Cristòfol Daudén Esmel

TWITTER BOTS DETECTOR USING MACHINE LEARNING TECHNIQUES

FINAL DEGREE PROJECT

tutored by Dr. Jordi Duch Gavaldà

Degree in Computer Science
-Computer engineering-

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2019

Resum.

Cada cop es veuen més notícies i articles que fan referència a l'aparició de comptes controlats per bots en les diferents xarxes socials, de l'efecte que poden tenir aquests en la societat i diferents formes de detectar-los per poder reportar aquests comptes i que siguin eliminats.

En aquest projecte té per objectiu explotar diferents camps envers aquest problema. Primer la implementació d'un codi senzill que permeti la creació de models, utilitzant mètodes d'aprenentatge automàtic, capaços de detectar si un compte de Twitter és controlat per un bot o no. Un cop s'hagin obtingut models prou robustos, s'usaran en una eina que, donat el nom d'un usuari de Twitter, retorni la probabilitat que tenen ell i els seus amics de ser bots. El tercer objectiu consisteix a dissenyar una eina que ens permeti avaluar la interacció entre usuaris legítims i comptes controlats per bots en el fil d'una conversa d'un tema concret, aquest serà especificat donant una sèrie de hashtags. Finalment, com a part addicional s'integraran les dues funcionalitats anteriors en un servei web per facilitar-ne l'ús i donar un servei centralitzat i de fàcil accés a tota mena d'usuaris.

Els millors models s'han obtingut mitjançant l'algorisme d'aprenentatge automàtic Random Forest obtenint un valor de F1 i grau d'encert superiors a 0.94. Les eines implementades compleixen amb els requeriments especificats, tot i que l'eina Hashtag Analyzer té uns costos temporal i computacional molt elevats.

Finalment, en les interaccions de Retweet durant les Eleccions Municipals Espanyoles del 2019 s'ha detectat que un 6% dels usuaris són bots i aquests generen quasi un 25% de les interaccions. D'altra banda, en les interaccions de Favorit, un 7% dels usuaris són bots, generant un 9% del total de les interaccions.

Resumen.

Cada vez se ven más noticias y artículos que hacen referencia a la aparición de cuentas controladas por bots en las distintas redes sociales, del efecto que estos pueden tener sobre la sociedad y distintas formas de detectarlos para poder reportar estas y eliminarlas.

Este proyecto tiene por objetivo explotar diferentes campos para con este problema. Primero la implementación de un código sencillo que permita la creación de modelos, utilizando métodos de aprendizaje automático, capaces de detectar si una cuenta de Twitter esta controlada por un bot o no. Una vez se hayan obtenido modelos bastante robustos, se usarán en una herramienta que, dado el nombre de un usuario de Twitter, devuelva la probabilidad que tienen él y sus amigos de ser bots. El tercer objetivo consiste en diseñar una herramienta que nos permita evaluar la interacción entre humanos y cuentas controladas por bots en el hilo de una conversación de un tema concreto, este será especificado dando una serie de hashtags. Finalmente, como parte adicional se integrarán las dos funcionalidades anteriores en un servicio web para facilitar su uso y dar un servicio centralizado y de fácil acceso a todo tipo de usuarios.

Los mejores modelos se han obtenido mediante el algoritmo de aprendizaje automático Random Forest obteniendo un grado de acierto y valor de F1 superiores a 0.94. Las herramientas implementadas cumplen con los requerimientos especificados, aunque la herramienta Hashtag Analyzer tiene unos costes temporal y computacional muy elevados.

Finalmente, en las interacciones de Retweet durante las Elecciones Municipales Españolas del 2019 se ha detectado que un 6% de los usuarios son bots y estos generan casi un 25% de las interacciones. Por otra parte, en las interacciones de Favorito un 7% de los usuarios son bots, generando un 9% del total de las interacciones.

Abstract.

There is an increasing number of news and articles that refer to the emergence of bot accounts in different social networks, the effect they could have on the society and different ways of detecting them, in order to report these accounts and eliminate to be eliminated.

This project aims to exploit different fields in relation to this problem. The first goal, consists of the implementation of a simple code that allows the creation of models, using automatic learning methods, capable of detecting whether a Twitter account is controlled by a bot or not. Once robust models have been obtained, they will be used in a tool that, given the username of a Twitter account, returns his probability and all his friends probabilities of being a bot. The third objective consists of designing a tool that allows us to evaluate the interactions between genuine users and *Twitterbots* in the thread of a conversation on a specific topic. Finally, as an additional part, the two previous functionalities are integrated into a web service, to facilitate the implemented tools use and give a centralized and easy to use service to all kind of users.

The best models were obtained using the Random Forest machine learning algorithm obtaining an accuracy rate and F1 score values greater than 0.94. The implemented tools fulfil with the specified requirements, although the Hashtag Analyzer tool requires a substantial amount of time and computing power.

Finally, in the Retweet interactions during the 2019 Spanish Municipal Elections, 6% of the users were found to be bots, generating almost 25% of the interactions. On the other hand, in Favourite Interactions, 7% of users were bots, generating 9% of total interactions.

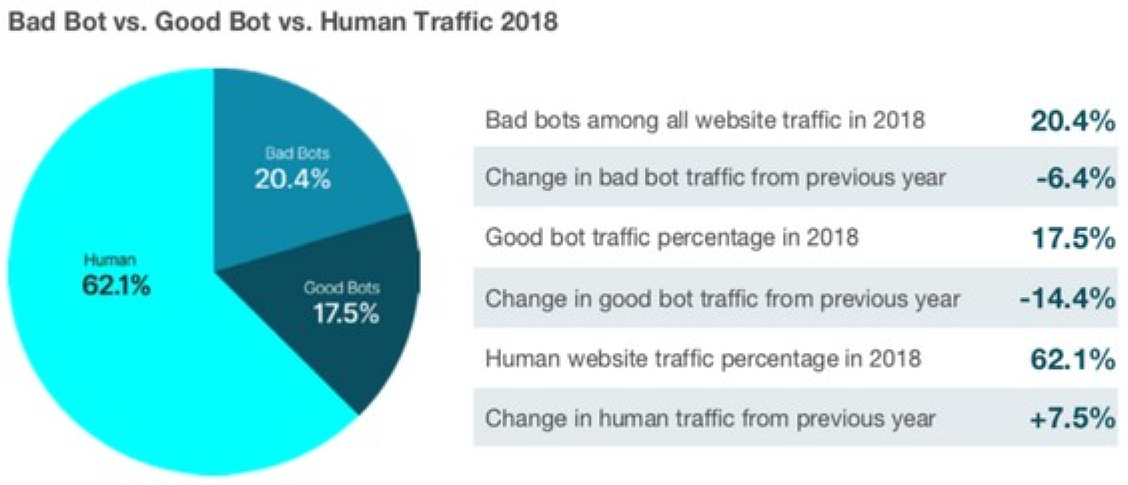
Index

Figure Index

Tables Index

1. Introduction

Nowadays, Social Media are the most powerful tools for connecting people all around the world. These connections allow information dissemination, which affects the ideas, news, and opinions to which we are exposed. So, there exist entities that exploits these online social networks with the aim of boost their popularity or to influence public opinion. It is not difficult to automatically target particular user groups and promote specific content or viewpoints. Is in this context when appears the concept of Social Bot [1]. Although year-over-year internet traffic decreases in both bad bot (-6,4%) and good bot (-14,4%), the 37,9% of all internet traffic still isn't human [2] (*Figure 1*).



Bad bots among all website traffic in 2018	20.4%
Change in bad bot traffic from previous year	-6.4%
Good bot traffic percentage in 2018	17.5%
Change in good bot traffic from previous year	-14.4%
Human website traffic percentage in 2018	62.1%
Change in human traffic from previous year	+7.5%

Figure 1: Internet traffic on 2018. Source: GlobalDots

A Social Bot, also known as a *Sybil account*, is a computer algorithm that automatically produces content and interacts with humans on social media. Many social bots perform useful functions, such as dissemination of news and publications [3] and coordination of volunteer activities [4]. However, the record of malicious applications of social bots is increasing every day. Some of them emulate human behaviour to create the appearance of widespread support for a political candidate or opinion [5], promote terrorist propaganda and recruitment [6, 7], manipulate the stock market [8], and disseminate rumours and conspiracy theories [9]. When working together in large clusters, bots have the ability to push narratives that could be false and misleading.

In this project, I have focused in the Twitter Social Media platform. In this context we introduce the concept of *Twitterbot*. A *Twitterbot* is an Internet bot that operates from a Twitter account. Some of the tasks that can be automated from a Twitter bot are writing tweets, retweeting, favouring a tweet, following someone, … Twitter does not mind the use of Twitter bot accounts while they do not break the Terms of Service through actions such as tweeting automated messages that are spam or tweeting misleading links. By some estimates, nearly 48 million Twitter accounts are automated [1]. Despite many of these accounts are easy to detect, such as the 'fake followers bots', there are bots that mimic human behaviour, posting and spreading information in a more complex way what difficulties its detection. So, the ability to detect bot accounts on social media sites, as Twitter in this case, is important for a healthy information exchange ecosystem.

Studies show that in the previous month leading up to the 2016 U.S. Presidential Election, a fifth of all tweets related to the election came from bot accounts [10]. These bot accounts interacted with normal users by refracting the natural conversations of the issues and events surrounding the election trying to influence them to support a concrete candidate. This suggest that the presence of social media bots can indeed negatively affect democratic political discussion rather than improving it, which in turn can potentially alter public opinion and endanger the integrity of Presidential election.

In this frame, this final degree project has, on the one hand, the goal of creating a library that allows users to generate a model and use it to detect if a certain Twitter account is controlled by a human or by a bot. On the other hand, there is another goal which is focused on the generation of tool that will be useful in the analysis of how Twitter bots interact with humans in the discussion of a certain topic.

1.1. Related Work

There already exist some platforms to evaluate whether a Twitter account is controlled by human or machine, such as the *BotOrNot* service [11], which is publicly available via the website[1] or via Python[2]. First of all you have to login with your personal Twitter account, then, the use of the *BotOrNot* service starts with a client specifying a Twitter screen name. The website and API use Twitter's REST API[3] to obtain the account's recent history, including recent tweets from that account as well as mentions of that screen name. Once the request data is received from Twitter's API, the *BotOrNot* website forwards it to the server. Then, the server computes the bot-likelihood score using its own classification algorithm. The server also generates data for the plots that will be displayed on the website when the request is originated from there (*Figure 2*).

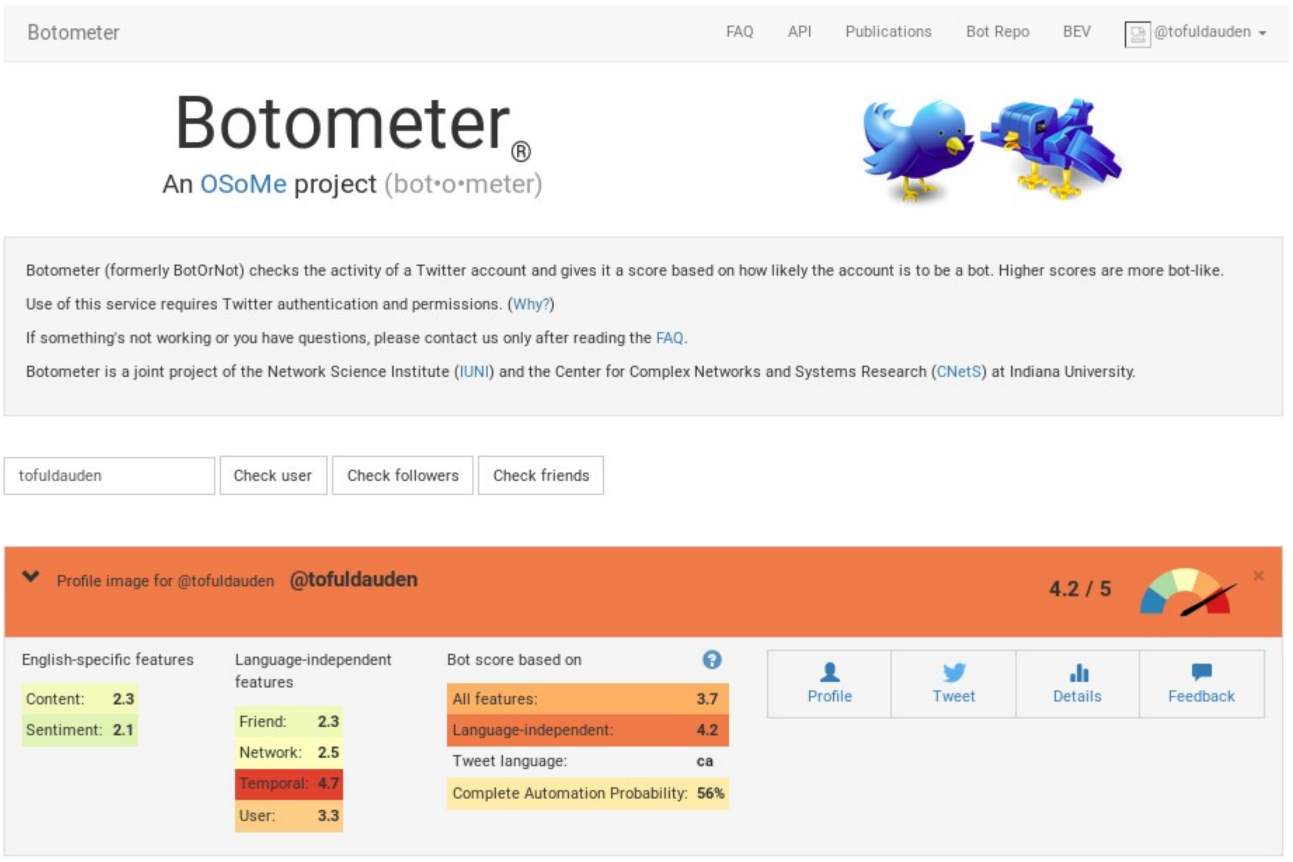
BotOrNot's classification system generates more than 1000 features using available meta-data and information extracted from interaction patterns and content. These features can be grouped into 6 main classes: Network, User, Friends, Temporal, Content and Sentiment. Once all data from a user is collected, *BotOrNot* uses Random Forest to calculate its bot-likelihood score. Extracted features are leveraged to fit seven different classifiers: one for each subclass of features and one for the overall score. This model has been previously trained with instances of both classes: social bot and human accounts.

Another approximation to solve the bot detection problem has been carried out by Efthimion, P.G. *et al.* [12]. In their work they take a special emphasis on the bot classification. Bots are categorized as "good" or "bad" based on the transparency with which they disclose their identity. Bad bots do not identify themselves to the web servers they access, while good bots declare and identify themselves.

To identify bots, they first analyse users data in order to get the variables shown in *Table 1* and put them in a binary matrix, these bot classification variables set three areas for analysis: profile, account activity and text mining. Then, logistic regression is applied on the data. This process outputs data into binary classifications which is required for support vector machine. Finally, support vector machining is used to test the bot detection model against different datasets of known Twitter bots.

1 truthy.indiana.edu/botornot
2 github.com/zafargilani/stcs/tree/master/botornot-python
3 dev.twitter.com/rest/public

The efficacy of the model is evaluated by the misclassification rate (a low misclassification rate means that the model is not misidentifying accounts owned by real people as bots) and the true positive rate (the rate that the algorithm correctly predicts that an account is a bot). When tested using all of the variables analysing the profile information, the model only misclassified 2.25% of the users and had a true positivity rating of of 98.98% with an accuracy value of 97.75%. A complete analysis using all variables analysing the profile and the text has a 100% accuracy, 0% of misclassification rate, and 100% of true positive rate. However, this analysis is done on a much smaller sample size, because text mining is computationally heavy and is most likely due to overfitting.



Botometer (formerly BotOrNot) checks the activity of a Twitter account and gives it a score based on how likely the account is to be a bot. Higher scores are more bot-like.

Use of this service requires Twitter authentication and permissions. (Why?)

If something's not working or you have questions, please contact us only after reading the FAQ.

Botometer is a joint project of the Network Science Institute (IUNI) and the Center for Complex Networks and Systems Research (CNetS) at Indiana University.

English-specific features

Content:	2.3
Sentiment:	2.1

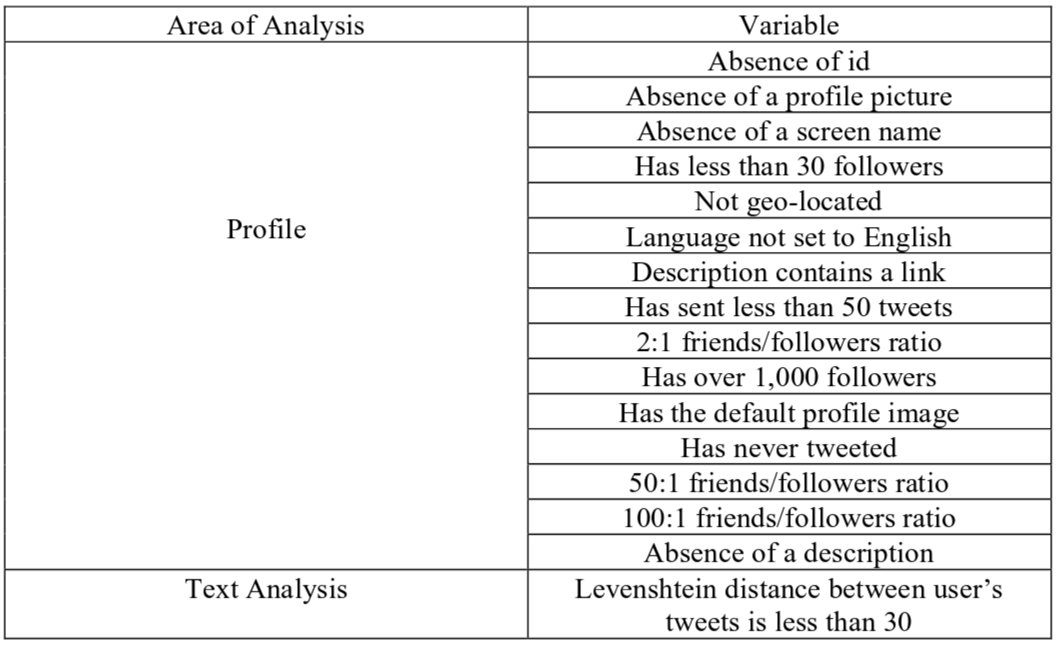
Language-independent features

Friend:	2.3
Network:	2.5
Temporal:	4.7
User:	3.3

Bot score based on

All features:	3.7
Language-independent:	4.2
Tweet language:	ca
Complete Automation Probability:	56%

Figure 2. Botometer analysis of user: “*tofuldauden*”



Area of Analysis	Variable
Profile	Absence of id
	Absence of a profile picture
	Absence of a screen name
	Has less than 30 followers
	Not geo-located
	Language not set to English
	Description contains a link
	Has sent less than 50 tweets
	2:1 friends/followers ratio
	Has over 1,000 followers
	Has the default profile image
	Has never tweeted
	50:1 friends/followers ratio
	100:1 friends/followers ratio
	Absence of a description
Text Analysis	Levenshtein distance between user's tweets is less than 30

Table 1. Bot Classification Variables By Area of Analysis

1.2. Goals

The first goal of the project is to generate a model that will be able to detect if a Twitter account is controlled by a human or by a bot, following the examples exposed in the previous section. For its generation, it is need to study and use the machine learning tools that already exist. Here I will analyse different models generated using different machine learning algorithms and choose the best one, in order to be used by the other tools of the project. For the training and validation of the different models are going to be used datasets collected by others.

Once an enough robust model is obtained, the second goal is to generate a tool that uses this model and the Twitter's REST API to get all information of a User and predicts its probability of being a bot.

The third main objective of the project consists of the generation of a tool that will help in the analysis of Twitter Users interactions around a certain topic. The execution result is a graph in which the nodes are all users that have interacted with the specified topic and the edges represents the interactions between the users. Furthermore, the node colour represents the probability of the user of being a bot, calculated with the model generated in the previous part, and its size is proportional to the number of interventions.

As a proof of concept, I will use the previous tool to study all interactions between Twitter users that took place during the previous week to the 2019 may Spanish Municipal Election. To achieve these goal it is necessary to give priority to the tweets collecting script, in order to let a computer capturing tweets while the other goals are carrying out.

