The Truth about Twitter

Final Year Project Report

DT228 B.Sc. (Hons) in Computer Science

Max MacDonald

C15740661

Cindy Liu

School of Computing

TUD

11-04-2019

Abstract

The rise of social media platforms in the last few decades have led to the surge of numerous types of fake accounts across them. These platforms have struggled to keep up with the massive volume of nefarious fake accounts being created everyday due largely in part to how hard it can be to identify them as many are created to masquerade efficiently as human coupled with said platforms initial disinterest in dealing with such a problem. As pressure from their user base and governments mount though many are now actively pursuing the agenda to cleanse their platforms of this taint.

The purpose of this report is to show how data mining and machine learning algorithms can be used to deal with the issue of correctly identifying whether a Twitter account is real or a form of fake account. Machine learning models are created and used in a simple web application where a user can check any and as many accounts as the wish to see what type of account the models predict it to be. They are then able to share their results on different social media platforms and give feedback on the web application.

The report starts off by explaining the background of the project including description, objectives and roadmap and all the research that has gone into it. Next, the design and development stages are delved into in detail with the various decisions made during creation of the web application explained. Evaluation of the experiments done on the datasets to find the most accurate classifier, test and train data split technique and parameters and the various forms of testing on the website are discussed. Lastly all conclusions drawn from the report and any future work planned is reviewed.

Keywords: Data mining, Machine learning, Twitter, Fake accounts

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

[illegible]

Max MacDonald
11/04/2018

Acknowledgements

I would like to thank my supervisor, Cindy Liu, who has helped me immeasurably throughout the project, always having time to meet me and offer feedback at every stage.

To my parents, I cannot possibly express my gratitude to you for always being there for me no matter what else is going on and especially for these last four years while undertaking this undergraduate degree.

To my thanks to my friends and family, you all made it possible for me to step away from college work when it got tough, recharge and unwind and come back to my work with renewed focus and vigour.

Lastly, I would like to express my thanks to all those who took part in the testing of my application and provided such great feedback.

Table of Contents

Table of Figures

1 Introduction

1.1 Project Background

Twitter, a free social networking microblogging service, is one of today's leading digital platforms with 326 million active users worldwide in the third quarter of 2018. [1] Registered users can broadcast short posts called tweets which can be liked, retweeted and responded to by other users. Just like other social media platforms such as Facebook and Instagram, Twitter has and still is facing a massive problem with fake or bot accounts. Estimates place the percentage of bot accounts on Twitter anywhere from 9 to 15% of the total user count. [2]

1.1.1 What is a Twitter Bot Account?

A Twitter bot account is an account that is controlled by a software application, via the Twitter API, which will automatically generate and publish new tweets, follow specific users, retweet other tweets and liking specific sets of tweets all based on content or hashtags included, depending on the settings of the controlling application. [2]

These bots can perform tasks at a much higher rate than a human user can and as such push out more content or tweets in the same timeframe, some even working around the clock. Bot accounts on other platforms are similar with any differences being based on the difference in platforms.

1.1.2 Types of Bot Accounts

There are many different types of bot accounts from helpful and informative ones tweeting spiritual wellbeing tips to ones which retweet tweets that push extreme ideologies to advertising accounts which are set up to tweet content about specific brands, products or services at certain times of the day.

Some accounts are even used to boost a person's fame or influence on Twitter by following that person's account and can be bought in packages. This is a massive industry in and of itself reportedly being a $40 to $360 million-dollar business annually. [3] Major celebrities such as 50 Cent and brands like Mercedes-Benz have come under scrutiny for possibly engaging in this practice. [4]

Then there is the complexity of the software applications behind these accounts. Older and more traditional bot accounts tend to be easier to detect as they follow much simpler patterns in their activities while newer social bots need far more complex algorithms to detect as they are set up to masquerade effectively as human accounts by mimicking human behaviour better.

1.1.3 Importance of Identifying Bot Accounts

The main reason why it is so important to be able to identify, unless the account states so itself, whether an account is real or not is the erosion of trust that can occur due to the account's activities. If the account is followed by one million other accounts, even if most of them are fake themselves, and posts something that, while untrue, pushes a narrative that certain groups would be inclined to believe then that post can gain a lot of traction and spread quickly all over Twitter and beyond to other social media platforms causing untold damage.

Many individuals or groups wish to affect the perception of specific events or entities through Twitter and this ranges from boosting their own profiles through fake followers as mentioned above to trying to influence public campaigns such as the 2016 US Presidential elections. Studies have estimated that in the lead up to this election, a fifth of all Twitter traffic related to the election came from a legion of bots. [5] That much traffic would have had a massive influence on people's views and how they voted and in turn the outcome of the election.

If a bot account is masquerading as a real human then, since it is inherently trying to deceive us, it is highly unlikely much good can come of its sustained existence and as such the sooner it is detected and shutdown the better.

1.1.4 Important Characteristics of Bot accounts

When trying to identify if an account is a bot or not there are some key characteristics that can help [2, 6, 7]:

- How often per day an account tweets and if there is a regular interval between tweets can both lead to suspicions as this is a hallmark of automation.

- How anonymous that account is trying to be, does it have a profile picture and if so is it of a person? Same for the background picture. Does their bio information help identify them or add to their anonymity? Is the account handle just an alphanumeric scramble?
- Generic bio or lack of one as the programs which create these bots are not set up to make completely unique bios.
- Ratio of how many other accounts an account follows to how many follow it.

1.2 Project Description

The focus of this project is to be able to differentiate between real and bot Twitter accounts and allow the public to make use of this ability. This will require a deep understanding into what constitutes each of the various types of bot accounts, research into the areas of data mining and machine learning, learning how to work towards the required insights from a procured dataset and lastly to create and deploy a web application. There are a lot of new subjects to learn about and interesting challenges to overcome.

1.3 Project Objectives

The main goal of this project is first and foremost to create accurate mathematical models, employing various data mining and machine learning techniques and classifiers, that can tell a real Twitter account from a fake one. The secondary goal is to create a simple web application that makes use of these models to allow users to get a prediction for any Twitter account they wish.

To be able to achieve both goals, the following list of objectives must be completed:

I. Research into what makes a fake Twitter account and alternative solutions.
II. Research data mining, machine learning, the various technologies involved and Project Management and Data Mining methodologies.
III. Gather historical Twitter user account data that has both real and fake accounts.
IV. Understand, clean and prepare gathered data.

V. Design the overall project architecture.
VI. Design a database to store data and read the prepared data into it.
VII. Experiment with the data and various classifiers until an acceptable level of accuracy is achieved.
VIII. Develop the web application to make use of the results from the previous part.
IX. Deploy the web application onto the server.
X. Perform usability and ad hoc testing to gain user feedback and ensure everything works as intended.

1.4 Project Challenges

With all the main objectives set out for project set out, there will invariably be many challenges along the way in completion of these. Here are the main challenges that could be encountered:

1.4.1 Time management

Time management for this entire project will be a major challenge as the year's modules, their workloads and exams must all be juggled with this project. While everything has been planned out into a project plan and tweaked as progress is made, unexpected situations can arise within college and in outside life which can throw specific pieces of work's timelines off.

1.4.2 Data Preparation and Feature Selection

One of the longest and most important parts to any data mining project is preparing the data and then selecting the right features for use in training models. Any errors at either of these stages will lead to inaccurate predictions thereby losing users trust and interest in using the web application. The data must be explored carefully, and time invested in understanding how to prepare the data correctly and what features are important, and which are not.

1.4.3 Accuracy of models

To be able to create accurate models requires that the previous challenge be overcome and then being to understand the various machine learning classifiers and how they arrive at their conclusions. This will require a significant portion of time dedicated to research and discerning between the different classifiers.

This challenge is the most important aspect that the success of the project relies on as without an accurate model the web application holds little value to the user as it cannot make accurate predictions.

1.4.4 Deployment of Web Application

The deployment of the web application onto a server is potentially a complicated challenge due to a lack of experience in this area. Each provider of deployment servers has their own steps needed to be taken and settings altered to ensure a web application can be deployed smoothly and correctly. To overcome this challenge will very much be a process of trial and error and reading through tutorials.

1.5 Dissertation Structure

Chapter 1: Introduction

An Introduction to the project background, a description of what it entails, a list of all major objectives requiring completion and what are the main set of challenges that will be faced to achieve completion.

Chapter 2: Research

Covers all research done before and throughout the design and development of the project. Alternative solutions to the core issue and similar past projects are investigated and detailed. Various relevant technologies are analysed, and an explanation given on why specific ones were chosen. Lastly the areas of data mining and machine learning are delved into, explaining the various techniques used.

Chapter 3: Approach & Methodology

The Agile and Kanban project management methodologies are explained and shown how they are used within this project. The CRISP-DM and SEMMA data mining methodologies are gone into in detail and compared with a conclusion on why one was chosen over the other.

Chapter 4: Design & Architecture

Details the various design elements of the project including the general technical architecture and how it was adapted for use, how the data mining and web application link together and discussions on Use Cases, Entity Relationship and Class Diagrams.

Chapter 5: Development

Goes in depth into all the development done for both parts of the project, describing how accurate machine learning models were created through the various stages of the CRISP-DM methodology and the construction of a website to make use of them such that accurate predictions are returned to the user.

