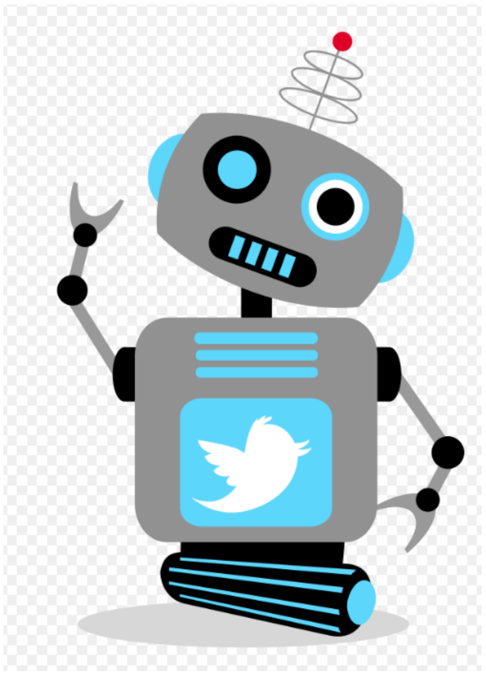
Feature extraction and ML for bot and activity classification of Twitter users

Hands-on tutorial on feature engineering for the supervised classification of bots in Twitter and for the unsupervised clustering of users based on their activity



This tutorial focuses on the feature extraction and engineering of features based on Twitter in order to classify twitter accounts to bot-human and cluster them based on their activity.

More specifically in this tutorial, we shall discuss :

A. Getting access to Twitter API using Python and Tweepy
B. Feature extraction based on categories
C. Evaluation of the time needed to extract these features
D. Supervised Machine Learning Model for binary bot-human classification
E. Unsupervised Machine Learning Model for grouping users in correspondence to their activity
F. Evaluation of models
G. Conclusions

Getting access to Twitter API using Python and Tweepy

In order to use Tweepy we should create a Twitter developer account following the next steps:

1. Create a standard account on Twitter and verify it
2. Go to https://developer.twitter.com/en
3. Sign up with your Twitter account and accept the terms and conditions
4. Verify the developer account from the mail your received

As soon as it is verified, you should get a set of keys: api_key, api_secret_key, token and token_secret.

In order to use them we are going to place them in a json file api_key.json:

```
{
      "api_key": "your_api_key",
      "api_secret_key": "your_api_secret_key",
      "token": "your_api_token",
      "token_secret": "your_token_secret"
}
```

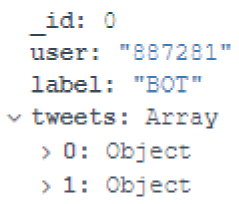
To continue, in our python script we are going to retrieve the keys from the json file and use them with tweepy to retrieve the data we need.

```
import tweepy, json
   try:
      keyFile = open('api_key.json')
      key = json.load(keyFile)
      keyFile.close()
   except:
      raise "Create a JSON file 'api_key.json' with the Twitter API
keys, that contains the properties 'key', 'key_secret', 'token' and
'token_secret'."

   auth = tweepy.OAuthHandler(consumer_key=key['key'],
consumer_secret=key['key_secret'])
   auth.set_access_token(key['token'], key['token_secret'])
   api = tweepy.API(auth, wait_on_rate_limit=True)
```

Finally, we are retrieving the data we need for each user in the file users_dataset.csv with:

```
tweets =
api.user_timeline(user_id=str(user_id),count=200,tweet_mode="extended
",wait_on_rate_limit=True)
```

and place them in a mongodb database with the following structure:



```
  _id: 0
  user: "887281"
  label: "BOT"
v tweets: Array
  > 0: Object
  > 1: Object
```

Feature extraction based on categories

In this section, we will present the feature extraction methodology and the categories that we have used in order to train our ML models that we will see in the next sections. We need to mention that all the features that we have added are language agnostic, so they can be used in any language and not only in English.

