_id (which is not allowed in Twitter), which are other 200.000 elements, have been removed.

retweet_count It is a numerical feature, recognized as Float by Pandas, but actually Int. Pandas recognizes the attribute as Float because it contains some NaN values (which are Float). After the handling of the missing values, the attribute has been correctly converted to Int.

The maximum number of retweets obtained by a tweet is less than 4.1 millions, then all the values above this limit have been removed and considered as missing values.

reply_count It is a numerical (Int) feature. Analyzing the values, it emerges the presence of a 'inf' value, which has been canceled.

favorite_count It is a numerical (Int) feature. The most liked tweet has received less than 7.2 million likes, then the values above this threshold have been removed.

The data set also contains an instance where the value of favorite_count is negative, which has been handled as a missing value.

num_hashtags It is a numerical (Int) feature. The maximum number of hashtags per tweet has been set to a "rough" limit of 140 (the maximum tweet length is 280).

num_urls It is a numerical (Int) feature. Twitter allows to insert at most 10 links per tweet.

num_mentions It is a numerical (Int) feature. Twitter allows to mention at most 10 users per tweet.

created_at It is a numerical (Datetime) feature. The range of allowed dates in which a tweet can been created is between 2006-03-21 (date of publication of the first tweet) and 2022-09-30 (date of publication of the project).

I also searched for tweets which have been published before the creation date of the account which are associated to (retrieved from the users data set).

After cleaning the attribute, it emerged that the range of publication of the tweets in the data set is between 2012-03-11 and 2020-05-03.

1.1.2 Duplicate data

Two tweets are considered equal if they have the same value for the attributes 'user_id', 'created_at' and 'text'. Between two (or more) duplicate tweets, I pick the one with less missing values for the numerical features.

The approach is to order (descending) the tweets by the numerical features, considering NaN as the smallest value, and keep only the first tweet between duplicated ones.

After the process, more than 3 millions tweets were marked as duplicates and removed.

1.1.3 Handling missing values

The tweets data set, after applying the phases described above, contains more than 500K elements containing at least a NaN value.

I have chosen to remove the tweets containing no text because for most of them also other features are missing, and so they are not very useful for the analysis.

For the numerical features containing NaN values (except created_at), it is possible to fill the missing values using as substitution value the mode (instead of the mean, which is more sensible to outliers) of the feature for each user. This is done grouping the tweets by user_id and then extracting the mode for each numerical attribute.

For the created_at feature instead, the mean is more meaningful, and the outliers have been already removed, so I used it.

After this process, there are still 1K tweets with at least a NaN value (because the users which are associated to have only that tweet in the data set, so it is not possible to fill the missing values with the approach used above). Having a look at these entries it is likely that these are erroneous data, which are not associated to users recorded in the users data frame, so I have chosen to drop these tweets.

1.1.4 Separate tweets data set

The tweets have a different behavior with respect to retweets (which are naively identified if the 'text' attribute is starting with 'RT @'). In particular they have an high 'retweet_count', and associating them to users who have retweeted them may create problems in the user profiling.

Separating tweets and retweets will hopefully allow to recognize better the behavior of an user.

1.1.5 Outlier detection

It is not likely that a tweet has a number of retweets greater than 1000 and a number of likes less than 10. These outliers values for the retweet_count are considered wrong data and removed (replaced with NaN).

1.2 Users data set

The users data set contains the information about 11508 users. Each account has associated 5 attributes.

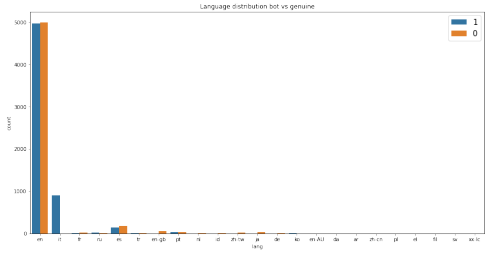

Figure 1: Language distribution bot vs genuine.

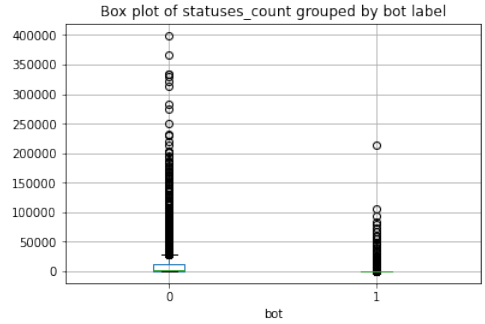

Figure 2: Box plot of statuses_count grouped by bot label.

1.2.1 Characterization of the features

name It is a categorical (String) feature. There is one user with no name associated. Since the feature is not particularly significant for the analysis, the missing value has been filled with the id of the user.

lang It is a categorical (String) feature.

The attribute contains two instances of the default value 'Select Language...'. This value has been inferred by the language used by the user to write the tweet, which is English ('en') in both cases.

There are also two pairs of values ('en-GB' - 'en-gb' / 'zh-TW' - 'zh-tw') which actually are the same value and then have been merged.

There are 23 distinct values for the language feature, considering also regional sub-languages ('en-gb', 'en-au', 'zh-ch', 'zh-tw'), which have been kept separated from the principal one.

The most frequent language is 'en' by far, followed by 'it' and 'es'.

It is interesting to see (Figure 1) that the users with Italian language are all (more than 900) labeled as bots.

bot It is a categorical (String) feature. Admitted values for the feature are '0' and '1', and in the data set there are not invalid or missing values for this attribute.

created_at It is a numerical (Datetime) feature. The range of allowed dates in which a user can been created is between 2006-03-21 (date of publication of the first tweet) and 2022-09-30 (date of publication of the project). There are no instances containing a wrong value for this feature.

statuses_count It is a numerical (Int) feature. Comparing the distribution of statuses_count for the bot and genuine users (Figure 2), it looks like that genuine users tend to have a number of tweets published higher with respect to bots.

1.2.2 Duplicate data

Two user entries are considered equal if they have the same value for 'user_id'. Using this criterion, in the data set there are no duplicate users.

1.2.3 Handling missing values

For the users data set, there is only one user which has no associated name. Since it is a categorical attribute and it does not influence the users profiling, I have chosen to assign the id as the name for this entry.

For the statuses_count instead I filled NaN values with the mode value.

1.3 Introduction of new features

After the cleaning of the data it is possible now to create new indicators which may be helpful in discriminating users' behavior.

I first merged the tweets and users data frame with a right join, so that only the tweets that have a correspondent user in the users data set are picked.

After the merging, the resulting data set is grouped by user_id, in order to easily extract the interesting new features.

num_tweets Counts the number of tweets associated to the user.

num_retweets Counts the number of retweets associated to each user. To obtain the value I merge the users and the retweets data frames similarly as before, and then retrieve the values.

avg_tweets_per_day Stores the average number of tweets per day for each user. To calculate the average first I retrieve the datetime of the first and the last tweet, and then divide num_tweets by the interval (in days) between the first and last tweet.

avg_[retweets/replies/favorites/hashtags/mentions/urls]_per_tweet For each numerical feature of the tweets data frame (except created_at) I create a new attribute containing the average of that feature per tweet.

cumulative_[retweets/replies/favorites/hashtags/mentions/urls]_per_tweet For each numerical feature of the tweets data frame (except created_at) I also create a new attribute containing the cumulative sum of that feature for each user.

avg_tweet_length Contains the average tweet length for each user.

entropy_hour Measures the entropy in terms of hour of publication of the tweets associated to each user.