Chapter6: Testing and Evaluation

Explains how the separate parts of the system were tested and evaluated. Usability and Ad hoc testing for the web application and what the results of these were. For the data mining side of the project, this revolved around the evaluation of various machine learning classifiers across multiple different output combinations. Various accuracy metrics are computed and explored leading to the final model/s selection used in the web application.

Chapter 7: Project Plan

Gives both the initial and final project plan, with explanations for any changes and how various Kanban boards were used to implement and keep track of the plan.

Chapter 8: Conclusion and Future Work

Wraps up the project, elaborating on the key points learned while undertaking this project, the parts that did and didn't live up to expectation and what additional work could be done going forward.

Chapter 9: Bibliography

Details all references used throughout the document. Websites, books, academic papers and studies are all included.

2 Research

This chapter will cover all research done for this project including applications or solutions akin this project, other final year projects like this, all technologies researched for this project, research into data science and its sub topics big data, data mining and machine learning and finally the results of all this research such as chosen technologies and areas that will prove challenging.

2.1 Alternative Existing Solutions to the Problem

This section explains how Twitter deals with bot accounts and looks at a similar application to this project, Botometer.

2.1.1 Twitter

It has only been in the last few years that Twitter has taken the detection and suspension of bot accounts seriously. Brexit and the US Presidential elections were the deciding factors as the activities of bot accounts in the lead up to these proved to be a liability for the company. After an internal investigation, Twitter announced it would not be selling any more advertising to Russia media outlets Russia Today and Sputnik as these organisations were found to have interfered with the Presidential election on behalf of their government. [8]

Twitter has also been quiet active this year in detecting and shutting down bot accounts, between May and July around 70 million fake and suspicious accounts were shut down, same in October to a bot network of a few hundred accounts, that were involved in a coordinated campaign to defend Saudi Arabia's Government's role in the disappearance of Jamal Khashoggi, and most recently in November around 10 thousand more, that were all aimed at discouraging Americans to vote in the midterm elections. [9,10,11]

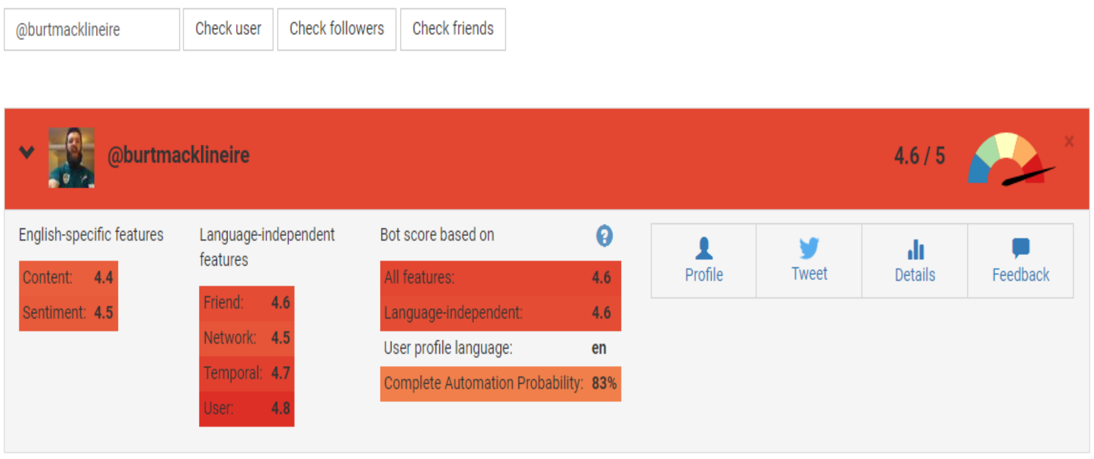
While the company has been trying, it is not an easy fight as they will always be on a reactive footing rather than a proactive one since the creation and running of bots, which are constantly evolving, can be automated but their large-scale detection relies on human intervention. This combined with the sheer volume of users and content through the site makes it a daunting and never-ending task. [12]

2.1.2 Botometer

Botometer is a joint project between Indiana University Network Science Institute (IUNI) and the Center for Complex Networks and Systems Research (CNeTS). It employs a machine learning algorithm trained to classify an account as real or bot based on a labelled dataset comprised of over 10 thousand. It uses the Twitter REST API to gather public data on an account and then passed to the Botometer API which "*extracts about 1,200 features to characterize the account's profile, friends, social network structure, temporal activity patterns, language, and sentiment*". [13] These are passed onto its models to compute the various scores which in turn go towards the overall score.

Its web front allows a user to check the activity of a Twitter account, after having the user's Twitter account authorised, and gives it a score, out of 5, based on how likely the account is to be a bot with the closer the number is to 5 the more likely it is. There is also an option to check that accounts followers and the accounts it follows as well. It is simple, easy on the eye and informative

I used my own Twitter account to test it and the results are shown below. As you can see it rates my account with a bot score of 4.6/5 and a Complete Automation Probability (CAP) of 83% which is the probability that this account is fully automated. I set my Twitter account up a few years ago, followed some people, sent out one tweet and then completely ignored it so it is not surprising that Botometer's models gave back these results even if they are wrong.


@burtmacklineire | Check user | Check followers | Check friends

@burtmacklineire 4.6 / 5

English-specific features
Content: 4.4
Sentiment: 4.5

Language-independent features
Friend: 4.6
Network: 4.5
Temporal: 4.7
User: 4.8

Bot score based on
All features: 4.6
Language-independent: 4.6
User profile language: en
Complete Automation Probability: 83%

Figure 1 Botometer GUI [13]

2.2 Existing FYPs

This section looks at existing final year projects related to this one.

2.2.1 Fantasy Premier League Predictive Analytics

This project involves the creation of a web application that allows the user to view important statistics about a football player from the Premier League and have recommendations made to them about what players to pick for their fantasy football team on a given week. Through use of this application, the user can gain an advantage over their friends and co-workers in any of the fantasy football leagues they participate in, with bragging rights or even a physical prize on the line.

Predictive data analytics was employed on historical Premier League player data using the CRISP-DM model, detailed in section 3.2.1, and combined with various machine learning algorithms, mainly linear regression, to arrive at the recommendation system for the four sections of players: goalkeeper, defender, midfielder and striker. The web application was then a combination of said recommendation system, a Flask RESTful API and AngularJS to present everything in a neat and aesthetically pleasing way to the user. [14]

This project showed that you can combine your work, in this case college-orientated, with leisure subject matter and create something greater than the sum of the two parts. This is a critical piece of information to not just know but understand and realise that the more invested one gets in what they are doing, mind and body, the greater the outcome will be, in quality, scope and self-fulfilment.

2.2.2 CrimAnalytics – A Crime Prediction Web Application

The creation of an accurate crime prediction model and subsequent use in a web application lies at the heart of this project. Users can avail of an interactive map to look up crime rates and other useful statistics within that area, based on the model and how local features are nearby. This is of great benefit to locals, as they can realise how big of a problem their area faces and can then move forward with positive action to try and reduce those rates and to tourists and those looking to move to a new area as this information can be a key factor in their decision making.

To arrive at this, as in the previous project, predictive data analytics and modelling is performed over a dataset, a government crime statistics dataset, by cleaning and preparing the data and then using R to implement the model training and application. The web application that has the results integrated into it is comprised of a front-end of HTML and JSP scripts and a back-end split into a middle layer of Java code that process all the business logic and then a final layer of the servers running docker containers of a MySQL database, an R server and an Apache Tomcat server. All this combines for a full stack architecture that has each part fully separated out and independent. [15]

This is an ambitious project that delivers on all fronts, providing in-depth analysis of the subject material, efficient and organised structure to the application and an interesting user experience. There is much to learn in the area of data analytics and this project shows that the same subject can be of interest and utilised by the general public, businesses and governments

2.2.3 Anti-Bullying with Machine Learning

This project revolves around being able to distinguish bullying content from normal text, in relation to messages and content shared on social media platforms such as Facebook and Twitter and then providing a simple web front end for any member of the public to check a piece of text to see would it be classified as bullying content. An incredibly sensitive and important subject, online bullying and trying to identify and stop it is something that always placed highly on any agenda within schools, the workplace and even government as most of the population as either been a victim of it or knows a friend who has.

Text analytics was done over various datasets, such as a Twitter dataset and a MySpace dataset both used in separate cyberbullying research. This involved various stages such as data cleaning, rebalancing and feature reduction before being used to train several different machine learning classifiers, including Naïve Bayes and Decision Trees across an array of experiments, each one tweaking the process with the aim of better classification of the data. The final models were then combined with an incredibly straightforward interface for demonstration purposes. [16]

The attention to detail in every part of the project from research and design to development and deployment and the similarities between it and the current undertaking are a welcome reminder that it is possible to see an idea through from start to finish no matter and that no matter what obstacles arise there is nearly always a solution even if that means taking a step backwards to eventually go forward.

2.3 Technologies Researched

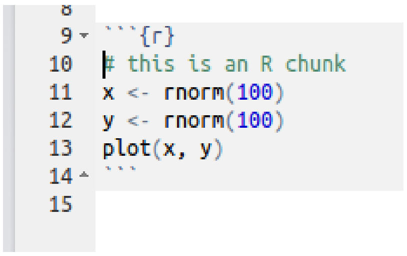
This section deals with all research into the various possible technologies that could be used in this project and their benefits and limitations.