The different features are categorized based on their relevance to the following categories :

A. Temporal
B. Content
C. Network - Social Neighborhood
D. Sentiment
E. User

Temporal Features

On the temporal features, we included information which is related to the timestamp of tweets and retweets (timestamp_ms) of the users.
Some illustrative examples for temporal features are:

- **Total_days** , which indicates the days between the first and the last tweet of a specific user
- **Total_hours**, which indicates the hours between the first and the last tweet of a specific user
- **Consecutive_days_of_activity**, which indicates the consecutive days that a user have posted at least one tweet (was active)
- **Consecutive_days_of_no_activity**, which indicates the consecutive days that a user didn’t post a tweet (was inactive)
- **Average_between_tweets**, which indicates the average difference between the tweets of a specific user,
- **Max_min_per_day**, which indicates statistical results of the number of tweets_per_day (including retweets)
- **Max_occurence_of_same_gap**, which indicates if there is a pattern in idle time and returns the maximum times a time frame that a user being idle exists.

Content Features

On the content features, we included information which is related to the content of the tweet text.
Some illustrative examples for content features are:

- **All_punctuation_marks,** which indicates the the amount of punctuation marks in the text of tweets of a specific user
- **Similarities**, which returns a list of statistical results taken from the list of similarities between the tweets,
- **Marks_per_tweet**, which indicates the amount of punctuation marks that are used in a tweet for a specific user
- **Marks_distribution**, which indicates the statistical results of the marks used per tweet
- **Tweet_retweet_ratio**, which indicates the ratio (number of tweets)/ (number of retweets) that the specific user tweeted,
- **Number _of_sources**, which indicates the number of devices the specific user used for his tweets and retweets,
- **Unique_mentions_rate**, which indicates how many many mentions the specific did per tweet
- **Average_marked_as_favorite**, which indicates the statistical results of the specific user's results being favorite,
- **Retweeted**, which indicates the statistical results of the specific user's results being retweeted,
- **Statistics_of_their_retweets**, which indicates the statistical results of the retweets from the tweets that our specific user retweeted

Network Features

On the network features, we included information which is related to the
Some illustrative examples for network features are:

- **Number_of_accounts_retweeted**, which indicates the number of the users our specific user chooses to retweet from

Sentiment Features

On the sentiment features, we included information which is related to the sentiment analysis of the text used in the tweets.
Some illustrative examples for sentiment features are:

- **Positive_sentiment_per_tweet**, which indicates the positive polarity score of the tweets of a specific user,
- **Negative_sentiment_per_tweet**, which indicates the negative polarity score of the tweets of a specific user,
- **Neutral_sentiment_per_tweet**, which indicates the statistical results of the tweets texts that have a neutral sentiment analysis,
- **Emojis_per_tweet**, which indicates the amount of emojis which are used in a tweet by a specific user,

- **Tweet_emoji_ratio**, which indicates the ratio (number of emojis)/(number of characters in tweet),
- **Most _common_emoji**, which indicates which emoji is used the most and how many times

User Features

On the user features, we included information which is related to the User object on the Twitter data.
Some illustrative examples for user features are:

- **User_screen_name_length**, which indicates the amount of letters that the screen name of the specific user consists of.
- **User_followers_count**, which indicates the amount of followers that the specific user has.
- **User_friends_count**, which indicates the amount of friends that the specific user has.
- **User_is_verified**, which returns 1 if the user is verified and 0 otherwise,
- **User_description_length**, which indicates the amount of letters that the description of the specific user consists of,
- **Numbers_in _screen_name**, which indicates the number of numbers in the screen name of the specific user,
- **Numbers_in_name**, which indicates the number of numbers in the name of the specific user,
- **User_tweets_count**, which indicates the number of tweets the specific user has tweeted,
- **Has_bot_word_in_name**, which returns 1 if the word bot exists if the specific user's name and 0 otherwise,
- **Has_bot_word_in_screen_name**, which returns 1 if the word bot exists if the specific user's screen name and 0 otherwise,
- **Has_bot_word_in_description**, which returns 1 if the word bot exists if the specific user's description and 0 otherwise,
- **Days_since_creation**, which indicates the number of days since the the specific user created his account

Evaluation of the time needed to extract these features

We gathered the tweet timeline of almost **2400** users and calculated the features mentioned above. The total time needed to calculate all the features of all users was about **6 minutes** and the mean time to get the features of a single user was at **0.157 seconds**.