To retrieve the entropy with respect to the hour of publication (and analogously with respect to the text length), I first count (for each user) for each distinct value of the hour (0 - 23) the number of occurrences, then I apply the entropy to the occurrences normalized as probabilities, diving them by the total number of tweets associated to the given user.

entropy_text_length Measures the entropy in terms of text length of the tweets associated to each user.

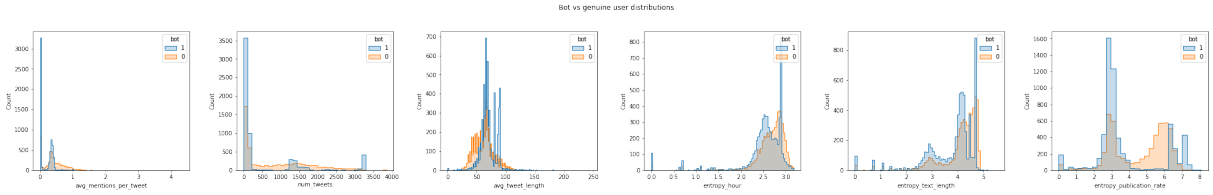

Figure 3: Bot vs genuine distribution among different features.

entropy_publication_rate Measures the entropy in terms of publication rate of the tweets associated to each user.

The entropy associated to the publication rate is calculated by, for each user, sorting the datetime of publication of his / her tweets, extracting the difference in seconds between the list of datetimes (excluding the first tweet) and then applying the entropy function.

If some users have just one tweet, the entropy for the publication rate will produce a NaN value, so I fill them with the value 0.

1.3.1 New features distribution and statistics

Most of new features show a distribution still highly skewed towards zero, as expected. They are not very informative, but we can see a more spread distribution for the features num_tweets, avg_text_length, entropy_hour and entropy_text_length.

If we try to compare the distribution of genuine vs bot users with respect to the most characterizing features (Figure 3), we can see that in fact the distribution are different depending on the bot label.

For example, for the num_tweets feature we can derive that the number of bots with 0 tweets is twice the number of genuine users with 0 tweets.

If instead we consider the entropy characterizing the hour of publication, the bots seems to have a more fixed variability (around 2.8) with respect to genuine users.

Comparing the interquartile ranges of the features for the genuine and bot users we can see a general lower activity for bots. It is possible to see this for example in the number of tweets per user, or in the number of retweets or in the number of likes received, which values are all considerably lower for the bots.

Concerning instead the entropy features, it is interesting to see a slightly higher variability for the bots. If we investigate into it anyway, it is possible to see from the mean value a more realistic ratio, where the genuine users have an higher mean variability. We can also say that the mean is a more appropriate estimate for the entropy features, given the limited presence of outliers for these attributes.

2 Clustering

2.1 Preprocessing

2.1.1 Dividing the data set

I divide the user features into categorical and numerical, into the data frames cat_df and num_df, respectively.

I put the feature 'created_at' into the categorical data frame even if it is not because it can't be used for the clustering.

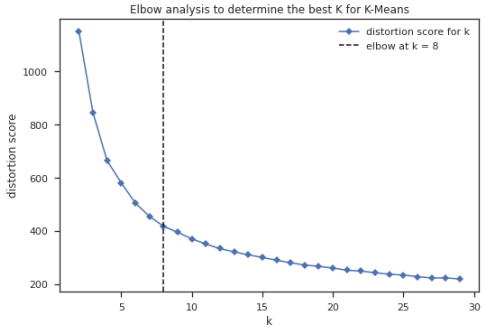

Figure 4: Elbow analysis to determine the best K for K-Means.

2.1.2 Elimination of highly correlated features

Highly correlated features may turn into a problem for the clustering analysis. There is an high correlation (greater than 0.9) between:

- avg_favorites_per_tweet and cumulative_favorites
- avg_retweets_per_tweet and cumulative_retweets
- cumulative_favorites and cumulative_retweets

For this reason I decided to drop the features cumulative_favorites and cumulative_retweets.

2.1.3 Normalization

Before starting the clustering analysis, the numerical features are normalized using a MinMaxScaler. I used the min-max method instead of the standard (Z-score) one because it fits better with the data in consideration.

2.2 K-Means

2.2.1 Determine K

To determine candidate values for the best K in K-means, I use the elbow and silhouette analysis.

Elbow From the elbow method I obtain as candidate K the value 8 (Figure 4).

Silhouette The silhouette score has its peak at value 3 (Figure 5).

Comparison The two candidate values for the K parameter are compared, using the SSE, Davies-Bouldin, Silhouette and Calinski-Harabasz scores.

- SSE: determines the accuracy of the model
- Davies-Bouldin: determines how well the clusters are separated
- Silhouette: gives an indication on how well the clusters are defined
- Calinski-Harabasz: similar to Silhouette

From the results (see Notebook) it is possible to see that the K-means clustering with 3 clusters performs in general better with respect to the one with 8, even thought it has a much higher SSE, which may be caused by outliers.

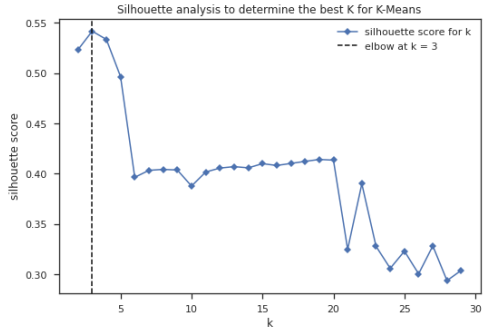

Figure 5: Silhouette analysis to determine the best K for K-Means.

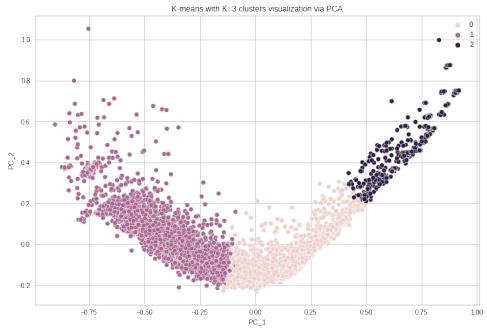

Figure 6: K-means with K=3 clusters visualization via PCA.

2.2.2 K = 3 evaluation

I select as K the value 3 and I proceed with the evaluation of the clustering.

I visualize how the data is separated into the clusters by using PCA to select the two most informative components, and I plot by using these coordinates (Figure 6).

By the parallel coordinates plot (Figure 7) it is possible to see that there is quite a good separation of the clusters if we consider the entropy features, in particular entropy_publication_rate.

If we take the three cluster centers, we can see that:

- the first cluster (cluster 0) contains medium active users
- the second cluster (cluster 1) groups the most active users, with many tweets associated and an

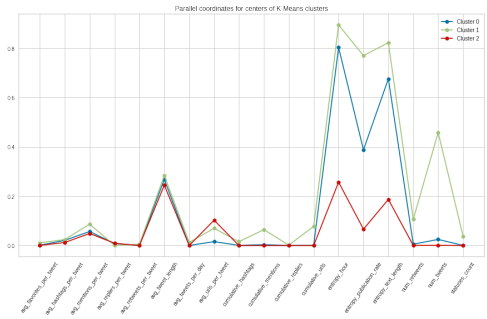

Figure 7: Parallel coordinates for centers of K-Means clusters.

