FINAL YEAR PROJECT

Preliminary Report

Author:
Marcell BATTA

Supervisor:
Dr. Lahcen OUARBYA

A thesis submitted in fulfilment of the requirements
for BSc Computer Science Degree

March 19, 2019

Contents

1 Introduction **1**
1.1 Aim . . . 1

2 Background Research **3**
2.1 Politics . . . 3
2.1.1 2016 US Elections . . . 3
2.1.2 2018 US Mid-Term Election . . . 3
2.2 Twitter junk . . . 4
2.3 Existing Systems . . . 4
2.3.1 Tweetbotornot . . . 4
2.3.2 DeBot . . . 4

3 System Design **5**
3.1 Method . . . 5
3.2 System Requirements . . . 5
3.3 Algorithm . . . 5
3.3.1 Random Forest . . . 5
3.3.2 Data . . . 8
3.3.3 Twitter API . . . 8

4 Planning **9**
4.1 Gantt chart . . . 9
4.2 Progress Logs . . . 9
4.3 Version Control . . . 10

Bibliography **11**

Chapter 1

Introduction

Social media has become an integral part of our lives in the past years as we spend more time online than ever before. Roughly 30% of our online time is spent on social media interactions, Twitter being one of them [1]. Twitter is arguably the largest source of news on the internet. This is due to the nature of information spreading on the platform through tweets and retweets. At this point, because of the scale, it is impossible to monitor it all to make sure everything is accurate and that there is no false information being spread.

1.1 Aim

The aim of this project is to create a program with an underlying algorithm that will attempt to figure out whether a Twitter account is under the control of a human or is purely being controlled by a script that someone wrote. The issue doesn't come from there being accounts not directly used by people or 'bots'. There are plenty of examples of public bot accounts for things such as weather or news. Instead the issues come with the ones that claim to be real individuals when they in fact are not. With the use of a program such as this, it can be possible to identify these bots by comparing their parameters like tweet content and see if they match patterns of other real people or not.
Ideally the program shouldn't misjudge too often and should be a reliable way to identify false accounts from real ones. This would be done through a supervised machine learning algorithm which would be fed with as much data of previously labelled accounts as possible.

Chapter 2

Background Research

This chapter contains the information found before beginning development of the program, along with some systems that are already available and a summary on them.

2.1 Politics

Politics is probably the biggest concern when it comes to these bot accounts. They are the reason why false information spreads so fast. This is because of the way Twitter works with its trending hashtags. These bots will tweet and retweet about important and most likely incorrect matters. They also make use of popular hashtags that basically define the topic of a tweet. This then leads to these malicious tags to become trending for everyone to see.

2.1.1 2016 US Elections

The 2016 elections in America was one of the, if not the biggest outburst of Twitter bots we have yet to see. It was found that by extrapolating some findings, roughly 19% of 20 million election related tweets originated from bots between September and October of 2016 [2]. According to the same study it was also found that around 15% of all accounts that were involved in election related tweets were bots. Now even though that is a lot of attention for these tweets containing false information, they will mostly only be seen by people who are already on the same side and agree. However, this doesn't rule out the affects. A study by the NBER(National Bureau of Economic Research) came to the conclusion that these bots were the cause of up to 3.23% of the votes that went towards Donald Trump [3]. This tells us that even if it's just marginal, it does still affect the outcomes.

The interesting part of all this is that the bots immediately went silent and disappeared as the election ended. The accounts though didn't get deleted but they simply went into hibernation waiting for their next bit of propaganda that needed to be spread. In 2017, 2000 of these bots reemerged to take part in the French and German elections as well, meaning they were run by the same people. They were discovered to make up for 1 in 5 election related tweets [4].

2.1.2 2018 US Mid-Term Election

Following the 2016 elections, the 2018 Mid-Terms were another prime target for Twitter bots. Before the voting took place, there were automated accounts trying to discourage people from voting. Of these, 10,000 were banned by Twitter. Even legislations were signed in an attempt to control the situation [5]. Interestingly, nearly

two weeks after the election day, there was still activity amongst these bots which accounted for a fifth of the #ivoted tweets [6]. The numbers recorded during this recent election compared to the one in 2016 was believed to be much lower. This could either be due to the reduced number of accounts being used or it could even be that the bots are now much more sophisticated and can remain undetected as they might be able to recreate human interactions and behaviour at a higher standard.