To calculate the mean time mentioned above, we gathered in a list the time needed to calculate all the features for each one of the 2400 users and got the mean value of that list.

Supervised Machine Learning Model for binary bot-human classification

For the supervised machine learning model for binary bot-human classification we have used a list of different classifiers in order to evaluate and compare our models.
More specifically, we have used the features that we have previously created and added it to our models.

We are going to quickly test the fit of 5 different models on our dataset. We have chosen to test:

1. Logistic Regression: basic linear classifier (good to baseline)
2. Random Forest: ensemble bagging classifier
3. K-Nearest Neighbors: instance based classifier
4. Support Vector Machines: maximum margin classifier
5. Gaussian Naive Bayes: probabilistic classifier

To run the initial experiments, we used the default parameters for each model. To get a more accurate representation of the fit for each of these models, the default parameters would need to be adjusted; however, for the purposes of this tutorial, the general idea is made more clear by not tweaking each model.

```
Y = df.label.copy()
X = df.copy()
X = X.drop(columns=['label'])
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

dfs = []
models = [
          ('LogReg', LogisticRegression()),
          ('RF', RandomForestClassifier()),
          ('KNN', KNeighborsClassifier()),
          ('SVM', SVC()),
          ('GNB', GaussianNB())
        ]
results = []
names = []
scoring = ['accuracy', 'precision_weighted', 'recall_weighted',
'f1_weighted', 'roc_auc']
    target_names = ['malignant', 'benign']
```

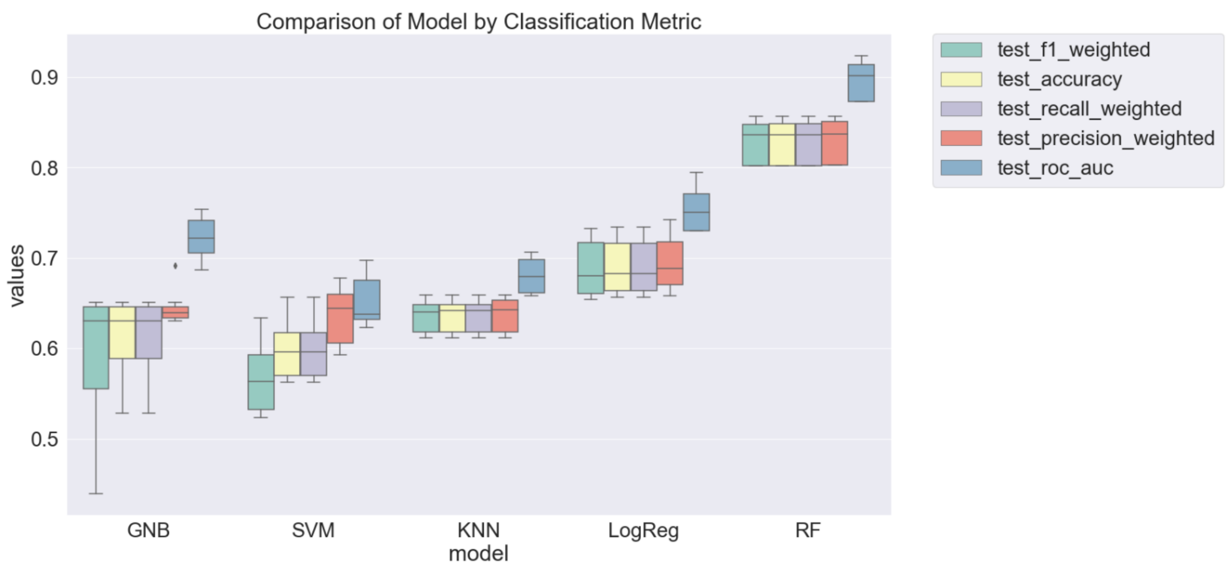
```
for name, model in models:
        kfold = model_selection.KFold(n_splits=5, shuffle=True,
random_state=90210)
        cv_results = model_selection.cross_validate(model, X_train,
y_train, cv=kfold, scoring=scoring)
        clf = model.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print(name)
        print(classification_report(y_test, y_pred,
target_names=target_names))
results.append(cv_results)
        names.append(name)
this_df = pd.DataFrame(cv_results)
        this_df['model'] = name
        dfs.append(this_df)
final = pd.concat(dfs, ignore_index=True)
```