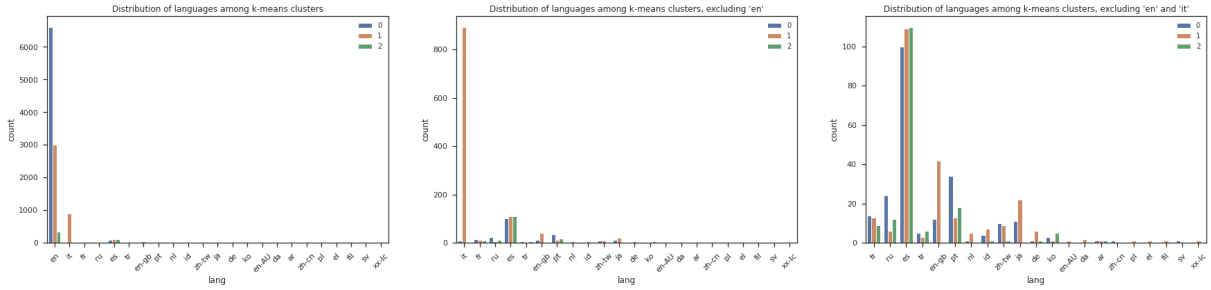

Figure 8: Distribution of languages among k-means clusters.

higher variability

- the third cluster (cluster 2) contains users with a scarce activity

The plots reported in Figure 8 show how the users with different languages are distributed among the clusters. To visualize better the data, I plotted three different charts: the first one takes into account all the languages, the second one excludes 'en' (which is the most frequent by large), and the third one also excludes 'it' (which is the second most frequent one).

From the plots it is possible to see that the users with language 'en' are principally assigned to clusters 0 (most of them) and 1. It is also interesting to see that for language 'it' (which are all bots), almost all the users are assigned to cluster 1.

Evaluation by external metrics I use the Similarity, Homogeneity, Completeness and Mutual Information metrics to evaluate the K-means clustering with K equal to 3, with respect to the categorical features 'bot' and 'lang'.

- Similarity: computed by the adjusted rand score, it gives a similarity evaluation between the categorical features and the clustering labels
- Homogeneity: measures the homogeneity of clusters with respect to the categorical features
- Completeness: measures how much the users with the same value for the categorical features are assigned to the same cluster
- Mutual Information: measures the mutual dependence between the categorical features and the clustering labels

The results (see Notebook) show a relatively good performance of the K-Means clustering with K equal to 3 with respect to the 'lang' feature.

2.3 Density-based

The density based clustering is performed using the DBScan algorithm.

2.3.1 Study of the clustering parameters

For the DBScan algorithm it is needed to set two main parameters, which are eps (maximum distance to consider two points neighbors) and minPts (minimum number of points to form a cluster). In order to establish the value for the parameter eps, I plot the sorted distances of every point to its 4-th nearest neighbour.

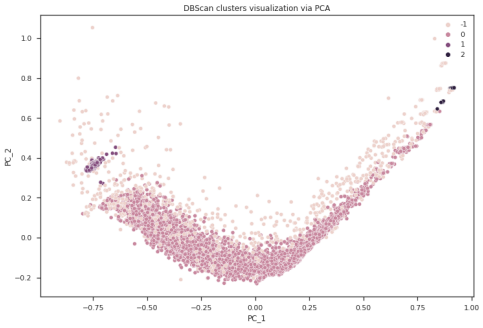

Figure 9: DBScan clusters visualization via PCA.

From the plot of sorted distances (see Notebook) it is possible to see that the distance is quite stable between 0.01 and 0.05, then there is a small step, and again a stable increase up to 0.2. From this plot then we can derive that a good value for the parameter eps is between 0.05 and 0.2.

For the minPts parameter instead some studies suggest that a good value for it is twice the number of features, which is 18 in our case. For completeness, I will try values for minPts from 18 to 36 (with a step of 2).

I perform a grid search for the best combination of the parameters, with the ranges defined above.

From the results of the search, the most promising combination of parameters seems to be 0.15 for the eps and 20 for the minPts.

DBScan as expected is quite good in finding outliers, marked as belonging to cluster -1. Anyway it does not seem able to create a good breakdown of the data because the cluster 0 contains almost all the points not labeled as noise, and the rest two clusters are very small (Figure 9).

2.3.2 Evaluation

From the evaluation of the clustering obtained using DBScan (see Notebook) it seems that density based clustering is not particularly suited for our data set.

2.4 Hierarchical

I build the dendogram using four different linkage methods:

- single (MIN)
- complete (MAX)
- average
- Ward

From the dendograms (Figure 10), it is possible to see that:

- Single: There are many singletons and a big cluster containing 11471 elements, it does not seem a good choice
- Complete: Cutting at distance 2 divides the data in three clusters, a bigger one (7500 points), and two smaller ones (1400 and 2500 points)
- Average: Similar to Single, does not seem a good choice
- Ward: Cutting at distance 20 produces three clusters (4500 / 400 / 6600 points)

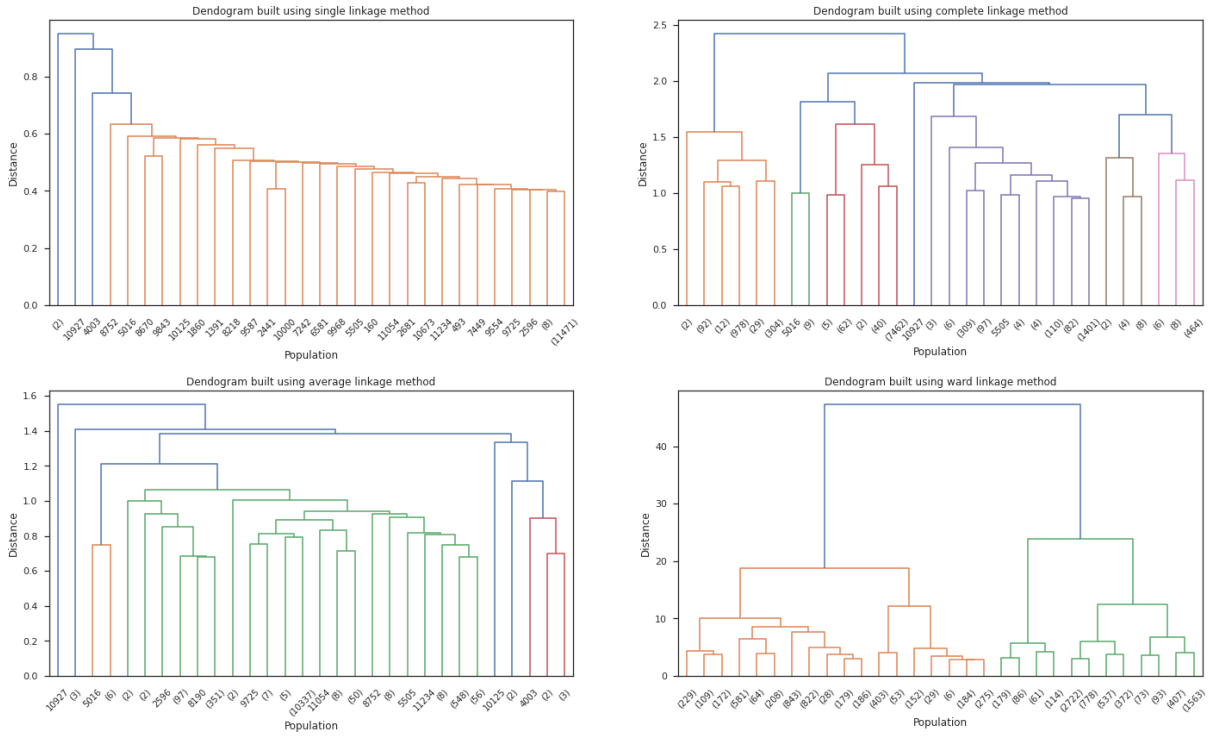

Figure 10: Dendograms built using different linkage methods.