Finally, the second and third main goals of the project, user and topic analysers, are going to be merged in a single web service that will allow a user to calculate the probability of a user of being a bot, and generate, in real time, a graph that represents all user interactions around a topic, including the users analysis. This web service will be a beta version of what an utility of this kind could be. It has implemented as a summary and presentation of all work done in the project.

The objectives can be subdivided in:

1. Study which of the different existing machine learning algorithms is the best one for the generation of a model that will allow us to detect a *Twitterbot*.
2. Analyse the Twitter data and specify the variables are going to be used by the model in order to process a user.
3. Implement the code that generates the desired model:
    a. Data preparation, pre-processing and feature extraction: loading, merging and shuffling the datasets, cleaning of data, etc.
    b. Model creation and validation: application of the different Machine Learning methods in order to generate different classification models and test its performance to select the best model for this problem.
4. Implement a tool that calculates a user's probability of being a bot, giving its username:
    a. Study Twitter's API REST: analyze all requirements needed to get information from Twitter using its API and study its methods in order to get all data that our classification model uses as an input.
    b. Write the code and test it.
5. Implement a tool that collects and analyses all Twitter Users interactions around a certain topic:
    a. Get all tweets published around a certain topic: Study the Twitter's Streaming API and implement a script that gathers, in real time, all tweets containing certain Hashtags specified as an input.
    b. Process these tweets: implement a script that from every tweet gets the user who has write it and the users that have interacted with it (favouring or retweeting the tweet). The output are two files with the users who have favour or retweeted each one of the tweets and two more files with the corresponding interactions between users.
    c. Process all users: calculate the probability of each user of being a bot.
    d. Generate the Interactions Graph: by using the processed users file and the interactions file, generate a graph in which the nodes are all users that have interacted with the specified topic and the edges represents the interactions between the users.
6. Use the Analyse Hashtag tool to study all interactions between Twitter users that took place during the previous week to the 2019 May Spanish Municipal Election.

7. Implement a web service that includes all two implemented functionalities:

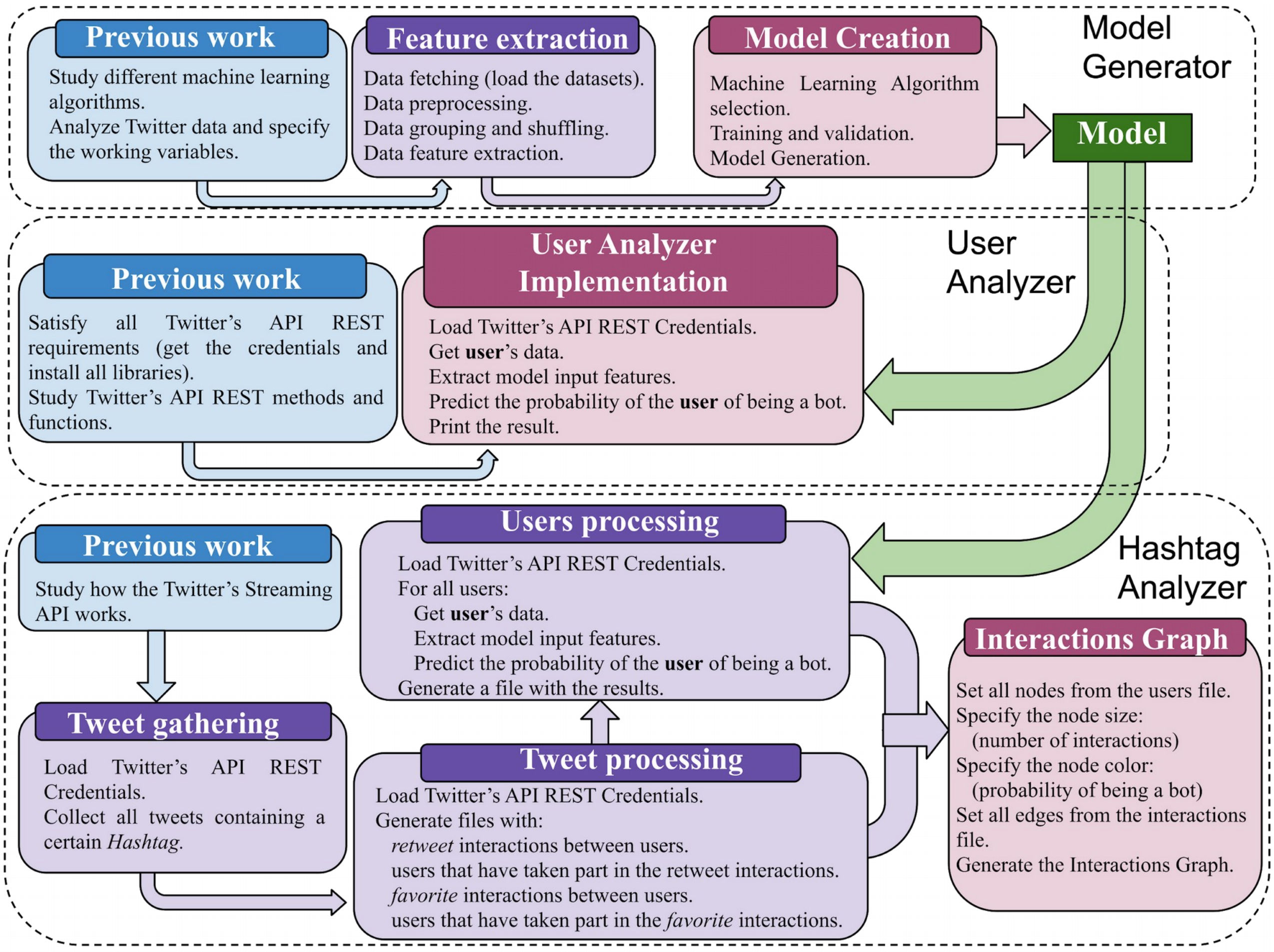
    a. Study and decide which is the best framework for the implementation of the web service: technologies, programming language, database, libraries, etc.

    b. Build the web service basic structure: define the different pages (home, user analyser and hashtag analyser pages), implement a basic version of the API rest, set the database, etc.

    c. Add the *Analyse User* functionality.

    d. Add the *Analyse Hashtag* functionality.

Figure 3 shows the three main goals of the project divided in the different parts of the different libraries and scripts that have to be designed, implemented and tested.



Previous work
Study different machine learning algorithms.
Analyze Twitter data and specify the working variables.
Feature extraction
Data fetching (load the datasets).
Data preprocessing.
Data grouping and shuffling.
Data feature extraction.
Model Creation
Machine Learning Algorithm selection.
Training and validation.
Model Generation.
Model Generator
Model
Previous work
Satisfy all Twitter's API REST requirements (get the credentials and install all libraries).
Study Twitter's API REST methods and functions.
User Analyzer Implementation
Load Twitter's API REST Credentials.
Get user's data.
Extract model input features.
Predict the probability of the user of being a bot.
Print the result.
User Analyzer
Previous work
Study how the Twitter's Streaming API works.
Tweet gathering
Load Twitter's API REST Credentials.
Collect all tweets containing a certain Hashtag.
Users processing
Load Twitter's API REST Credentials.
For all users:
Get user's data.
Extract model input features.
Predict the probability of the user of being a bot.
Generate a file with the results.
Tweet processing
Load Twitter's API REST Credentials.
Generate files with:
retweet interactions between users.
users that have taken part in the retweet interactions.
favorite interactions between users.
users that have taken part in the favorite interactions.
Hashtag Analyzer
Interactions Graph
Set all nodes from the users file.
Specify the node size:
(number of interactions)
Specify the node color:
(probability of being a bot)
Set all edges from the interactions file.
Generate the Interactions Graph.

Figure 3. Project main goals structure.

2. General Project Description

This project consists of:

- The generation of a, easy to understand, model able to identify a *Twitterbot.*
- The implementation of a tool that uses the model in order to calculate a Twitter user probability of being a bot, by indicating its username.
- The implementation of a tool that collects all tweets about a certain topic, gets all user interactions around these tweets, analyse these users and generate the interactions graph.
- The implementation of a web service that uses these previous tools in order to provide an easy way to use them.

Apart from the points shown above, the project also includes the corresponding documentation in which are explained all these objectives, the decisions taken during the project development, the project requirements, the libraries and web service designs, how the implementation has carried out, etc.

Finally, the project also has an investigation goal which consists of the use of Hashtag Analyser tool to study all interactions between Twitter users that took place during the previous week to the 2019 may Spanish Municipal Election.

As this Final Degree Project has been proposed by its current developer, with the support of its tutor, all requirements, functionalities and design have been decided by them through different meetings.

2.1. Functional Requirements

This project consists of four different libraries/tools/applications so, the functional requirements of each component are explained in different sections.

2.1.1. Data Analysis and Model Generator Functional Requirements

The Model Generator consists of a code that generates a classification model that calculates a probability of Twitter user of being a bot. This script needs to be filled with an existing dataset, which is loaded and preprocessed at the beginning of the execution. The model's input is a binary vector containing the features extracted and processed from the user's data. The script that generates the model should fulfil the following requirements:

R1.1. The script must be able to load data from different datasets.
R1.2. The script must fill data gaps in order to avoid posterior errors.
R1.3. The script must be able to identify bots from human users in the loaded data from the datasets in order to test and validate the generated model.
R1.4. The script must generate the feature vector from the dataset.
R1.5. The script should be flexible enough to accept new features in order to generate new models.
R1.6: The script should support multiple machine learning and statistics methods that allows the developer/user to compare the results and extract conclusions.
R1.7: The script must be able to test and validate the created models.
R1.8: The script must be able to save the created models.

2.1.2. User Analyser Functional Requirements

The User Analyser consists of a tool that outputs the probability for a Twitter account of being controlled by a bot. The account with which this tool has to work is specified as a parameter by indicating its *username*. It also has the functionality of analyse all friends of the indicated account. The tool should fulfil the following requirements:

R2.1. The tool has to check that the username indicated as a parameter belongs to a real Twitter account.
R2.2. The tool must be able to load the created models by the previous script.
R2.3. The tool should be able to let the user choose which model wants to use from the installed one.
R2.4. The tool should show the results by the terminal in an easy comprehension way.
R2.5. The tool must be able to let the user choose if he want to analyse all user friends.

2.1.3. Hashtag Analyser Functional Requirements

The Hashtag Analyser consists of a tool that generates an interaction graph of all users that have tweeted or favour a tweet about a certain topic. The first step consists of the data collection, on which a script collects, in real time, all tweets that contain a certain hashtag (this hashtag is specified by parameter and indicates the topic we are going to work with).

On the second step these tweets are processed. From each tweet it is get the user who has published it and all users who have retweeted and favourite it. This information is stored in four different files:

- **users_fav.csv**: contains all users who have published a tweet and those who have favourite that tweet. It also contains the number of publications and favourite tweets.
- **users_rt.csv**: contains the same information than the previous file but taking into account the users that have retweeted a tweed instead of those who have favourite it.
- **links_fav.csv**: contains the relations between who has published a tweet and those who have favourite it, storing the two users id and the number of interactions between them.
- **links_rt.csv**: contains the same information than the previous file but taking into account the users that have retweeted a tweed instead of those who have favourite it.

Once the previous files are generated, the users files are processed in order to generate another file which also contains the probabilities of each user of being a bot. Finally, the processed user file and its corresponding links file are used for the generation of the interactions graph. Merging the individual requirements of each step, the tool should fulfil the following requirements:

R3.1. The tool should be able to allow the user specify one or more hashtags to indicate the tweets capturing topic.
R3.2. The tool must store all tweets collected in a CSV file with the proper headers.
R3.3. The tool must be able to check if all the intermediate files used in by the different scripts exists and if they have the proper format.
R3.4. The tool must store all users and interactions in CSV files with the proper headers.
R3.5. The tool must be able to load an existing model for the users analysis.
R3.6. The tool should be able to plot the result Graph in 2D.

R3.7. The tool should set the nodes size of the Graph depending on the number of interactions of each user.

R3.8. The tool should set the nodes colour of the Graph depending on the *twitterbot* probability of each user.

2.1.4. Web Service Functional Requirements

The web service is a beta utility that allows a normal user to use the previous tools (User Analyser and Hashtag Analyser) in a simple and easy way. It will consist of three web pages: the home page, the User Analyser page and the Hashtag Analyser page. The first one contains a little summary of the project and links to the other ones. The others contain a form where the user has to input the data required by the tools and a submit button. When the submit button is pressed the API run the corresponding tool and forwards the result to the server. The website should fulfil the following requirements:

R4.1. The web service should have an easy navigation.

R4.2. The web service should be easy to use, nice and attractive for the users.

R4.3. The web service must check that cases in which the forms is not filled or it is filled wrongly.

R4.4. The web service must be error tolerant.

2.2. Non Functional Requirements

The non functional requirements have been defined by the project developer and its tutor, taking into account the current knowledges that the first one has and the knowledges he want to acquire during the project development.

R1. Full code is going to be written in python.

R2. Jupyter Notebook[4] is going to be used for the data analysis and the generation of the model.

R3. For the rest of the project, a it is going to be used a text editor or an IDE.

R4. The models generated are going to be stored in a special folder, under the 'models/' name, as arbitrary Python objects by using the Joblib[5] library.

R5. Pandas[6] and Numpy[7] libraries are going to be used for working with the datasets.

R6. The metrics and machine learning used must be from the ScikitLearn[8] library.

R7. To work with Twitter, the Tweepy[9] library is going to be used.

R8. All intermediate files generated by the different scripts and the datasets used are going to have a CSV format.

R9. For printing the results in 2D it is going to be used the matplotlib[10] library.

R10. For the web service implementation it is going to be used the django framework[11].

R11. The different web pages are going to be written by using html and CSS languages and the Bootstrap[12] library.

R12. All tools should be error proof and print an error message if something wrong happens during the execution.

4 https://jupyter.org/
5 https://joblib.readthedocs.io/en/latest/index.html
6 https://pandas.pydata.org/
7 https://www.numpy.org/
8 https://scikit-learn.org/stable/
9 https://www.tweepy.org/
10 https://matplotlib.org/api/pyplot_api.html
11 https://www.djangoproject.com/
12 https://getbootstrap.com/

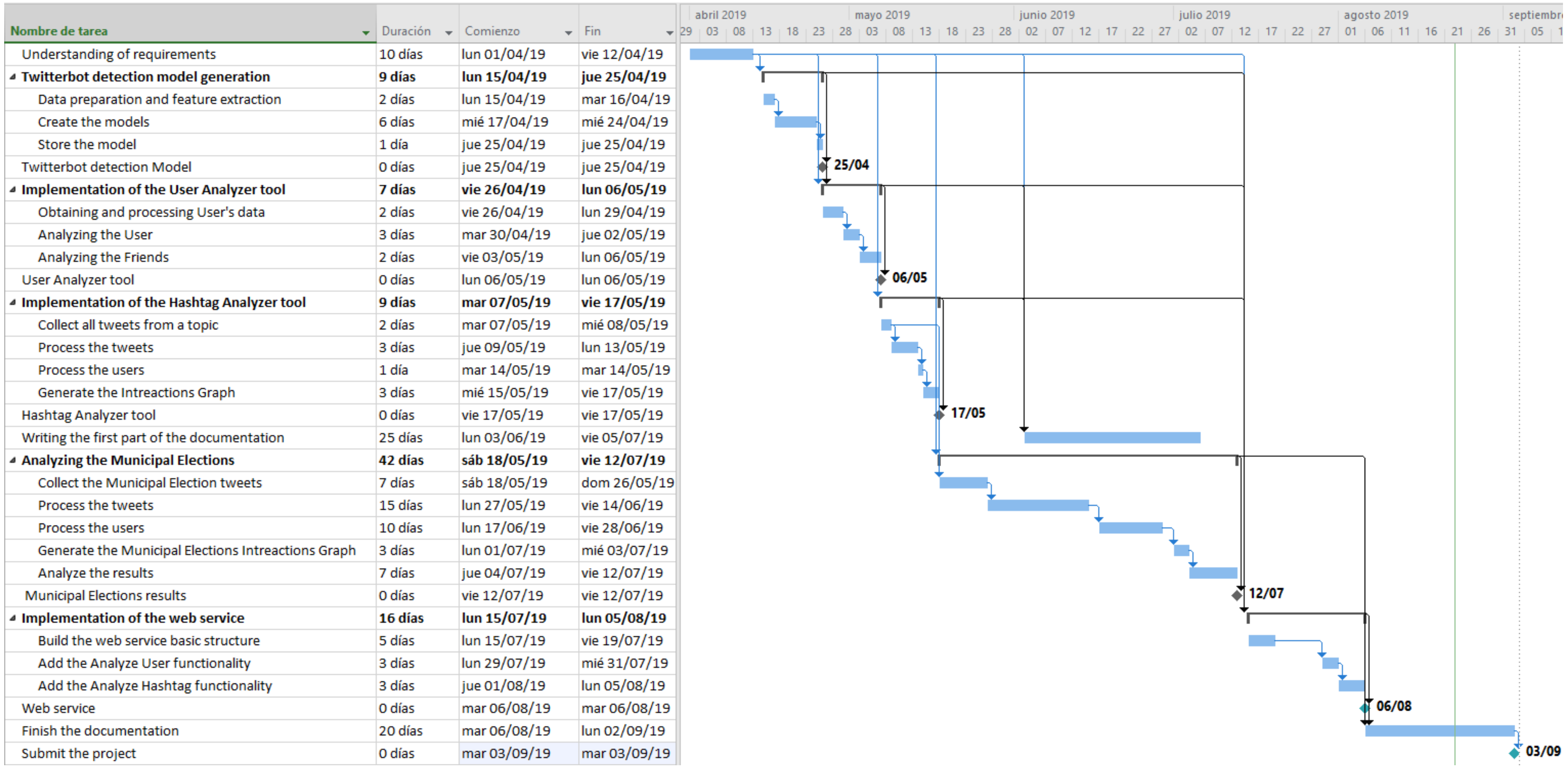
2.3. Project Planning

The project is constituted by the following activities:

1. Understanding of requirements: This activity consists of setting all the requirements and getting all the information that will be need for the project development. This includes: study different machine learning algorithms that will be used in the classification model generation; study the Twitter's API REST in order to know which methods provide, the credentials you need to use it and understand the data format it returns; decide which features are going to be used by the model to classify users; search from where can be obtained databases for the model training and validation; understand how the Django framework works, etc.
2. *Twitterbot* detection model generation: This activity consists of designing and coding a script that will analyse different bot detection models generated by using different machine learning algorithms. Once the selection of the best model is done, it is stored for being used by the other tools of the project.
    a. Data preparation and feature extraction: In this process the datasets are loaded and preprocessed: it is added the class to which each user belongs (human or bot) and then are merged into a single dataset. Next, a feature vector of each user is generated from the previous dataset, what is going to be used to train and validate the models.
    b. Create the models: Create different models by using different machine learning algorithms, validate them and test its performance to select the best one.
    c. Store the model: Generate and store the best model by training the machine learning algorithm, that had the best performance, using all the dataset.
3. Implementation of the User Analyser tool: This activity consists of designing, coding and testing a tool that has as an input parameter a Twitter account *username* and outputs its probability of being a bot.
    a. Obtaining and processing User's data: Develop a method that from the Twitter username gets all User information using the Twitter's API and generates the feature vector that uses the model as an input.
    b. Analysing the User: Write a method that loads the decision model and predicts the users probability of being a bot using the feature vector. Then, it prints the prediction decision in an easy way to understand for the user.
    c. Analysing the Friends: Add an option from which the user can also analyse all friends of the analysed user.
4. Implementation of the Hashtag Analyser tool: This activity consists of designing, coding and testing a tool that generates the graph of all Twitter users interactions around a topic.
    a. Collect all tweets from a topic: Implement a script that gathers, in real time, those tweets that contain certain hashtag/s from all tweets that are being published.
    b. Process the tweets: This activity consists of the implementation a script that from every tweet gets the user who has write it and the users that have interacted with it (favouring or retweeting) and writes these information in different files.

    c. Process the users: Write a script that for each user calculates its probability of being a bot and stores all data in a new file.
    d. Generate the interactions graph: This activity consists of the implementation of a script that generates an interactions graph by using the processed users file and the interactions file.
5. Writing the first part of the documentation: Set the document structure and start with the writing of the documentation for the final degree project.
6. Analysing the Municipal Election: Use the Analyse Hashtag tool for study all interactions between Twitter users that took place during the previous week to the 2019 may Spanish Municipal Election.
    a. Collect the Municipal Election tweets: Collect all tweets that contain the hashtags: #Elecciones2019, #EleccionesMunicipales2019, #EleccionsMunicipals2019 and #Elecciones26M using the tool implemented in the activity 4.a.
    b. Process the tweets: Use the tool implemented in the activity 4.b to process all captured tweets and generate corresponding intermediate files.
    c. Process the users: This activity consists of using the tool developed in the activity 5.c in order to analyse all users in the files generated by the previous activity.
    d. Generate the Municipal Election interactions graph: This activity consists of using the tool created in the activity 5.d. to generate the interactions graph.
    e. Analyse the results: In this activity, the Municipal Election interactions graph is going to be analysed and studied in order to get some conclusions about the presence and effect of bots during the Election.
7. Implementation of the web service: This activity consists of the implementation of a web service that includes the User Analyser and Hashtag Analyser functionalities, in order to create a beta version of an easy to use service for all kind of users.
    a. Build the web service basic structure: In this activity the tasks to do are defining the different pages (home, user analyser and hashtag analyser pages), implement a basic version of the API rest and set the database.
    b. Add the Analyse *User* functionality: This activity consists of adding the Analyse *User* functionality to the API and finish the user analyser main page.
    c. Add the Analyse *Hashtag* functionality: This activity consists of adding the Analyse *Hashtag* functionality to the API and finish the hashtag analyser main page.
8. Finishing the documentation: Finish the writing of the documentation by adding all work carried out, the Municipal Election Users Interactions analysis results and the conclusions of all work done.