Firstly, we create a variable “dfs” to hold all of the datasets that will be created from the application of 5-fold cross validation on the training set.
Next, “models” in a list of tuples holding the name and class for each classifier to be tested. After this, we loop through this list and run 5-fold cross validation. The results of each run are recorded in a pandas dataframe which we append to the “dfs” list.

To further aid in evaluation, a classification report on the test set is printed to screen. Finally, we concatenate and return all of our results.

In the following figures, we see our results (classification report) of the different models that we have used and more specifically the Accuracy, Recall, F1-Score and support. Also, we display a plot in which, it is more clear the results and the comparison between the different algorithms and models.

```
LogReg
              precision    recall  f1-score   support

         BOT       0.67      0.71      0.69       262
       HUMAN       0.62      0.58      0.60       218

    accuracy                           0.65       480
   macro avg       0.65      0.64      0.64       480
weighted avg       0.65      0.65      0.65       480

RF
              precision    recall  f1-score   support

         BOT       0.83      0.84      0.83       262
       HUMAN       0.80      0.79      0.80       218

    accuracy                           0.82       480
   macro avg       0.82      0.81      0.81       480
weighted avg       0.82      0.82      0.82       480

KNN
              precision    recall  f1-score   support

         BOT       0.65      0.65      0.65       262
       HUMAN       0.58      0.58      0.58       218

    accuracy                           0.61       480
   macro avg       0.61      0.61      0.61       480
weighted avg       0.61      0.61      0.61       480


SVM
              precision    recall  f1-score   support

         BOT       0.57      0.80      0.67       262
       HUMAN       0.54      0.28      0.37       218

    accuracy                           0.56       480
   macro avg       0.56      0.54      0.52       480
weighted avg       0.56      0.56      0.53       480

GNB
              precision    recall  f1-score   support

         BOT       0.68      0.61      0.64       262
       HUMAN       0.58      0.65      0.61       218

    accuracy                           0.63       480
   macro avg       0.63      0.63      0.63       480
weighted avg       0.63      0.63      0.63       480
```



After plotting our performance metrics from the 5-fold cross validation, it is immediately clear that GNB, SVMs and KNNs fit our data rather poorly across all metrics and that the Random Forest tree models fit the data very well.

Unsupervised Machine Learning Model for grouping users in correspondence to their activity

For the unsupervised machine learning model for grouping users in correspondence to their activity, we have used different clustering algorithms like K-means, Hierarchical Clustering. Also in these models, we have used the features that we have previously created and added it to our models.