The complete and ward seem the best options.

Figure 11 reports the clusters obtained by agglomerative clusterings, using the number of cluster (which is required in this implementation) identified in the previous phase, for the linkage methods Complete, Average and Ward.

As expected, the clustering using the Average linkage method did not produce significant results.

2.4.1 Evaluation

For the evaluation of the hierarchical clustering I also used the Cophenetic Correlation Coefficient (CPCC), which is the correlation between the entries of the matrix built by the hierarchical clustering and the original dissimilarity matrix; it is a standard measure of how well a hierarchical clustering

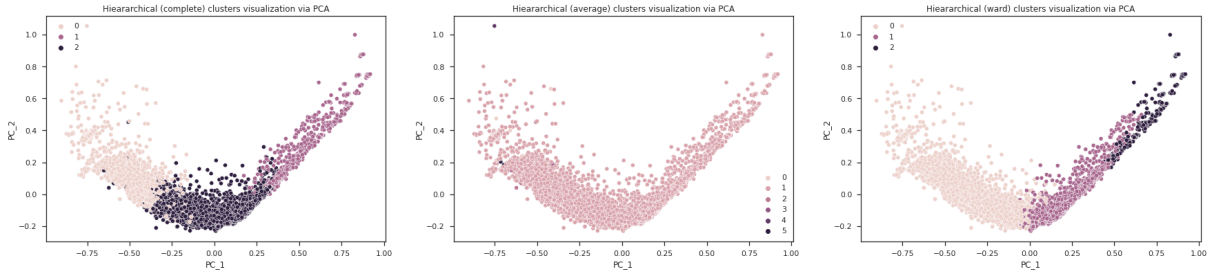

Figure 11: Hierarchical clusters (for Complete, Average and Ward linkage methods) visualization via PCA.

(of a particular type) fits the data. One of the most common uses of this measure is to evaluate which type of hierarchical clustering is best for a particular type of data.

From the results of the evaluation (see Notebook) it is possible to see that the Average method scores good for Davies-Bouldin and Silhouette because it divides the data set principally in one cluster. This means that even if the evaluation via the metrics is quite good, the results are not so significant (in fact it is very poor in the Calinski-Harabasz score).

The Ward method instead seems to be the best method.

2.5 Comparison

I compare the clustering obtained from the three different techniques described above, which are:

- K-Means with 8 clusters
- K-Means with 3 clusters
- DBScan with eps 0.15 and minPts 20
- Hierarchical using Ward linkage method, cutting at distance 20, obtaining 3 clusters

The clusters are compared between them using the Davies-Bouldin, Silhouette and Calinski-Harabasz indexes.

Type	Number_of_clusters	Davies_Bouldin	Silhouette	Calinski_Harabasz
kmeans	8	1.059017	0.404314	7355.954167
kmeans	3	0.718478	0.541799	9812.475244
dbscan	3	1.908947	0.367459	1223.815200
hierarchical (Ward)	3	0.714640	0.527053	9117.141303

The comparison shows that the best clusterings for the data set in consideration seems to be K-Means with 3 clusters and Hierarchical clustering using Ward as linkage method, with both similar scores (K-Means scores slightly better).

2.6 pyclustering

X-Means X-Means clustering is an extended K-Means which tries to automatically determine the number of clusters. The approach is to use KMeans++ to determine the initial (two) centroids given in input to the X-Means algorithm. As K-Means, X-Means tends to form globular clusters. The results of the X-Means clustering (Figure 12) show that the algorithm tends to split the data in the maximum number of clusters, because it gives the best BIC score. Anyway, even if the score is better, the clusters do not represent well the actual structure of the data.

BSAS BSAS creates new clusters as the algorithm evolves. The basic idea of the algorithm is that as each new vector is considered, it is either assigned to an existing cluster or assigned to a newly created cluster, depending on its distance from the already formed ones. BSAS clustering (Figure 13) seems to make a good separation of the data, quite similar to the one obtained by K-Means. An advantage of using this kind of clustering is that it is incremental, and then suitable for an online setting where the data is given as a stream.

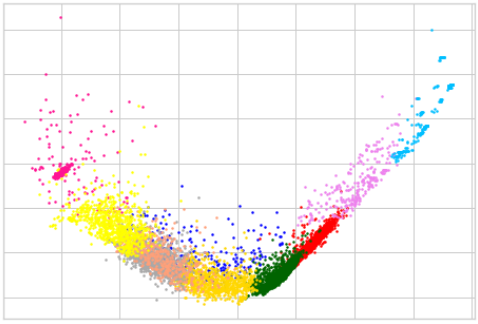

Figure 12: X-Means clusters visualization via PCA.

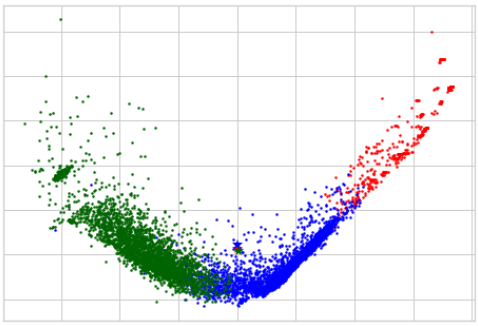

Figure 13: BSAS clusters visualization via PCA.

3 Classification

3.1 Preprocessing

- The 'id' and 'name' features are not useful for the classification task, so I drop them.
- The binary target label 'bot' is first converted into an integer (0 / 1) and then removed from the data set.
- The 'lang' feature contains many entries with low frequency (less than 30), which can be grouped together into the single entry 'Other' to reduce possible overfitting.
- The datetime 'created_at' feature can't be used as it is for the classification task. From that indicator it is possible to extract four new variables which could improve the classification performance:
  - 'created_at_year': year of subscription
  - 'created_at_weekday': day of the week of subscription
  - 'created_at_day': day of the month of subscription
  - 'created_at_hour': hour of the day of subscription

3.2 Approach

The approach adopted to build each classification model is composed by multiple steps:

1. Preprocess the data to perform actions over the data set specifically suited for the individual models.

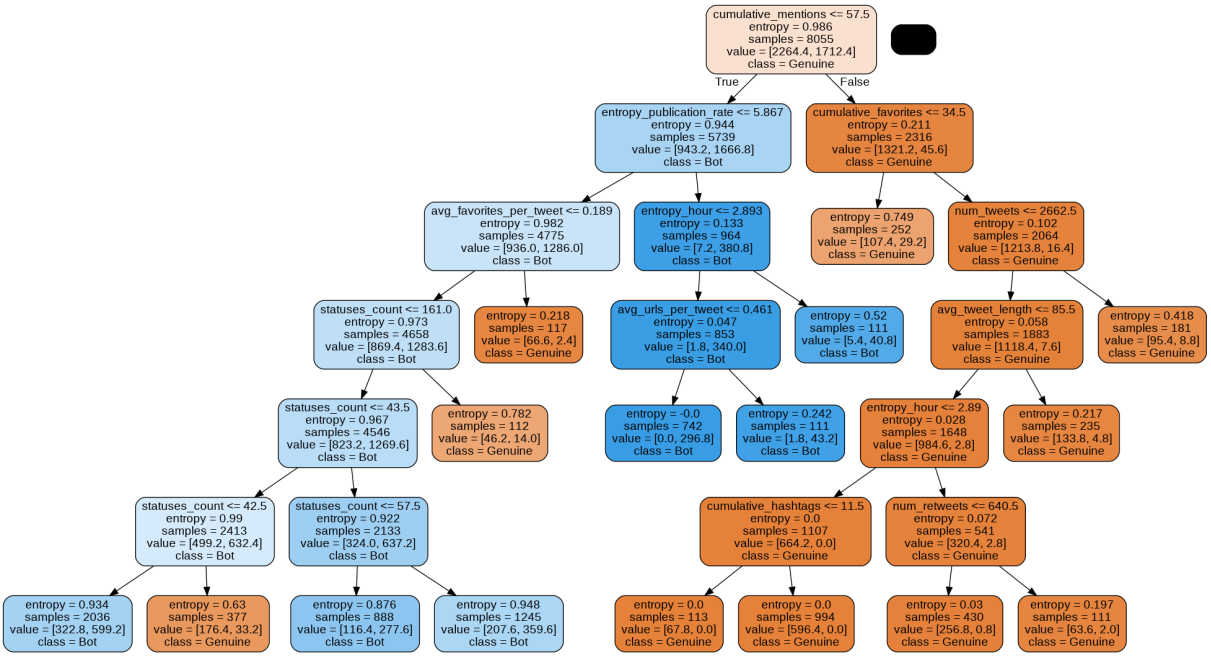

Figure 14: Decision tree obtained by the classifier.

2. Split the data set in training and test set in a 70/30 ratio, keeping the same proportions of the target class label 'bot' for both the derived sets.

3. Find the best combination of hyper-parameters via a grid search (or randomized, where the exhaustive search is infeasible) with cross validation over the training set.

4. Train the model using the combination of parameters obtained from previous step.

5. Make the predictions over the test set and evaluate the results using multiple scores (accuracy, recall, precision, F1), plotting the confusion matrix and comparing the distribution of the users according to the actual and to the predicted class labels.

3.3 Models

3.3.1 Decision Tree

Preprocessing As a preprocessing step, the 'lang' feature is converted as a one-hot numeric array in order to let the Decision Tree classifier to handle the indicator properly.

Building the model In Figure 14 it is possible to visualize the tree constructed by the classifier, which shows the features it uses to make the predictions and how the two classes of users are discriminated depending on their attributes.

Results The classifier is able to reach an accuracy of 83% over the test set, and from Table 1 it is possible to see that the model is very good in predicting the actual bots, but lacks in recognizing correctly genuine users, with a recall of 70% for the genuine users in the test set.

The problem of high false positives (see Figure 15) does not seem related to a possible poor generalization ability because the predictions on the training set are similar and neither to an unbalancement on the data because both classes are quite equally represented.

	precision	recall	f1-score
Genuine	0.91	0.70	0.80
Bot	0.78	0.94	0.86
accuracy			0.83

Table 1: Decision tree classifier scores.

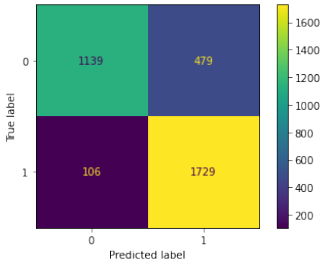

Figure 15: Confusion matrix of decision tree classifier predictions.

	precision	recall	f1-score
Genuine	0.94	0.64	0.76
Bot	0.75	0.96	0.85
accuracy			0.81

Table 2: K-NN classifier scores.

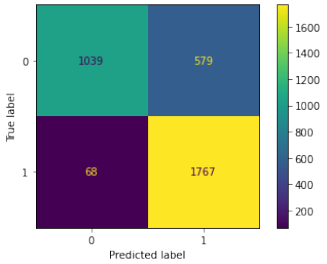

Figure 16: Confusion matrix of K-NN classifier predictions.

A possible explanation is that the features associated to the users do not permit to distinguish easily between the two classes because both genuine users and bots show a similar behavior in some cases.

From the plots of distribution of the users (see Notebook) in particular it is possible to see that the classifier is able to distinguish well the two classes when there is a clear separation between them, and tends to misclassify the instances when they are mixed.

3.3.2 K-NN

Preprocessing K-NN is based on a notion of distance between instances in a multi-dimensional space, but the categorical feature 'lang' can't be converted to numerical with a meaningful distance between languages, so I decided to drop that attribute for this type of classifier.

In addition, given that the number of features that characterize a user is quite high, to reduce possible overfitting due to the curse of dimensionality I decided to drop some features which, after some experiments, do not seem to be very informative for the actual classification model.

As a last preprocessing step, in order to avoid the distance measure to be dominated by attributes with high values, the features are scaled using a Min-Max Scaler.

Results The K-NN classifier results to be less accurate in the predictions with respect to the Decision Tree, in particular with a significant increase of false positives (see Table 2 and Figure 16).

3.3.3 Rule-based classifier (RIPPER)

The Rule based classifier used in this context is RIPPER, which is a direct method (extract rules directly from data). For 2-class problem, as in this case, it chooses one of the classes as positive class, and the other as negative class; then it learns rules for positive class, and the negative class will be default class.

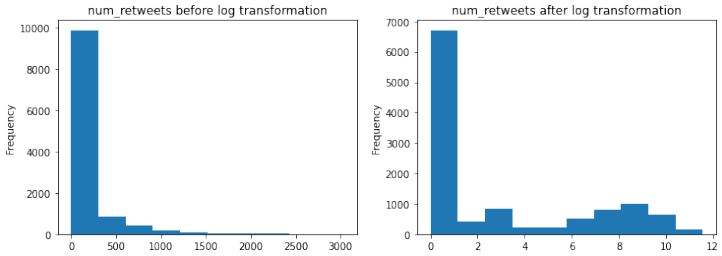

Figure 17: Log transformation of the 'num_retweets' feature.

Preprocessing Binning can be applied as a preprocessing step for the Rule based classifier to smooth the input data and then reduce the chances of overfitting caused by excessively specific rules, which reduce the generalization capabilities of the model. Anyway, the implementation of RIPPER used to build the model automatically applies binning to the features.

The output of RIPPER is a set of rules, which is not reported here but can be investigated in the notebook.

Results The Rule based classifier results to be slightly less accurate in the predictions with respect to the Decision Tree, especially with a smaller generalization capability, probably due to an excessive complexity of the rule set.

3.3.4 Naive Bayes

The very highly skewed distribution of the features will probably preclude the Naive Bayes classifiers to produce good results, given that it is more suited for Gaussian distributed data.

Preprocessing In order to try to reduce the skewness, I apply a log transformation to the features. The distribution resulting from log transformation still shows an high skewness, but it is difficult to reduce it more due to the nature of the data (see Figure 17).

Building the model Two different versions of the Naive Bayes classifier are used: Gaussian, which assumes the likelihood of the features to be Gaussian, and Multinomial which is more suited for multinomially distributed data.

Results As expected, the Gaussian Naive Bayes classifier is not particularly suited for this classification task, due to distribution of the features, which is very highly skewed.

On the other hand, Multinomial Naive Bayes classifier shows good performances in predicting true genuine users and true bot users, but is very poor in the number of false positives detected.