2.2 Twitter junk

Political issues aren't the only thing being caused by these bots. A study done at the University of Iowa has shown that through the third-party applications that Twitter allows its users to utilise can be, and is often abused in malicious ways such as phishing or even just spam. Twitter themselves have a way of dealing with these toxic accounts and do most of the time eventually ban them, however this study has found that 40% of the accounts that their algorithm detected as in some way benign were located about a month before Twitter took any action towards them [7]. As this does show that there are still improvements to be made even at Twitters end in terms of detection speed, we mustn't forget that there are many other parameters we must watch out for, some unknown outside of Twitter. A Twitter spokesperson wrote:

> "Research based solely on publicly available information about accounts and tweets on Twitter often cannot paint an accurate or complete picture of the steps we take to enforce our developer policies" [8]

2.3 Existing Systems

There are a handful of algorithms or programs that have been designed to detect these bots. Most of them tend to use a machine learning algorithm as a way to classify accounts and tell them apart from each other, while others have attempted to use deep neural network architectures such as long short-term memory(LSTM).

2.3.1 Tweetbotornot

Tweetbotornot is a package built in R that uses machine learning to classify Twitter accounts. It has two 'levels'. One for users where it uses information related to an account such as location or number of followers. The other is a tweet-level which checks for details like hashtags, mentions or capital letters out of the user's more recent 100 tweets. This could prove useful when testing my program to compare results as the accuracy of this library is 93.8 percent. As this is just a package created, it doesn't have any user-interface program built around it or anything like that, therefore it is unusable by anyone not knowledgable in R.

2.3.2 DeBot

DeBot is a fully functioning python API for bot detection on Twitter. It has the ability to obtain a list of bots already detected by DeBot, or just simply checking individual accounts. You can also get a list of bots which appear in the archives more than a given number of times. They can even be requested based on topics. It does however have a somewhat working built in search mechanism on its website.

Chapter 3

System Design

This chapter is about the methodology behind the program and some of the features and requirements.

3.1 Method

The core part of the system will be deciding whether the account it is checking is a bot or not. It will use a form of machine learning to classify the account as either 'bot' or 'not bot'. Initially it would seem as if this was a binary classification issue, however it makes more sense to treat it as a regression problem. This makes it much easier to interpret the results for the user as a probability is easily understood and is a much more honest answer compared to just giving the user a 'yes' or 'no' since we can never be too sure either way.

3.2 System Requirements

In order to achieve the aims of the system, there are a few things the program will need to do.

1. Allow users to input a Twitter account.
2. A connection using the Twitter API must be established in order to retrieve users' data.
3. The system must be able to determine whether an account is a bot or not.
4. Display the likelihood that an account is a bot or not.

3.3 Algorithm

The algorithm for my system will consist of a supervised machine learning algorithm called random forest. This works by creating a multitude of decision trees and outputting the mean prediction of these trees. However, depending on how things go during development, it might make more sense to use a deep learning neural network instead for the regression.

3.3.1 Random Forest

Random forest is a supervised learning algorithm. This means the data must be labelled, otherwise the algorithm won't know what to do with it. It's a useful algorithm, as it can be utilized for both classification and regression problems. In order

to understand and implement the random forest algorithm, we first need to know about its building blocks, the decision tree.
A decision tree is made up of an ensemble of branches and leaves. The branches contain the decisions, whilst the leaves determine the label that the tree believes the data belongs to. These decisions are made based on the features that best help us determine what label the data belongs to. A major downside to decision trees is that they suffer from overfitting when they become too deep with many branches.
This is where random forest comes in. As the name implies, it combines many decision trees into a forest like object where each trees outcome is thrown together and averaged out to get one answer. However, the clever thing it manages to do is the random feature selection. Each tree takes a limited number of random features from the original total and creates its own decision tree based on that. This helps it give a much more accurate result and mostly remove the overfitting aspect of the algorithm. The other form of randomness comes from the random subset of data selected with replacement for each individual tree, also known as bagging.
Once the subsets are decided upon and the trees are split up we must go through the nodes and decide on how to set these rules and which features to base them on. This is done using the gini impurity equation. It's the probability of any given node that a randomly selected sample would be incorrectly labelled if it was labelled by the distribution of samples in that node [9]. The gini impurity is worked out using the following equation:

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)$$

The gini impurity of a node n is 1 minus the sum over all the classes J of the fraction of examples in each class p_i squared [9]. At nodes beyond the root the gini impurity is additionally weighted by the fraction of points from its parent node. As it is the probability of incorrect labelling, we are looking for values as small as possible here. This is repeated throughout the algorithm recursively, finding the best possible values for the best features to pick until a given depth or if there is only one class' samples remaining.

To further help understand the algorithm it's crucial that we look at the pseudocode as that's much easier to translate into code.

Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

```
function RandomForest(S, F)
    H ← ∅
    for i ∈ 1, ..., B do
        S^(i) ← A bootstrap sample from S
        h_i ← RandomizedTreeLearn(S^(i), F)
        H ← H ∪ {h_i}
    end for
    return H
end function
function RandomizedTreeLearn(S, F)
    At each node:
        f ← very small subset of F
        Split on best feature in f
    return The learned tree
end function
```

FIGURE 3.1: Random forest pseudocode

As previously mentioned, the algorithm works by creating a forest of decision trees. This means that for B number of trees we take a bootstrap or bagging sample of features from the original data and create these decision trees by splitting the nodes on the best features. Once this has been complete all the way down the trees recursively, we return the finished trees and decide based on majority vote which outcome is the most likely.

To give a visual representation of a decision tree it would look something like the following:

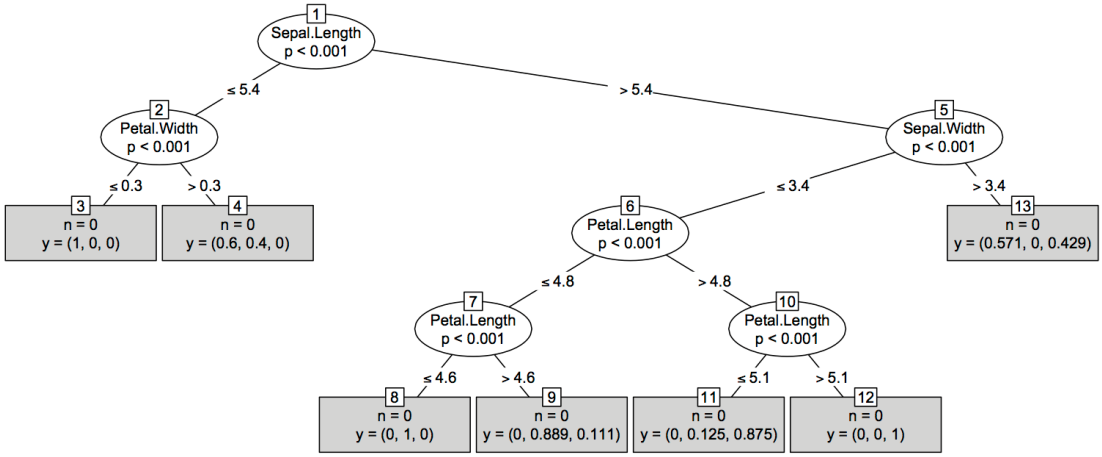

FIGURE 3.2: An example decision tree using the iris dataset

Here you can see that at each node there is a condition being made. This is the 'best' feature that is decided based on the earlier mentioned gini impurity being minimised. Then based on the value of these features for a given datapoint, the algorithm will traverse down a branch of the tree all the way to a root node. This then gives us our prediction for that one decision tree within the forest.

3.3.2 Data

Regardless of which algorithm I end up finalising with, I will need data for training. This is important since everything will be based on it. This data needs to also be correctly labelled and will most likely need to be pre-processed somewhat before being fed to my algorithm. It will then be used to train the system.