The plan for the project activities is shown in *Figure 4*.



Nombre de tarea	Duración	Comienzo	Fin
Understanding of requirements	10 días	lun 01/04/19	vie 12/04/19
Twitterbot detection model generation	**9 días**	**lun 15/04/19**	**jue 25/04/19**
Data preparation and feature extraction	2 días	lun 15/04/19	mar 16/04/19
Create the models	6 días	mié 17/04/19	mié 24/04/19
Store the model	1 día	jue 25/04/19	jue 25/04/19
Twitterbot detection Model	0 días	jue 25/04/19	jue 25/04/19
Implementation of the User Analyzer tool	**7 días**	**vie 26/04/19**	**lun 06/05/19**
Obtaining and processing User's data	2 días	vie 26/04/19	lun 29/04/19
Analyzing the User	3 días	mar 30/04/19	jue 02/05/19
Analyzing the Friends	2 días	vie 03/05/19	lun 06/05/19
User Analyzer tool	0 días	lun 06/05/19	lun 06/05/19
Implementation of the Hashtag Analyzer tool	**9 días**	**mar 07/05/19**	**vie 17/05/19**
Collect all tweets from a topic	2 días	mar 07/05/19	mié 08/05/19
Process the tweets	3 días	jue 09/05/19	lun 13/05/19
Process the users	1 día	mar 14/05/19	mar 14/05/19
Generate the Intreactions Graph	3 días	mié 15/05/19	vie 17/05/19
Hashtag Analyzer tool	0 días	vie 17/05/19	vie 17/05/19
Writing the first part of the documentation	25 días	lun 03/06/19	vie 05/07/19
Analyzing the Municipal Elections	**42 días**	**sáb 18/05/19**	**vie 12/07/19**
Collect the Municipal Election tweets	7 días	sáb 18/05/19	dom 26/05/19
Process the tweets	15 días	lun 27/05/19	vie 14/06/19
Process the users	10 días	lun 17/06/19	vie 28/06/19
Generate the Municipal Elections Intreactions Graph	3 días	lun 01/07/19	mié 03/07/19
Analyze the results	7 días	jue 04/07/19	vie 12/07/19
Municipal Elections results	0 días	vie 12/07/19	vie 12/07/19
Implementation of the web service	**16 días**	**lun 15/07/19**	**lun 05/08/19**
Build the web service basic structure	5 días	lun 15/07/19	vie 19/07/19
Add the Analyze User functionality	3 días	lun 29/07/19	mié 31/07/19
Add the Analyze Hashtag functionality	3 días	jue 01/08/19	lun 05/08/19
Web service	0 días	mar 06/08/19	mar 06/08/19
Finish the documentation	20 días	mar 06/08/19	lun 02/09/19
Submit the project	0 días	mar 03/09/19	mar 03/09/19

Figure 4. Project activities planning.

3. Design

3.1. Architecture

The project consists of different tools which are connected between them, this is reflected in the web service. First, I will introduce the global architecture of the project and then I will explain the architecture of each one of the tools.

For using any of the implemented tools, a model is needed. So, first of all, the Model Generator code is executed in order the create the best possible model for *Twitterbot* detection. Then, the web service's API REST, when a user requests to analyse a Twitter account or a hashtag, executes the User Analyser or Hashtag Analyser tools. This tools use the generated model and get the information they need by using the Twitter's API REST (*Figure 5*).

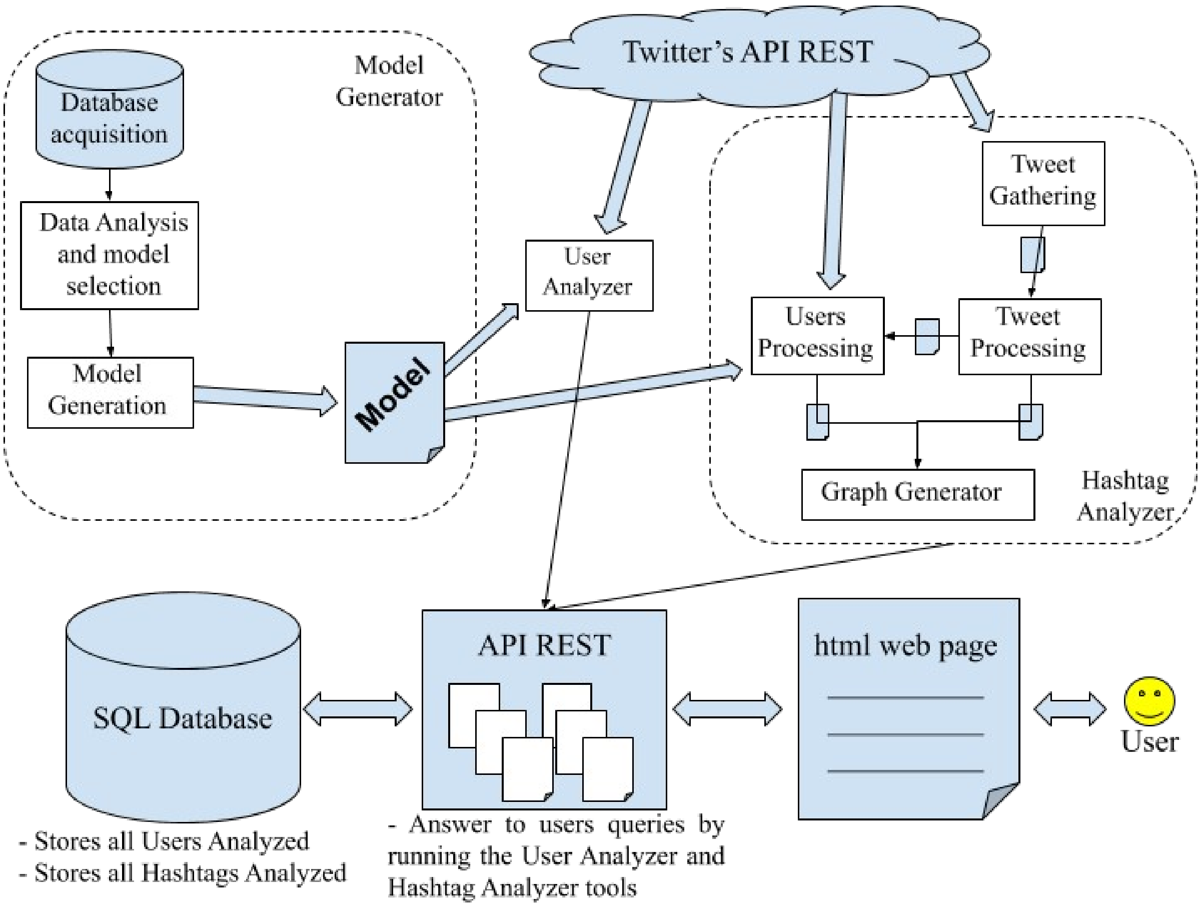


Figure 5. Project Architecture

As it is shown in *Figure 6*, each one of the tools has its own workflow with the exception that they all need a *Twitterbot* prediction model. The code that generates this model, first of all, loads the datasets and process them. In order to generate the best possible model, datasets containing different kinds of Twitter bots are used, such as fake followers, spambots, social bots, etc. and datasets containing genuine Twitter accounts. All this data has to be labelled (0 for the genuine accounts and 1 for the bot accounts) and merged into a single dataset. Then, from each account in this new dataset, a feature vector is generated, which are going to be used for training, testing and evaluating the different models. Finally, the model that has highest performance is generated and stored for being used by the other tools.

The User Analyser tool consists of a single script which needs as an input a Twitter account's *username*. The user analysing process is divided in three steps: first the user data is acquired using Twitter's API REST, next this data is processed obtaining the feature vector that the model uses as an input. Finally, the user is evaluated using the model and its feature vector and the result is printed.

On the other hand, the Hashtag Analyser tool consists of different scripts. This decision comes from the fact that each user may want to analyse Twitter users interactions around a topic during different periods of time. So, while tweets are being collected the other scripts of the tool can be used for obtaining partial results. Another reason for dividing the tool in different subprocesses is the computing cost of analysing huge amount of data and the Twitter's API REST limitations (180 calls every 15 minutes)[13]. Once the tweets are gathered, they are processed for getting all users in *Retweet interactions*: the users that have published the tweets and the ones who have retweet them. This information is stored in two different files, one containing the all users and the number of times they have interacted and another one with the user pairs: the user who published the tweet and the user who retweeted it. The same is done for the Favourite *interactions*. Before generating the interactions graph, the files that only contain users are processed in order to analyse each one of the Twitter accounts. This process is the same than the one carried out by the User Analyser tool, but instead of printing the results they are written back in another file. Finally, using a processed users file and its corresponding links file, the graph generator, creates and draws the corresponding interactions graph.

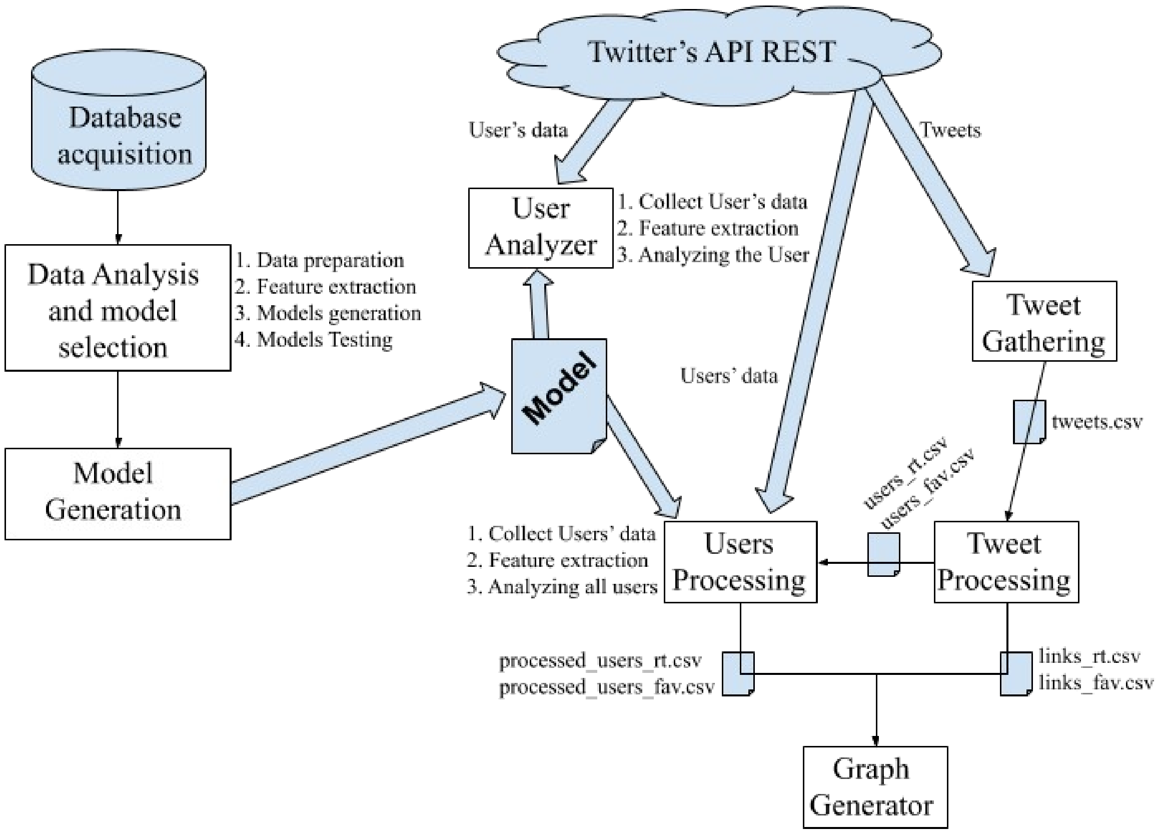


Figure 6. Project tools Architecture

13 https://developer.twitter.com/en/docs/basics/rate-limiting.html

The web service, as it is implemented using the framework Django, has follows its MTV (Model-Template-View) architecture (see *Figure 7*). The Model is the part of the web-app which acts as a mediator between the website interface and the database. It is the object which implements the logic for the application's data domain. The View is formatting the data via the model. In turn, it communicates to the database and that data which transfer to the template for viewing. Template's main goal is to keep everything that browser renders. The model's data that's coming from the server in different parts while integrating the same when the user interacts with the website [13].

When a user requests a web page, the controller forwards the URL to the appropriate view. Next, the View processes the request by communicating with the database through Model, formatting the data and transferring it to the template. Then, the template integrates the data with the html code, which is sent to the user as a http response. Finally, the user will display the response on his browser.

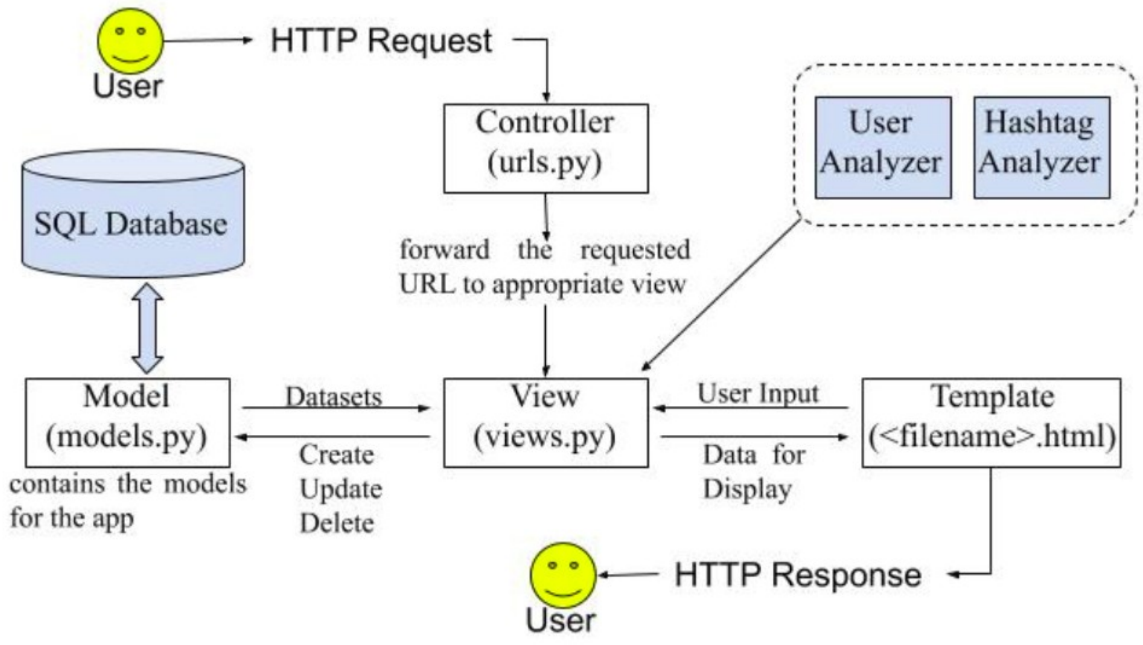


Figure 7. Web Service Architecture

3.2. Functionalities

As it is said, the project consists of different tools where each tools has its own functionalities so, these are going to be explained tool by tool.

The Data Analysis and Model Generator script can:

1. Read data from different Twitter Users datasets (R1.1).
2. Preprocess the data to fix gaps and label the different users depending on if they are bot or genuine accounts (R1.2, R1.3).
3. Apply different functions of data manipulation to extract relevant information from data in order to generate the feature vectors (R1.4, R1.5).
4. Create models from data using Logistic Regression, Random Forest and KNN as statistical and classification methods (R1.6).
5. Test and validate the created models to insure their correctness and evaluate their performance in order the select the best (R1.7).
6. Store the model (R1.8).

The User Analyser tool can:

1. Load Twitter bot classification models (R2.2).
2. Ask the user which model wants to use (R2.3).
3. Use the selected model to analyse the indicated user.
4. Show the results in an easy to understand way (R2.4).
5. Ask the user if he wants to analyse all user analysed tools (R2.5).

The Hashtag Analyser tool is also divided in different scripts, so its functionalities are also going to be explained by modules.

The tweet collector module can:

1. Capture all tweets containing a certain hashtag in real time.
2. Store the captured information in a CSV file (R3.2).

The tweet processor module can:

1. Read the captured tweets CSV file (R3.3).
2. Process a tweet and get: the user who published it, the users who retweeted it and the users who favourite it.
3. Store the captured information in 4 different CSV files (R3.2).

The user processor module can:

1. Read the Twitter users CSV file (R3.3).
2. Load Twitter bot classification model (R3.5).
3. Analyse all users in the file using the model.
4. Store the processed users results in a CSV file (R3.2).

The graph generator module can:

1. Read the Twitter processed users CSV file (R3.3).
2. Read the inks CSV file (R3.3).
3. Define the graph nodes as all processed users.
4. Define the graph links as the interactions between the users.
5. Set the node colour depending on that's user probability of being a bot (R3.8).

6. Set the node size depending on that's user number of interactions (R3.7).
7. Draw the resulting graph (R3.6).

Finally, the web service functionalities are the following ones:
1. Show the probability of being a bot of the specified Twitter user.
2. Show all users already analysed and their probability of being a bot.
3. Show in real time the interactions graph of all users that tweet about a specific topic.

3.3. Data Persistence Design

The project, as its functioning consists of data flows, does not need the design of data persistence, with the exception of the model, which needs to be stored, and the web service, which has a SQL database. On the other hand, many datasets are used for generating the model and some temporary files are generated during the tools execution.
The datasets used by the model generator are CSV files with the fields shown in *Table 2*. This fields are the same that the ones contained in the metadata of a Twitter User account. When the users' data is processed during the feature extraction step, from the information of all these fields we set the variables that will be used as an input for the models. The feature vector will also contain the *knownbot* variable which indicates if the current user is a genuine or a bot account, this variable is only used during the model evaluation and validation. This feature vector is also generated by User Analyser tool when gets the User account's metadata using the Twitter's API REST.

id	name	screen_name	statuses_count
followers_count	friends_count	favourites_count	listed_count
URL	lang	time_zone	location
default_profile	default_profile_image	geo_enabled	
profile_image_url	profile_banner_url	profile_use_background_image	
profile_background_image_url_https	profile_text_color	profile_image_url_https	
profile_sidebar_border_color	profile_background_tile	profile_sidebar_fill_color	
profile_background_image_url	profile_background_color	profile_link_color	
utc_offset	is_translator	follow_request_sent	
protected	verified	notifications	description
contributors_enabled	following	created_at	
timestamp	crawled_at	updated	

Table 2: CSV Datasets fields

The intermediate CSV files headers, generated by the Hashtag Analyzer tool scripts (*Figure 6*), are shown in *Table 3*.