3.3.5 AdaBoost

AdaBoost is an ensemble method that manipulates data distribution. It performs multiple boosting rounds, each of which consists of training a base classifier based on a sampling (with replacement) of the data according to weights; the weights are updated at every round, depending on the performance of the classifier.

Preprocessing As a preprocessing step, the 'lang' feature is one-hot encoded.

The base estimator chosen for the AdaBoostClassifier is a DecisionTreeClassifier (which showed to perform well on the data set), with a maximum depth of 4.

	precision	recall	f1-score
Genuine	0.94	0.70	0.80
Bot	0.79	0.96	0.86
accuracy			0.84

Table 3: AdaBoost classifier scores.

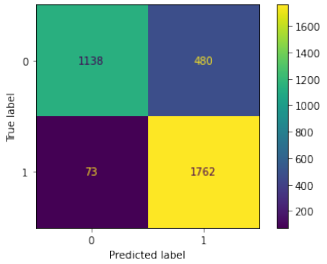

Figure 18: Confusion matrix of AdaBoost classifier predictions.

	precision	recall	f1-score
Genuine	0.92	0.71	0.81
Bot	0.79	0.95	0.87
accuracy			0.84

Table 4: Random forest classifier scores.

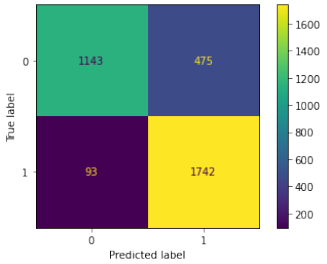

Figure 19: Confusion matrix of Random forest classifier predictions.

Results The AdaBoost classifier shows very good results in predicting actual genuine and bot users, but the number of false positives is still high and similar to the one achieved by the Decision Tree (see Table 3 and Figure 18).

3.3.6 Random Forest

Is a class of ensemble methods specifically designed for decision trees, which combines the predictions made by multiple decision trees and outputs the class that is the mode of the class's output by individual trees.

Each decision tree is built on a bootstrap sample based on the values of an independent set of random vectors; unlike AdaBoost, the random vector are generated from a fixed probability distribution.

Preprocessing As for AdaBoost, the 'lang' feature is converted to a one-hot array.

The number of trees (n_estimators) used in the Random Forest is set to $\sqrt{m}$ where m is the number of features.

In addition, the class 0 is weighted 0.6 and the class 1 is weighted 0.4, so as to improve the recall for the class of genuine users.

Results The performance of the Random Forest classifier are very comparable with the ones of the other ensemble method AdaBoost. It is able to reach very high precision for the genuine class and recall for the bot class, but the results for the recall of genuine users are still in the order of 70% (see Table 4 and Figure 19).

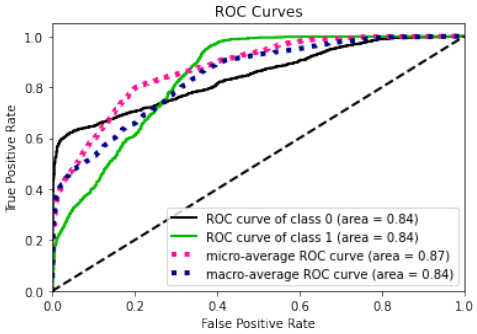

Figure 20: ROC curves relative to the classification obtained by SVC.

3.3.7 SVC

SVC (Support Vector Classifier) is based upon support vector machines (SVM), which are a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes.

SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Preprocessing Given that the notion of distance is not meaningful for the attribute 'lang', I decided to drop it.

In addition, the SVC is not scale invariant, then it is necessary to normalize the data before training the model.

Results By using the prediction probabilities computed by SVC for the elements in the test set, it is possible to plot the ROC curves for both classes of users, and the relative micro- and macro-average ROC curves (see Figure 20).

By interpreting the plot, we can understand that the classifier, for the class 0 (genuine users) is able to get a true positive rate of 60% quite easily, keeping the false positive rate very low, but when it tries to reach higher TP rates, the FP rate increases very quickly. This tendency is also evident in the poor recall obtained by the model for the class of genuine users.

For what concerns the class 1 (bot users), even if the area under the curve is the same for both classes, the model gets better results with respect to class 0, obtaining TP rates higher than 85 - 90% with a FP rate lower than 40%.

The SVC does not seem particularly suited for this data set; the number of true positives and negatives is quite high, but the model is penalized in the recall obtained for the class of genuine users, which is fairly lower if compared with other models such as Decision Tree or Random Forest, for instance.

These results probably come from the fact that the two classes does not separate well in the hyperplane created by the SVC, and then the model is not able to identify support vectors that can make a clear subdivision between different classes instances.

3.3.8 Feed-forward neural network

A feed-forward neural network has connections only between nodes of level L_i and the next one L_{i+1}.

Preprocessing The 'lang' feature is converted as a one-hot numeric array in order to let the neural network to handle the indicator properly.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 33)                0

