Data Mining Project Report

Bucci Alessandro (638619) - a.bucci12@studenti.unipi.it
Cignoni Giacomo (581112) - g.cignoni3@studenti.unipi.it
Marinelli Alberto Roberto (638283) - a.marinelli9@studenti.unipi.it

Master Degree Computer Science, AI curriculum
Data Mining, Academic Year: 2022-2023

Contents

1 Data understanding

The **data understanding** phase aims to unify the dataset in order to improve the result of subsequent operations. In particular, we focused on data analysis in order to eliminate duplicate values, handle missing or corrupted data, and identify outliers.
The data on which this work is done are derived from crawling on the **Twitter** platform from which two datasets were extracted, one about **users** and one about **Tweets**; these contains *11.508* and *136.648* records, respectively.

1.1 Features informations

Before manipulating the data, the first task performed is to understand the semantics of of each attribute we need to work with. For each attribute we also differentiate between the original data type and the cast data type.

Features of Tweets DataFrame			
Feature name	Initial Type	Cast Type	Description
id	int64	Int64	The identifier of the single tweet
user_id	object	Int64	Is the identifier of the who wrote the tweet
retweet_count	object	Int64	Number of retweets of the single tweet
reply_count	object	Int64	Number of replies of the single tweet
favorite_count	object	Int64	Number of favorites (or likes) of the single tweet
num_hashtags	object	Int64	Number of hashtags in the single tweet
num urls	object	Int64	Number of urls in the single tweet
num_mentions	object	Int64	The number of urls in the single tweet
created_at	object	Datetime64	Timestamp of when the single tweet was created
text	object	String	Text of the single tweet

Table 1: Features overview of the Tweets DataFrame

Features of Users Dataframe			
Feature name	Initial Type	Cast Type	Description
id	int64	Int64	The identifier of the single user
name	object	String	The name of the user
lang	object	String	The user's chosen language
bot	int64	Boolean	A binary variable that indicates if a user is a bot or a genuine user
created_at	object	Datetime64	The timestamp of when the user was created
statuses_count	float64	Int64	The count of the tweets made by the user at the moment of data crawling

Table 2: Features overview of the Users DataFrame

1.1.1 Datatype Casting

In both datasets, due to incorrect or generic initial data types, an attribute casting operation was required to obtain the data in the format described in Table 1 and Table 2. During this operation, values that did not conform to their type (e.g. a string or an infinite value to an integer value) were set to null. In order to allow null values we preferred Pandas nullable data types (e.g. Int64 instead of np.int64) when possible.

1.2 Joining tables

Tables are joined with an inner join on the *user_id* attribute. Thus, the resulting DataFrame has number of rows equal to the tweets with a joinable user and each row contains the user features other than the tweet features.

1.2.1 Correlation of initial features

Checking the correlation map of the features we discovered that all features are practically uncorrelated, then no drop of feature was needed.

1.3 Management of damaged or missing values

1.3.1 Outliers and noise Detections

In this section, we explored the distributions of the significant attributes in both tables, with boxplots and barplots, in order to find eventual outliers and noise in the data. We identified outliers and noise following a logical principle, values that are not plausible were set to null.

In tweets DataFrame, values from *retweet_count*, *reply_count* and *favorite_count* were substituted if they exceeded the values from the record holder tweets respectively for most retweet, most replies and most likes [2] [3].

Instead, values from *num_hashtags*, *num_urls* and *num_mentions* were removed if they exceeded the number of max allowed characters in a tweet (280), as all these features have to be explicated in the text.

For the user dataset, we found no value in *stauses_count* attribute being invalid. Instead, the *lang* attribute featured various values as *"Select Language..."* , obviously being invalid, and one value being *"xx-lc"*, that do not correspond to any ISO language code [1]. The occurrences of the first case were substituted with the mode of the *lang* attribute (*"en"*), while for the second case we analyzed the tweets of said user and observed that his tweets were in English and so substituted with the assumed language *"en"*. We also decided that if the "lang" attribute contains a dash, only the language preceding it is selected (e.g., "en-gb" become "en", "zh-tw" become "zh"), thus excluding regional differences of the same language.

1.3.2 Manage Duplicates

Duplicate management was done in two steps, in both datasets; initially all rows that were exactly identical were removed.

Then, in the **tweets dataset**, all duplicate records were identified by considering only *id* (of the Tweet) and *id_user* (who wrote the tweet). The goal of this operation was to search for duplicates of the same Tweet written by the same person in order to be able to match attributes and correct those with corrupted or null values. Nonetheless, the result of this analysis did not lead to the correction of any value because, when analyzing the result of this operation matching row by row, the texts and related data of the Tweets were completely different so it was assumed that an error was made in the extraction of the Tweet id during crawling, thus the **presence of different Tweets with the same id** was discovered.

Finally, the same operation was performed for the user dataset, searching for duplicate users with the same id produced no records.

1.3.3 Manage Null

We tried to replace null values with significant aggregator functions, such as median, to measure the central tendency of each attribute.

In the Users DataFrame we replaced null values for the *stautses_count* attribute with two possible medians, one calculated over the users with a true *bot* attribute value and the other over the users with a false *bot* attribute value. The null *stautses_count* value is substituted with the corresponding median to the *bot* value of its row.

A similar approach is used for the numeric attributes in the tweets dataset (*retweet_count*, *reply_count*, *favorite_count*, *num_hashtags*, *num_urls*, *num_mentions*). We substitute each null attribute with the median of that attribute calculated on **tweets of the same user**. If the number of tweets for that user with valid (non-null) attribute is less than 20, we considered the user attribute median as unreliable, due to the scarcity of tweets. In this case, we used the median for that attribute calculated differentiating between tweets belonging to a *bot* or *not bot* user.

The median and number of valid attribute tweets for each user were calculated using supporting tables.

The null values in the *text* attribute of the tweets dataset were substituted with the void string, as it makes no sense trying to recover information about a portion of text using central tendency measures, and other methods would be too computationally expensive.

2 Data preparation

2.1 Indicators

Examining the data, we obtained several indicators for each user.

- **tweet_count** is obtained counting the tweets published by an user.

- **tweet_with_text_count** is obtained counting the tweets published by an user with *'text'* attribute not null.

- **avg_text_len** is obtained through averaging the tweets' length published by an user.

- **entropy_*time_precision*** is the entropy of a user given ***time_precision***. The ***time_precision*** we chose are *seconds, minutes, 15_minutes, hours, days.* The entropies are further explained in the next subsection.

- **post_in_*valid_year*** is the count of tweets of a user in a ***valid_year***. The ***valid_year*** we decided to use are ranging from *2012* to *2020*, since the tweets published before *2012* are placed before the creation of Twitter and the tweets published after *2020* are in *2032* and after.

- **years_outside_of_plausible_range** is the count of all tweets published before *2012* and after *2020* for each user.

- **creation_year** is the creation year of the user.

- ***numeric_attribute*_avg** is the average of the ***numeric_attributes***. The ***numeric_attributes*** we used are *'retweet_count', 'reply_count', 'favorite_count', 'num_hashtags', 'num_urls', 'num_mentions'*

- ***numeric_attribute*_sum** is the sum of the ***numeric_attributes*** described above.

- The **years_outside_of_plausible_range** is obtained by counting all the tweets, for each user, that falls outside the year range *2006-2020* for each user, where *2006* is when twitter was released and *2020* is the year in which the crawling was performed.

- **total_success_score** is an aggregation of multiple numeric tweet attributes as explained in the following subsection.

2.1.1 Entropy

The **entropy** is calculated for each user using the timedeltas between the datetime of each tweet. A timedelta is obtained by subtracting two datetimes and the result is the time between the two datetimes:

```
2022-01-01 02:00:00 - 2022-01-01 01:00:00 = Timedelta('0 days 01:00:00')
```

The main idea is to subtract from each datetime of a tweet of a specific user, the datetime of the tweet before published by the same user.