2.3.1 Technologies for Data Mining & Machine Learning

2.3.1.1 R

R, a GNU project, is a programming language and environment for statistical computing and graphics. It is a variation on the S language and can run code from other languages such as C, C++ and Foltran. It has a wide and enthusiastic community worldwide ensuring there is plenty of support for beginners and its functionality can be extended through numerous packages found online. It has a wide, coherent and well-developed suite of facilities for data handling, storage, data analysis and graphical displays. [17]

Even with all this it does have its limitations such as memory management, R can consume all available memory, since some packages are created by normal users they might not always be up to industry standard and a basic knowledge of statistical vocabulary is needed as it was written by statisticians for statisticians.



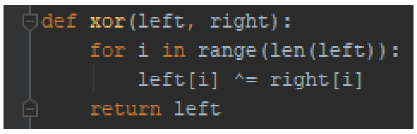
~~~
```{r}
# this is an R chunk
x <- rnorm(100)
y <- rnorm(100)
plot(x, y)
```
~~~

Figure 2 R code example

2.3.1.2 Python

Python is an interpreted, high level programming language that places a lot of emphasis on code readability. It is Open Source, friendly and easy to learn with one of the largest communities in the programming world. [18] It also has a wide variety of packages covering nearly any topic a user might need or need, entire frameworks that can be used to get a project up and running quickly and simply and is supported across multiple platforms and systems.

It does have its downsides though, due to the fact it is compiled at run time it can be quiet slow running, it is also not a good choice if mobile development is at the core of your work or if your project is a game with high-end graphics.

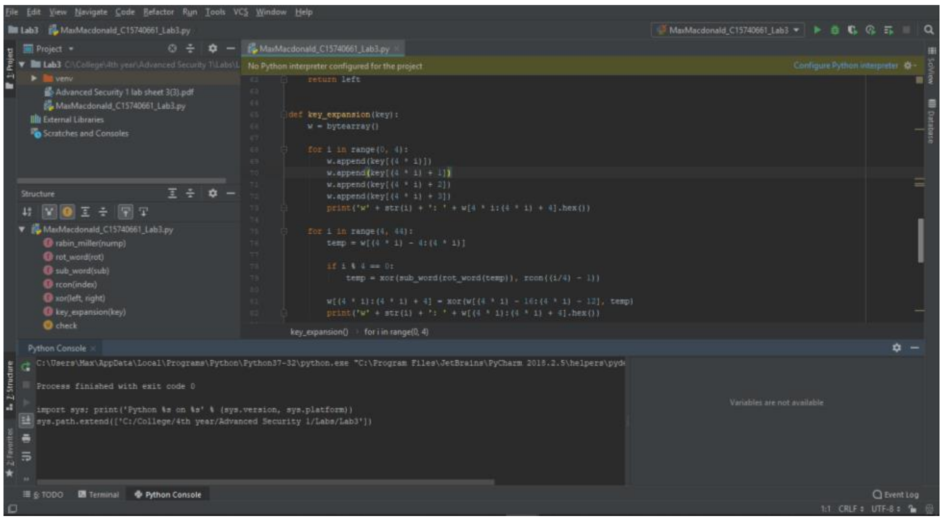


```python
def xor(left, right):
    for i in range(len(left)):
        left[i] ^= right[i]
    return left
```

Figure 3 Python code example

2.3.1.3 PyCharm

PyCharm is a Python IDE developed by JetBrains that includes intelligent code compilations, error checking and quick fixes and easy project navigation. [19] The professional version being made available to students, allowing access to many great other features such as starting a project with a framework already in place, database and SQL support and a Python Profiler.


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Lab3
MaxMacdonald_C15740661_Lab3.py
MaxMacdonald_C15740661_Lab3
Project
Lab3 C:\College\4th year\Advanced Security 1\Labs\L
venv
Advanced Security 1 lab sheet 3(3).pdf
MaxMacdonald_C15740661_Lab3.py
External Libraries
Scratches and Consoles
Structure
MaxMacdonald_C15740661_Lab3.py
rabin_miller(nump)
rot_word(rot)
sub_word(sub)
rcon(index)
xor(left, right)
key_expansion(key)
check
No Python interpreter configured for the project
Configure Python interpreter
key_expansion() › for i in range(0, 4)
Python Console
Variables are not available
6: TODO
Terminal
Python Console
Event Log
1:1 CRLF UTF-8
1: Project
Z: Structure
2: Favorites
SciView
Database

```python
    return left


def key_expansion(key):
    w = bytearray()

    for i in range(0, 4):
        w.append(key[(4 * i)])
        w.append(key[(4 * i) + 1])
        w.append(key[(4 * i) + 2])
        w.append(key[(4 * i) + 3])
        print('w' + str(i) + ': ' + w[4 * i:(4 * i) + 4].hex())

    for i in range(4, 44):
        temp = w[(4 * i) - 4:(4 * i)]

        if i % 4 == 0:
            temp = xor(sub_word(rot_word(temp)), rcon((i/4) - 1))

        w[(4 * i):(4 * i) + 4] = xor(w[(4 * i) - 16:(4 * i) - 12], temp)
        print('w' + str(i) + ': ' + w[(4 * i):(4 * i) + 4].hex())
```

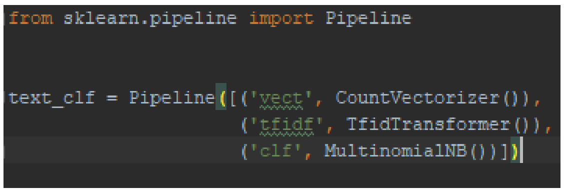
```
C:\Users\Max\AppData\Local\Programs\Python\Python37-32\python.exe "C:\Program Files\JetBrains\PyCharm 2018.2.5\helpers\pyd

Process finished with exit code 0

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:/College/4th year/Advanced Security 1/Labs/Lab3'])
```

Figure 4 PyCharm IDE

Below are a few integral Python libraries for this project if it is to be used.

2.3.1.3.1 Scikit-learn

Easily the most important and fundamental library to this project, Scikit-learn facilitates machine-learning for users of all levels by supporting various classification, regression and clustering algorithms including gradient boosting, random forests, support vector machines and k-fold cross validation. [20] It allows a user to easily create models, run them and compare their accuracy scores.

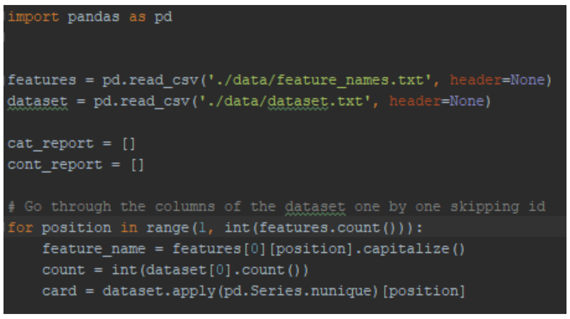


```
from sklearn.pipeline import Pipeline

text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB())])
```

Figure 5 Scikit-learn code example

2.3.1.3.2 Pandas

The Pandas library provides high-performance, accessible data structures, such as Series and Data Frames, and data analysis tools. Data Frames are two-dimensional arrays while Series are only one-dimensional, and both offer huge array of features across them such as easily sorting them, iterating through them, searching across them for a count of specific entries or gaining stats on each column like mean or mode. [21]

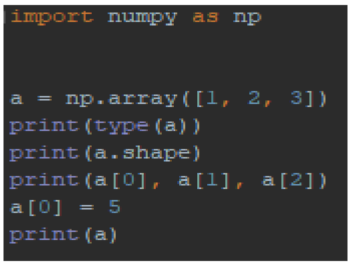


```
import pandas as pd

features = pd.read_csv('./data/feature_names.txt', header=None)
dataset = pd.read_csv('./data/dataset.txt', header=None)

cat_report = []
cont_report = []

# Go through the columns of the dataset one by one skipping id
for position in range(1, int(features.count())):
    feature_name = features[0][position].capitalize()
    count = int(dataset[0].count())
    card = dataset.apply(pd.Series.nunique)[position]
```

Figure 6 Pandas code example

2.3.1.3.3 NumPy

The NumPy library offers numerous features for scientific computation and works well with the Scikit-learn library. It offers a powerful N-dimensional array object which can be used as an efficient container of generic data, as well as several sophisticated functions and tools. [22]



```
import numpy as np

a = np.array([1, 2, 3])
print(type(a))
print(a.shape)
print(a[0], a[1], a[2])
a[0] = 5
print(a)
```

Figure 7 NumPy code example

2.3.2 Technologies for Web Application

2.3.2.1 Flask

Flask is a web micro-framework for Python that provides users with a simple and effective core of tools, libraries and technologies to build a web application while also allowing it to be easily extended. [23] This has its benefits, as it is light with little need to keep an eye out for security bugs, but also has its limitations as the user will still have to do a lot of work themselves or increase the list of dependencies within the project.

Flask
web development,
one drop at a time

Figure 8 Flask Logo [23]

2.3.2.2 Django

Django is a full web framework for Python that enables rapid deployment and elegant, practical design. It was built by skilful developers that abstracted much of the work required to get a web application of the ground such as managing views and templates, URL endpoints and security features, allowing users to focus on the nuts and bolts of their application instead. [24]

django

Figure 9 Django Logo [24]

2.3.2.3 Python to Twitter API

There are numerous Python libraries or wrappers that can connect to and gather data from the Twitter API such as Tweepy, Twython or Python Twitter. As Tweepy provides some great documentation and examples and is brilliantly supported I will be starting with that library. [25] If it does everything I need of it I will not need to use any of the other libraries.