 dense (Dense)               (None, 5)                 170

 dense_1 (Dense)             (None, 1)                 6

=================================================================
Total params: 176
Trainable params: 176
Non-trainable params: 0
```

Figure 21: Feed-forward neural network summary.

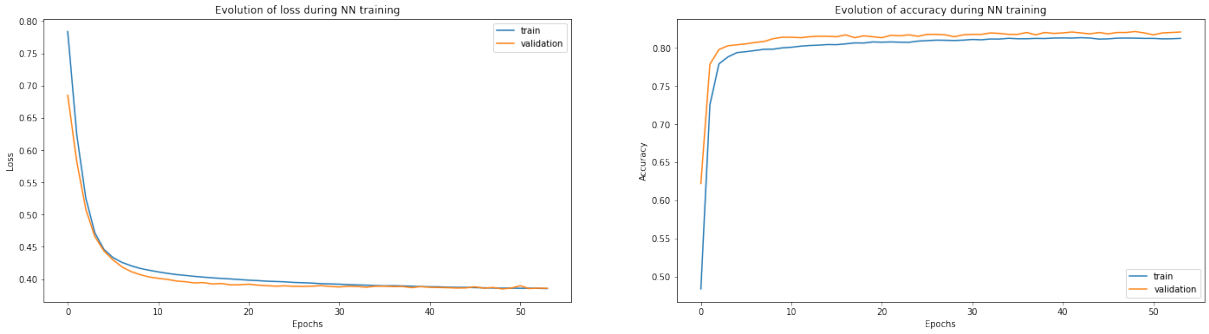

Figure 22: Evolution of loss and accuracy during feed-forward NN training.

Before giving in input the data set to the neural network, the values are normalized to have mean 0 and variance 1.

Building the model The neural network used for the classification is composed by three layers (see Figure 21):

1. The input layer is composed by as many neurons as the number of features of the data set. The layer is set as Flatten so as the output is flattened to a 1-D array, suitable for the next hidden layer
2. The hidden layer is composed by a number of neurons equal to the square root of the number of features and is a Dense layer, then each node is connected to every node of the previous layer
3. The output layer has a single neuron, since the classification task is binary

The plots reported in Figure 22 show the evolution of the loss and the accuracy through the epochs of the neural network.

Results The feed-forward neural network shows good results (See Table 5 and Figure 23) in the precision for the genuine users and the recall for bot users, but the number of false positive is fairly higher if compared to other models such as Random Forest.

Neural networks are not particularly suited for this kind of task, and the built feed-forward model is maybe too simple to obtain better performances over the data set.

3.3.9 Multi-layer perceptron neural network

The MLPClassifier is based on a multi-layer perceptron neural network that trains using back-propagation.

Preprocessing As for the feed-forward neural network, the 'lang' feature is converted as a one-hot numeric array and the values of the features are normalized.

	precision	recall	f1-score
Genuine	0.95	0.64	0.76
Bot	0.75	0.97	0.85
accuracy			0.81

Table 5: Feed-forward NN classifier scores.

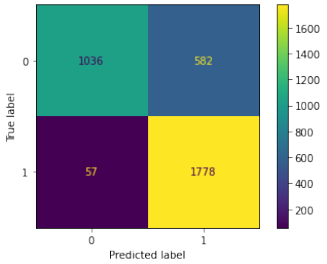

Figure 23: Confusion matrix of feed-forward NN classifier predictions.

	precision	recall	f1-score
Genuine	0.92	0.65	0.76
Bot	0.75	0.95	0.84
accuracy			0.81

Table 6: MLP NN classifier scores.

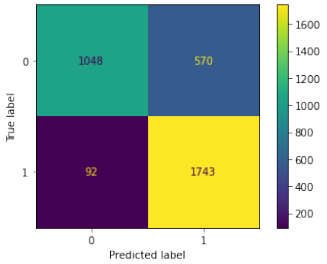

Figure 24: Confusion matrix of MLP NN classifier predictions.

Results The MLPClassifier shows similar performance with respect to the classifier based on the previous feed-forward network, but still it can't reach a recall for the genuine users comparable to the one obtained by the Random forest classifier (see Table 6 and Figure 24).

From the ROC curves plotted in Figure 25 it is possible to see that the behavior of the MLPClassifier is very similar to the one of the SVC. Also in this case, for the class 0 (genuine users), the classifier need to significantly increase the FP rate to obtain good values for the TP rate.

3.4 Comparison

Comparing the accuracy obtained by each model built during this classification task (Figure 26) it is possible to observe that:

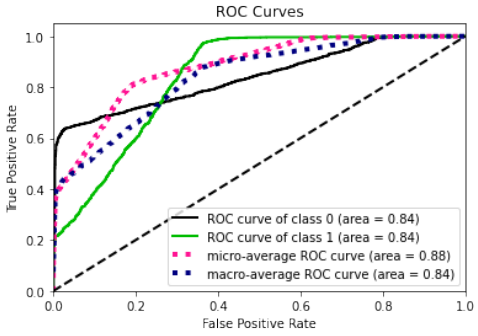

Figure 25: ROC curves relative to the classification obtained by MLP classifier.

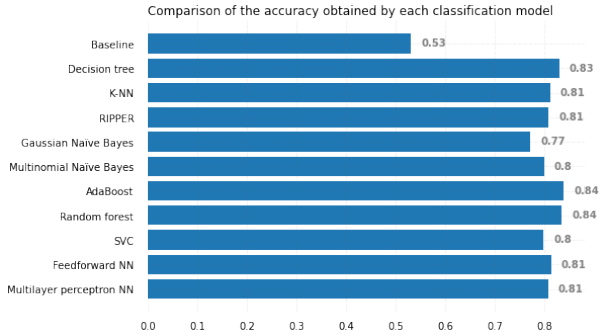

Figure 26: Comparison of the accuracy obtained by each classification model.

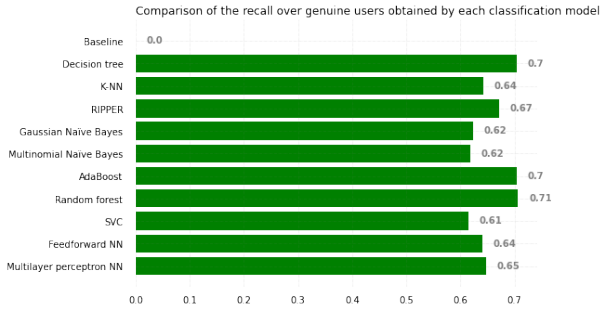

Figure 27: Comparison of the recall over genuine users obtained by each classification model.

- All the models outperform the baseline performance by a 25/30% factor
- The Naive Bayes classifiers are the worst performing, due to the general high skewness of the features included in the data set
- The classifiers based on neural networks are not the best models for this classification task, probably due to the unclear separation between bots and genuine users for many attributes' values
- Decision tree classifier shows good performance, being able to produce a remarkable accuracy
- Ensemble methods based on decision trees are the best performing ones, reaching an accuracy of 84%, thanks to their ability to exploit the conjunction of simple classifiers to better fit the data

For what concerns the recall over genuine users obtained by each classification model the results are quite analogous (Figure 27); the models with the best recall score for genuine users are the ensemble methods Random forest and AdaBoost, in addition to the Decision tree classifier, whereas the worst performing ones are the Naive Bayes classifiers and SVC.

4 Time series analysis

4.1 Preprocessing

In order to extract the success score time series for the year 2019 for each user I perform the following steps:

1. Take only the tweets published in 2019

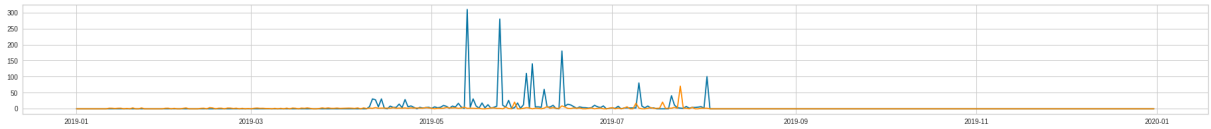

Figure 28: Comparison of two example time series.

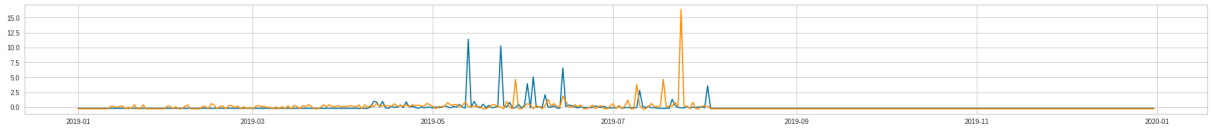

Figure 29: Time series obtained from the ones of Figure 28 after applying amplitude scaling.

2. Filter the tweets taking only the ones belonging to a user also listed in the 'users' data set (this step is necessary for the shapelet extraction)
3. For each user compute the Success score for each day of 2019 where she / he has published at least a tweet (see Listing 1). The computation is performed by:
   (a) Group the tweets by 'user_id' and 'created_at'
   (b) For each group, take the sum of 'retweet_count', 'reply_count', 'favorite_count', 'num_hashtags', 'num_mentions', 'num_urls'
   (c) By using the sum of the values, compute the success score for each user in each day of 2019.
4. For each user fill the missing days of the time series with the value -1