In this way we obtained the time passed between the publishing of each tweet. Given all the timedeltas of an user we then count for each unique timedelta the number of occurrences of that timedelta, then we calculate the entropy of that user using the unique timedeltas as categories.

We used different time precision to calculate different entropies to help us clusterize the data in the next task. The time precision that were used are: seconds, minutes, quarters of an hour, hours and days.

2.1.2 Total Success Score

The *'total_success_score'* is defined as:

$$\text{Total Success Score} = \frac{\text{Total Acceptance Score}}{\text{Total Diffusion Score} + 0.1} \quad \text{where}$$

- Total Acceptance Score = *retweet_count_sum*+*reply_count_sum*+*favorite_count_sum*

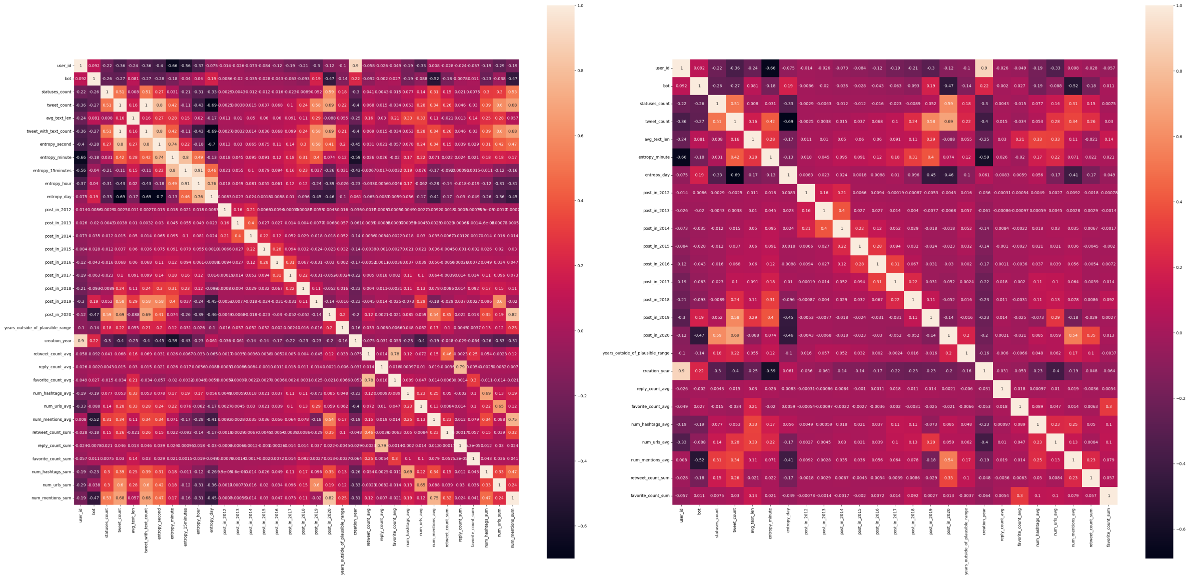
- Total Diffusion Score = *num_hashtags_sum* + *num_mentions_sum* + *num_urls_sum*

In our cleaned dataset, every user has published at least one post, so the *total_success_score* will be always non-NaN.

2.2 Choosing uncorrelated indicators

We can see in Figure 1a that there are many positively and negatively correlated features. The first two rows are just for checking how much the indicators are correlated to bot feature, we do not use them in the clustering examination nor do we use them to eliminate correlated features.

We chose to delete the features with the correlation higher than 0.7 and lower than -0.7, the result can be seen in Figure 1b.



(a) **(b)**

Figure 1: Correlation map of the chosen indicators before (a) and after (b) cleaning the highly correlated

3 Clustering

3.1 Preprocessing

After selecting the most significant features during the data preparation phase, the data had to be pre-processed before clustering could begin. In particular, two datasets were generated with different features within them. One consisted of almost all indicators (with more emphasis on temporal information) and the second consisted of the smallest number of representative features.

3.1.1 Dataset configurations

For both datasets, features such as *'user_id'*, *'lang'* and *'created_at'* were discarded because they did not add any useful information for clustering purposes. In addition, for the **complete dataset**, *'total_success_score'* was discarded because all the indicators from which it was derived were included. In contrast, for the **minimal dataset**, only the *'total_success_score'* and other indicators such as *'statuses_count'*, *'tweet_count'*, *'avg_text_len'*, *'entropy_minute'* and *'entropy_day'* were maintained. Finally, the 'bot' column was saved separately to perform evaluations and checks on clustering results.

3.1.2 Logscale reduction

Due to some attributes having some excessively high values, it was necessary to apply a reduction in logscale values. More specifically, the attributes subjected to this operation were: 'statuses_count', 'favorite_count_avg', 'favorite_count_sum', 'retweet_count_sum' and 'reply_count_avg'.

3.1.3 Normalization

A good practice in clustering to avoid the bias given by the range of the different attribute is normalization. The most common adopted normalizations are: Z-Score and Min-Max. We tried both normalizations by selecting the best one for each type of clustering and to the results produced.

3.2 DBSCAN

To detect irregularly shaped clusters, **DBSCAN** was used on both data configurations. In this section, only the one on the *minimal dataset* is documented, because it produces better results than the *complete dataset*, in which only a large cluster containing 99% of the samples was generated.

3.2.1 Determining eps and min points

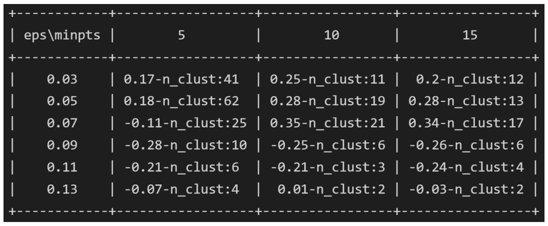
To select the best ranges of the eps in which to start clustering, the Knee Method was used, both for the normalized dataset with min max and the standard scaler. As a result of this, *0.03-0.13* was chosen as the range for the eps of the experiment on the dataset normalized with min max and *1.0-0.1* for that normalized with the standard scaler. The min points were selected through a grid search by selecting three possible values: *5*, *10* and *15*.

3.2.2 Clustering results

Using the ranges listed above, grid searches were performed for both the normalised dataset with min max and the standard scale. The results are shown in Figure 2.

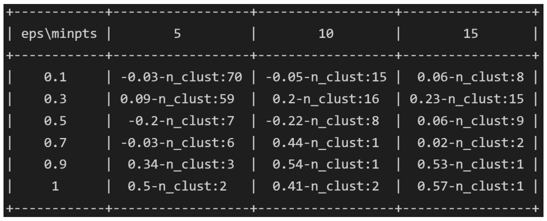
The clusters obtained were subsequently analysed in more detail. Discarding the results with too many clusters and those with only one large cluster, clustering with 0.03 eps and 10 minpts from the grid search on the MinMax was selected for further analysis.