We are going to quickly test the fit of 2 different models on our dataset. We have chosen to test:

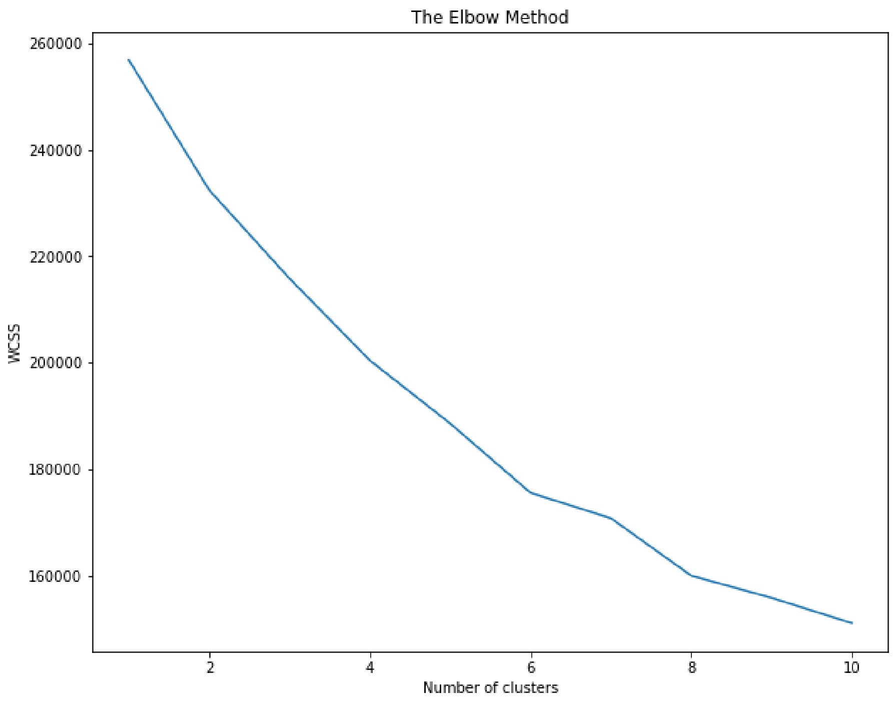
1. K-means Clustering
2. Hierarchical Clustering

K-means Clustering

First of all, standardizing the dataset is essential , as the K-means and Hierarchical clustering depend on calculating distances between the observations. Due to different scales of measurement of variables, some variables may have higher influence on the clustering output.

```
#standardize the data to normal distribution
import pandas as pd

from sklearn import preprocessing
df_std = preprocessing.scale(df)
df_std = pd.DataFrame(df_std)
```

In K-means, the number of clusters required has to be decided before the application, so some level of domain expertise would have helped. Else we can use a scree plot to decide the number of clusters based on reduction in variance.



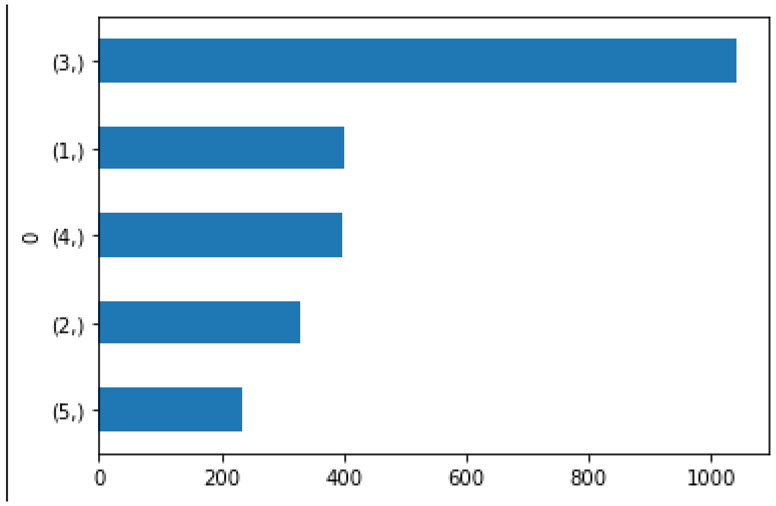
As of the results, a good amount of clusters that we can use for our models are 5.