2.3.3 Technologies for Web Server

As Django and Flask web hosting only truly fulfil the role of development servers another web hosting service must be chosen for deployment onto a production server.

2.3.3.1 Apache HTTP Server

Apache HTTP Server is a free and open-source HTTP server built to operate on numerous different operating systems such as UNIX, Windows or Mac. [26] It is developed and maintained by an open community of developers, has a strong community of users willing to help first timers and is the most widely used web server in the market today. It does have its restrictions though: a strict updating policy must be put in place and the ability to modify its configuration can potentially cause a serious threat to the security of the web application.

Figure 10 Apache HTTP Server Logo [26]

2.3.3.2 Heroku

Heroku is a cloud platform as a service (PaaS) that allows users to deploy applications onto its servers and supports several programming languages including Python. It "*makes the processes of deploying, configuring, scaling, tuning, and managing apps as simple and straightforward as possible*" enabling developers to focus on building their app. [27] With this though comes a lack of control as the exact configuration of an application is set by them and if there is a high volume of data traffic then there is a premium charged.

Figure 11 Heroku Logo [27]

2.3.3.3 Amazon Web Services

Amazon Web Services (AWS) provides on-demand cloud computing platforms to users on a paid subscription basis. [28] This means that for many users it eliminates capacity constraints while mitigating the costs involved as well as adding in global reach and scalability. It is a high-tier grade service, but you are also paying for it unlike many others. It does offer a first-year free tier of all its services for first time customers and for students and educators there is an AWS Educate account that gives credits enabling hands on experience with their services.

Figure 12 AWS Logo [28]

2.3.4 Technologies for Version Control

2.3.4.1 Git

Git is an open source distributed version control system that has seen a massive surge in popularity in recent years. It is free and easy to use and learn and can be run from the Git Bash client or from its integration in PyCharm, making it even easier to use and track changes in the process. Using either of these ways, it is simple to connect to GitHub, a web-based hosting service for Git repositories and ensure that a project is backed up with required access given to specific team members as well as giving public access to view the project and its code base. [29]

Figure 13 Git Logo

2.3.4.2 Mercurial

Mercurial is a free, distributed source control management tool that prides itself on how fast and powerful it is, it claims it can handle any project no matter the size or type. It is easy to learn and offers an instinctive interface. It is platform independent and extensible. For Mercurial, “history is permanent and sacred.” It only allows the rollback of the last pull or commit although there are extensions if more is needed. [30]

Figure 14 Mercurial Logo [30]

2.3.5 Technologies for Data Storage

2.3.5.1 MySQL

MySQL is an open source relational database management system and one of the most popular systems in the world due to how easy it is to use, its nature as a relational database and how much investment and innovation has gone into it. It allows for powerful joins as well as standard features such as triggers, stored procedures and cursors. [31]

Due to its acquisition by Oracle though there have been some negatives: It is no longer completely open-source as some modules for it are now closed-source and it is no longer community driven.

Figure 15 MySQL Logo [31]

2.3.5.2 PostgreSQL

PostgreSQL is an open source object-relational database management system with a big emphasis on extensibility and creating features which safely store and scale the most complicated data workloads. [32] It essentially is a combination of relational and NoSQL databases, giving the best of both worlds through its extensions. It is highly scalable and supports JSON

Even with this it has some drawbacks: Its documentation has been known to be spotty and its configuration can be confusing to an inexperienced eye.

Figure 16 PostgreSQL Logo [32]

2.3.5.3 MongoDB

MongoDB is a free and open-source distributed NoSQL, or document, database that is scalable and flexible. It stores data in JSON- like documents which can be of any desired structure, removing the need for schemas, as in relational databases and allows for powerful ways to access and analyse data using Ad-hoc queries, indexing and real-time aggregation. [33] What is given up for this is the lack of functions or stored procedures as well as loss of strength in terms of ACID (Atomic, Consistency, Isolations, Durability).

mongoDB®

Figure 17 MongoDB Logo [33]

2.4 Other Relevant Research Done

This section covers all other relevant research done for this project. Research into approaches and methodologies will be dealt with in their own section.

2.4.1 Big Data

Over the last few decades the amount of data we have gone from using has increased from kilobytes to megabytes to gigabytes and has now hit terabytes in the everyday home. Many businesses have gone beyond this and are now dealing in petabytes or even exabytes of data.

These large datasets, both structured and unstructured, that are beyond the scope of traditional techniques to process due to their size or complexity and are used heavily within the domain of data science are what have been termed: Big Data. [34]

Organisations the world over have been investing into this area in the last number of years as the results that can arise from proper storage and use, through data mining and data analysis projects, of Big data can lead to massive returns or scientific breakthroughs.

2.4.2 Data Mining

Data mining is the process of detecting anomalies, correlations and patterns within Big data to make predictions using a wide range of methods including various machine-learning algorithms. [35] There are various project models that can be used, although all of them are built upon the same foundation of stages

2.4.2.1 Data Acquisition

At the beginning of every data mining or analytics project, the first stage of the project is to ensure there is data to use. Where this data comes from varies from project to project as it may come from an inhouse databases or from surveys carried out with a business's customer base.

The data used is referred to as a dataset with each row in the dataset being an instance of the data and each column being a descriptive feature of the data. [35]

2.4.2.2 Data Understanding

Understanding the business logic and context behind the acquired data and having a base knowledge of the project's domain are integral parts of any data mining project as these helps to make sense of the relationships between features and enable easier selection of machine learning algorithms and improving their accuracy.

2.4.2.3 Data Preparation

This stage deals with the pre-processing of the data to ensure it's in the correct state to be used for various business purposes such as in a machine-learning algorithm or data analysis. It includes the sub-stages of data cleaning and feature selection.

2.4.2.4 Data Cleaning

Data cleaning is the process of finding and removing entries in the data that has been either entered or formatted incorrectly. [35] Without proper data cleaning, when passing the data into a model various errors can arise, leading to program failure or completely inaccurate results.

2.4.2.5 Feature Selection

In any data mining project, the aim is to produce accurate predictions as efficiently as possible. To do this, we want to minimise the number of features without affecting the accuracy. [35] Therefore, feature selection is a key component that must be carefully deliberated and decided on.

2.4.2.6 *Training & Testing*

The data is used to train and test the models using various splitting techniques and the results are saved to be used in the following stage.

2.4.2.7 *Evaluation*

Where the results from the previous stage are evaluated based on various accuracy metrics to see how certain models perform. If they have reached the required level of accuracy then they can start to be used in an application or to help make business decisions, otherwise a return to a previous stage is needed.

2.4.3 Machine Learning

Acquiring and preparing large sets of data is only part of the battle, the next major stage is to be able to detect patterns within this data and then make predictions from this. This is the core of Machine Learning, enabling us to extract significant insight from Big Data through complex, mathematical algorithms with minimal human intervention. [36] These algorithms are trained on sub-sets of the data to grow more accurate in their predictions.

2.4.3.1 *Algorithms*

Machine learning algorithms are divided into two sections

- ***Unsupervised Learning***
    - An algorithm is given a set of inputs without any outputs known. It learns by itself, through specific methods, what outputs it should prescribe to each input.
    - Uses of Unsupervised learning algorithms
        - Find hidden patterns within data
        - Face recognition software
    - Example of Unsupervised learning algorithms
        - Clustering
- ***Supervised Learning***
    - An algorithm is given a set of inputs with all outputs known. Using these it learns what outputs to prescribe to any future inputs.
    - Uses of Unsupervised learning algorithms

- Predicting Football scores based on previous years data
- Selection of advertising to be displayed to specific users

- Examples of Unsupervised learning algorithms:
    - Support Vector Machines
    - K-nearest Neighbour
    - Naïve Bayes
    - Decision Trees

- ***Examples of Both***
    - Artificial Neural Networks
    - Anomaly Detection

2.4.3.1.1 Unsupervised Learning Algorithms

2.4.3.1.1.1 Clustering

There are multiple different types of clustering algorithms such as K-Means Clustering and Hierarchal Clustering. All of them revolve around grouping the data based on each input's feature similarity.

The difference between algorithms, for example, K-Means Clustering and Hierarchal clustering is that the former separates the data points iteratively into K clusters based on the features of the data while the latter considers each data point a cluster then identifies the clusters that are closest to each other and merging them, while taking note of the hierarchal relationship between them, and so on until only one cluster remains with one large hierarchy.[35]

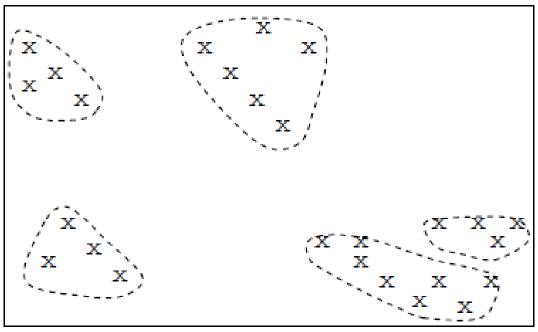


Figure 18 Clustering Feature Space Example [35]

2.4.3.1.2 Supervised Learning Algorithms

2.4.3.1.2.1 Naïve Bayes

Naïve Bayes classifiers belong to the family of probability-based classifiers and are based on Bayes' theorem with the added assumption of conditional independence between all the features in the data. [36] This added assumption allows for the model to drastically reduce the amount of probabilities it must compute.