Using the *'bot'* data, it was noted that clusters in this configuration almost always have a well-defined distribution of only bots or only users. Despite this, half of the clusters are very small in size.



```
+------------+------------------+-----------------+-----------------+
| eps\minpts |        5         |       10        |       15        |
+------------+------------------+-----------------+-----------------+
|    0.03    | 0.17-n_clust:41  | 0.25-n_clust:11 |  0.2-n_clust:12 |
|    0.05    | 0.18-n_clust:62  | 0.28-n_clust:19 | 0.28-n_clust:13 |
|    0.07    | -0.11-n_clust:25 | 0.35-n_clust:21 | 0.34-n_clust:17 |
|    0.09    | -0.28-n_clust:10 | -0.25-n_clust:6 | -0.26-n_clust:6 |
|    0.11    | -0.21-n_clust:6  | -0.21-n_clust:3 | -0.24-n_clust:4 |
|    0.13    | -0.07-n_clust:4  |  0.01-n_clust:2 | -0.03-n_clust:2 |
+------------+------------------+-----------------+-----------------+
```

(a) MinMax normalization grid search



```
+------------+------------------+------------------+-----------------+
| eps\minpts |        5         |        10        |       15        |
+------------+------------------+------------------+-----------------+
|    0.1     | -0.03-n_clust:70 | -0.05-n_clust:15 |  0.06-n_clust:8 |
|    0.3     | 0.09-n_clust:59  |  0.2-n_clust:16  | 0.23-n_clust:15 |
|    0.5     |  -0.2-n_clust:7  | -0.22-n_clust:8  |  0.06-n_clust:9 |
|    0.7     | -0.03-n_clust:6  |  0.44-n_clust:1  |  0.02-n_clust:2 |
|    0.9     |  0.34-n_clust:3  |  0.54-n_clust:1  |  0.53-n_clust:1 |
|     1      |  0.5-n_clust:2   |  0.41-n_clust:2  |  0.57-n_clust:1 |
+------------+------------------+------------------+-----------------+
```

(b) Z-score normalization grid search

Figure 2: DBSCAN grid seaerches (silhouette scores and number of clusters)

3.3 Hierarchical clustering

Exploring agglomerative **hierarchical clustering**, only results on the *complete dataset* were reported because on the *minimal dataset*, no sufficiently discriminative clusters (between bots and non-bots) were obtained.

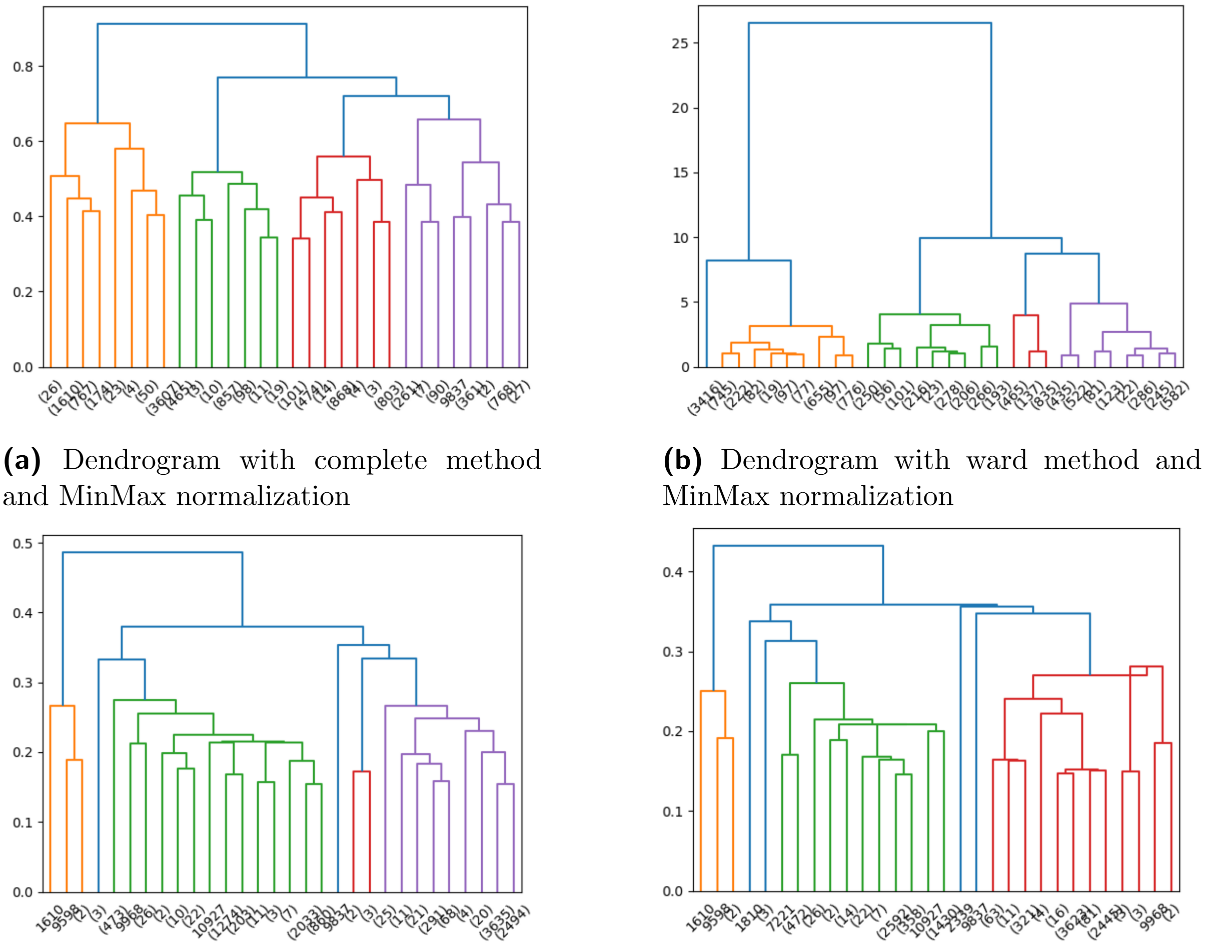
3.3.1 Distance metric

Before starting agglomerative **hierarchical clustering**, it was necessary to select a metric for distance, in our case, after some initial trials we identified **cosine** as the best.

3.3.2 Dendrogram for hierarchical clustering

Dendograms were produced for both normalizations on the dataset, trying different methods and keeping only those that produced better results. In particular, we chose the *"complete"*, *"ward"*, *"mean"*, and *"centroid"* methods, discarding *"single"* because it is sensitive to outliers and in our dataset we did not remove outliers based only on statistical methods.

The cutoff values were selected for each dendogram in order to avoid cutoffs that would produce classes that were too small or had an unbalanced number of elements.



(a) Dendrogram with complete method and MinMax normalization

(b) Dendrogram with ward method and MinMax normalization

(c) Dendrogram with average method and MinMax normalization

(d) Dendrogram with centroid method and MinMax normalization

Figure 3: Dendrogram of hierarchical clustering with different methods and MinMax normalization

3.3.3 Clustering results

Observing the distributions of the elements and on the silhouette values, among the results of hierarchical clustering we selected two of them for further analysis.

The most interesting clustering were the following:

1. MINMAX normalization - dist: cosine, method: ward, n_clusters: 5, silhouette: 0.39, whose dendogram is shown in Figure 3a

2. STD normalization - dist: cosine, method: ward, n_clusters: 5, silhouette: 0.31

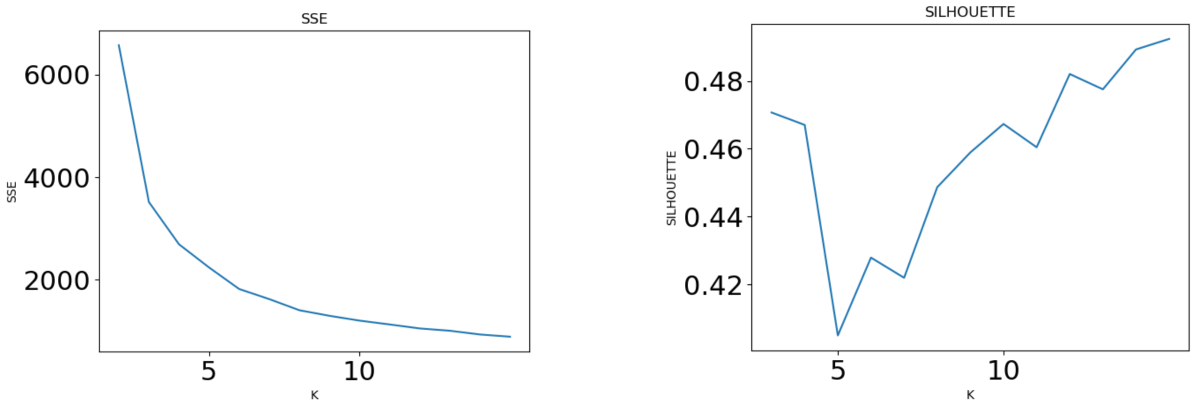
In both, a good balance of bot and non-bot distributions as well as a fair size of clusters can be achieved.