3.3.3 Twitter API

As the user will be able to enter a Twitter account name themselves, the program will need to have Twitter REST API functionality. This is to access the relevant accounts information such as tweets and account details. There is a limited number of requests allowed within a 15 minute window therefore I need to make sure to keep the API requests to a bare minimum.

Chapter 4

Planning

In this section I will talk about some of the things I have done to make sure everything goes as planned and my time is used efficiently.

4.1 Gantt chart

Gantt charts are a nice way to keep track of your time available over a whole project. It really helps understand the scope of things and to give a better estimate on how to split up your time into smaller chunks.

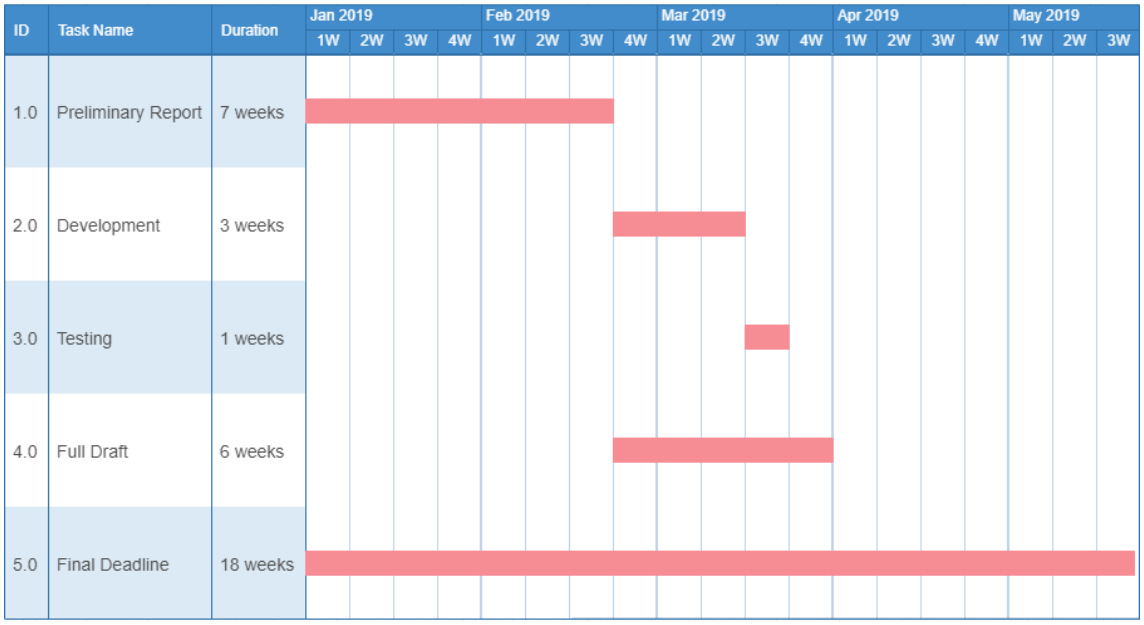

FIGURE 4.1: Gantt chart

Here you can see the gantt chart created for this project. It has a rough sketch of the bigger tasks, nothing too precise, as it is helpful enough this way to keep track of the weeks overall.

4.2 Progress Logs

The weekly progress logs act as a weekly sprint. In them I write down what I did, what I wanted to do and what I will do by next week. This helps keep track of the more short term week-by-week goals. I also find the recaps at the end of the week to be quite helpful to remind myself of what I did and what I still need to do.

4.3 Version Control

Version control is an essential part of any project. I'm using GitHub, but there are many others out there that do the same job. It helps mainly because of the ability to go back to previous versions whenever needed, for example when something goes horribly wrong.