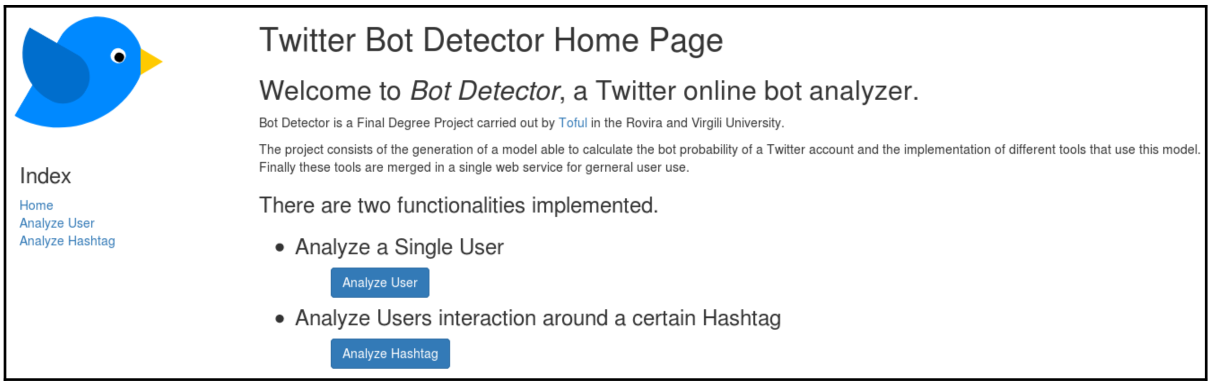
tweets.csv			
tweet_id	user_id	in_replay_tweet_id	in_replay_user_id
users_{fav/rt}.csv			
user_id	num_interactions		
links_{fav/rt}.csv			
user1	user2	num_interactions	
processed_users_{fav/rt}.csv			
user_id	user_name	num_interactions	bot_prob

Table 3. Hashtag Analyzer tool CSV files headers.

3.4. Graphical Interface Design

In the project, the unique component that has a graphical interface is the web service. The web has 3 different pages: the home page, the User Analyzer page and the Hashtag Analyzer page. All three pages have the same index at the left side which consists of a bird picture, representing Twitter, and links to the other pages.

The home page, *Figure 8*, has a brew description of what is the project about and direct links, by clicking a button, to the pages that contain the User Analyzer and the Hashtag Analyzer functionalities.



Twitter Bot Detector Home Page

Welcome to *Bot Detector*, a Twitter online bot analyzer.

Bot Detector is a Final Degree Project carried out by Toful in the Rovira and Virgili University.

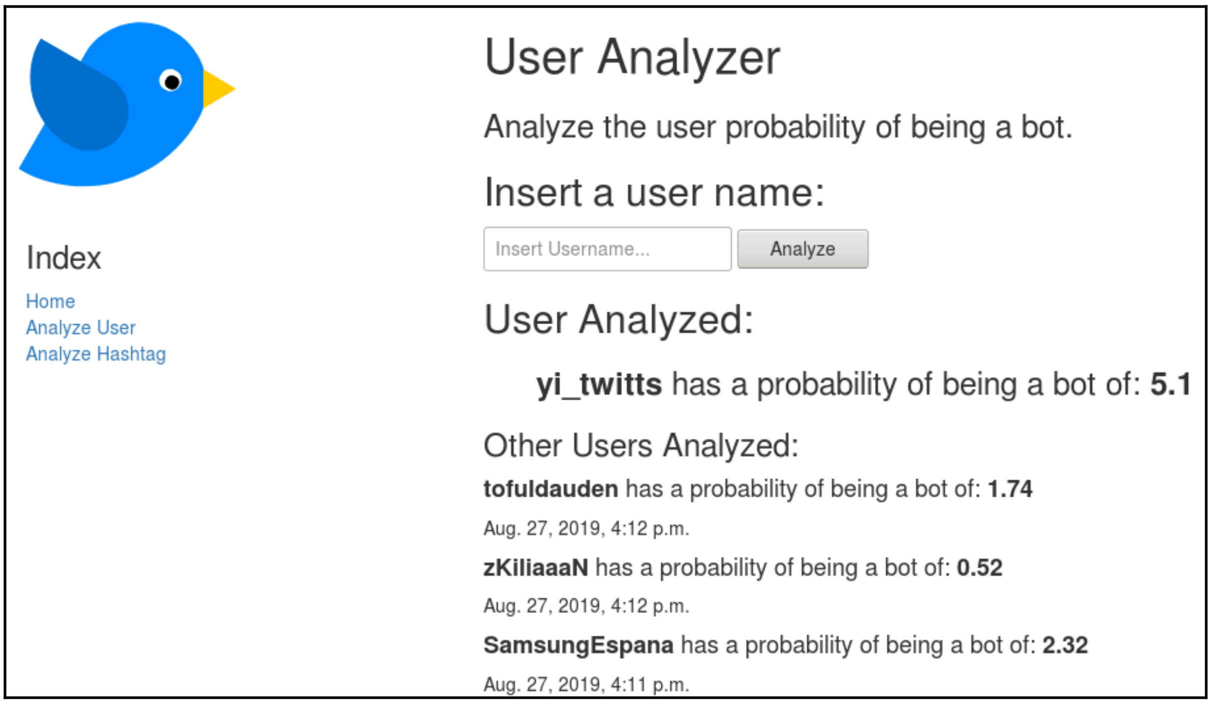
The project consists of the generation of a model able to calculate the bot probability of a Twitter account and the implementation of different tools that use this model. Finally these tools are merged in a single web service for gerneral user use.

There are two functionalities implemented.

- Analyze a Single User
- Analyze Users interaction around a certain Hashtag

Figure 8: Web service Home page.

The User Analyzer web page, *Figure 9,* is divided in different parts. In the first one there is the title “User Analyzer”, a brew description of the tool and a form to introduce a Twitter username with its corresponding submit button named “Analyze”. The second part is only shown when a correct Twitter account is submitted for its analysis. Then, it appears a section with the title “User Analyzed:” and the account analysis result. Finally, in the last section, are shown all Twitter accounts already analyzed and when they were evaluated under the “Other Users Analyzed:” title.


User Analyzer

Analyze the user probability of being a bot.

Insert a user name:

User Analyzed:

yi_twitts has a probability of being a bot of: **5.1**

Other Users Analyzed:

tofuldauden has a probability of being a bot of: **1.74**

Aug. 27, 2019, 4:12 p.m.

zKiliaaaN has a probability of being a bot of: **0.52**

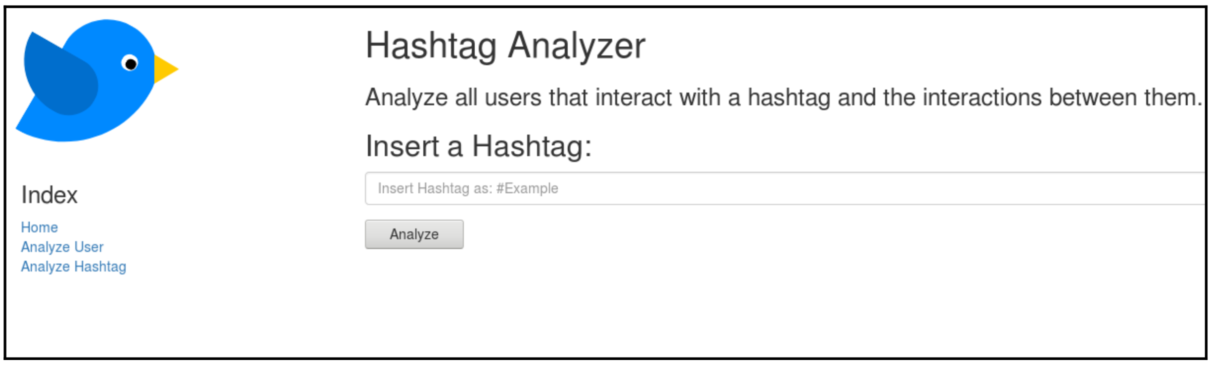
Aug. 27, 2019, 4:12 p.m.

SamsungEspana has a probability of being a bot of: **2.32**

Aug. 27, 2019, 4:11 p.m.

Figure 9: Web service User Analyzer page.

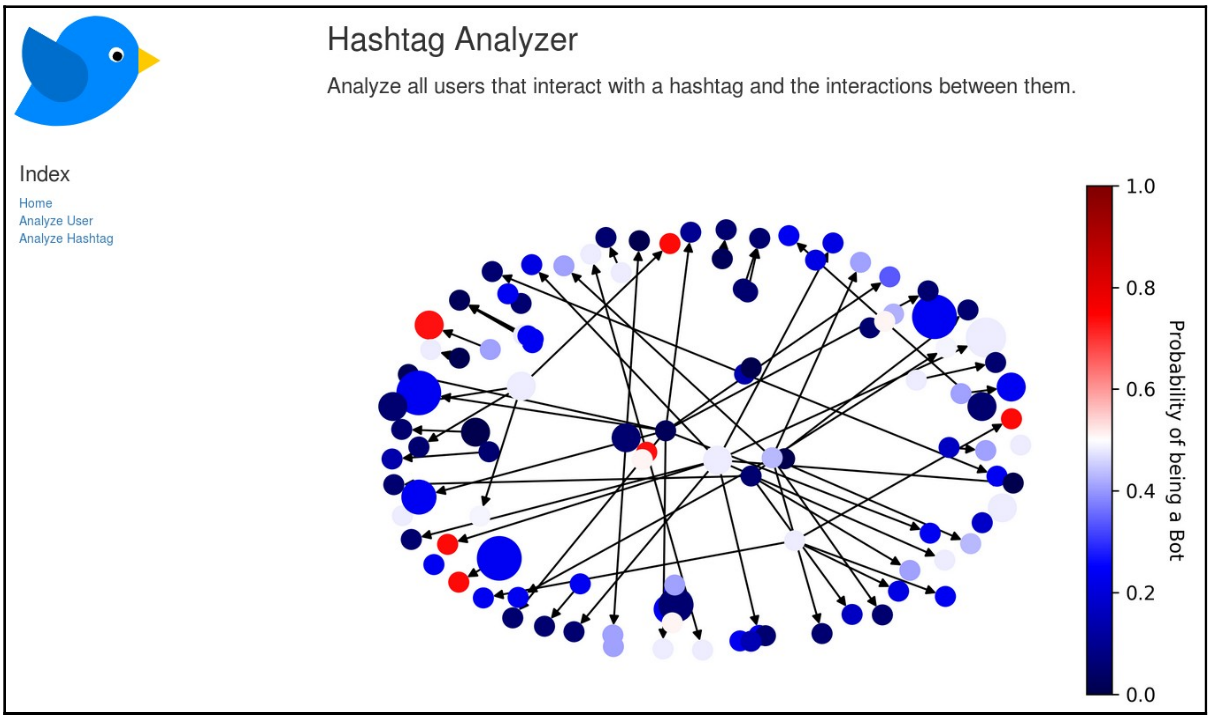
Finally, the Hashtag Analyzer web page has two main views. In the first one there is a form for introduce the hashtag you want to analyze, under the "Insert a Hashtag:" title, and its corresponding submit button (*Figure 10*). The other one is shown while the indicated hashtag is been analyzes, which simply consists of the interactions graph picture (*Figure 11*).


Hashtag Analyzer

Analyze all users that interact with a hashtag and the interactions between them.

Insert a Hashtag:

Figure 10: Web service Hashtag Analyzer initial page.


Hashtag Analyzer

Analyze all users that interact with a hashtag and the interactions between them.

Figure 11: Web service Hashtag Analyzer processing hashtag page.

4. Development and Implementation

The data analysis and model generation script and the User and Hashtag Analyser tools of the project have been designed by using the Python programming language. I decided to use this high-level dynamic programming language because of the huge amount of data science libraries it provides, which have been helpful during the data analysis and the model generation; its easy to understand syntax, so other users can modify the provided tools without many problems; and its compatibility with the Django framework, which has been used on the web service development.

Another step previous to the development and implementation of the project tools is getting the necessary keys for accessing to Twitter's API REST. In order to get this keys I only had to follow the steps below:

1. Create a twitter account if you do not already have one.
2. Go to https://apps.twitter.com/ and log in with your twitter credentials.
3. Click "Create New App".
4. Fill out the form, agree to the terms, and click "Create your Twitter application".
5. In the next page, click on "API keys" tab, and copy your "API key" and "API secret".
6. Scroll down and click "Create my access token", and copy your "Access token" and "Access token secret".

This 4 keys (API key, API secret, Access token and Access token secret) are stored in the "credentials.txt" file, located at the projects root, and is not going to be shared. So, if anyone wants to use the project tools has to previously generate its own "credentials.txt" file.

In the following points I will explain the project tools development and implementation step by step. Here I also will explain which libraries, machine learning techniques and algorithms I have used and why.

4.1. Data Analysis and Model Generation Development and Implementation

The first script implemented has many functions, such as understanding the datasets with which I am going to work, analyse the data, generate and evaluate some classification models with machine learning algorithms and save the best model in order to be used later by the other tools. For doing that I opted to work with *Jupyter Notebook* (R2), which is an open-source web application that allows you to create documents that contain live code, equations, visualizations and narrative text. In addition to this technology, I also used *Numpy* and *Pandas* libraries (R5) for dataset management and the *Scikit-Learn* library (R6) which contains methods and machine learning algorithms for models generation.

4.1.1. Data Preparation

The first thing I do, is reading the dataset I am going to use for the models generation using the Pandas read CSV function. This dataset is the **cresci-2017**[14] which consists of (i) *genuine*, (ii) *traditional*, (iii) *social spambot*, and (iiii) *fake followers* Twitter accounts, annotated by CrowdFlower contributors. Released in CSV format. From this accounts I have used the

14 https://botometer.iuni.iu.edu/bot-repository/datasets.html

genuine, the social spambots and the fake followers ones, generating 3 different sets:

- A set containing *genuine* and *fake followers* accounts.
- A set containing *genuine* and *social spambots* accounts.
- A set containing all accounts from the other two.

Before merging this accounts for the generation of the 3 different datasets, the accounts are labelled by adding a new feature to the ones they already have (see *Table 2*), the *knownbot* feature. This new variable will allow the models results validation. All accounts are also processed in order to fix those variables in which there is Nan value. Finally this sets are stored in a Python list.

4.1.2. Feature Extraction

All 3 datasets are then processed in order to get the feature vectors that are going to be used as an input for the models. This process consists of applying different lambda functions to each one of the datasets and store the result in a new Pandas' dataframe. The functions definitions used by the lambdas are stored in the "modules/aux_functions.py" auxiliary file. I have done in this way because, this functions are going to be reused later when using the generated model to analyse a new user. Another reason for doing it in this way, is the simplicity in adding new functions for new features extraction. By now, the resulting feature vector is a binary vector with the following variables:

Feature	Explanation
id	User Id, not used during the analysis and model generation.
profile_pic	Absence of profile picture.
def_profile_pic	Has the default profile image.
has_screen_name	Absence of a screen name.
30followers	Has less than 30 followers.
1000followers	Has over than 1000 followers.
1000friends	Has over 1000 friends.
30friends	Has less than 30 friends.
twice_num_followers	Has two friends for each follower it has.
fifty_FriendsFollowersRatio	Has fifty friends for each follower it has.
hundred_FriendsFollowersRatio	Has on hundred friends for each follower it has.
geoloc	Is not geo-located.
banner_link	Description contains banner links.
50tweets	Has sent less than 50 tweets.
20statuses	Has more than 20 statuses.
NeverTweeted	Has never tweeted.
has_description	Absence of description.
knownbot	Is a bot account. This feature is only used for the testing and validation.

Table 4: Features extracted from Twitter User's Data

Finally, I organize each one of the working datasets into two different data frames (the Y and X variables) The Y variable has the "knownbot" feature, which indicates if the account is a bot or not. On the other hand, the variable X contains all features the excluding the "id" and the "knownbot" ones. Variable X is going to be used for the training and testing of the model and the Y subset is going to be used for testing results validation.

4.1.3. Machine Learning Algorithms used

Before continuing with the models generation explanation, I will introduce the machine learning algorithms I have used for these models creation.

Random Forest Classifier

Random Forest Classifier (RF) is an ensemble machine learning method for classification.

RF operates with decision trees (see *Figure 12*). A decision tree is a representation of a decision procedure for determining the class (y) of a given instance (x). Each node of the tree represents a feature that partitions the space of instances at the node according to the possible outcomes of the test. Each subset of the partition corresponds to a classification sub-problem for that subspace of the instance, which is solved by a subtree. So, each link (branch) represents a decision (rule) that will generate a new subtree to solve a part of the initial problem and each leaf represents an outcome (categorical or continuous value)[14].

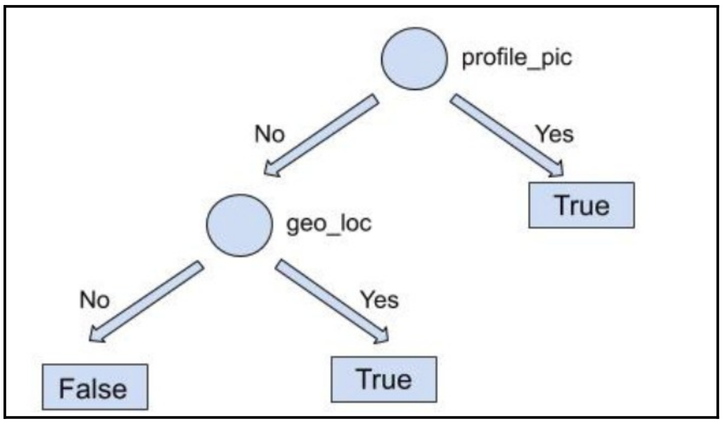


Figure 12: Decision tree structure.

During the training phase, RF operates by constructing an ensemble of decision trees (*Figure 13*), where each one of them will fit to a subset of the training dataset, this is what we call a forest. After, when using this model to do a prediction over a sample, the algorithm works as follows: each one of the decision trees makes a prediction, and the forest prediction comes from the all individual trees prediction mode/average. The main difference between a normal decision tree, like the ID3 decision tree, and the ones used in this algorithm is that when considering the features to do the space partition on each node, it only considers a random subset of all the available features (from here comes the “Random” of the algorithm name).

This classifier fixes the overfitting problem that appears in decision trees. The RF is less sensible to little input data changes than the decision tree and has a better global precision in classification tasks. However, the computational cost in creating the relational forest and using it is higher and explaining the results obtained is harder than using a single decision tree.

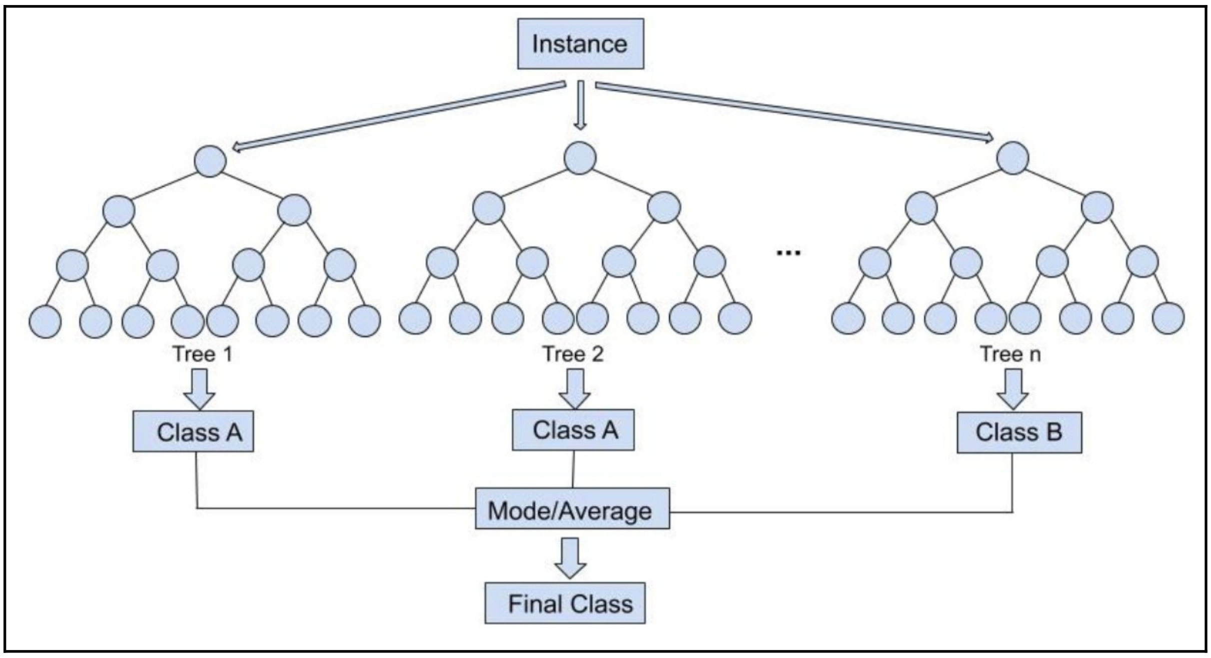


Figure 13: Random Forest structure.