While this is quite a leap of faith to make, it still results in a robust model that delivers strong results and, when coupled with its scalability, efficiency and simplicity, is the reason it is normally the starting point for most data mining projects.

$$\mathbb{M}(\mathbf{q}) = \underset{l \in levels(t)}{\arg\max} \left(\left(\prod_{i=1}^{m} P(\mathbf{q}[i] \mid t = l) \right) \times P(t = l) \right)$$

Figure 19 Naive Bayes Model [36]

2.4.3.1.2.2 Support Vector Machines

Support Vector Machine classifiers belong to the family of error-based classifiers. It maps all the data as points in an N-dimension space, N being the number of features, and then tries to find a hyperplane, or decision boundary, that distinctly classifies the data points. [36]

It tries to maximise the distance between the hyperplane and data points from both classes. Those points closest to the hyperplane are called support vectors and have a significant impact on its placement.

New points are mapped to this space and classified depending which side of the hyperplane they belong to. It has a high degree of accuracy, takes up less computation power than other algorithms and can be used for both regression and classification task.

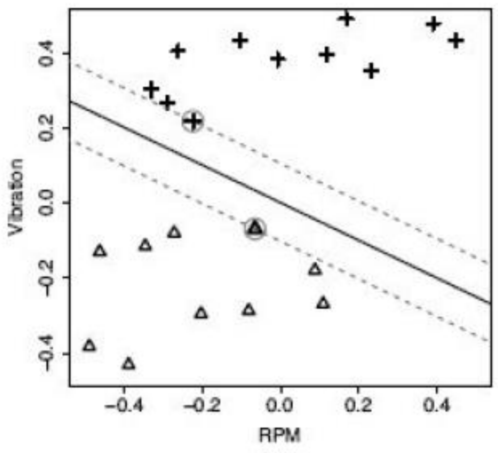


Figure 20 SVM Hyperplane selection [36]

2.4.3.1.2.3 K-Nearest Neighbour

K-Nearest Neighbour (KNN) is a simple, non- parametric classifier and belongs to the family of instance-based classifiers and as such has either no or a very small training phase to it.

To classify new data points, the feature similarity of its k-nearest neighbours is used, with the new data point going to the class with the majority count. [35] While it makes no assumptions about data and is versatile it can be computationally expensive and sensitive to irrelevant data.

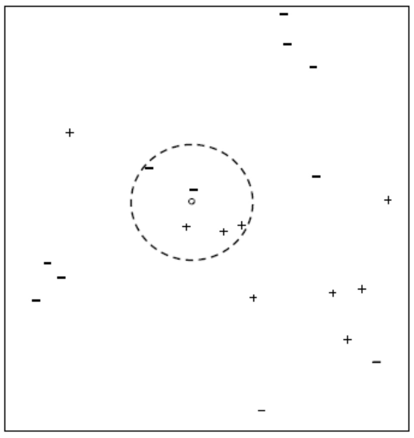


Figure 21 KNN Feature Space Example [35]

2.4.3.1.2.4 Decision Trees

Decision Tree classifier belongs to the family of information-based classifiers that takes a dataset and one feature at a time, branches out for all possible values if categorical and uses thresholds for continuous, until each branch can be labelled with only one classification. When trying to classify new instances, the tree is followed down to a branch based on its features the path those lead down the tree. [35]

The order in which the features are split upon depends on which metric is used, either the Information Gain, based on the entropy or measure of disorder of the feature, or the Gini Index, which is the measure of impurity of the dataset in regards that feature, with both trying to make the most efficient and smallest tree possible.

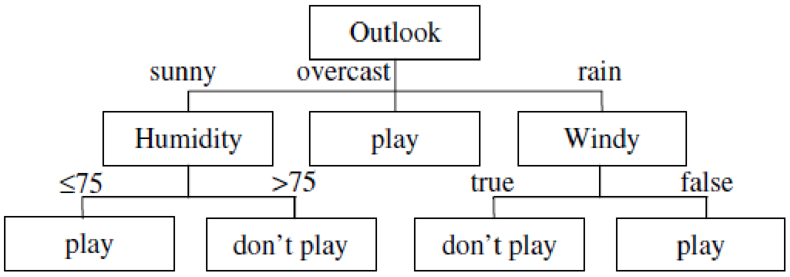


Figure 22 Decision Tree Example [35]

2.4.3.1.3 Both Learning Type Algorithms

2.4.3.1.3.1 Artificial Neural Networks

Artificial Neural Networks are "*biologically inspired computer programs designed to simulate the way in which the human brain processes information*". Using artificial neurons, the computerized version of a brain cell, a network is formed by connecting the output of specific neurons to the input of other neurons, forming a directed, weighted graph. A neurons weights and activation functions can be tuned over the learning process to increase the networks accuracy. [37]

Neural networks can be used with datasets where either both the inputs and outputs are provided or only the inputs are provided allowing for use in a wider range of scenarios. [38]

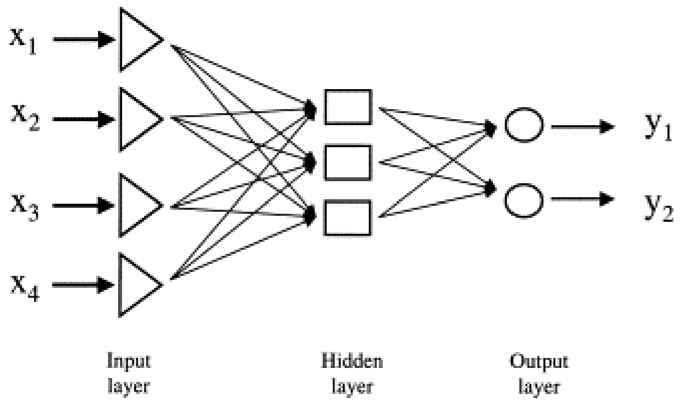


Figure 23 ANN Structure Example [37]

2.4.3.1.3.2 Anomaly Detection

Anomaly detection revolves around identifying outliers, those instances which display odd behaviour and can from unconventional patterns, within the dataset. These can be point anomalies, where a single instance has some features too far out of range when compared to other instances, or collective anomalies, where the behaviour of a group of instances helps to detect such anomalies. It is like noise removal but where that wants to clear noisy data from the dataset, anomaly detection wants to keep such abnormal data in and learn from it. [39]

Different techniques can be utilised with anomaly detection such as density-based which works like K-nearest neighbour detailed above and clustering-based, again like a previously mentioned algorithm: Clustering. Anomaly detection algorithms can be used with input and output pairs where it is known what the range of normal behaviour is to find ones that fall outside this or with just inputs and allow the algorithm to figure out what constitutes normal behaviour and raise a flag on instances that do not match this.

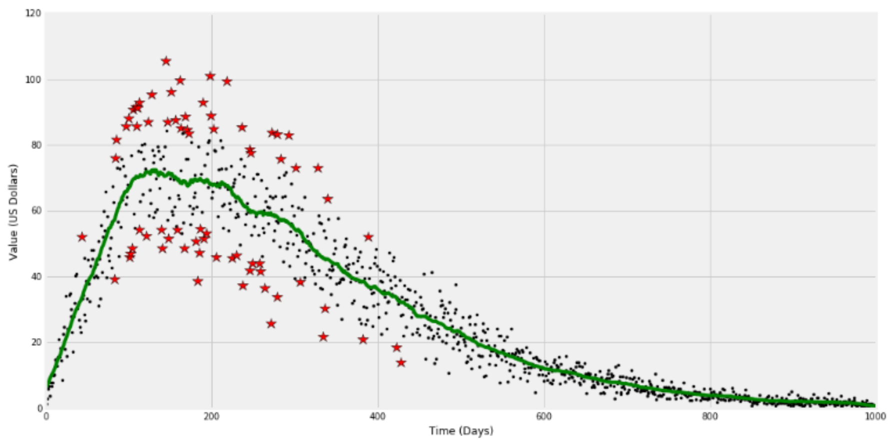


Figure 24 Anomaly Detection in Stock Value Example [39]

2.4.3.2 Training & Testing

Once a dataset has been fully prepared for use in a machine learning algorithm it must be divided up into training and testing datasets. The training dataset is a sub set of the original dataset used to train the model while the testing dataset is what is left. [35]

The training dataset is passed through a working model, with the results compared against the actual outcomes enabling the accuracy of the model to be measured. There are various ways that the base dataset can be divided up into training and testing datasets.

2.4.3.2.1 Holdout

This is the most basic division of the original set into training and testing sets with the partitioning of the original into two mutually exclusive sets. The split is usually taking a 2:1 ratio. The main problem with this method is that as more training data is used there is fewer testing data to be used. Ideally you want both the training and testing sets to be as large as possible. [36]

2.4.3.2.2 K-fold Cross Validation

Using this method, the original set is divided up into K partitions of equal size. Then for each partition, that partition acts as the testing set with the remaining partitions becoming the training set. A model is fitted using this training set and evaluated using the testing set. The model is discarded with the results being held onto before moving onto the next partition. [35]

This method deals with the main issue of the Holdout method, ensuring the entire dataset is used for both training, each partition being used K-1 times, and testing, each partition used once, with the results being of significant use at the end for evaluation.

2.4.3.3 Evaluation

After a model has been created and data run through it, results will have been produced. The accuracy of these results must be measured carefully. Only by truly understanding the accuracy of the results and what influenced it will someone be able to improve the model and its accuracy.