3.4 K-Means

For K-Means we chose to use only minmax normalization, as it is known to be a more suited normalization for this algorithm. In addition, the results on the *complete dataset* have been reported because they are slightly more accurate than those of the *minimal dataset*.

3.4.1 Choosing the best K

The best value of k was found by initiating a grid search in which all values from *2* to *15* were tried and the corresponding silhouettes and SSE calculated (Figure 4).



(a) K-Means SSE graph

(b) K-Means Silhouette graph

Figure 4: K-Means graphs for SSE and Silhouette

After analyizing SSE and Silhouette, we selected *K=5*, as it is approximately at the last significant SSE decrease in the SSE graph, while maintaining a relatively high Silhouette.

3.5 X-Means

We tried the X-Means implementation from Pyclustering library, with Bayesian Information Criterion (BIC) splitting criterion. We used *2* as the minimum number of clusters and a variable between *20* and *100* as the maximum. Centroids were initialized with K-Means++ heuristic.

With both complete and minimal features configurations, X-Means fails to achieve a good clustering, always selecting as the number of clusters the allowed maximum.

3.6 Final clustering selected

The final selected clustering method is the **hierarchical clustering**, executed with *minmax normalization* and *"ward"* using the *complete dataset*.

This choice was made by looking at the distribution of the clusters, their consistency in the scatter plot and to the distribution of bots and non-bots in each cluster.

In particular, of the clusters generated, one composed of almost only users and one composed of only bots are of particular interest (Figure 5).

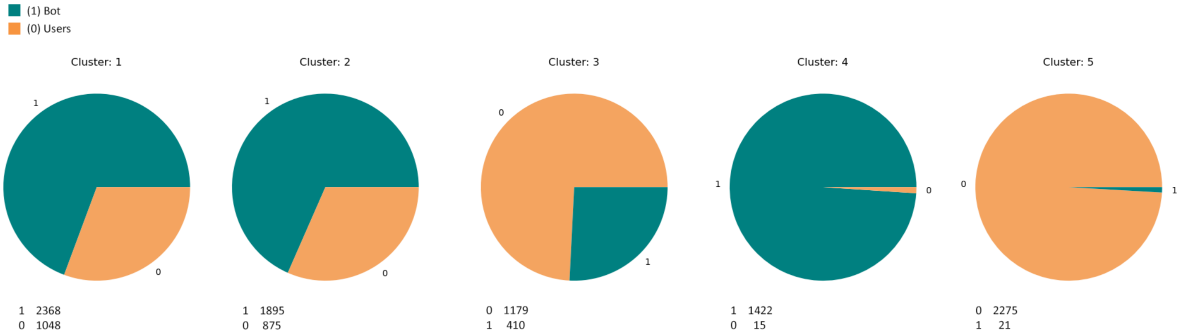


Figure 5: Distribution of bots in the clusters of final clustering.

The pie plots show 5 types of clusters:

1. Predominantly bot
2. Predominantly bot
3. Predominantly non-bot
4. All bot
5. All non-bot

Analysing each of these results by using radar plots, it was found that clusters including mostly bots have a lot of tweets from *2019*, *2018* and *2017*, while clusters with mostly humans have tweets mostly in *2020*. Cluster *2* of mostly bots also has much less tweets for user. Clusters *4* and *5* instead have many more tweets per user on average (over 2000). Among all, average tweet text length does not change much. Clusters *4* and *5* of exclusively bots and exclusively humans have higher favorite sum, but cluster *5* of humans have the highest retweet sum on average, indicating that humans (at least in this cluster) gets more retweets. Clusters *1* and *2* instead present users with few likes and retweets on average.

4 Classification

4.1 Dataset preparation

4.1.1 Normalization and log scale

We converted some numeric features of aggregated tweet parameters (e.g. *favorite_count_avg*) and *total_success_score* to log scale and normalized with Z-score normalization all the features.

4.1.2 Train-Test split

We reserved 15% of the dataset for testing and another *15%* for validation, the rest *70%* is the training set. We defined also a *development set*, composed by the training and validation set (*85%* of the data) as some models do not need validation step or use the entire development set for cross validation.

4.1.3 Score Metric

As the dataset (and its splits) are almost balanced in term of bot and non-bot distribution (*53%* bots, *47%* non-bots), we decided to use accuracy as the chosen metric for evaluating the explored models.

4.2 K-NN

We trained K-Nearest-Neighbours model on the training set, for values of K from *1* to *20* and plotted the training and validation accuracy, as shown in Figure 6. The selected K is the smallest for which the 2 training and validation curves get near each other, that is the model with the smallest K such that it does not fall into overfitting. The best K-NN are tested on the test set.

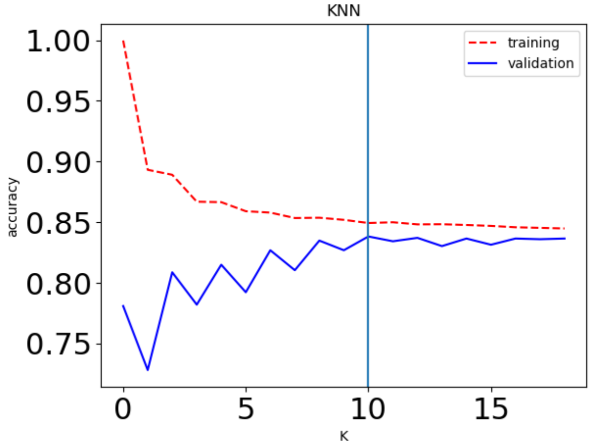


Figure 6: Curves of the training and validation accuracy for KNN at increasing K (classification with full set of features). The selected K is also shown.

4.3 Naive Bayesian classifiers

We tried both Bernoulli and Gaussian Naive Bayesian classifiers, trained on the training set and tested on validation. The best of 2 was also chosen for testing on the test set. (We recall that Bernoulli applies a binarization on the features).

4.4 Decision Tree

Decision tree was trained performing 6-fold cross validation on the entire development set to find the best hyperparameters. After CV, the model with the best found set of hyperparameters is retrained on the whole development set data, then tested on test set.

The model is implemented with *Scikit-learn* and the tuned hyperparameters are: *criterion, max_depth, max_features, min_samples_split, min_samples_leaf.*

4.5 Random Forest

Random forest model was trained performing 6-fold cross validation on the entire development set to find the best hyperparameters. After CV, the model with the best found set of hyperparameters is retrained on the whole development set data, then tested on test set.

The model is implemented with *Scikit-learn* and the tuned hyperparameters are: *max_depth, max_features, n_estimators, min_samples_leaf.*

4.6 SVM

SVM model was trained performing 6-fold cross validation on the entire development set to find the best hyperparameters. After CV, the model with the best found set of hyperparameters is retrained on the whole development set data, then tested on test set.

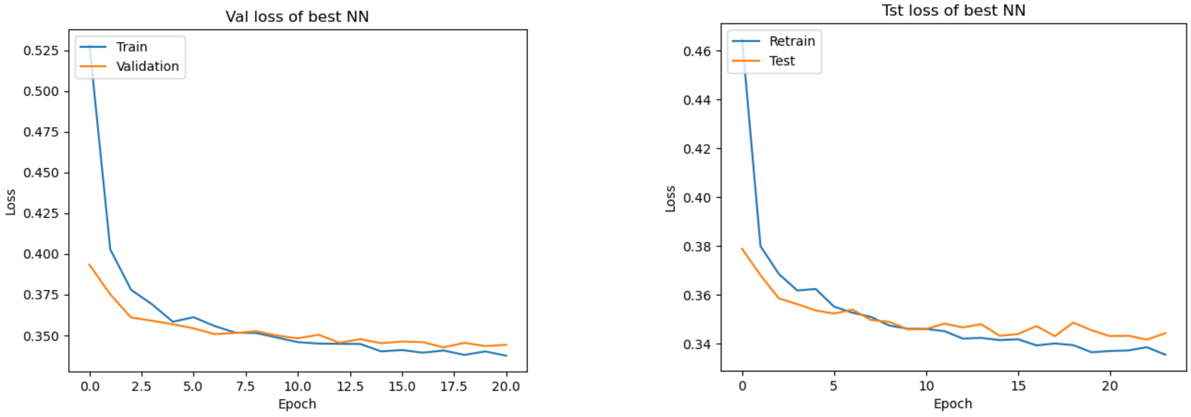
The model is implemented with *Scikit-learn* and the tuned hyperparameters are: *kernel, gamma, C*. The hyperparameter *degree* was tuned only when *kernel = poly*.