Logistic Regression

In statistics, it is a model that uses a logistic function to model a binary dependent variable. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

The model dependent variable has two possible values, in our case, the Twitter account is a human or a bot, where these values are labelled “0” and “1”. The probability of each one of the two values is expressed in terms of a linear combination of one or more independent variables (“predictors”) which can be categorical (two classes) or continuous (any real value).

In logistic regression, $p(y=1|x)$ is modelled via the logistic function - a sigmoid function that takes any real input and outputs a value between zero and one. The logistic function $g(z)$ is defined as follows:

$$g(z)=\frac{e^z}{e^z+1}=\frac{1}{1+e^{-z}} \tag{1}$$

If we assume that z is a linear combination of multiple explanatory variables $x_1, x_2, \dots, x_n$. We can then express z as follows:

$$z=\alpha_0+\alpha_1 x_1+\alpha_2 x_2+\dots+\alpha_n x_n=\alpha_0+\sum_{i=1}^{n}\alpha_i x_i \tag{2}$$

So $p(y=1|x,\theta)$ can be written as:

$$p(y=1|x;\theta)=\frac{1}{1+e^{-(\alpha_0+\sum_{i=1}^{n}\alpha_1 x_1)}} \tag{3}$$

where θ are the parameters of the linear model $(\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n)$.

Note that by consistency, $p(y=0|x;\theta)=1-p(y=1|x;\theta)$.

To train the model, I compute the error of the predictions with respect to the real values with Cost function (Maximum log-Likelihood)

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)})-y^{(i)}) \quad (4)$$

$$A=\frac{-1}{m}[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))] \quad (5)$$

and I adjust the initial θ values by applying the Gradient descent algorithm to minimize the cost function value.

$$\theta_j=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)} \quad (6)$$

Now, the Logistic regression model is ready to predict new data. Using the new x values as an input we get the predicted y values.

K Nearest Neighbours

K nearest neighbours is an algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

A sample, x, is classified by a majority vote of its neighbours, with the case being assigned to the class (y: *Twitterbot* or human) most common amongst its K nearest neighbours measured by a distance function. If K = 1, then the sample is simply assigned to the class of its nearest neighbour. The distance function I have used is the Euclidean distance:

$$D=\sqrt{\sum_{i=1}^{K}(x_i-y_i)^2} \quad (7)$$

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN [15].

4.1.4. Models Generation and Evaluation

For the generation and evaluation of the models, I have used the Cross-Validation technique. In a prediction problem, cross-validation works as follows. Suppose we have a dataset D:={(y_i , $\mathbf{x_i}$)} where y is typically the dependent variable we want to predict (in this case if its a bot or not) and **x** i is the vector of independent variables/features we want to use for prediction. Our goal is thus to develop a model/algorithm that predicts y from **x**. To assess the predictive power of a predictive model, we split the dataset into two parts: the training set and the test set. Then, first, we fit our model using the training set and, after that, we evaluate the trained model using the test set. Once the evaluation is done, we use the prediction results (y_{pred}) and the dependent variable real values (y) of the test set to evaluate the model performance.

This process explained above has been carried out for the models generated by using the the 3 datasets defined in section 4.1.1. and the machine learning algorithms: Logistic Regression, Random Forest Classifier and KNN.

A simple/way to visualize the performance of a model/algorithm in a cross-validation strategy is to build a confusion matrix (see *Figure 14*). A confusion matrix compares predicted classifications (y_{pred}) with real classifications (y). In this case, there are only two possible values for y=1 (meaning that the Twitter account is controlled by a bot) and y=0 (meaning that the Twitter account is controlled by a human), therefore we can define:

- **True positives (TP):** When a true sample is predicted as true (equivalent to hit).
- **True negative (TN):** When a false sample is predicted as false (equivalent to correct rejection).
- **False positive (FP):** When a false sample is predicted as true (equivalent with false alarm).
- **False negative (FN):** When a true sample is predicted as false (equivalent with miss).

Confusion Matrix		Predicted Values: ypred	
		False	True
Real Values: y	False	True Negatives (TN)	False Positives (FP)
	True	False Negatives (FN)	True Positives (TP)

Figure 14: Confusion Matrix

In order to evaluate the test results and decide which is the best model/algorithm for Twitter bot detection, I focused in the analysis of the metrics:

- Accuracy Rate: It's the ratio of the correctly labelled accounts to the whole pool of accounts.

$$Accuracy\,rate = \frac{1}{m}\sum_{i=1}^{m} I\left(y^{(i)} = y^{(i)}_{pred}\right) \tag{8}$$

- Misclassification Rate: Is the proportion of misclassified observations.

$$Misclassification\,rate = \frac{1}{m}\sum_{i=1}^{m} I\left(y^{(i)} \neq y^{(i)}_{pred}\right) \tag{9}$$

- F1 Score: It is the harmonic mean of precision and recall.

$$F1\,Score = 2 \times \frac{Precision\ x\ Recall}{Precision + Recall} \tag{10}$$

Where the Precision is the proportion of predicted positive values that are correctly real positive values (11), and the Recall is the fraction of positive values that have been retrieved over the total amount of positive values (12).

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{11}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{12}$$

Note that F1 Score maximum value is 1 when Precision=Recall=1 and its lowest value is equal to 0 when either Precision or Recall are equal to 0.

4.1.5. Saving the Model

Once I have analysed with which algorithm I get the best prediction results, is time to generate a model with this algorithm and with all the available data in order to have more training examples. This process is quite similar to the model evaluation process, but this time, instead of using the cross validation technique for generating the model (which splits the dataset into the training and the test sets), I fit the algorithm with the entire dataset. When the model is trained and ready to be used I store it as a Python object in the "models/" folder by using the *dump* function of the *joblib* library (R4).

4.2. User Analyzer Implementation

This tool consists of a simple script named "user_analyzer.py" which has as an input a Twitter account *username*. First of all the auxiliary functions used for the feature extraction (explained in section 4.1.2.) are imported. Then the credentials file is read (using the *read_credentials()* method) and the Tweepy API object is initiated (R7), it will allow us to get data from Twitter using its API REST. This two steps are also done in other scripts.

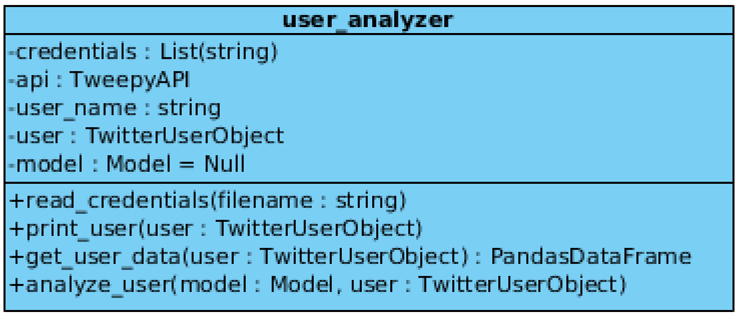

user_analyzer
-credentials : List(string)
-api : TweepyAPI
-user_name : string
-user : TwitterUserObject
-model : Model = Null
+read_credentials(filename : string)
+print_user(user : TwitterUserObject)
+get_user_data(user : TwitterUserObject) : PandasDataFrame
+analyze_user(model : Model, user : TwitterUserObject)

Figure 15: Variables and methods used in the user_analyzer.py script.

After this initialization steps, the Twitter User object of the input account is achieved by using the *get_user()* method of the API object. If it exists (the username specified as an argument is from a real Twitter account) a form asks the user which of the available models he want to use in order to analyze the Twitter account (the delivered project provides 3 models: one for detecting fake followers, another to detect social spam bots and the last to detect all kind of Twitter bots). This form is implemented with a switcher object. Once the model is selected, it is loaded and the analyzing process starts.

Next, the *analyze_user()* method is called which has as parameters the model and the Twitter User object. In this function, first, the *get_user_data()* method is used for getting the user's feature vector from the Twitter User object using the auxiliary functions imported in the first step. When the feature vector is build, the model makes the prediction and the result is printed through the terminal.

Finally, another form asks the user if he wants to also analyze all the analyzed-account friends. In an affirmative case, the *analyze_user()* method is used for each one of the friends inside a *for* loop.

4.3. Hashtag Analyzer Implementation

The Hashtag Analyzer tool consists of 4 scripts/modules that communicate between them by using intermediate files:

- **tweet_collector.py**: Uses the Twitter Streaming API to capture all tweets containing a hashtag specified as a parameter.
- **tweet_processor.py**: Processes all tweets captured by the tweet collector module, obtaining the users that have interacted around the specified topic and the relations between them.
- **user_processor.py**: Calculates the probability of being a bot of all users specified in the input file.
- **graph_generator.py**: Generates the interaction graph between all processed users.

Figure 16 shows the variables used and methods implemented in each one of the Hashtag Analyzer tool scripts.

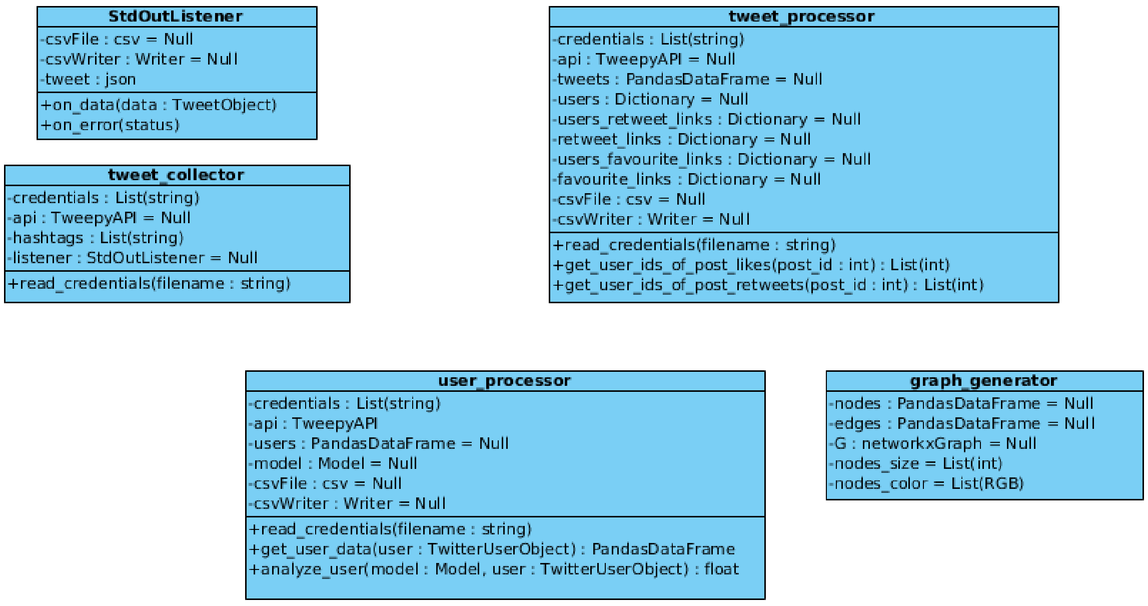

StdOutListener
-csvFile : csv = Null
-csvWriter : Writer = Null
-tweet : json
+on_data(data : TweetObject)
+on_error(status)
tweet_collector
-credentials : List(string)
-api : TweepyAPI = Null
-hashtags : List(string)
-listener : StdOutListener = Null
+read_credentials(filename : string)
tweet_processor
-credentials : List(string)
-api : TweepyAPI = Null
-tweets : PandasDataFrame = Null
-users : Dictionary = Null
-users_retweet_links : Dictionary = Null
-retweet_links : Dictionary = Null
-users_favourite_links : Dictionary = Null
-favourite_links : Dictionary = Null
-csvFile : csv = Null
-csvWriter : Writer = Null
+read_credentials(filename : string)
+get_user_ids_of_post_likes(post_id : int) : List(int)
+get_user_ids_of_post_retweets(post_id : int) : List(int)
user_processor
-credentials : List(string)
-api : TweepyAPI
-users : PandasDataFrame = Null
-model : Model = Null
-csvFile : csv = Null
-csvWriter : Writer = Null
+read_credentials(filename : string)
+get_user_data(user : TwitterUserObject) : PandasDataFrame
+analyze_user(model : Model, user : TwitterUserObject) : float
graph_generator
-nodes : PandasDataFrame = Null
-edges : PandasDataFrame = Null
-G : networkxGraph = Null
-nodes_size = List(int)
-nodes_color = List(RGB)

Figure 16: Variables and methods used in the Hashtag Analyzer tool scripts.

4.3.1. Collecting the tweets

The tweet_collector.py's script, as the user_analyzer.py tool, starts reading the credentials file and initiating the Tweepy API object. Continuously, the file where the tweets are going to be saved is created/opened using the Python CSV library (R8). This file has the default name "tweets.csv" and is stored under the "results/" folder.

Now is time to create and instantiate a stream listener object by defining a class that inherits from *StreamListener*. The *on_data()* method of a stream listener receives all messages and calls functions according to the message type. In our case, when the listener receives a Tweet, captures: the id of the tweet, the id of the user who have published it and, if the tweet is in reply to another tweet also captures the replayed tweet id and the id of the user who published it. Then, this information is stored into the "tweets.csv" file. I do not capture other information such as the users who have retweeted it because, the Tweet is collected at the instant it is published, so this value would be 0 and no interactions will be shown in the graph.

Once we have an API and a status listener we can create our stream object using the *Stream(auth, listener)* method. Finally, we use *filter(track)* method to stream all tweets containing the hashtags specified as the script input by assigning them to the *track* parameter (*track* parameter is an array of search terms to stream).

4.3.2. Processing the tweets

In an initial version of this tool, Twitter's API REST was used for gathering all users who retweeted a tweet. The problem in using the API for doing this task is that you are limited to 180 requests every 15 minutes, so if a tweet is retweeted 181 times, it will take half an hour to process a single tweet. Another issue is the absence of a method that returns all users who have favoured a tweet. Therefore, I opted for getting this information by using a http request, what has resulted a more efficient method.

The script that processes the tweets generated by the tweet_colector.py code is named tweet_processor.py. So, first of all, the "tweets.csv" file generated by the previous module is read and stored into a pandas dataframe. Then, the code results in a process which is repeated twice, first to capture all users and its relations around the *Retweet* interaction and after, for the *Favourite* interaction.

The process works as follows (see *Code 1*):

- For each tweet in the tweets.csv file, it is checked if the user who have published it is already stored in a users dictionary (in the dictionary, the key is the user id and the value is the number of tweets he has published plus the number of interactions with other users). If it exists, the number of interactions is incremented, if not, the user is stored and the number of interactions is initialized to 1.

- Then, all users who have interacted with the tweet (first, those who have retweeted it and, in the second iteration, those who have favourite it) are obtained by using the *get_user_ids_of_post_retweets(tweet_id)* method (in the second iteration, the method used is *get_user_ids_of_post_likes()*). This method returns a list of all users who have retweeted the tweet specified as a parameter by using a http request. For each user in this new list, the first step is repeated: if the user exists in the dictionary, it is updated, if not, the user is added. Additionally to this step, the tupla formed by the user who published the tweet and the user who interacted with it is stored in another dictionary, in the same way is done in the users dictionary (in this new dictionary, the key is the tupla formed by user who published the tweet and the user who interacted with it; and the value is the number of interactions between them).

- Finally, these dictionaries are stored in two different files. The first one will contain the users dictionary, named as "users_rt.csv", and the second one, will contain the interactions dictionary (the interaction is defined by the tupla: user who published the tweet - user who interacted with it). The second file is named "links_rt.csv"

In the second iteration, the files are named as “users_fav.csv” and “links_fav.csv”. This files are generated under the “results/” folder.

```
users_retweet_links = {}     #Users dictionary
retweet_links = {}           #Interactions dictionary
for i in range( 0, len(tweets) ):
    tweet_id = tweets['tweet_id'][i]
    user = tweets['user_id'][i]
    if user in users_retweet_links.keys():
        users_retweet_links[ user ] += 1
    else:
        users_retweet_links[ user ] = 1
    try:
        for user_rt in get_user_ids_of_post_retweets( tweet_id ):
            if (user, user_rt) in retweet_links.keys():
                retweet_links[ (user, user_rt) ] += 1
            else:
                retweet_links[ (user, user_rt) ] = 1

            if user_rt in users_retweet_links.keys():
                users_retweet_links[ user_rt ] += 1
            else:
                users_retweet_links[ user_rt ] = 1
    except:
        print("An exception occurred with user", user_rt)
```

Code 1: Algorithm for processing the tweets.csv file content.

4.3.3. Processing the users

The user_processor.py script is very similar to the user_analyzer.py tool. The main differences between them are:

- The first one has filename as an input parameter instead of a Twitter account screen name.
- The user can not choose which model wants to use, it has to be modified inside the code.
- The prediction results are not shown in the terminal.
- Do not analyse all user friends.

Internally, it works in this way: first the credentials file is read and the API object is instanced. Next, the “users_fav.csv”/”users_rt.csv” file is loaded and stored a Pandas dataframe. Finally, in a *for* loop, each user of the file is analyzed by using the model. The prediction, the user id, the user screen name and the number of interactions are stored in a new CSV file. This new CSV file is named “processed_users_fav.csv”/ ”processed_users_rt.csv” and it is also stored in the “/results” folder.

4.3.4. Generating the interactions graph

The graph_generator.py is the last script of the Hashtag Analyzer tool. It has as input parameters two files, one containing the processed users generated by the user_processor.py module, and the other containing all links between these users, generated by the tweet_processor.py module.

The code, first, reads both files and store them in two Pandas dataframes, variables **nodes** and **edges**. Continuously a Network X graph object is instanced and all nodes of the **nodes** variable are added using a *for* loop. In this loop there are also stored in two arrays the size of each node (the user number of interactions) and its colour (using the *cmap* function over the user’s probability). The colormap used is the seismic, where the nodes representing a Twitter account with a high probability of being a bot will be red and the accounts with a low probability of being a bot will be blue. After adding the nodes, the edges are added using the tuples stored in the **edges** variable.

Before plotting the graph I had to add a correcting algorithm for those accounts that appear in the **edges** list but not in the **nodes** list. These accounts are lost during the processing users step, if an account has been deleted or has a private status, it is not processed and saved. So, for solving this error I implemented an algorithm that gives the minimum size value and colour green to these users.

Finally the interactions graph with its corresponding colour bar are plotted in a 2D figure using the matplotlib library.

4.4. Web Service Implementation

The motivation of implementing the web service is a presentation of all developed tools in an easy way to use for users who are not familiarized with Python scripting and the used technologies. The development of the web service has consisted of following the "*Tutorial Django: El Sitio Web de La Biblioteca Local*" tutorial[15] by adapting it to my project requirements. The web service development can also be divided in three main steps: the generation of the web service basic structure, adding the *Analyze User* functionality and adding the *Analyze Hashtag* functionality.

Another fact to take into account, is that the web service tools are using my own Twitter's API credentials which is something that should not happen. The correct way to proceed is, before allowing any user to use any of the tools, force him to login with its Twitter account with the purpose that each one would use his own credentials. But as I said this is only a beta version of what the web service could be, so I opted not implementing this requirement.

4.4.1. Build the web service basic structure

First of all I need to generate the web service project with:

```
mkdir twitterBotAnalyzerWeb
cd twitterBotAnalyzerWeb
django-admin startproject twitterBotAnalyzerWeb
cd twitterBotAnalyzerWeb
```

Code 2: Creating the web service project.

Then I create the *hashtagAnalyzer* application inside the project (see *Code 3*). The name was decided in the moment of the application creation because of the most remarkable functionality of the web service, the possibility of generating a interactions graph in real time. As I will explain later, using this name was a mistake.

```
python3 manage.py startapp hashtagAnalyzer
```

Code 3: Creating the *hashtagAnalyzer* application.

After this steps the project main skeleton is created, see *Code 4*. There are files and folders which have to be self created such as the forms.py that contains the forms I will use and the "src/" and "static/css/" folders that contain the tools implemented and the css objects definitions respectively.