Now that we have selected the amount of clusters and the data are in standard format, the data are ready to be clustered. The K-Means estimator class is where you can set the algorithm parameters before fitting the estimator to the data.
We have tried with several parameters and the final arguments are the following :

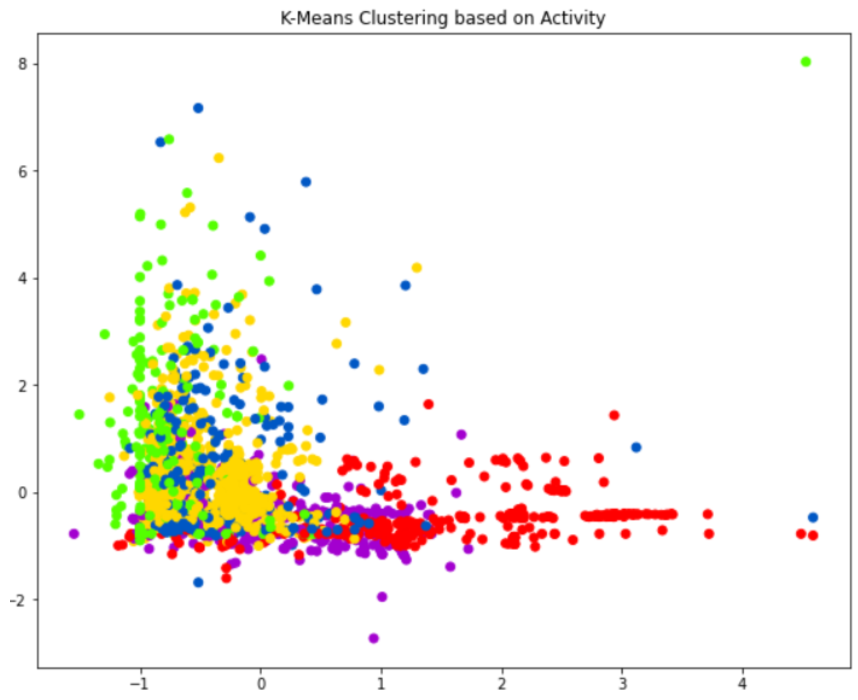
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300,
random_state = 42)
```

With the following script, we can see the cluster sizes for each of the clusters :

```
cluster.value_counts().sort_values().plot(kind = 'barh')
```



Finally, let's plot the clusters to see how actually our data has been clustered. Here’s what the plot looks like:

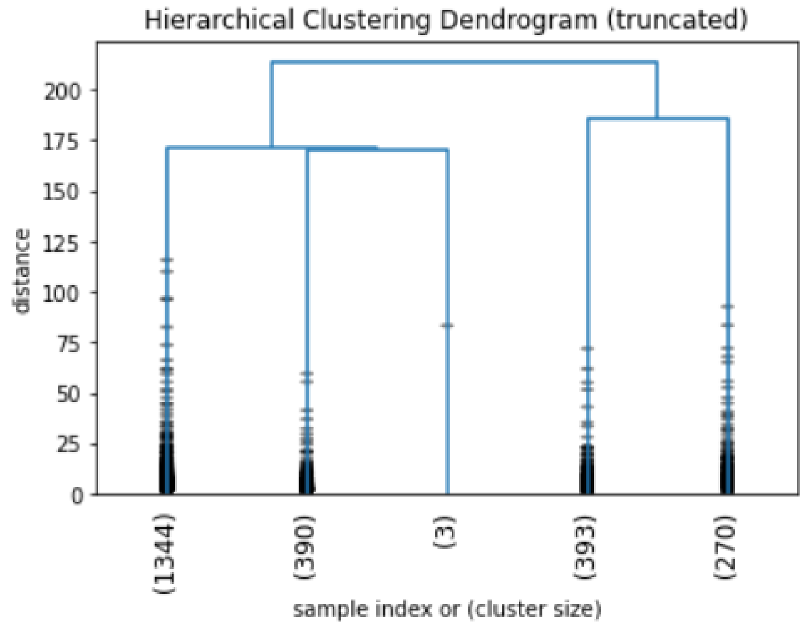


You can see the data points in the form of five clusters. The data points from :

- Cluster 0 are represented in red color (extremely active)
- Cluster 1 are represented in blue color (very active)
- Cluster 2 are represented in pink color (active)
- Cluster 3 are represented in yellow color (slightly active)
- Cluster 4 are represented in green color (inactive)

Hierarchical Clustering

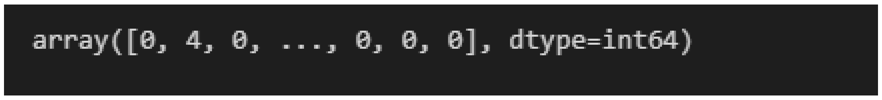
The sole concept of hierarchical clustering lies in just the construction and analysis of a dendrogram. A dendrogram is a tree-like structure that explains the relationship between all the data points in the system.
Like K-means clustering, hierarchical clustering also groups together the data points with similar characteristics. In the following figure, we can see the dendrogram - Hierarchical Clustering of User profiles based on the activities.



On the axis X we can see the cluster size for each different cluster.
Now we know the number of clusters for our dataset, the next step is to group the data points into these five clusters.
To do so we will again use the AgglomerativeClustering class of the sklearn.cluster library. Take a look at the following script :