4.7 MLP

In regard to the Multi Layer Perceptron model we used training and validation set respectively to train and validate different fully connected NN architectures. Early stopping based on validation loss was employed in this phase.

After the best architecture was found, we retrained the final model with the whole development set, using a number of epochs slightly larger than the one used in the first training phase. The final model is then tested on the test set. An example of the training, validation and test curves for can be seen in Figure 7

The explored architectures have between 1 and 3 hidden layers with ReLU as activation functions and the output 1 neuron layer using sigmoid activation. Each hidden layer has 8 to 64 neurons and a dropout layer is present between them.



(a) Train and validation loss of the best MLP

(b) Retraining and test curves of the best MLP

Figure 7: Loss curves of the best MLP (full set of features classification)

4.8 Results

We tested and compared the classification models across **2 different sets of features** extracted from our original dataset.

The first one, referred as **full set of features**, contains the following features: *statuses_count, tweet_count, avg_text_len, entropy_minute, entropy_day, reply_count_avg, favorite_count_avg, num_hashtags_avg, num_urls_avg, num_mentions_avg, retweet_count_sum, favorite_count_sum* and the categorical attribute *lang*.

The second one, referred as **minimal set of feature**, has the same features, except for *lang* and the substitution of numerical tweet aggregated attributes (*reply_count_avg, favorite_count_avg, num_hashtags_avg, num_urls_avg, num_mentions_avg, retweet_count_sum, favorite_count_sum*) with *total_success_score*.

Both dataset were tested with the same models and the same hyperparameters were tuned.

4.8.1 Full set of features

Model	**Test Accuracy**	**Model Hyperparams**
KNN	0.82	K=10
Naive Bayes	0.76	Bernoulli Naive Bayes
Decision Tree	0.84	criterion: *entropy*, max_depth: 10, max_features: *None*, min_samples_leaf: 2, min_samples_split: 6
Random Forest	**0.85**	max_depth: *None*, max_features: *log2*, min_samples_leaf: 2, n_estimators: 128
SVM	0.84	C: 1, gamma: *scale*, kernel: *rbf*
MLP	**0.85**	[64, 24, 6] hidden layers neurons

Table 3: Classification test scores and best hyperparams for **full set** of features

As shown in Table 3, the best models on test for the full set of features are **Random Forest** and **MLP** with **0.85** accuracy score. Nonetheless, with much simpler models such as Decision Tree and even K-NN we can get comparable accuracy, 0.84 and 0.81 respectively. The worst model seems to be Naive Bayes, but it still reaches a decent accuracy of 0.76.

4.8.2 Minimal set of features

Model	**Test Accuracy**	**Model Hyperparams**
KNN	0.80	K=16
Naive Bayes	0.65	Bernoulli Naive Bayes
Decision Tree	0.84	criterion: *entropy*, max_depth: 5, max_features: *None*, min_samples_leaf: 2, min_samples_split: 4
Random Forest	**0.85**	max_depth: 15, max_features: *None*, min_samples_leaf: 4, n_estimators: 128
SVM	0.84	C: 100, gamma: *scale*, kernel: *rbf*
MLP	0.84	[48, 16] hidden layers neurons

Table 4: Classification test scores and best hyperparams for **minimal set** of features

As shown in Table 4, the best models on test for the minimal set of features is **Random Forest** with **0.85** accuracy score, although with much simpler models such as Decision Tree and even K-NN we get comparable accuracy, 0.84 and 0.80 respectively. The worst model seems to be Naive Bayes, failing to go over 0.65 of accuracy.

It is important to note that, in the process of selecting the best Naive Bayes among Gaussian and Bernoulli, we selected Bernoulli even if it had a slightly lower validation score (0.67 vs 0.70), it had a more balanced non-bot recall score (0.58 vs 0.48). That means that more than half of the samples predicted as non bot by the Gaussian model are bots instead.

4.8.3 Conclusions on classification

Comparing the test scores model by model between the two sets of features, we can see that results are almost equal. That means that the features we removed from the full set were not discriminating for classification or were well enough represented by the substitute *total_success_score* in the minimal set. Performance decrease with less features being minimal or absent, advocates for the use of the minimal set for bot classification in our dataset.

The only exception is the Naive Bayes, which recorded a decrease of over 0.10 in accuracy when using the minimal set.

Another interesting result to report is that in all of the models the recall for non-bot class (and precision for bot class) is lower (around 0.70), meaning that there are a significant number of bots classified as non-bot. At the same time precision for non-bot (and recall for bot) is high (around 0.95), meaning that very few non-bots are erroneously classified as bots.

5 Time Series Analysis

In this section we extracted the time series from the tweets of the 2019, where for each user who has published any tweets is 2019, there will be a time series of length 365 (the number of days in 2019) where each entry is a *Success score* defined as:

$$\text{Success Score} = \frac{\text{Acceptance Score}}{\text{Diffusion Score} + 0.1} \quad \text{where}$$

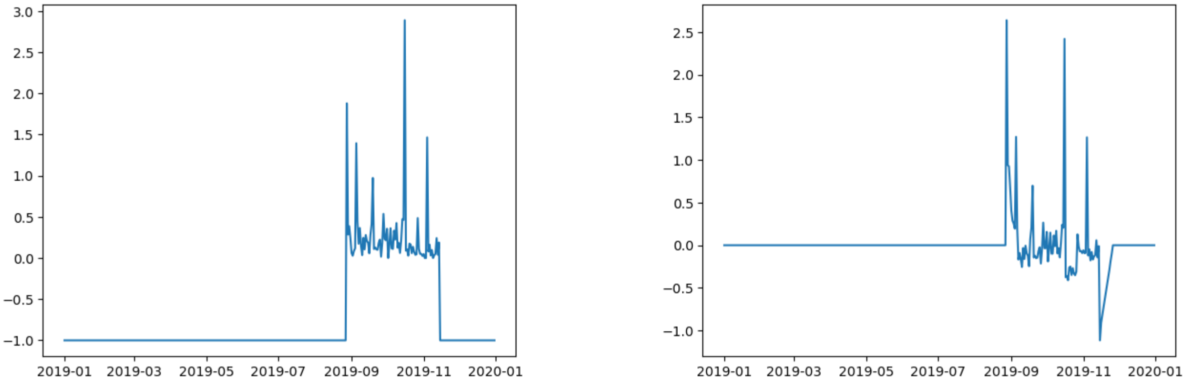
- Acceptance Score = retweet_count + reply_count + favorite_count
- Diffusion Score = num_hashtags + num_mentions + num_urls

If a user has not published any posts in a specific day, the entry will be -1 instead of *Success score*. We want to group similar users through the use of the created time series and extract shapelets, exploiting the binary variable bot.

5.1 Preprocessing

5.1.1 Detect and trend removal

We used the Augmented Dickey-Fuller, a statistical test to get the non-stationary time series, where if the p-value is higher than *0.05*, then the time series is non-stationary.



(a) A time series with a trend

(b) The same time series de-trended

Figure 8: Results of the detrending preprocessing

To correct it, we subtract a moving average of the series from the original series. An example is shown in Figure 8.

5.1.2 Noise removal

To find the best sliding window for the noise we used the mean of the **sums of absolute differences (SAD)** between the original timeseries and the smoothed one, then we resorted to the Knee method to choose the best window that ”doesn’t smooth too much” the original timeseries, thus selecting the smallest window size value after the Knee.