Once the project skeleton is created I finish with the configuration process by:

- Register the hashtagAnalyzer application. I have to append app name to the INSTALLED_APPS list located into the settings.py file.
- Connect the application urls.py file to the project's urls.py file, in order to map all project requests to the application. By adding the lines shown in *Code 5* to the twitterBotAnalyzerWeb/twitterBotAnalyzerWeb/urls.py file.

15 https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Tutorial_local_library_website

```
twitterBotAnalyzerWeb/
    manage.py
    setup.py
    twitterBotAnalyzerWeb/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    hashtagAnalyzer/
       __init__.py
       admin.py
       apps.py
       models.py
       tests.py
       views.py
       urls.py
       forms.py
       migrations/
       src/
       static/
          css/
       templates/
```

Code 4: Web service skeleton.

```
from django.conf.urls import include
from django.urls import path
urlpatterns += [
      path( 'hashtagAnalyzer/', include('hashtagAnalyzer.urls') ),
    ]
from django.views.generic import RedirectView
urlpatterns += [
      path('', RedirectView.as_view( url='/hashtagAnalyzer/',
permanent=True) ),
    ]
from django.conf import settings
from django.conf.urls.static import static
urlpatterns += static( settings.STATIC_URL,
document_root=settings.STATIC_ROOT )
```

Code 5: Modifying the twitterBotAnalyzerWeb/twitterBotAnalyzerWeb/urls.py file.

From here on, the implementation of the web service consist of a process of knowledge acquisition - programming. During this step I learn how the Django framework works, how the models.py, urls.py, views.py and the templates files interact between them and where I have to define each one of the elements that will perform the web service. As a result I implemented the home page with the corresponding links to the tool pages which will be implemented in a near future (see *Figure 8*).

4.4.2. Add the User Analyzer functionality

The process of adding this functionality to the web service consists of some steps. First I create the class *AnalyzeUserPost* in the models.py file. Here I define all the properties I want store from each user analyzed such as the user name, the user id, the probability of being a bot and when the model was analyzed for first time/updated. I also define the *AnalyzeUserForm* in the forms.py file. This class implements a model form that will allow me to ask the user for the screen name Twitter account he wants to analyze. Finally in the views.py file I define the *AnalyzeUserView* class which has two basic methods.

- The *get()* method creates an *AnalyzeUserForm* instance where the user will be able to write the Twitter account screen name he wants to analyze. It also gets all *AnalyzeUserPost* objects from the database and sort them by date. Then, these variables are transferred to the *analyzeUser.html* template for viewing.

- The *post()* method, a part from doing the same than the get method, also processes the requested form. It uses the User Analyzer tool to process the Twitter account specified by the web user in the form. If the new Twitter account does not exists, a new *AnalyzeUserPost* object is created and stored in the database; if it already exists the *AnalyzeUserPost* object is updated. Finally, all variables are transferred to the *analyzeUser.html* template for viewing, which in this case will also show the new analyzed user.

In addition to this, the *analyzeUser.html* template is also implemented. The template has three main sections: in the first one the form asks the user to introduce a Twitter account screen name; the second section only appears if the user submits a real Twitter account name, there is shown the user's screen name and its calculated probability of being a bot; finally, in the last section there are shown all users that have been already analyzed (see *Figure 9*).

4.4.3. Add the Hashtag Analyzer functionality

This functionality could not be completely added to the web service, here my mistake of naming the application *hashtagAnalyzer*. The original idea was:

> When the user specifies the hashtags he wants to analyze in the form, an asynchronous process starts capturing and storing all tweets published containing those hashtags by using the tweet_collector.py script. Simultaneously, every minute, another thread, process the tweets file, analyzes all the captured users, generates the graph and transfers it to the *analyzeHashtag.html* template for viewing.

The problem here is that process the captured tweets requires a substantial amount of time and computing power, what is unsustainable for a web service. While trying to find another way to implement the functionality the service shows an example of the final idea.

Despite this issue, I defined the *AnalyzeHashtagPost* class which contains a variable for storing the name of the hashtag analyzed and the *generate_graph()* method, which does the same than the genrate_graph.py script.I also defined the *AnalyzeHashtagForm* class in the forms.py file and in the views.py file I implemented the 0 version of the *AnalyzeHashtagView* class. This last class, has the same methods than the *AnalyzeUserView* class: *get()* and *post()*.

- In this case, the *get()* method only creates an *AnalyzeHashtagForm* instance where the user will be able to write the hashtag he wants to analyze and transfers the variable to the *analyzeHashtag.html* template for viewing.
- The *post()* method do not process the requested form. It uses the *generate_graph()* method of a *AnalyzeHashtagPost()* object instance, specifying as the parameters the path of two predefined files for the generation of an example graph. The method's output is a figure that contains the interactions graph generated from the example files. Finally this figure is transferred to the *analyzeHashtag.html* template for viewing.

Finally, I also implemented the *analyzeHashtag.html* template. This template has two main sections: one in which the form asks the user to introduce the hashtag he wants to analyze and the second in which the interactions graph is shown. Result is shown in *Figures 10* and *11*.

5. Evaluation and Results

In this section I focus on the data analysis results and on the evaluation of the generated models. After that I also compare the User Analyze tool with other existing tools that implement the same service and finally I briefly comment the results obtained by the Hashtag Analyzer tool.

5.1. Data Analysis Results and Evaluation of the Models

The data with which I worked consists of 3474 genuine Twitter accounts, 3351 fake followers accounts and 4912 spambot accounts. From this data, after preprocessing and feature extraction I obtain 3 datasets:

- Dataset 1 contains *genuine* and *fake followers* accounts, making a total of 6825 Twitter users.
- Dataset 2 contains *genuine* and *social spambots* accounts, making a total of 8386 Twitter users.
- Dataset 3 contains *genuine* accounts and both, *fake followers* accounts and *social spambots* accounts. What makes a total of 11737 Twitter users.

Each one of the users of in our datasets is defined by the 18 variables shown in *Table 4*. From which, the user id is not going to be used, and the *knownbot* variable is only used for the test validation.

Each model is validated using shuffle split cross validation. So, the dataset is split into the training set, which is used for training the model and the test set, which is used for its validation and performance evaluation. For the experiments I setted the training set size to an 80% of all dataset, leaving a 20% for the test set.

5.1.1. Random Forest Models Performance

The first machine learning algorithm with which I worked is Random Forest (RF). The parameters of the forest that I modified, in order to find the best configuration for my problem, are the maximum depth of the trees and the number of trees. The proceeding consisted on generating different models by changing the value of these parameters, evaluate them and define the optimum configuration for the algorithm.

Analyzing the results (*Table 5*), the results obtained by all implemented models are satisfactory. All models have achieved a F1 Score and Accuracy Rate metric values over 0.92 and all misclassification rates are under 0.08, when I start considering a good model with F1 Score and Accuracy Rate metric values over 0.80 and misclassification rate values under 0.20.

Taking a look with more detail to the results, I summarize that the performance is globally better when setting the max depth to 20. On the other hand, each dataset seems to have his own best value for the number of estimators parameter. When I will generate the model than only detects fake followers bots I will set the number of estimators equal to 100, while when I generate the model for detecting social spam bots I will set it to 150. Finally, when generating the model for detecting all kinds of *Twitterbots* I will also set it to 100, as we first fixed the max depth parameter to 20 (seeing the results, when the max depth parameter is set to 10, the best performance is gained by setting the number of estimators parameter to 200).

Comparing the use of each dataset for model generation between them, the best results are gained when working only with fake followers, while worst results are achieved when working only with social spam bots. As I initially thought, detecting fake followers Twitter bots is easier than trying to detect that social spam bots that imitate human behaviour. Getting better results when trying to detect all kind of bots than when trying to detect only social spam bots comes from the fact that the detection of the fake followers bot accounts contained in Dataset 3, improve the metric values.

Taking into account that values fluctuations between models are very small, I will not focus on the study of metric values evolution in front of the modified parameters. For more information, figures generated during the analysis process are shown in *Appendix A*.

Dataset	Max Depth	Number of Classifiers	Precision	Recall	F1 Score	Accuracy	Missclassification Rate
1	10	100	0,971	0,971	0,971	0,971	0,029
		150	0,965	0,965	0,965	0,965	0,035
		200	0,966	0,966	0,966	0,966	0,034
	20	100	0,971	0,970	0,970	0,970	0,030
		150	0,970	0,970	0,970	0,970	0,030
		200	0,965	0,965	0,965	0,965	0,035
2	10	100	0,937	0,936	0,936	0,936	0,064
		150	0,942	0,941	0,941	0,941	0,059
		200	0,932	0,931	0,931	0,931	0,069
	20	100	0,933	0,933	0,933	0,933	0,067
		150	0,943	0,943	0,943	0,943	0,057
		200	0,936	0,935	0,936	0,935	0,065
3	10	100	0,943	0,942	0,942	0,942	0,058
		150	0,942	0,941	0,941	0,941	0,059
		200	0,944	0,943	0,944	0,943	0,057
	20	100	0,946	0,945	0,946	0,945	0,055
		150	0,941	0,940	0,940	0,940	0,060
		200	0,944	0,943	0,943	0,943	0,057

Table 5: Random Forest models results.

To summarize, for the three models generation with the Random Forest algorithm (each one with one of the datasets), the parameters will be set as follows:

Model 1	Max Depth	Number of Estimators	Objective
1	20	100	Detecting Fake Followers bots
2	20	150	Detecting Social Spam bots
3	20	100	Detecting all kind of TwitterBots

Table 6: Parameters used on each model generation.

5.1.2. Logistic Regression Models Performance

The second machine learning algorithm used for the generation of Twitter bot detection models is the Logistic Regression algorithm (LR). Initially, my expectations about the models created with this algorithm were poor, but surprisingly the performance of the models was close to the obtained by the random forest models.

For model generation, I set the algorithm default parameters, getting the results shown in *Figure 17*. Such as with the RF models, the best results are achieved when using the Dataset 1 (*fakeFollowers*) and the worst when working with Dataset 3 (*spamBots*).

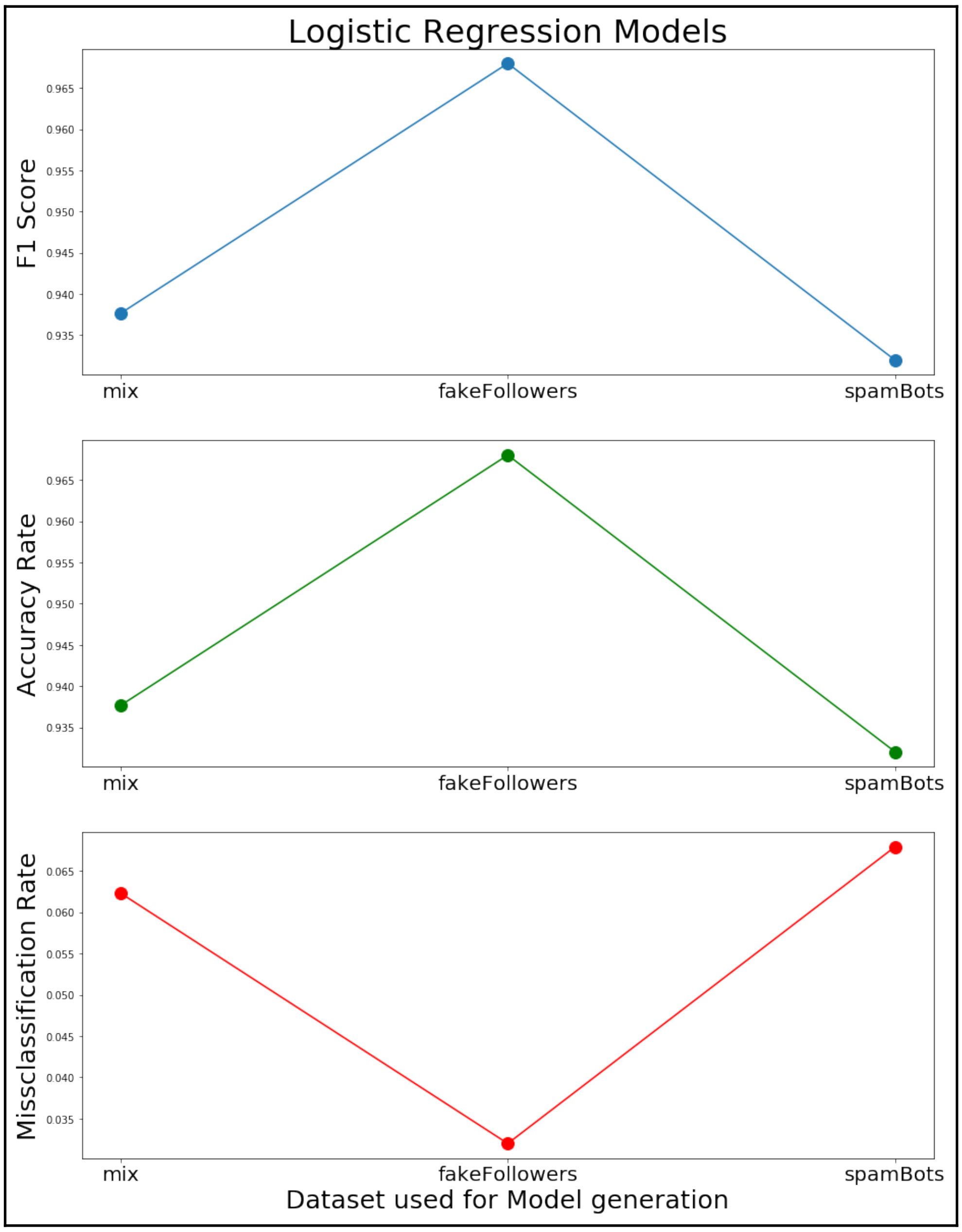


Figure 17: Logistic Regression models results.

5.1.3. KNN Models Performance

Finally, the last machine learning algorithm with which I worked is K-Nearest Neighbour algorithm (KNN). The generation of a model using this algorithms requires of two processing steps. First, I need to set the appropriate K-value (number of neighbours) taking into account the working data, so I have to search the best K-value for each dataset. The correct setting of this value will define the model performance. Once these values are specified, I use the cross validation technique and the corresponding k-value to train and validate the model.

In order to find the best K-value, I train the algorithm using a different number of neighbours and I annotate the error rate of each one of the models generated. Then I show the error rate evolution in a figure in order to decide the best K-value (*Figures 18-20*). The optimums K-value is the smallest value with the minimum error rate. So, when working with Dataset 1 I will user a K-value of 4, when working with Dataset 2 a K-value of 10 and when working with Dataset 3 a K-value of 7.

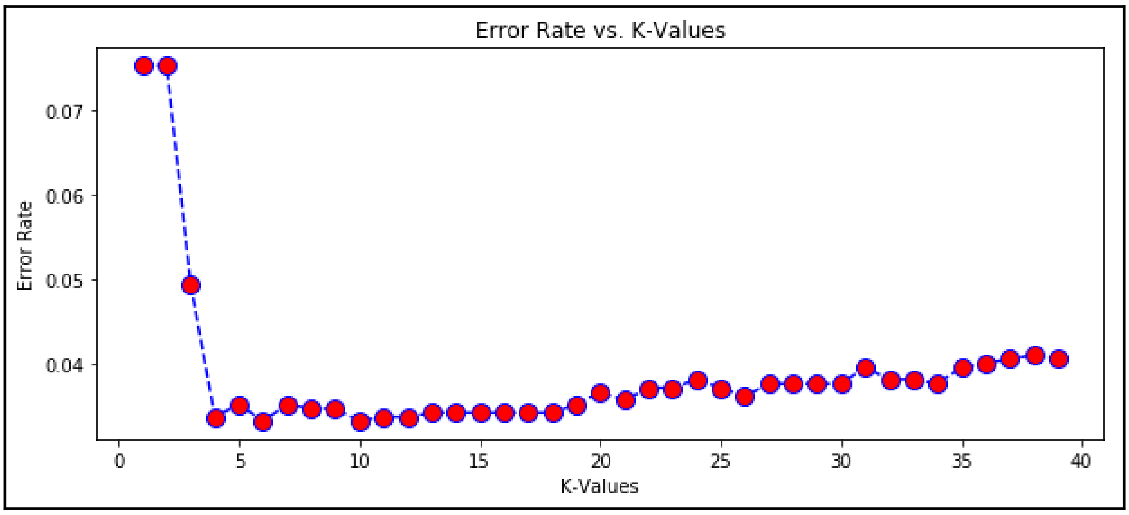


Figure 18: KNN Error Rate vs K-Values for Dataset 1

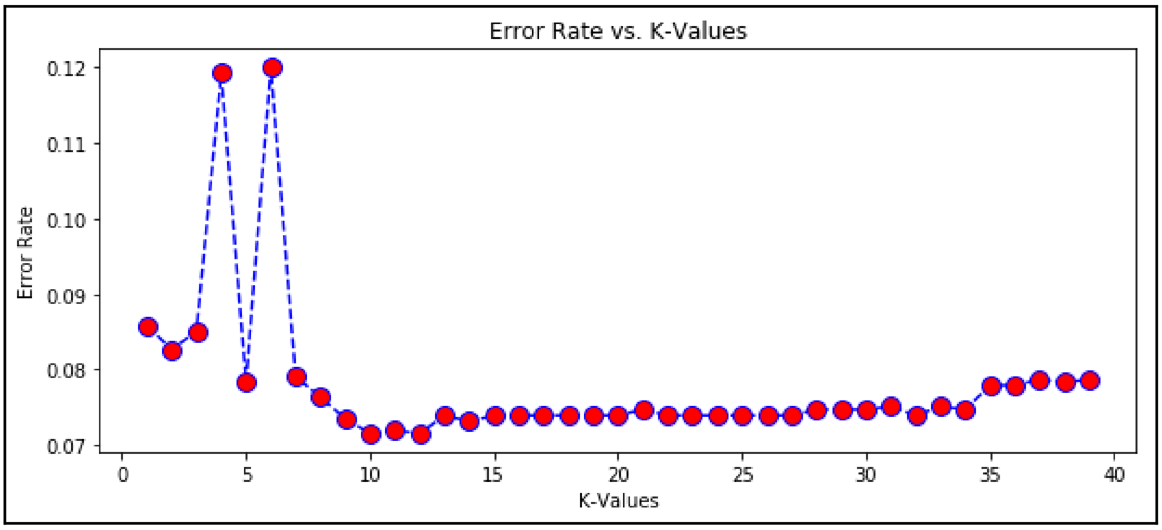


Figure 19: KNN Error Rate vs K-Values for Dataset 2

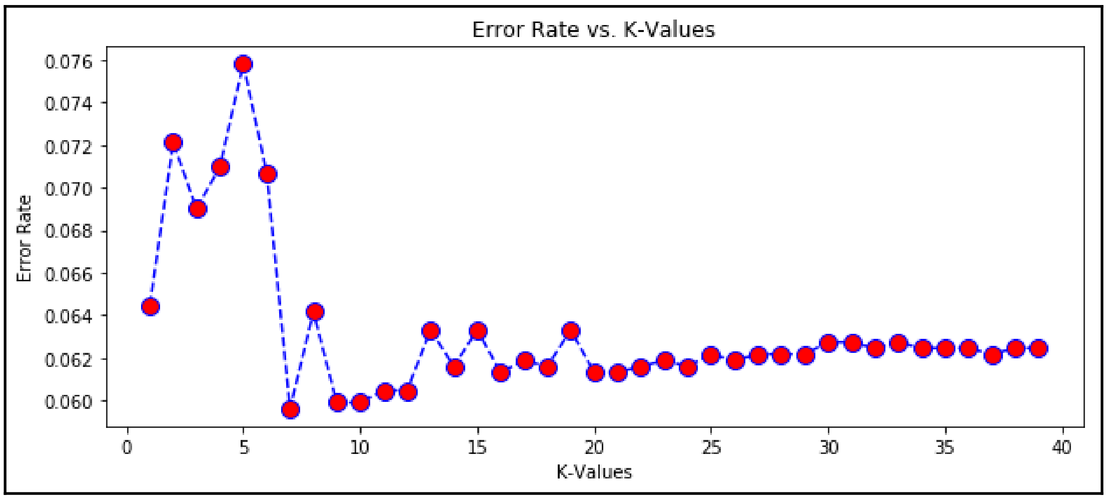


Figure 20: KNN Error Rate vs K-Values for Dataset 3

Finally, I train and test the models getting the results shown in *Figure 21*. Such as with the other models the best results are achieved when using Dataset 1 (*fakeFollowers*) and the worst when working with Dataset 3 (*spamBots*).