2.4.3.3.1 Classification Accuracy

Simply put, the accuracy of a model is the amount of predictions it got right. Put into formulaic terms: Accuracy = No. of correct predictions / Total no. of predictions. [36] This in and of itself is not enough in terms of detail for a proper model evaluation and as such other methods must also be employed.

2.4.3.3.2 Confusion Matrix

A confusion matrix is a table layout for the visualisation of the performance of a model. The totals of correct and incorrect predictions are calculated and broken down by class. These values are placed into a matrix with predicted across the top and expected down the side.

When looking at a two-class instance or one class against all the others this matrix will then hold the values for True Positives, False positives in the 1st row and False Negatives and True Negatives in the 2nd row. [36] This data holds much more meaning then the previous method of evaluation and can help in knowing what part of the model needs to be tuned to gain a better accuracy level.

2.4.3.3.3 F1 Score

The F1 score is another method of measuring a model's accuracy. It is obtained by computing the weighted average of the Recall and Precision. [36] The closer this score is to 1 the more accurate the model is.

$$F_1\ measure = 2 \times \frac{(precision \times recall)}{(precision + recall)}$$

Figure 25 F1 Measure Formula [36]

Recall is found by dividing the total correct predictions by the sum of the total correct predictions and false negatives.

$$recall = \frac{TP}{(TP + FN)}$$

Figure 26 Recall Formula [36]

Precision is found by dividing the total correct predictions by the sum of the total correct predictions and false positives.

$$precision = \frac{TP}{(TP + FP)}$$

Figure 27 Precision Formula [36]

2.4.3.3.3.1 Average Class Accuracy (Harmonic Mean)

The Average Class Accuracy (HM) is an accuracy metric that many people favour over the previously mentioned. Compared to the arithmetic mean version, it places importance on smaller values and this allows it to give a more realistic measure of how well models are performing. [36]

$$average\ class\ accuracy_{HM} = \frac{1}{\frac{1}{|levels(t)|} \sum_{l \in levels(t)} \frac{1}{recall_l}}$$

Figure 28 Average Class Accuracy (HM) Formula

2.5 Resultant Findings and Requirements

This section will deal with what technologies were chosen to use in this project and why and what datasets will be used to create my machine-learning models.

2.5.1 Chosen Technologies

Python will be used over R for this project due to its familiarity, design tropes and high extensibility with a wide variety of libraries supporting every aspect of what is need in this project. PyCharm Professional IDE was chosen due to its wide variety of features including integrated Git support connecting to a GitHub repository with Git being the preferred VCS due to it being well supported, easily operated and widely recognised.

Django will be the framework of choice as it just provides more structure out of the box then Flask does. Amazon Web Services will be used for hosting the production server as the first-year free tier completely nullifies the costs and offers a great range of services, although if this project was to be extended beyond final year there would be more consideration to swapping to Apache HTTP Server due to fiscal reasons. For data storage PostgreSQL was chosen as it combines the best of both a relational and NoSQL databases.

2.5.2 Chosen Datasets

For the data that my machine-learning models will use, the cresci-2017 dataset was selected. [40] This dataset has been used in academic studies in the field of Twitter bot detection, [7] is part of the datasets used by the Botometer application and covers an excellent range of different accounts.

It is split further into several smaller datasets. First there is a dataset of genuine account, then there are three groups of traditional spambots, the first group are general spambots without any focus, second group are spambots attempting to promote a web URL to try and get users to click it and lastly a group of spambots attempting to push job offers on users as well as getting them to click a specific URL.

Next is a group of fake follower bots which exist purely to make a user appear more popular or influential on the platform. Lastly are three groups of social spambots, the first group retweeted a specific political candidate in Italy, the second group were attempting to get people to download a specific mobile application and the last group were trying to sell products on Amazon.com.

3 Approach & Methodology

This project will be performed iteratively, using the Kanban sub-discipline of the Agile methodology for the overall scope of the project, including the web application, while for the data mining aspect of the project the CRISP-DM methodology was chosen from a few data mining project management models.

Both the Agile methodology and its sub-discipline Kanban and their use in this project as well as the CRISP-DM methodology and a similar methodology: SEMMA, will be explained in this section. A comparison will be made between them and reasons given why I chose the former over the latter.

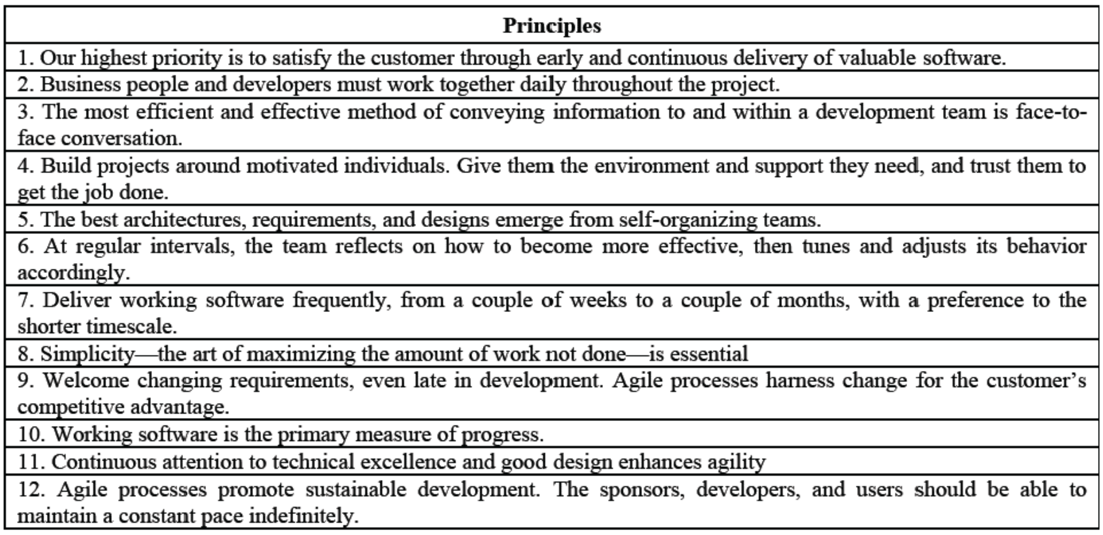
3.1 Agile & Kanban

3.1.1 Agile

The Agile concept is an approach to project management within the domain of software development and includes various principles such as Feature Driven Design, Scrums, XP and Kanban, the last of which I have incorporated into my project and discuss after this.

Agile and its sub-principles revolve around being adaptable, collaborative and versatile and with the focus on iterative and incremental development. [41] Projects that adopt Agile need an approach that facilitates rapid and flexible responses to change as well as continuous improvement.

The 12 Agile principles outlined in the Agile manifesto are:



Principles
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Business people and developers must work together daily throughout the project.
3. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
4. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
5. The best architectures, requirements, and designs emerge from self-organizing teams.
6. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
7. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
8. Simplicity—the art of maximizing the amount of work not done—is essential
9. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
10. Working software is the primary measure of progress.
11. Continuous attention to technical excellence and good design enhances agility
12. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Figure 29 12 Agile Principles [42]

Some of the key differences between Agile and traditional methodologies are that the management style of a project is to lead and collaborate in the former and command and control in the latter, that communication throughout the project is informal vs formal, that the developmental model is an evolutionary-delivery model vs a life-cycle model and that implementation is the focus vs spending large quantities of time on design. [41]

3.1.2 Kanban

Kanban is a part of the Agile family with a heavy focus on continuous delivery while at the same time ensuring that the development team do not become overburdened. [43]

The Kanban methodology was named in 2007 after several presentations given by David Anderson of his management approach at various companies. The word "*kanban*" is the Japanese for sign or signal card and the methodology's roots date back to the middle of the 20th century in Japan where Taiichi Ohno, while working for Toyota, employed the first Kanban system to regulate the workflow in the company. [44]

Kanban has three main principles behind it: [43]

- Visualise the flow of work: Set up an environment, either through post its on a board or digitally to visualise all the work items to give them context.
- Limit the work in progress: Limit each team member to at most 3 pieces of work at any given time to ensure the team does not start and commit to too much work.
- Enhance the flow: Once a team member has finished a piece of work, they take the highest priority work piece in the backlog.

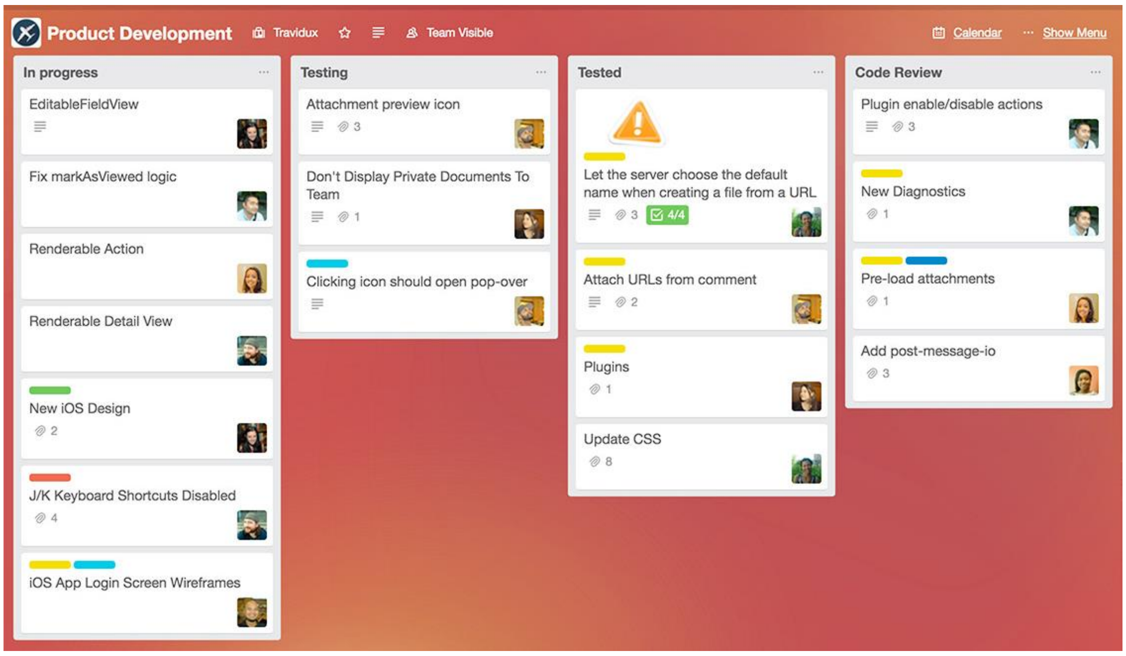
Some of the benefits of the Kanban methodology are shorter cycle times ensuring that new features can be delivered quicker, easy adaption to frequent changes in work piece priority and requires less oversight ensuring team and project leads have more time to focus on other activities.