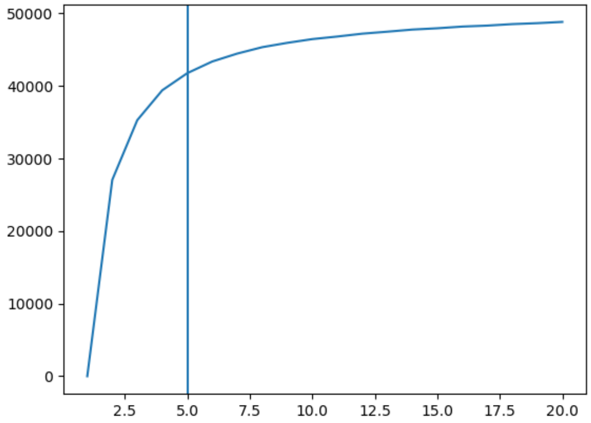


Figure 9: Knee method applied to the sums of absolute differences.

Using the Knee method we chose *5* as sliding window, as it can be seen in Figure 9. In Figure 10, an example of a denoised timeseries is shown.

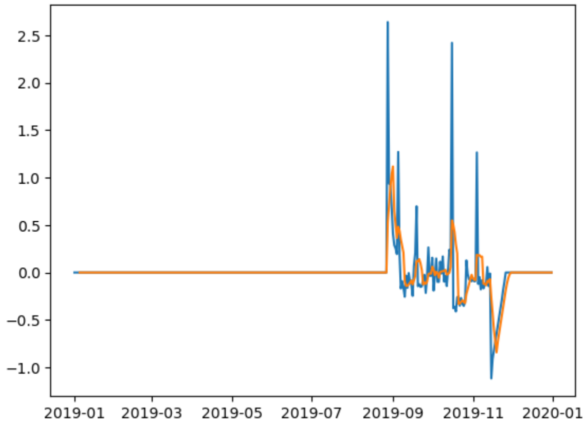
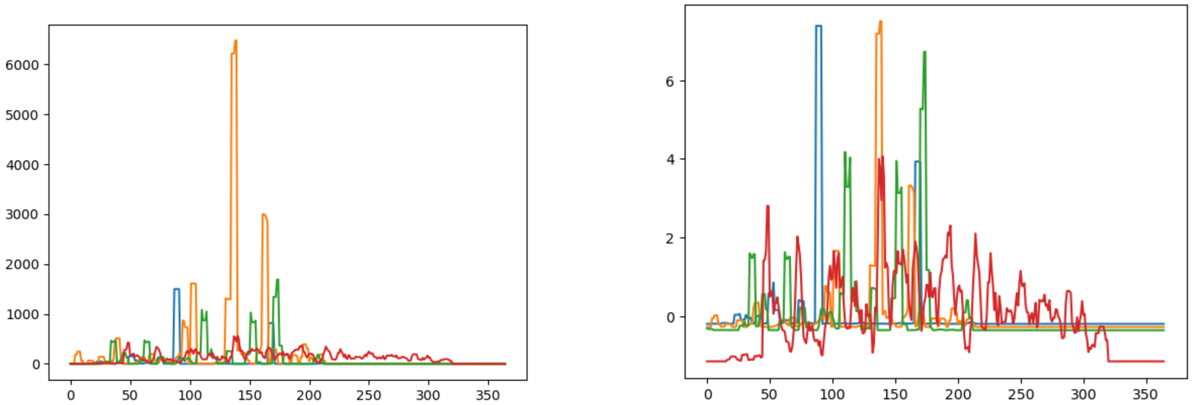


Figure 10: Denoise results with the sliding window = 5. In blue the original timeseries, in orange the denoised timeseries.

5.1.3 Scaling

As for scaling the timeseries we resorted to the Mean-Variance scaler (Figure 11)



(a) Not normalized time series

(b) Normalized time series

Figure 11: Results of the Mean-Variance scaler

5.2 Clustering

We clusterized the timeseries using 3 different methods: *Shape based*, *Feature based* and *Compression based*.

5.2.1 Shape based

In this clustering we compared the "shape" of the timeseries and the metric we used is the Euclidean one, the Dynamic Time Warping was too computationally expensive to apply for all the different K as it required too much time to be applied, so we used it with the resulting K obtained by the Euclidean metric.

Choosing K

To choose K we used the elbow method (only for the euclidean metric), the SSE graph is shown in Figure 12.

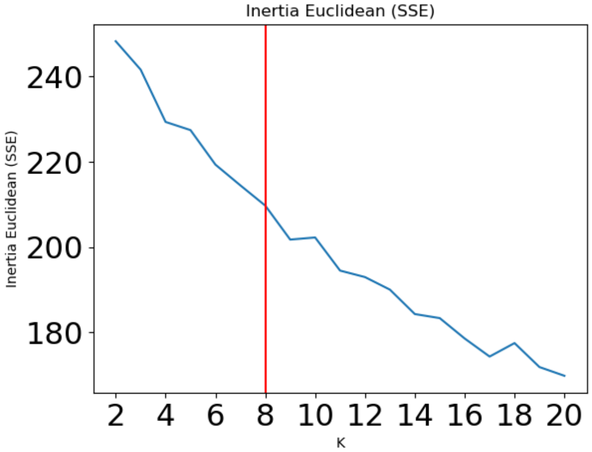
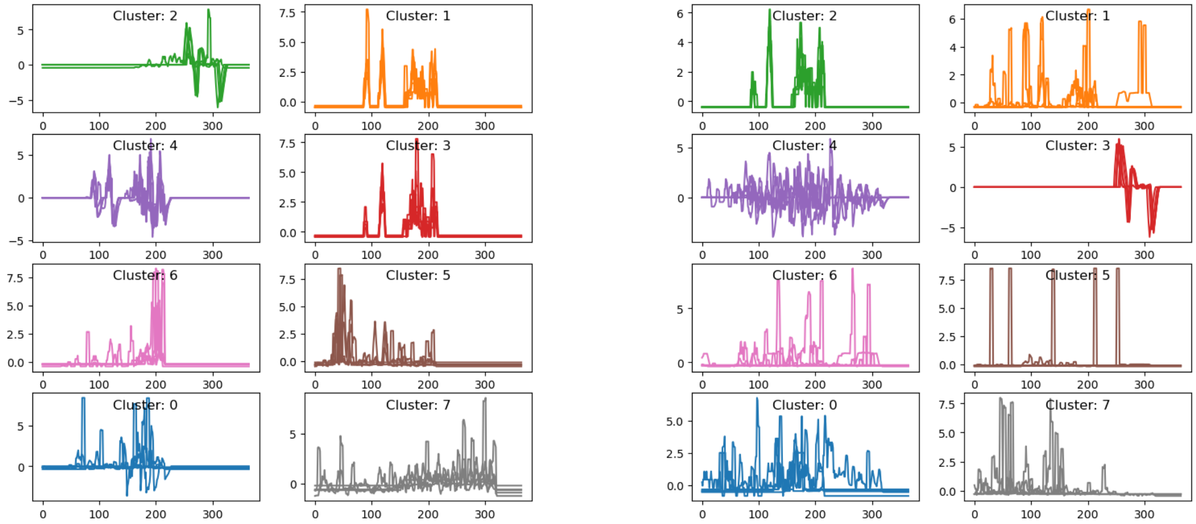


Figure 12: SSE Graph of the shape based clustering using the euclidean metrics

Although there is not any clear curve to apply the elbow method, *K=8* was the best as it had a balanced number of clusters in terms of SSE and cluster sizes.