```
from sklearn.cluster import AgglomerativeClustering

cluster2 = AgglomerativeClustering(n_clusters=5, affinity='euclidean',
linkage='ward')
cluster2.fit_predict(dataset2_standardized)
```

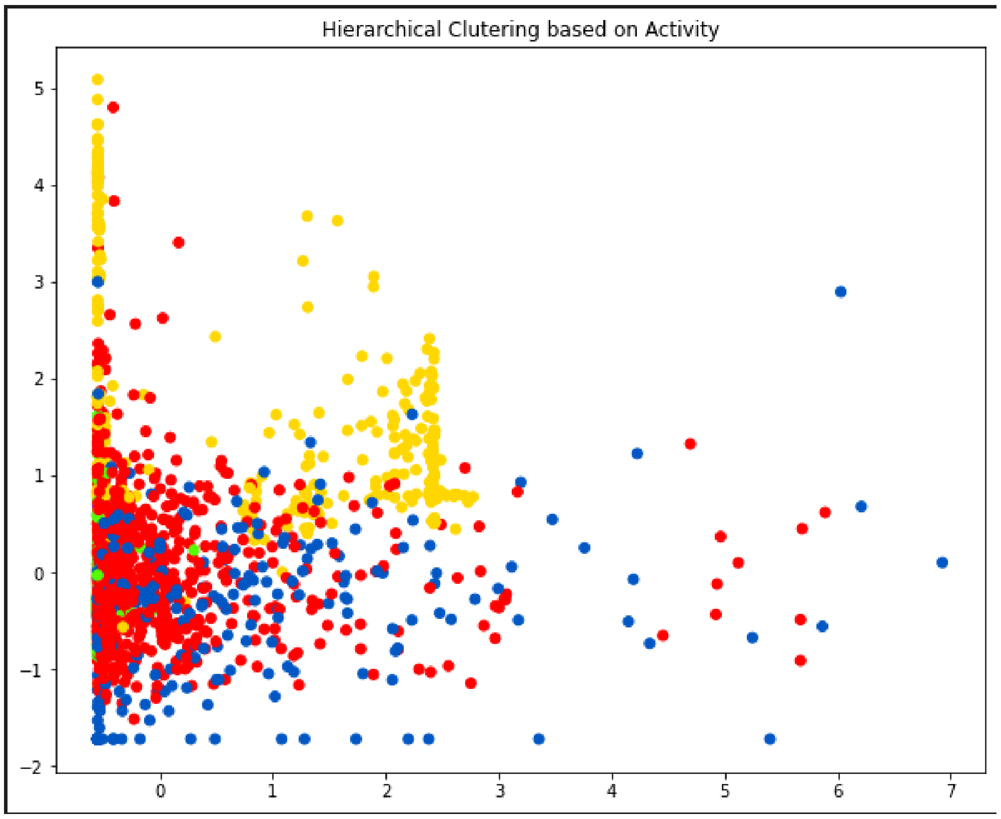
The output of the script above looks like this :