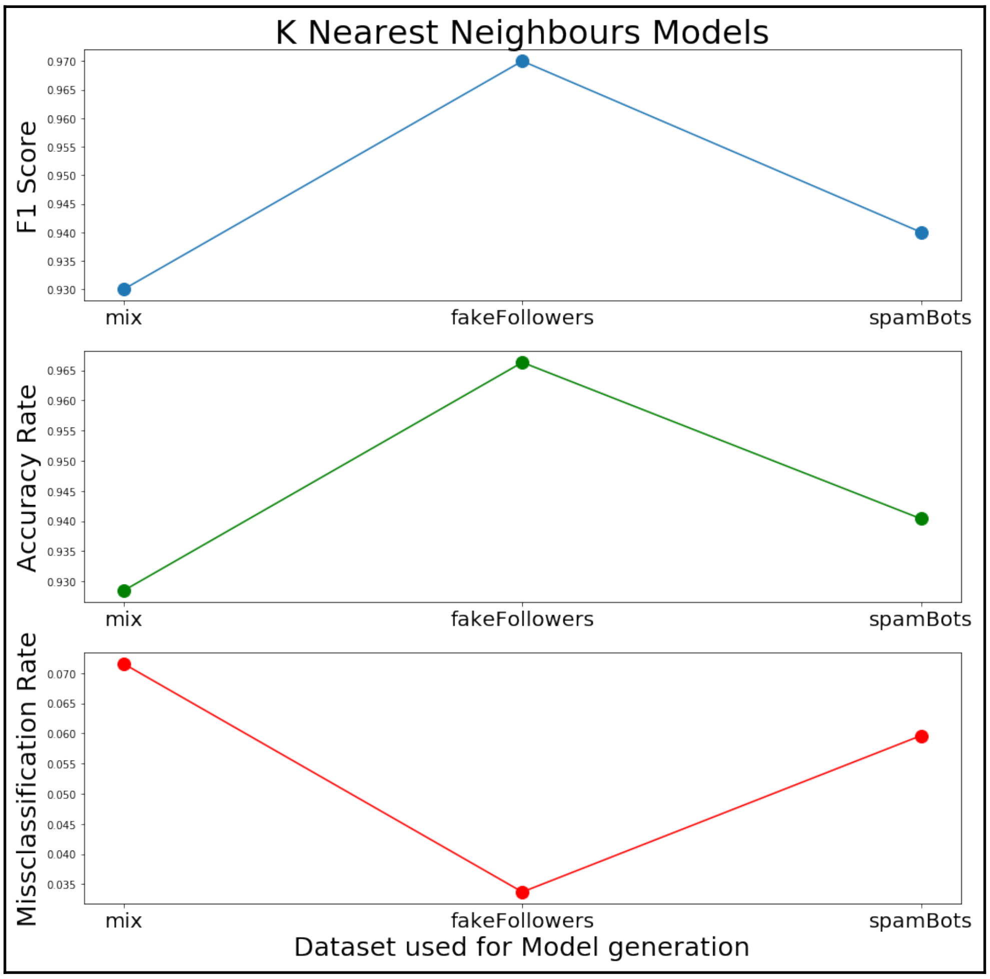


Figure 21: KNN models results.

5.1.4. Overall Comparison of the algorithms performance

To finish with the data analysis results, I present the overall Comparison of the models generated by the distinct machine learning algorithms performance. As can be seen in *Figures 22*, *23* and *24*, the models generated using the Random Forest algorithm with any of the datasets have higher performance than those generated by the other two algorithms.

Comparing KNN and LR models between them, when using the Dataset 1, the KNN model gets a higher score in the F1 metric while the LR model gets better score values in the Accuracy rate and Misclassification rate metrics. On the other hand, the model generated using the Logistic Regression Model and Dataset 2 gets better value in all metrics than the one generated using the same dataset and the KNN algorithm. Finally, when working with Dataset 3, of the two models, the KNN model is the one that achieves the best values in all metrics.

Taking this results into account, the models that have been generated and stored for being used by the other tools are the ones generated by the Random Forest algorithm. The User Analyzer tool will allow the user to choose between the three generated models (one detects fake followers bots, the other detects social spambots and the last, detects all kind of *Twitterbots*) while the Hashtag Analyzer will process all users using the third Model (the one able to detect all kind of *Twitterbots*).

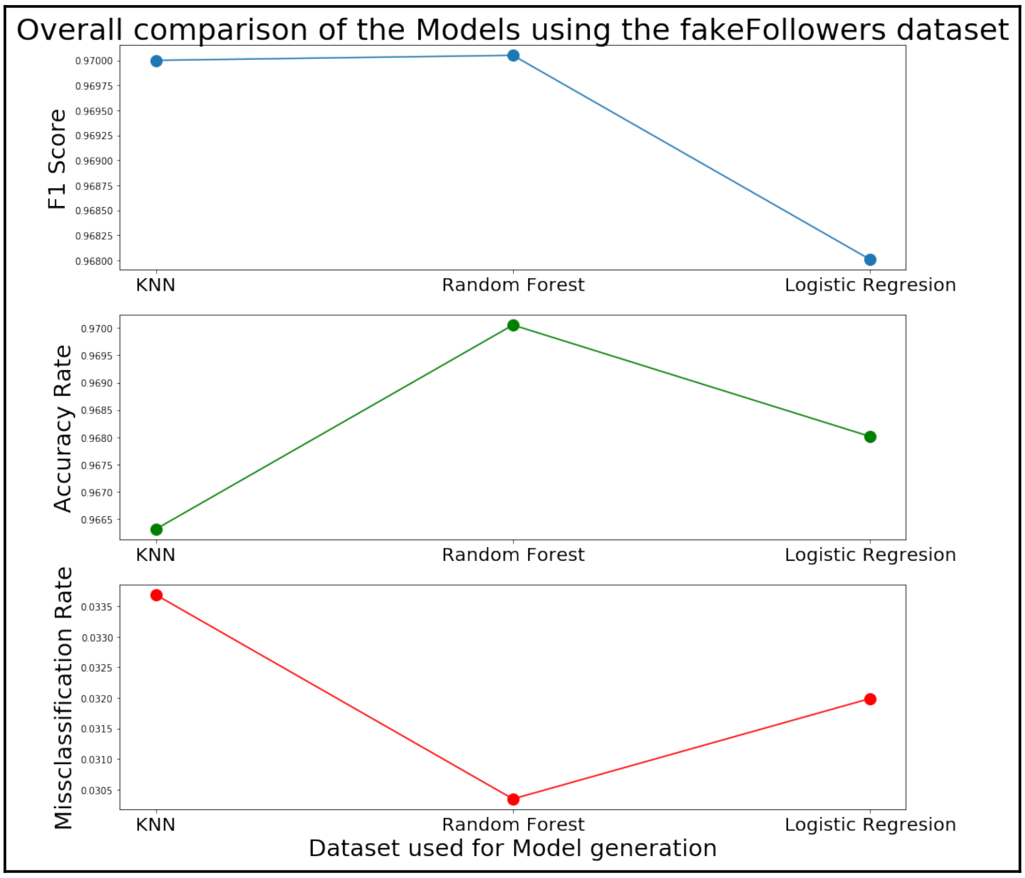


Figure 22: Overall comparison of the Models using Dataset 1.

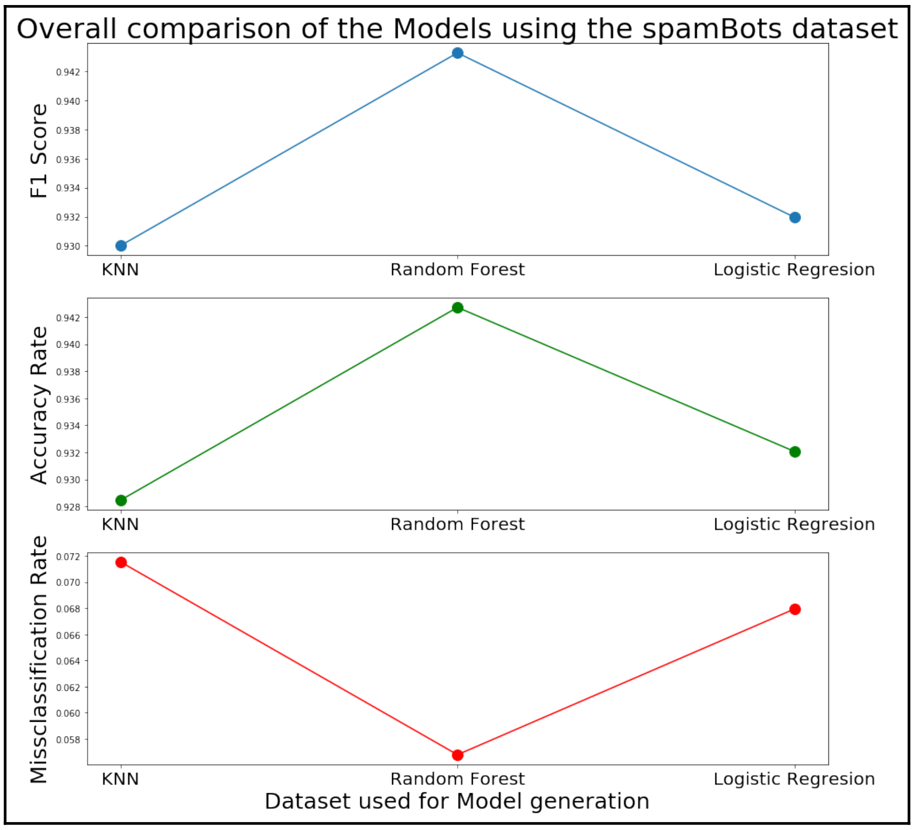


Figure 23: Overall comparison of the Models using Dataset 2.

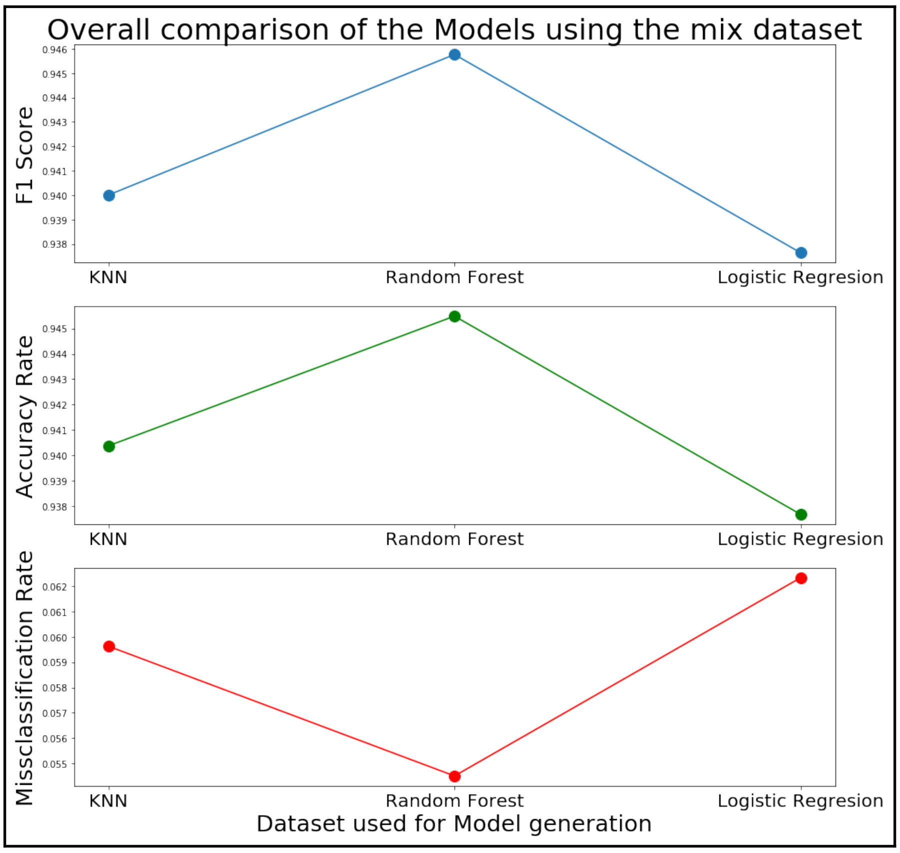


Figure 24: Overall comparison of the Models using Dataset 3.

5.2. User Analyzer tool evaluation

The resulting tool fulfils with all specified requirements, is fail-safe and the processing time of a single user takes less than 1 second, which are the most important characteristics a user wants when working with any application.

In order to evaluate my tool, I analyzed 10 random Twitter accounts from the **varol-2017**[16] dataset (5 genuine accounts and 5 bot accounts) with the my own software comparing the results with the ones obtained by the Botometer software.

As can be seen in *Table 7,* in spite of my models detect perfectly all real user accounts, when analyzing bot accounts with the model that detects all kind of *Twitterbots* the predictions are very poor (a prediction value of 0 indicates that the account is not a bot, and a prediction value of 10 indicates that the model is completely sure that the account is controlled by a bot). The other two models are able to detect 2 of the 5 bot accounts, and they predict a probability of being a bot to a third account close to 5. A fact to take into consideration is that, even though Botomoter classifies perfectly the accounts has treated, is unable to process those accounts that lack some of the features it needs to do the predictions.

Although, the results obtained by my models are considerably worse than the ones obtained by the Bototmeter software, I am happy with them, taking into account that I am only using 16 features to do my predictions in front of the over 1000 features that use Botometer. Furthermore, I have only used the **cersci-2017** dataset to train my models while he has used **varol-2017**, **caverlee-2011**, **cresci-2017** and **cresci-2015** datasets.

Comparing the predictions carried out by my models between bots and real users, I realize that the predictions values of the genuine accounts are all under 2.5. So, if I set the bot or not threshold to this value maybe the prediction results of my models would increase considerably. But this is only an hypothesis, so I lets it study to the future work.

User	Random Forest Model			Botometer	Is a Bot?
	Fake Followers	Social Bots	All Twitterbots		
tammylou01	8,39	8,66	2,38	9,40	1
d12cube_Don	0,00	0,20	0,40	ERROR: unauthorized	1
teakhollow	7,29	7,97	1,74	7,60	1
savinthegreat	4,90	4,60	1,40	ERROR: No Tweets	1
Kuma_chan_87	2,94	3,71	1,55	6,60	1
kelseycadden	0,12	1,28	1,75	0,20	0
bhaveshc_	0,00	0,90	1,10	1,00	0
alishas_wonder	0,12	1,28	1,75	0,40	0
Mr_Florida31	2,30	1,72	0,97	0,40	0
sharoonp143	0,12	1,28	1,75	2,00	0

Table 7: User Analyzer evaluation results.

As a last comment, analysing my own personal account with our implemented tool and comparing the results with Botometer ones, I realize that while my models are predicting my account as human (probabilities: 0.56, 3,96 and 3,75 respectively for each one of the models), Botometer is predicting it as bot with a probability value of 8.20. Which gives me a new hope in favour of my models capabilities.

16 https://botometer.iuni.iu.edu/bot-repository/datasets.html

5.3. Hashtag Analyzer tool evaluation

In order to evaluate this tool, and taking into account the Twitter bots influence on the 2016 U.S. Presidential Election, I decided to study Twitter interactions around the 2019 may Spanish Municipal Election.

The first step consisted on gathering tweets about the topic, so I captured all tweets containing the #Elecciones2019, #EleccionesMunicipales2019, #EleccionsMunicipals2019 and #Elecciones26M hashtags, from the 17th to the 26th of May 2019 using the tweet_collector.py script. The process finished with no problems, collecting an amount of 208157 tweets.

```
python3 tweet_collector.py #Elecciones2019 #EleccionesMunicipales2019
#EleccionsMunicipals2019 #Elecciones26M
```

Code 6: Collecting Municipal Election tweets.

The tweets processing process took more time than I expected. Process all tweets, in order to generate the users files and the links files, consumed more than 20 days, what caused me to delay on the project development. Finally, the resulting files contained:

- users_fav.csv → 229193 users.
- links_fav.csv→ 338060 links.
- users_rt.csv→ 98144 users.
- links_rt.csv→ 55002 links.

```
python3 tweet_processor.py results/tweets.csv
```

Code 7: Processing the tweets captured.

After that I also need to process both users files in order to analyze each one of the users, this process also consumed a huge amount of time. The generation of the processed_users_fav.csv file took 9 days and the processed_users_rt.csv file took 4 days.

```
python3 user_processor.py results/users_rt.csv
python3 user_processor.py results/users_fav.csv
```

Code 8: Process users files.

Finally, I generated the graph in order to study user interactions around the Municipal Election but, because of the huge amount of data I have to process, it took half a day to be performed. Another handicap is that the graph is too condensed and no conclusions can be retrieved from there. *Figure 25* corresponds to the retweet interactions graph, which has half of the users that the favourite interactions graph has and the sixth part of its links. I could not generate the *Favourite* interactions graph due to my personal computer hardware limitations.

```
python graph_generator.py results/processed_users_rt.csv results/links_rt.csv
```

Code 9: Generating the interactions graph.

A clearest example is shown in *Figure 11*, the one loaded by the Web Service. This graph is generated by processing the first 60 tweets of the 208157 captured during the week previous to the Election.

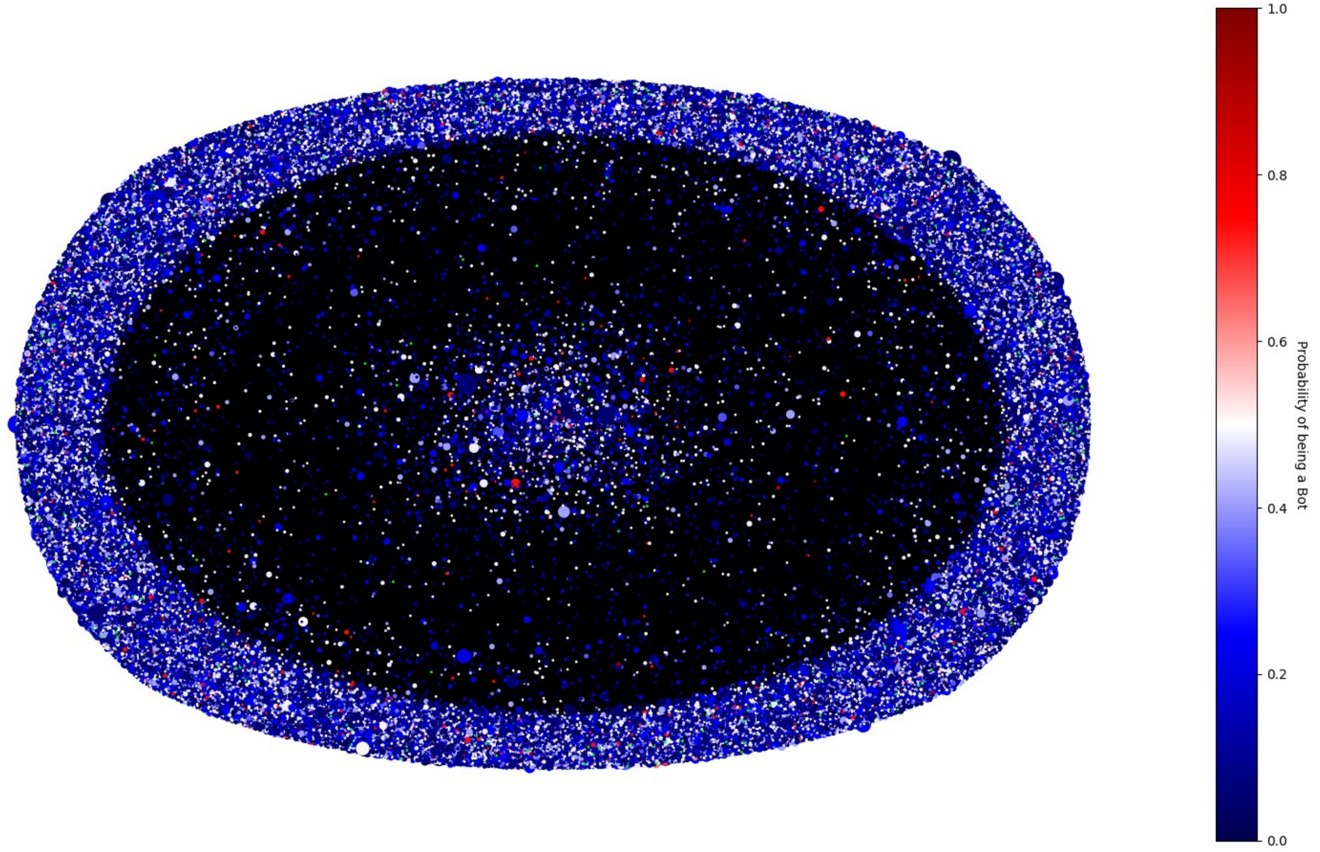


Figure 25: 2019 Spanish Municipal Election Retweet interactions Graph

5.4.1. Analyzing the 2019 Spanish Municipal Election interactions results

Here I will present some results obtained when analysing the data in files generated by the hashtag analyse process:

From the 98144 users who have interacted using the retweet action, 6068 have a probability of being a bot over 0.5. This means that more than the 6% of the Twitter accounts that have published or retweet a tweet during the study period about the Election were bots. Furthermore, the bot accounts interactions represent more than the 23% of all captured retweet interactions.