Results



(a) Euclidean metrics (K=8)

(b) Distance Time Warping metrics (K=8)

Figure 13: Results of the shape based clustering

As for the cluster size and distribution, the DTW wins (The clusters are ordered from 0 to 7, so we can also compare using the Figure 13):

Euclidean clusters : [307, 333, 2559, 270, 366, 196, 1847, 909]

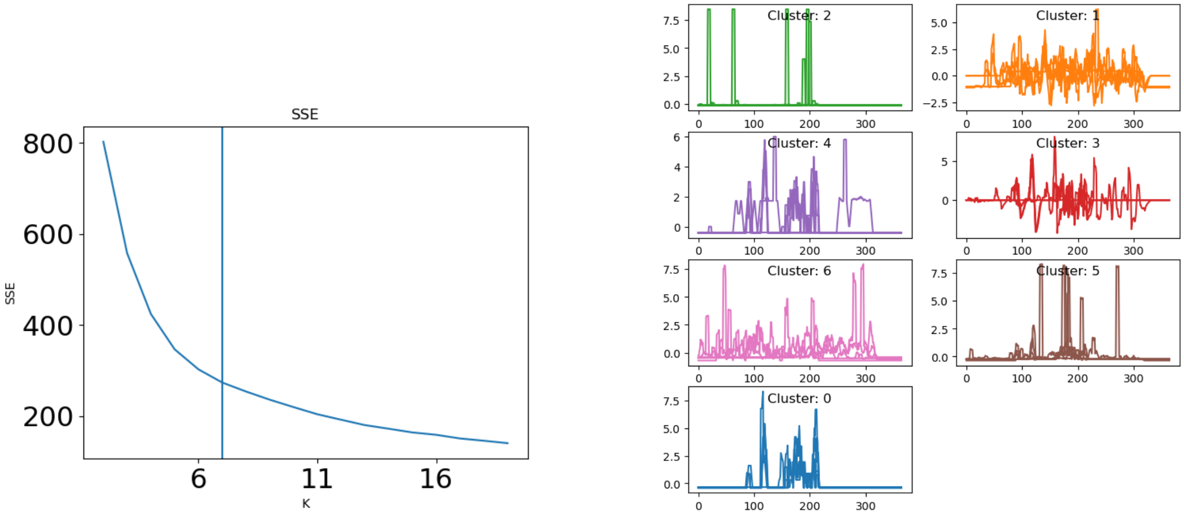
DTW clusters : [2246, 1294, 336, 566, 892, 531, 284, 638]

The euclidean metrics gave us the most "coherent" results, the nature of the metrics does not take into the account the evolution at different speeds of the series.
In both clusterings we can see some resemblances like: *Euclidean-Cluster-1* and *Euclidean-Cluster-3* is almost identical *DTW-Cluster-2*, and *Euclidean-Cluster-2* is almost identical *DTW-Cluster-3*. The other clusters share some similarities between the two metrics, but are not so strikingly similar as the ones stated before.

5.2.2 Feature based

In this clustering we defined a set of features to extract from the timeseries to then perform the standard clustering using tabular data. We extracted from each time series: *Average, Standard deviation, Variance, Median, 10th percentile, 25th percentile, 50th percentile, 75th percentile, 90th percentile, interquartile range and the coefficient of variation*.

K-Means



(a) SSE graph

(b) K-Means results

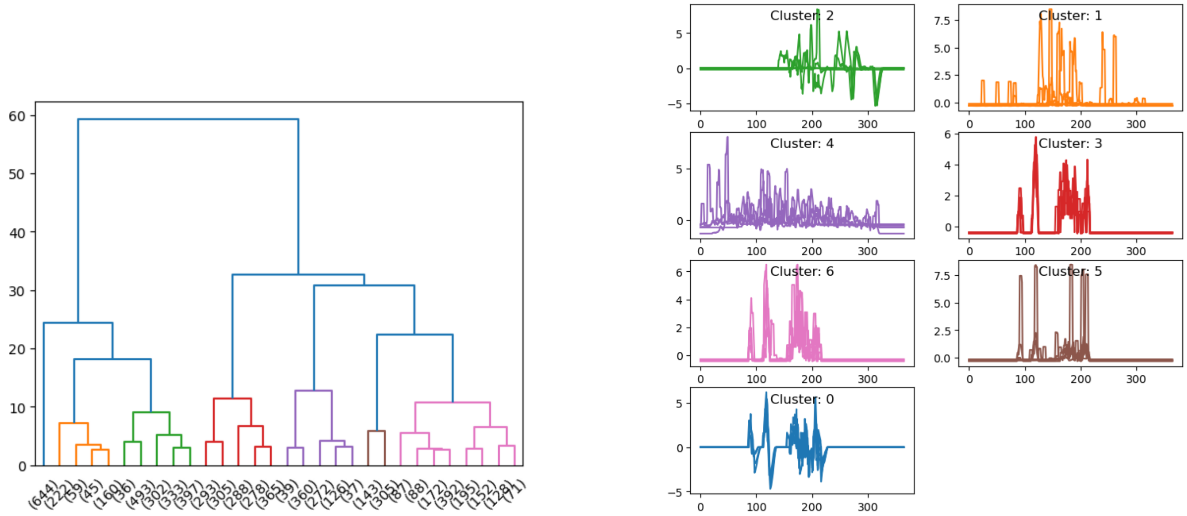
Figure 14: Results of the kmeans in the feature based clustering

In this case we choose K=*7* as we can deduct from the knee method (Figure 14a).

Feature based K-Means clustering seems to work well on the timeseries, clustering with good precision the ones with spikes in the same positions. It seems that it also separates in specific clusters the timeseries having negative spikes. We can also see that the resulting clusters are worse than the K-Means performed during the Shape based clustering, there is also one less cluster (Figure 14b).

Hierarchical

For the hierarchical we tried all the variation we already tried in the Task 2. The best clustering we found is the one using the Ward's method and the standard scaled values. We chose *14* as a threshold since it gave us the best clustering possible in term of size of the clusters and also the best distance in the dendogram.



(a) Dendogram of ward method with threshold = 14

(b) Clustering results of ward method with threshold = 14

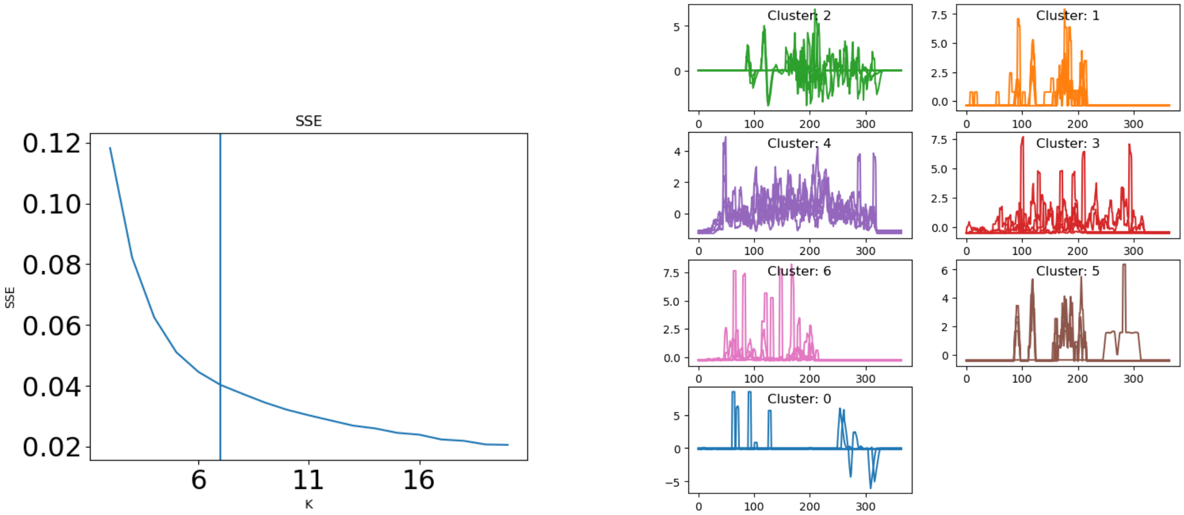
Figure 15: Dendogram and resulting clusters of the hierarchical clustering using ward method and the Z-score scaled timeseries

We can see that this clustering gave us 3 clusters with negative valued time series (*2*, *4* and *0*), and 4 positive valued time series, although cluster *3* and cluster *6* should be considered as one single cluster for being almost identical in the distribution.