```
array([0, 4, 0, ..., 0, 0, 0], dtype=int64)
```

You can see the cluster labels from all of your data points. Since we had five clusters, we have five labels in the output i.e. 0 to 4

As a final step, let's plot the clusters to see how actually our data has been clustered:
Here’s what the plot looks like:



You can see the data points in the form of five clusters. The data points from :

- Cluster 0 are represented in red color (extremely active)
- Cluster 1 are represented in blue color (very active)
- Cluster 2 are represented in pink color (active)
- Cluster 3 are represented in yellow color (slightly active)
- Cluster 4 are represented in green color (inactive)

As a result most of the user profiles are clustered as extremely active, followed by a similar amount of data points for very active and slightly active and inactive. Finally, our clusters found only 3 data points considering the user profiles as active, so the medium category is not represented efficiently in comparison to the K-means clustering, which was the superior cluster.

Evaluation of models

Lasty, we tested our models in the following 12 real accounts:

1. @Aristoteleio: 234343780
2. @ylecun: 48008938
3. @nportokaloglou: 249130209
4. @greeceinfigures: 1348351605654106118

5. @big_ben_clock: 86391789
6. @adonisgeorgiadi: 70340615
7. @kostasvaxevanis: 289527193
8. @vanitysthasmin: 1005766052607938560
9. @akurkov: 62213337
10. @ve10ve_ghost: 1495855082734297095
11. @adarshburman2: 1976335622
12. @bassaces1: 1066275282653536256

and produced the following results:

User \ Model	RF	KNN	SVM	GNB
@Aristoteleio	BOT	BOT	BOT	BOT
@ylecun	BOT	HUMAN	BOT	HUMAN
@nportokaloglou	BOT	BOT	BOT	BOT
@greeceinfigures	BOT	BOT	BOT	BOT
@big_ben_clock	BOT	HUMAN	BOT	HUMAN
@adonisgeorgiadi	BOT	BOT	BOT	HUMAN
@kostasvaxevanis	BOT	HUMAN	BOT	HUMAN
@vanitysthasmin	BOT	BOT	BOT	BOT
@akurkov	HUMAN	BOT	BOT	BOT
@ve10ve_ghost	HUMAN	HUMAN	BOT	HUMAN
@adarshburman2	HUMAN	HUMAN	BOT	HUMAN
@bassaces1	HUMAN	HUMAN	BOT	HUMAN

Conclusions

We hope you enjoyed this tutorial!
You should now have a good understanding of how to get access to Twitter API using Python and Tweepy and extract tweets of users and store them in your database. Furthermore, we worked on how to extract and create features based on different categories of Twitter data focused on being language agnostic. Also, we have developed both a supervised ML model for binary bot-human classification comparing the results between different algorithms, as well as developing an unsupervised ML model for grouping users based on their activity level on

different clusters. Finally, we tested our models on real accounts in order to further test their performance levels.

The above features can be found here:
https://github.com/MnAppsNet/Twitter-Bot-Detection/tree/master/features

The jupyter notebooks used for the Machine Learning models and evaluation can be found here: https://github.com/MnAppsNet/Twitter-Bot-Detection

We hope you found this information useful and thanks for reading!

Authors

1. Chatziara Dimitra, 98*
2. Kalyvas Emmanouil, 93*
3. Grigoroudis Efstratios, 74*

** All authors contributed equally to this tutorial*