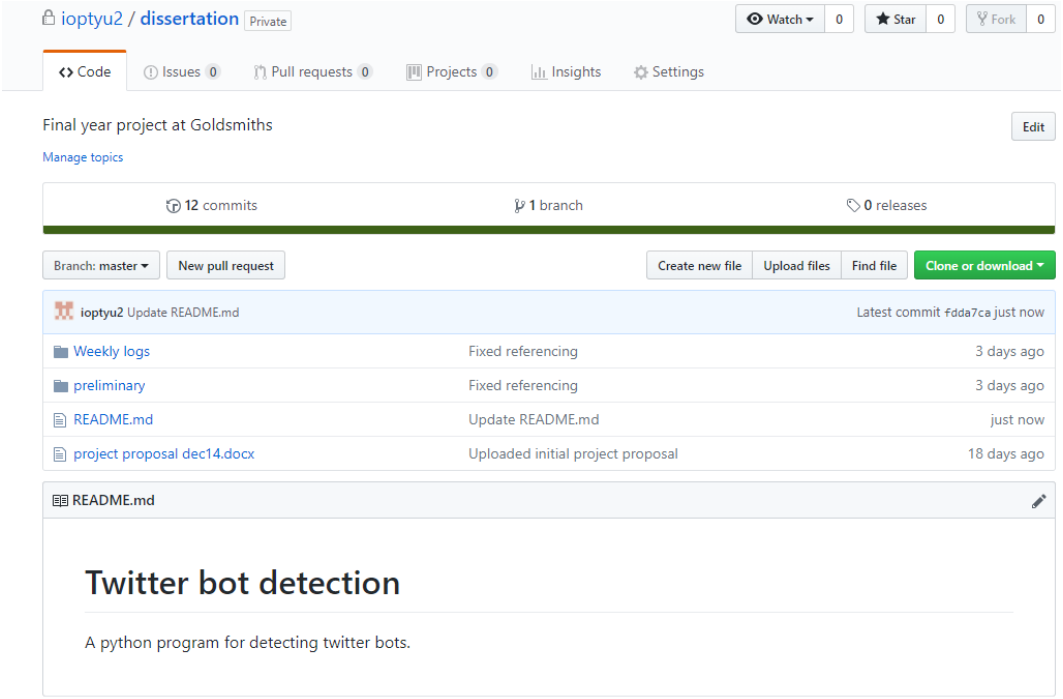

FIGURE 4.2: Github Repository

Bibliography

[1] Katie Young. *Social Media Captures Over 30 percent of Online Time*. URL: https://blog.globalwebindex.com/chart-of-the-day/social-media-captures-30-of-online-time/.

[2] Alessandro Bessi and Emilio Ferrara. "Social bots distort the 2016 U.S. Presidential election online discussion". In: *First Monday* 21.11 (2016). ISSN: 13960466. DOI: 10.5210/fm.v21i11.7090. URL: https://firstmonday.org/ojs/index.php/fm/article/view/7090.

[3] Yuriy Gorodnichenko, Tho Pham, and Oleksandr Talavera. *Social Media, Sentiment and Public Opinions: Evidence from #Brexit and #USElection*. Working Paper 24631. National Bureau of Economic Research, 2018. DOI: 10.3386/w24631. URL: http://www.nber.org/papers/w24631.

[4] Denise Clifton. *Twitter Bots Distorted the 2016 Election—Including Many Likely From Russia*. URL: https://www.motherjones.com/politics/2017/10/twitter-bots-distorted-the-2016-election-including-many-controlled-by-russia/.

[5] BBC. *US mid-terms: Twitter deletes anti-voting bots*. URL: https://www.bbc.co.uk/news/technology-46080157.

[6] The Economic Times. *Thousands of Twitter bots active during 2018 US mid-term elections*. URL: https://economictimes.indiatimes.com/news/international/world-news/thousands-of-twitter-bots-active-during-2018-us-mid-term-elections/articleshow/67850597.cms.

[7] Shehroze Farooqi and Zubair Shafiq. *Measurement and Early Detection of Third-Party Application Abuse on Twitter*. URL: http://homepage.divms.uiowa.edu/~sfarooqi/Files/Farooqi-AbusiveTwitterApplications.pdf.

[8] Andy Greenberg. *TWITTER STILL CAN'T KEEP UP WITH ITS FLOOD OF JUNK ACCOUNTS, STUDY FINDS*. URL: https://www.wired.com/story/twitter-abusive-apps-machine-learning/?mbid=social_twitter&utm_brand=wired&utm_campaign=wired&utm_medium=social&utm_social-type=owned&utm_source=twitter.

[9] Will Koehrsen. *An Implementation and Explanation of the Random Forest in Python*. URL: https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76.