```
successScores = tweets_df
    .groupby(['user_id', tweets_df['created_at'].dt.date])
    .sum()
    .agg(successScore, axis="columns")
```

Listing 1: Step 3 of success score computation

From Figure 28 it is possible to see two examples of resulting time series.

Amplitude scaling In order to reduce distortion due to different amplitudes, the time series are transformed by removing the mean and dividing by the standard deviation. The resulting time series, obtained from the ones plotted in Figure 28, are reported in Figure 29.

4.2 Clustering

Two types of clusterings are performed over the time series:

- Partitional clustering, using TimeSeriesKMeans to group similar time series based on their Dynamic Type Warping distance
- Feature-based clustering, extracting relevant features from the time series and using them to produce the clustering

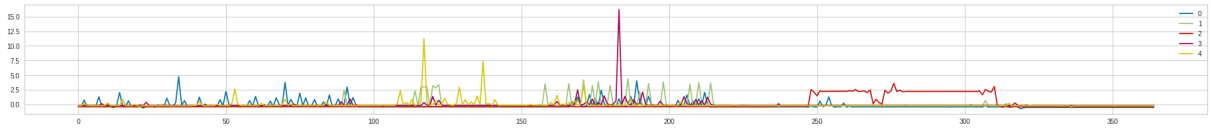

Figure 30: TimeSeriesKMeans cluster centers comparison.

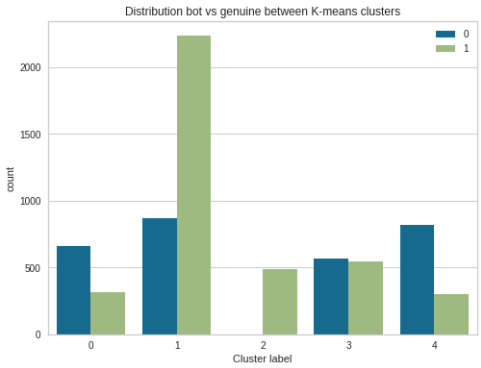

Figure 31: TimeSeriesKMeans clusters distribution comparison.

4.2.1 Partitional clustering

To determine the most suitable value for the best K in TimeSeriesKMeans, I use the elbow and silhouette analysis. From these analysis (and after experiments), the number of clusters has been set to 5.

The resulting clustering has a SSE value of 79, which is quite good.

Comparing the cluster centers time series (Figure 30) we can see that:

- The users into cluster 0 have an oscillating success score for the first three months of the year and in June; for the rest of the year the score is limited and stable
- The users into cluster 1 have a limited activity for the whole year, with some small peaks between May and July
- The users into cluster 2 have a stable and very low value of success score up to August, and then their tweets become trending for 2 / 3 months
- The users into cluster 3 have a very high peak by the end of June and at the beginning of July, whereas the rest of 2019 is quite stable
- The users into cluster 4 have a small success score for the majority of the year, except for April where there are some significant peaks

By plotting the distribution of users between the clusters with respect to the class label (see Figure 31), it is interesting to see that:

- Cluster 2 contains almost no genuine users
- Cluster 0 and 4 have more than double of genuine users with respect to bots
- In cluster 1 the number of bots is almost three times the number of genuine users

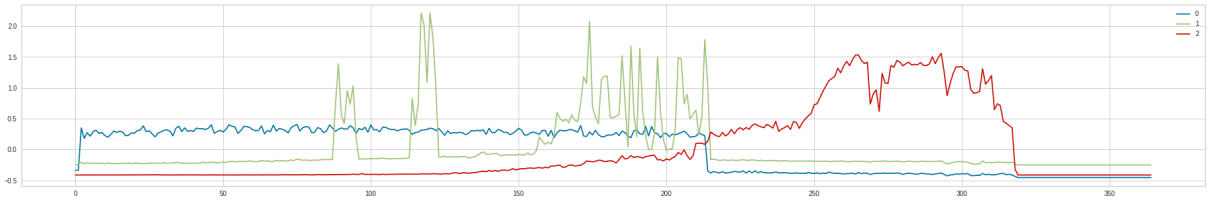

Figure 32: KMeans cluster centers comparison.

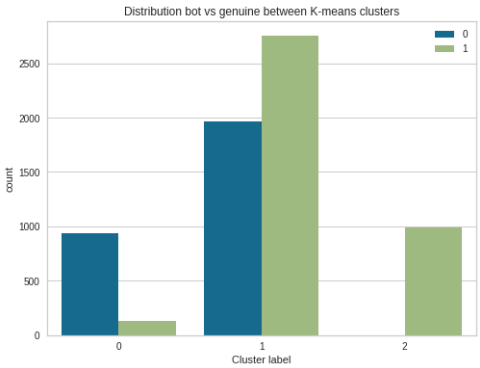

Figure 33: KMeans cluster distribution comparison.

4.2.2 Feature-based clustering

The relevant features used for the clustering are computed using tsfresh. First the relevant features are extracted from the time series, then the relevance table is calculated, and the 11 most relevant features are chosen to be used for the clustering.

As for TimeSeriesKMeans, the best K is determined using the elbow and silhouette method. The resulting value from the analysis of the number of clusters is 3.

From the plot in Figure 32 we can see the mean behavior of users for each of the three clusters identified:

- Cluster 0 groups users with a limited and stable Success score towards the year
- Cluster 1 identifies users having multiple peaks of Success score, especially up to July
- Cluster 2 represents users with a limited Success score up to August, then the tendency changes, gaining a high Success for a couple of months, probably because the topic of their tweets was trending for that period of 2019

From Figure 33, it is possible to see that cluster 2, which has a similar tendency of cluster 2 from the partitional clustering, only contains bot users. On the other hand, cluster 0 contains genuine users for the vast majority.

4.3 Shapelet extraction

Shapelets, being time series subsequences which are maximally representative of a class, can be used for a classification task. In our setting, we can exploit the shapelet extracted from the time series to classify the users being genuine or bots.

The number and length of the shapelets to extract are computed according to the heuristic developed by J. Grabocka.

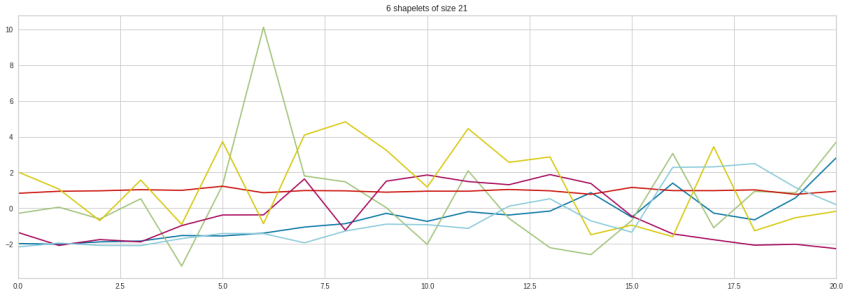

Figure 34: Shapelets extracted by LearningShapelet.

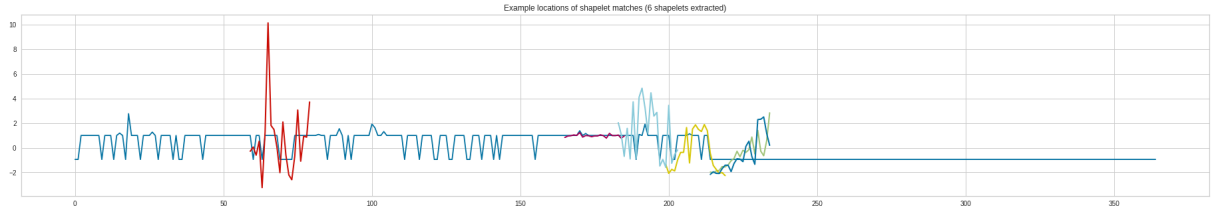

Figure 35: Shapelets location over a success score time series.

In LearningShapelet from the package tslearn, shapelets are learned such that their distances to series can linearly separate the time series instances by their targets.

The NN classifier based on shapelets achieves a classification accuracy of 72%.

After that the model has been trained, it is possible to plot the discovered shapelets (see Figure 34).

Shapelet location From the model it is also possible to compute the shapelet match location for any input time series. With the aid of this functionality it is possible to plot a time series of the data set, the shapelets extracted by the learner and their best matching location over the time series (Figure 35).