3.1.3 Project Use

While Agile and Kanban methodologies revolve around a team of multiple members working together it isn't hard to adapt them to this projects case of a team of one for this project. All roles will be taken on including all members of the team, the team lead and the shareholders that commission the project.

- Shareholders: All the requirements of this project were laid out by at its outset and priorities will continue to be assigned to all the deliverables and smaller pieces of work by the shareholders.
- Team lead & members: Every piece of work on this project has been and will be worked on and completed by the single team member. That member will oversee everything ensuring all the work is done in a timely manner and to a high quality.

Trello will be used, a free to use web-based project management application, to keep track of and manage the work load through the Kanban method. This was a piece of software that was introduced to during a previous piece of work and is extremely simple and easy to use. [45]



In progress
- EditableFieldView
- Fix markAsViewed logic
- Renderable Action
- Renderable Detail View
- New iOS Design 2
- J/K Keyboard Shortcuts Disabled 4
- iOS App Login Screen Wireframes

Testing
- Attachment preview icon 3
- Don't Display Private Documents To Team 1
- Clicking icon should open pop-over

Tested
- Let the server choose the default name when creating a file from a URL 3 4/4
- Attach URLs from comment 2
- Plugins 1
- Update CSS 8

Code Review
- Plugin enable/disable actions 3
- New Diagnostics 1
- Pre-load attachments 1
- Add post-message-io 3

Figure 30 Sample Trello Board [45]

3.2 Data Mining Project Management Models

3.2.1 CRISP-DM

CRISP-DM model is an acronym that stands for Cross-Industry Standard Process for Data Modelling and is a cycle which consists of 6 stages. The sequence of these stages is not strict which allows for movement between any stage if so required. [46]

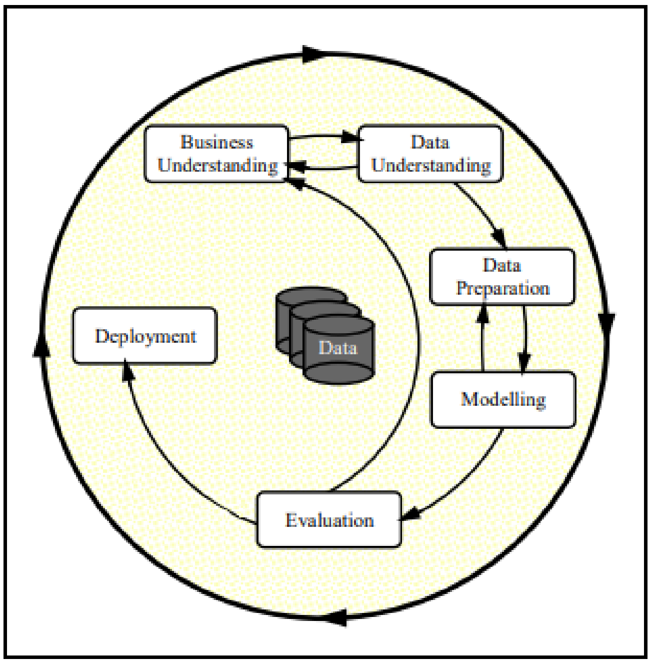


Figure 31 CRISP-DM Model [46]

3.2.1.1 Business Understanding

This stage revolves around understanding the business side of the data mining project:

- What is the domain of the business?
- Assessing the situation in terms of hardware, software, data sources and knowledge bases.
- Transforming business goals into data mining objectives.
- Producing a project plan.

3.2.1.2 Data Understanding

This stage is where data is acquired, and time spent exploring it, trying to understand any correlations between features as well as finding any data quality issues present in the data through a data quality report and data visualisations.

3.2.1.3 Data Preparation

This is the most important stage of the model and requires an excellent understanding of the previous two stages. This is where numerous tasks are performed on the data such as the pre-processing of data, cleaning and reformatting data to remove missing or fix corrupt values and feature selection. Aggregation or merging of data may also occur here. [46]

If the data is not prepared properly then the accuracy of any models it is passed to will suffer greatly and can take multiple iterations to get right.

3.2.1.4 Modelling

This stage revolves around the selection of modelling technique or machine-learning algorithms, decide how to measure the model's validity or accuracy, the building of the model and its assessment to fix any mistakes that can occur in the building process. [46]

These mistakes arise due to *noise*: outliers or missing values in the data that have managed to slip through the previous stage. This then leads to *overfitting*, where the model is too complex for the data and therefore is highly sensitive to noise, or *underfitting*, where the model is too simplistic and unable to detect patterns within the data. Either of these have a negative impact on the accuracy. [35]

3.2.1.5 Evaluation

Evaluation of results from the models, once data has been passed through, is done here. This allows us to get an understanding of the suitability of the models as well as any errors we may have made at an earlier stage. The entire process is reviewed and the next iteration, if needed, is planned.

3.2.1.6 Deployment

Once a model has been created that hits a satisfactory accuracy level it can be deployed. This can have a variety of different meanings, dependent on the project itself, from being used in a work report or a scientific paper or in a data mining application. The deployment must be planned with any monitoring and maintenance considered.

3.2.2 SEMMA

SEMMA is another model, developed by the SAS Institute, which is used to manage a data mining application and is an acronym for the 5 stages that comprise the model: [47]

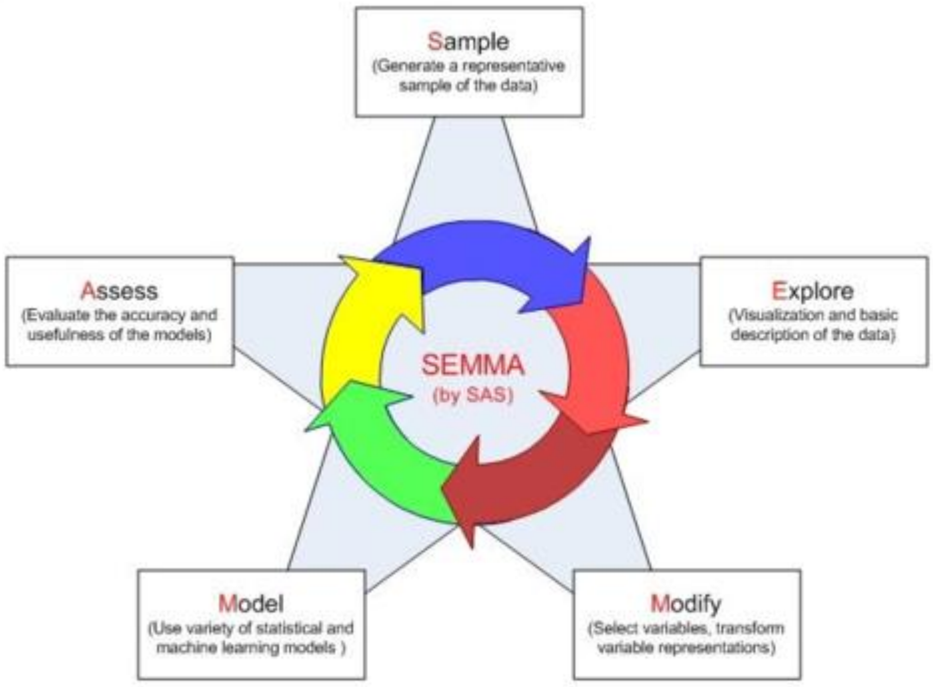

Sample
(Generate a representative sample of the data)
Explore
(Visualization and basic description of the data)
Modify
(Select variables, transform variable representations)
Model
(Use variety of statistical and machine learning models)
Assess
(Evaluate the accuracy and usefulness of the models)
SEMMA
(by SAS)

Figure 32 SEMMA Model [48]

3.2.2.1 Sample

This stage focuses on taking a sample of the dataset for use in the model. The sample must contain enough data such that accurate patterns can be drawn from it while small enough that it can still be used efficiently. Data partition of the sampled dataset into training and testing sets is also done here.