5.2.3 Compression based

In this clustering we compress the time series using the Piecewise Aggregate Approximation (PAA) which approximate the original time series using piecewise line.

We used *27* as the number of segments which compress the timeseries, because in preliminary trials we found that it provided a good and representative compression. Intuitively, we approximate the series going from a measurement day by day to a measurement of two weeks by two weeks.



(a) SSE graph

(b) K-Means results

Figure 16: Results of the K-Means in the compression based clustering

The results of this clustering (Figure 16b) is similar to the one obtained with shape clustering, but some clusters (such as *4* and *3*) seems to have less recognizable common patterns for the timeseries. The cluster *5* cluster *1* resembles together one of the most recognized clusters in all types of clustering which have always around 2400 time series, although in this clustering the depiction seems a little bit chaotic.

5.2.4 Final clustering selected

Of the methods used for clustering, the one which, subjected to further analysis, produced the best results was the Shape-Based K-Means with Euclidean metric.

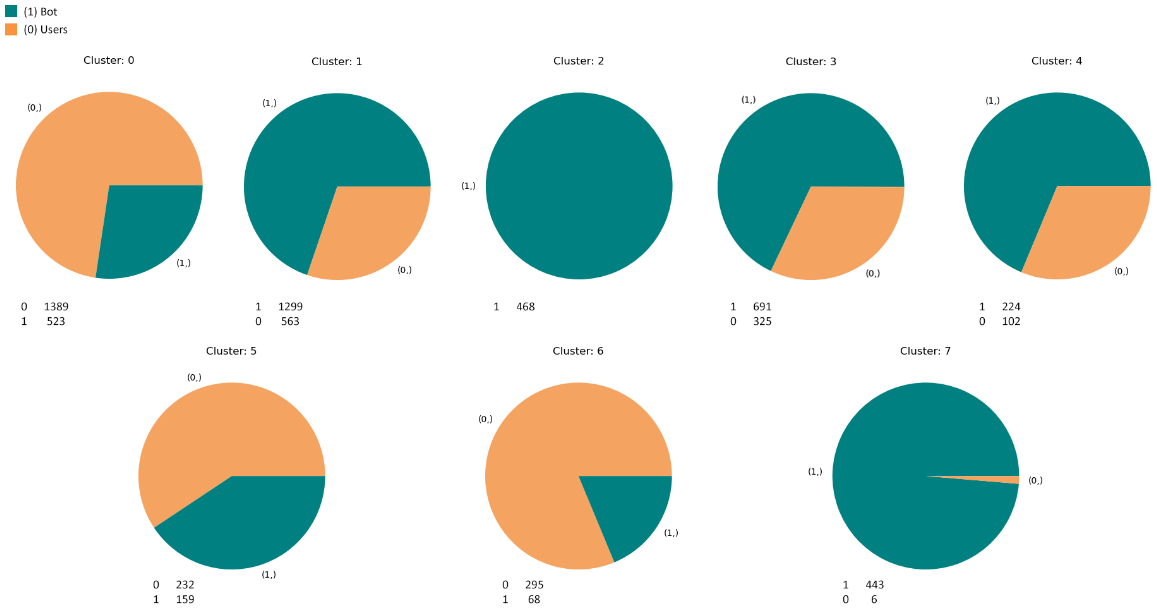


Figure 17: Distribution of bots in the clusters of final clustering.

As for the clustering composition, shown in the Figure 17, we can see that is well executed from the fact that there are not all cluster have a 50% split between bot and non-bot, the only notable exception is cluster *5* that is composed in total of 391 user which has still a majority of non-bot users. We can also see that the cluster *2* has generated a cluster composed only by bots, and in the time series depiction of the cluster *2*, the pattern very specific, the same cannot be said of cluster *7*, that, although has generated an almost perfect cluster of bot, there are not any recognizable patterns in its depiction. As for the cluster *1* and *3*, they are almost identical, in their depiction, but probably the euclidean metrics has not picked up their similarities, but it must be noted that their ratio of bot/non-bot is identical (around 68% of bots).

5.3 Shapelets

In order to identify shapes within the time series that distinguish them according to their classes, we extracted shapelets.

The first operation performed was to divide all time series into two classes, bots and non-bots respectively, and display them (Figure 18).

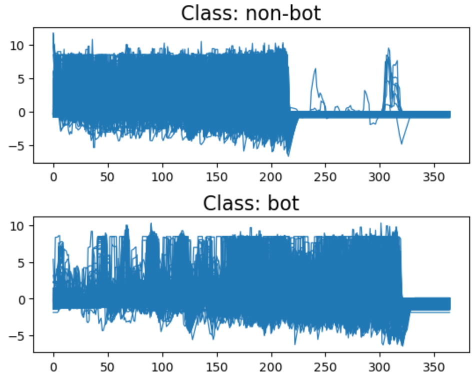


Figure 18: Plot timeseries separated by bot or non-bot label.

Subsequently, the number and size of shapelets within the time series were found using Grabocka heuristics. After an adjustment of the parameters, we selected shaplets of size 21. Grabocka returned 6 shapelets for this size.

Given this information, it was possible to train a Shapelet model consisting of a time series transformation and a logistic regression layer on top. Shapelet coefficients and logistic regression weights are optimised by gradient descent on on a L2-penalized cross-entropy loss, in our case with a maximum of 10 training epochs and a regularisation of 0.005.

We split the timeseries dataset with a stratifeid train-test split of 75-25. At the end of the testing phase, an accuracy of about 72% was achieved and a set of extracted shapelets was obtained.

Finally, using this model, we are able to analyse the time series and observe the number and position of shapelets found within each one, such as in Figure 19.

All of the extracted shapelets can be seen in Figure 20.

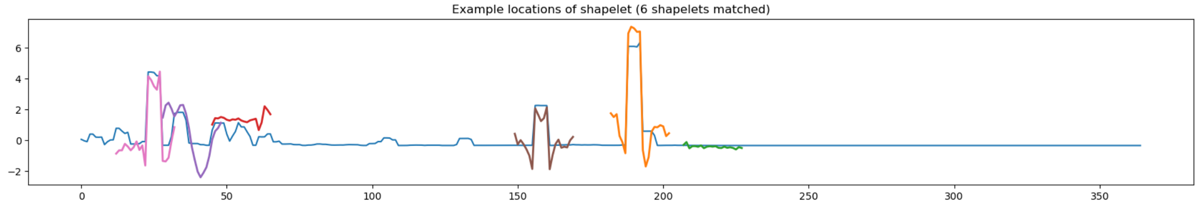


Figure 19: Example of a test set timeseries with matching shapelets.

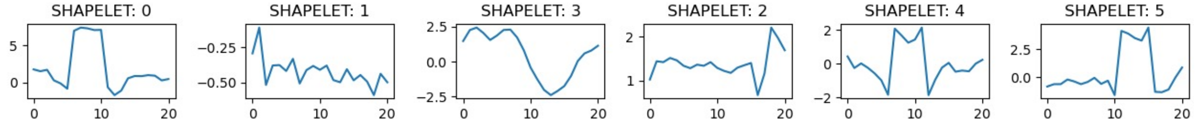


Figure 20: The 6 extracted shapelets.

References

[1] Wikipedia. List of ISO 639-1 codes — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=List%20of%20ISO%20639-1%20codes&oldid=1128323180`, 2023. [Online; accessed 07-January-2023].

[2] Wikipedia. List of most-liked tweets — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=List%20of%20most-liked%20tweets&oldid=1132172172`, 2023. [Online; accessed 07-January-2023].

[3] Wikipedia. List of most-retweeted tweets — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=List%20of%20most-retweeted%20tweets&oldid=1130592255`, 2023. [Online; accessed 07-January-2023].