Analyzing the Favourite interactions, from the 229193 captured users, 16394 have a probability of being a bot over 0.5, what represents the 7% of the Twitter accounts that have published or favoured a tweet during the study period about the Election. On the other hand, in this case, even though having detected a higher proportion of bot accounts, only the 9% of the interactions have been carried out by these accounts.

From this results, we conclude that bots are more likely to favourite a tweet than retweeting it. On the other hand, the proportion of bots retweeting a tweet over all retweet interactions is higher that the proportion of bots favouring a tweet, being the first close to the 25% of all retweet actions. Also comment that the amount of bot accounts I detected is higher than what I expected.

Finally, take into account that the number of real bot accounts can be higher than the detected, as I explained in the point 5.2. the results obtained by my models are worse (detecting less bot accounts) than the ones obtained by other tools such as Botometer.

6. Conclusions, discussion and future work

This project is based on the design of tools that, I hope that, will help in the study and detection of bot accounts in the Twitter Social Media platform. It started with the idea of the generation of a model able to distinguish a Twitter account controlled by a human from one governed by a bot. But as the project progressed, new functionalities came up to my and my tutor's minds and the project became what now is:

> This project is not like a conventional software engineering project, this project, first, has a research part where the results obtained are used later in a software project.

The first part, the research part, consists of the use of different machine learning algorithms in order to generate the classification models that are going to be used later by the other parts of the project. Working with the used dataset the generated models have promising results, getting scores for the evaluating metrics over 0.9. On the other hand, when using these models in the User Analyzer tool and comparing their results with the ones obtained with other tools, such as Botometer, I realize that my models need to be improved. In order to improve the models, leaving it as future work, some things I want to do, are:

- Using more data when training the models, use many datasets instead of only one (the **cersci-2019**).
- Use more features, the models use only 16 features for doing their predictions, while Botometer uses more than 1000.
- Include text mining of the tweets published by the users, in order to have more information for a more accurate prediction.

The User Analyzer tool fulfils with all his requirements resulting in an easy to use, easy to modify and error-free terminal application. For using it with new models, you only need to modify the *get_user_data()* method, in order the generate the input feature vector for the new model from the Twitter User Object, and load them from the "models/" folder.

The other tool of the project is the Hashtag Analyzer tool which, at the same time, is composed by a subset of tools. All of the tools also fulfil with their requirements. A problem detected when using it, is that when you have collected a considerable amount of tweets, process them requires a substantial amount of time and computing power. In my use case, analyze Twitter users interactions around the 2019 Spanish Municipal Election, from when I had all tweets collected to when I generated the first interactions graph, took more than a month. Another issue when working with a huge amount a tweets is that the resulting graph is too compressed to extract any conclusion. In order to improve this tool, as future work:

- Add the study of *Replay* interactions.
- I should find a faster way to process the data.
- Add an option to allow the user to indicate the number of Twitter users he want to show in the interactions graph.
- Use a python library that generates graphs that allow real time user interaction.

Results obtained on the 2019 Spanish Municipal Election Twitter interactions analysis are very interesting. I detected, using my own model, that a 6% of all users who published a tweet that was retweeted or retweeted a tweet containing the use case hashtags (#Elecciones2019, #EleccionesMunicipales2019, #EleccionsMunicipals2019 and #Elecciones26M) were *T*witterbots. Furthermore, this 6% of the users generate close to the 25% of all *Retweet* interactions. In the same way, the 7% of all users who published a tweet that was favoured or favoured a tweet containing the use case hashtags were classified as a bots. Despite the increase in the proportion of bot accounts, this 7% of users only represents the 9% of all *Favourite* interactions.

The Web Service, was not an original goal of the project, but as my knowledges in web technologies were limited, I found interesting learn how to build a full web service architecture while developing my final degree project. So I decided to implement a web service that offers the project tools functionalities in an easy to use way for all kind of users. The final result consists of a simple web service implemented using technologies that I have not used before such as the django framework, the html and css languages, bootstrap, etc. that allows a web user to analyze a Twitter account by writing his screen name in a form and pressing a submit button. Unfortunately, the Hashtag Analyzer functionality could not be added because of the time and computer power that requires processing the captured tweets. Nevertheless I am satisfied with the result, as it is only a beta version what it could really be (needs extra security such as TLS and dataset encryption, it should use the Twitter credentials of the users that want to use the web service tools by allowing them to login with their Twitter account, ...).

If I could continue this work, I would like to improve the generated models by fulfilling the points aforementioned above, try to speed up the Hashtag Analyzer tool and when the last is accomplished start a new project consisting of the generation of a real Web Service that includes all this project tools and much more.

All project tools are free available in the following Github repository:
https://github.com/toful/BotDetector

The web service is not available because it requires my credentials in order to work and I can not make them public.

References

[1] Varol, O.; Ferrara, E.; Davis, C.A.; Menczer, F. and Flammini, A. Online Human-Bot Interactions: Detection, Estimation, and Characterization. *ArXiv*abs/1703.03107, **2017**.

[2] GlobalDots. https://www.globaldots.com/bad-bot-report-2019/. [Industry Report: Bad Bot Landscape 2019] **14/08/2019**

[3] Lokot, T. and Diakopoulos, N. News Bots: Automating news and information dissemination on Twitter, *Digital Journalism*, **2016**, **4**(6), 682-699.

[4] Savage, S.; Monroy-Hernandez, A. and Höllerer, T. Botivist: Calling Volunteers to Action using Online Bots. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, **2016**, 813-822.

[5] Ratkiewicz, J.; Conover, M. D.; Meiss, M.; Goncalves, B.; Flammini, A. and Menczer, F. Detecting and Tracking Political Abuse in Social Media. *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, **2011**, 297-304.

[6] Ferrara, E.; Wang, W.-Q.; Varol, O.; Flammini, A. and Galstyan,A. Predicting online extremism, content adopters, and interaction reciprocity. *In Social Informatics: 8th Intl. Conf., SocInfo*, **2016**, 22–39.

[7] Abokhodair, N.; Yoo, D. and McDonald, D. W. Dissecting a social botnet: Growth, content and influence in twitter. *In Proc. of the 18th ACM Conf. on Computer Supported Cooperative Work & Social Computing*, **2015**, 839–851.

[8] Ferrara, E.; Varol, O.; Davis, C.; Menczer, F. and Flammini, A. The rise of social bots. *Comm. ACM*, **2016**, **59**(7), 96–104.

[9] Bessi, A.; Coletto, M.; Davidescu, G. A.; Scala, A.; Caldarelli, G. and Quattrociocchi, W. Science vs conspiracy: Collective narratives in the age of misinformation. *PLOS ONE*, **2015**, **10**(2), e0118093.

[10] A. Bessi and E. Ferrara. Social bots distort the 2016 U.S. Presidential election online discussion. *First Monday*, **2016**, vol. 21, no.11.

[11] A. Davis, C.; Varol, C.; Ferrara, E.; Flammini, A. and Menczer, F. BotOrNot: A System to Evaluate Social Bots. *In Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*, **2016**, 273-274.

[12] Efthimion, P.G.; Payne, S. and Proferes, N. Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots. *SMU Data Science Review*, **2018**, vol. 1, no. 2, Article 5.

[13] Data Flair. data-flair.training/blogs/django-architecture/. [Django Architecture – 3 Major Components of MVC Pattern] **20/08/2019**

[14] Utgoff, P. E. Incremental Induction of Decision Trees, *Machine learning*, **1989** **4**(2), 161-186.

[15] Saedsayad. http://www.saedsayad.com/k_nearest_neighbors.htm. [K Nearest Neighbors – Classification] **24/08/2019**

Appendix A.

Metric results of all Random Forest models using different datasets and parameter values:

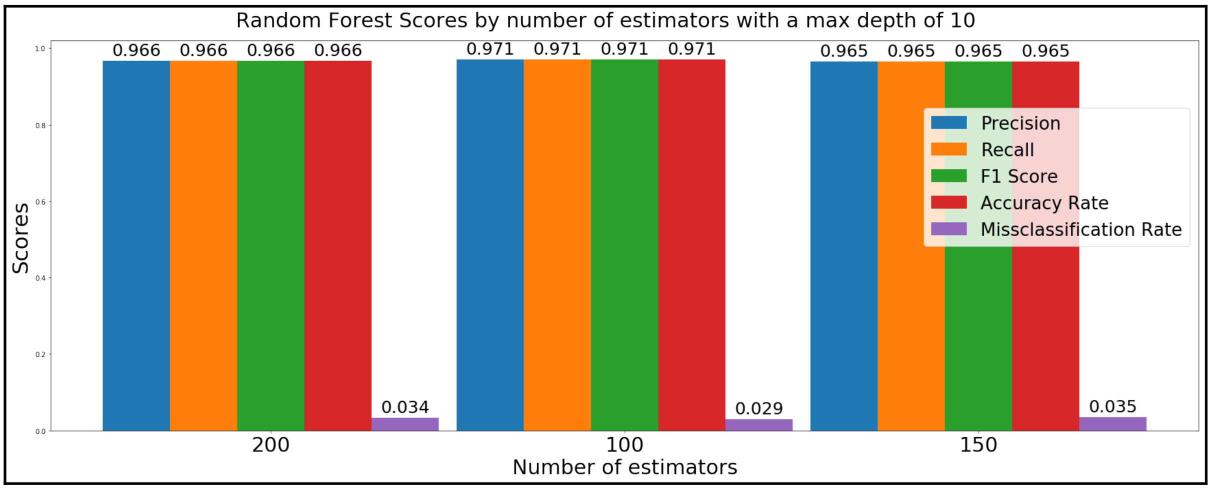


Figure 26: Random Forest Result with Dataset 1 and setting max_depth = 10.

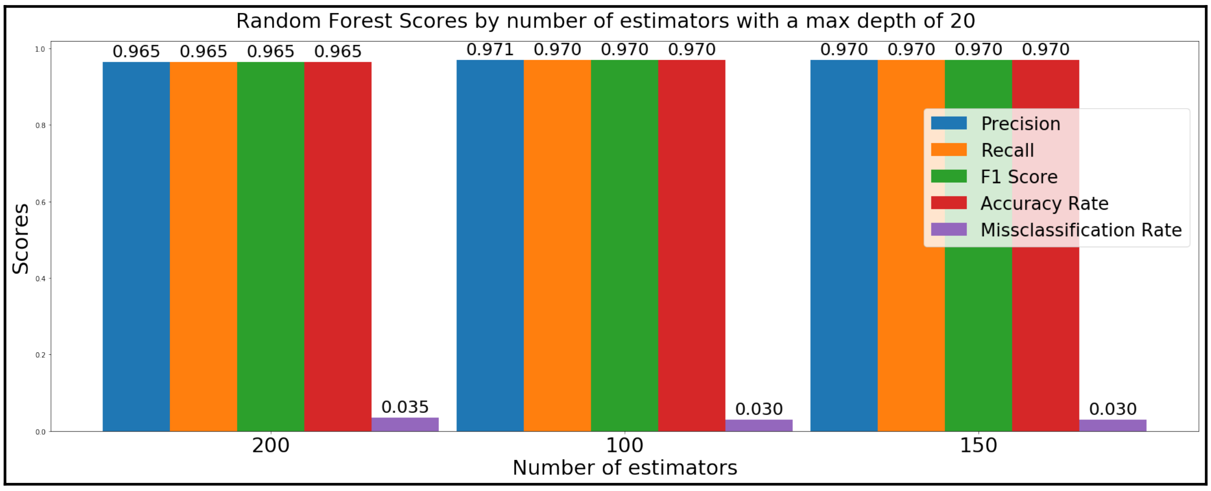


Figure 27: Random Forest Result with Dataset 1 and setting max_depth = 20.

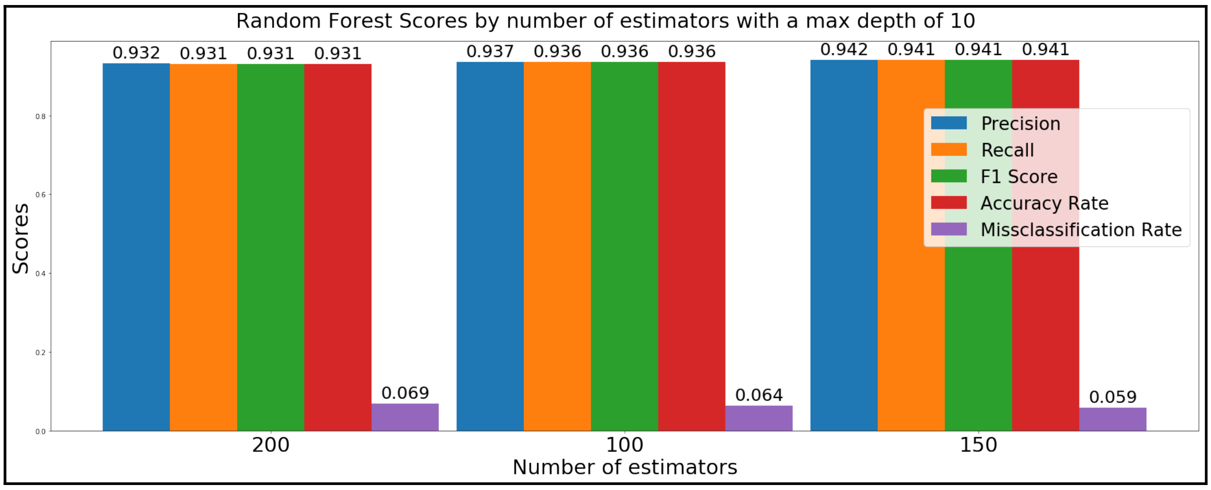


Figure 28: Random Forest Result with Dataset 2 and setting max_depth = 10.

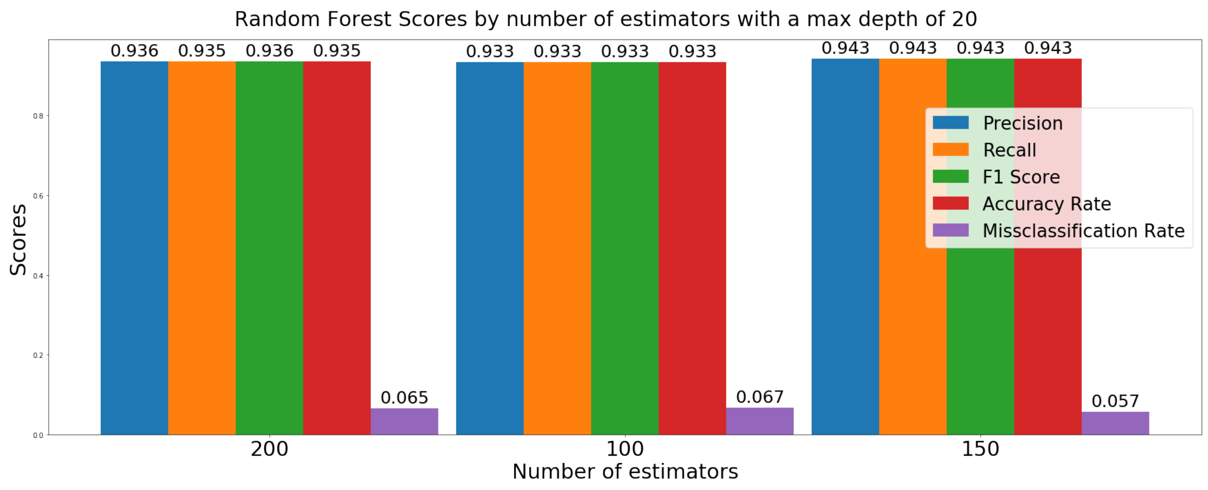


Figure 29: Random Forest Result with Dataset 2 and setting max_depth = 20.

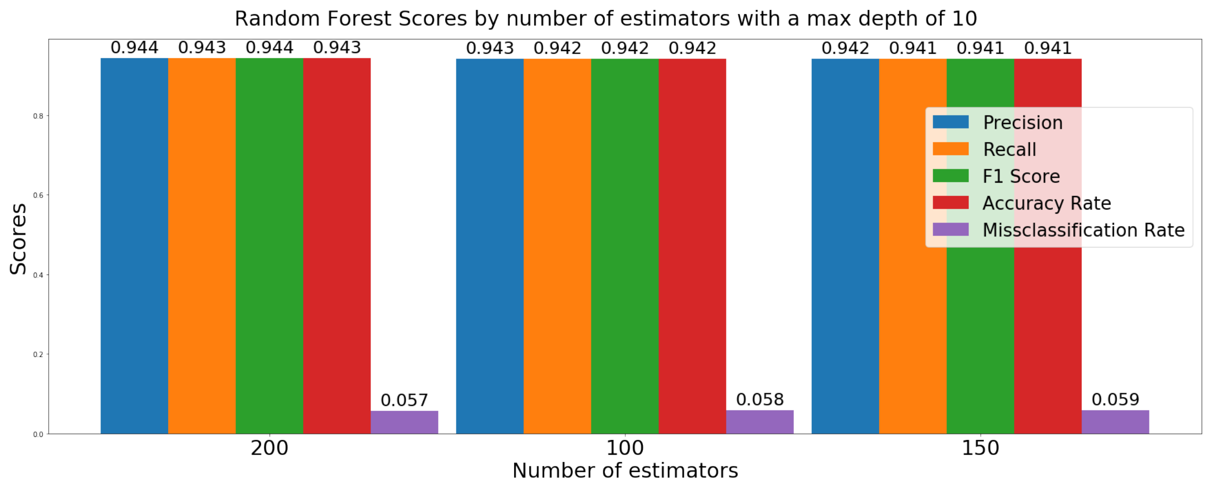


Figure 30: Random Forest Result with Dataset 3 and setting max_depth = 10.

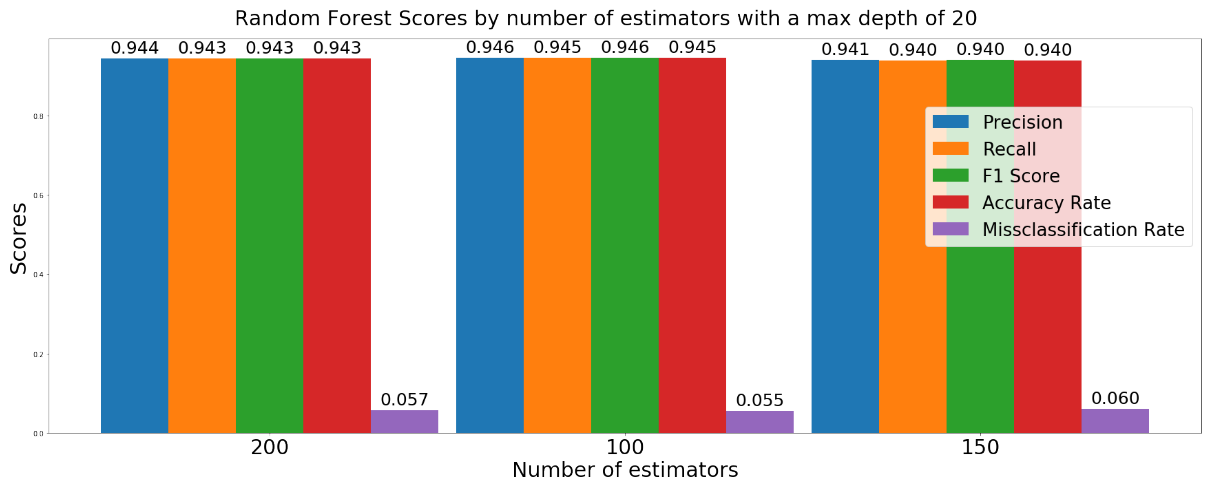


Figure 31: Random Forest Result with Dataset 3 and setting max_depth = 20