3.2.2.2 Explore

Exploration of the data is done here to detect any unexpected patterns, anomalies such as missing or corrupted data, or instances in the data that prove to be unnecessary while also gaining a better understanding of the data. This is done through visual representations and statistical techniques. [47]

3.2.2.3 Modify

The data is modified by creating, deleting, selecting and/or transforming variables within it to ensure the data being passed into the model is of high quality and considers any issues that arose in the exploration stage. Not every issue will be perfectly solvable, and it will be a case of applying the best fit solution. [47]

3.2.2.4 Model

The creation of the model is dealt with here, where selected modelling techniques are employed to build a model that will accurately make predictions based upon the data that is passed to it.

Again, like in CRISP-DM, the problems of *overfitting* and *underfitting* due to *noise* must potentially be dealt with.

3.2.2.5 Assess

The outputs of the model, after inputting the training set, that was set aside during the sampling stage, are compared to the actual outcomes of that dataset with the model's accuracy and usefulness being evaluated.

3.2.3 Difference in Models

Looking at both models it is easy to see that they are quite similar, in fact SEMMA appears to be akin to a slimmed down version of CRISP-DM with the first, Business Understanding, and last stage, Deployment, omitted such that all the focus is on data modelling aspect of the application, though realistically some knowledge of the business domain must be known or else the final product will be lacking in various areas.

This table clearly demonstrates this comparison:

CRISP-DM	**SEMMA**
Business Understanding	-----------
Data Understanding (Part 1)	Sample
Data Understanding (Part 2)	Explore
Data Preparation	Modify
Modelling	Model
Evaluation	Assess
Deployment	-----------

Figure 33 Model Comparison

3.2.4 Conclusion

The CRISP-DM model was chosen for use within this project as it is an industry standard model that is used widely and as such is well proven, is robust and versatile allowing for movement between any stage if needed and finally inclusion of the deployment stage, which SEMMA is missing, as the data mining aspect of the project will need to be integrated and deployed within the overall web application.

4 Design

This chapter details all the design elements of the project including the technical architecture chosen for this project and how it has been adapted from the Model View Controller design pattern and all other design documents such as the Use Case, Entity Relationship, Flow Chart and Source Code layout diagrams.

4.1 Technical Architecture

4.1.1 Model View Controller

The Model View Controller (MVC) architecture is used across a wide range of applications where there is a need to provide a User Interface through a desktop or web front-end. It is a three-tier architecture which uses the Controller, comprised of several classes such as a Command Factory class and Command, Service, and DAO classes, to pass information between the View, i.e. the front end, and the Model, i.e. the backend. [48]

This ensures the separation of roles between the different sections of code in a project. This makes it easier to divide up the work in a project as team members can focus on specific sections without worrying too much about the other parts enabling better development and testing.

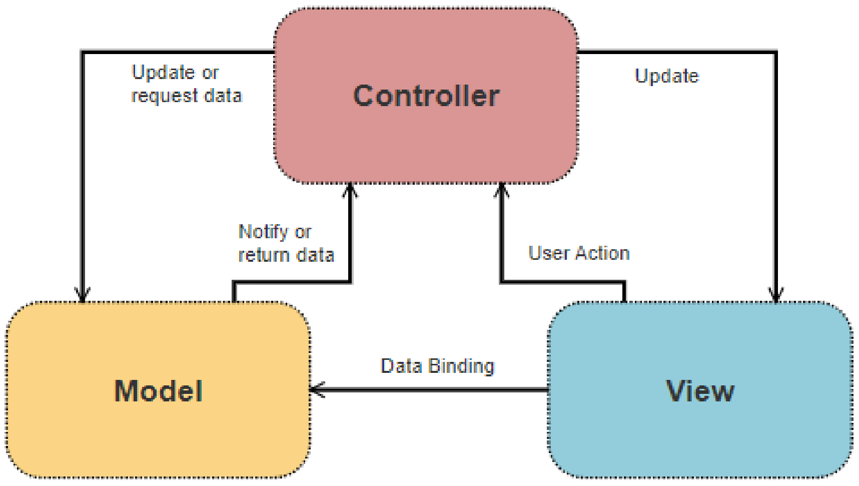


Figure 34 Model View Controller

4.1.2 Model View Template

The Django framework has been chosen for the web application aspect of this project which uses its own modified version of the MVC called the Model

View Template (MVT). In this adaption, Django takes care of the Controller role and replaces it with the Template section, which takes the role of the presentation layer by containing all the HTML and Static, such as CSS and image, files while the View section deals with all business logic and handles all requests from and responses to the User. The Model section stays the same and deals with everything to do with the database. [49]

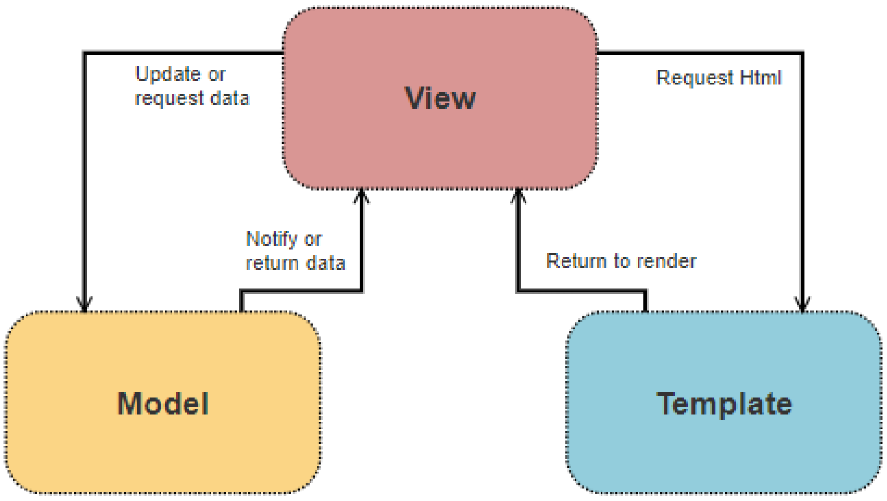


Figure 35 Model View Template

4.1.3 Project Technical Architecture

The layout of the project technical architecture can be seen in Figure 32 and shows how each of the different parts of the project interact with each other. The model, or database, part of the Django MVT has been replaced with the prediction modals and act as the only link between the web application, which make use of them, and the data mining application, which creates and saves them to the web application.

Only the data mining app interacts with the PostgreSQL database, both of which are held locally with the web application being hosted on AWS. While the project has three tiers, the web application has a simple client-server architecture. Any change to the models requires only that the web application be quickly and seamlessly redeployed, meaning only minor disruption to the user experience, as those are the only changed elements.

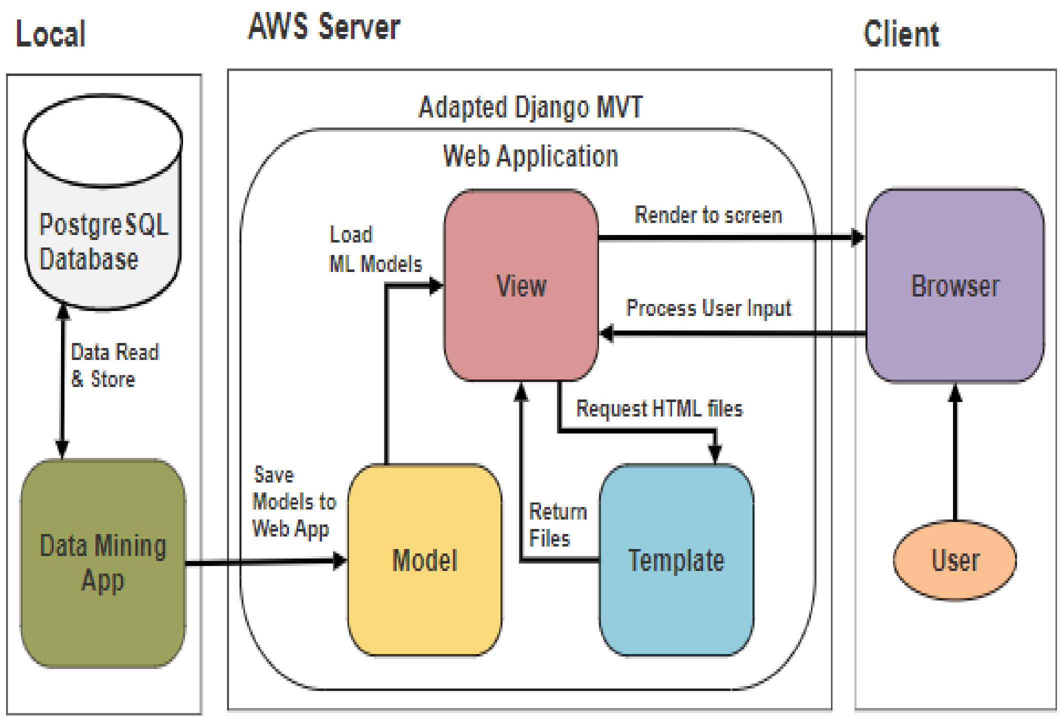


Figure 36 Project Technical Architecture Diagram

4.2 Other Design Documents

4.2.1 Use-Case

The Use-case in Figure 33 below details how a user will interact with the system. The user can enter in their own Twitter handle, if they have one, or any other one they wish, be it a celebrity's, one of their friends or any other handle they wish to check. There is no need for any form of validation from the user due to the use of the OAuth2 two-legged authentication protocol.

They will then be able to view the prediction results about what type of real or fake account it is. They can then share this result to Twitter and/or Facebook, fill out a feedback form about the site and their experiences and go back to try another account. Lastly at the main screen they will also be able to view information about the site.

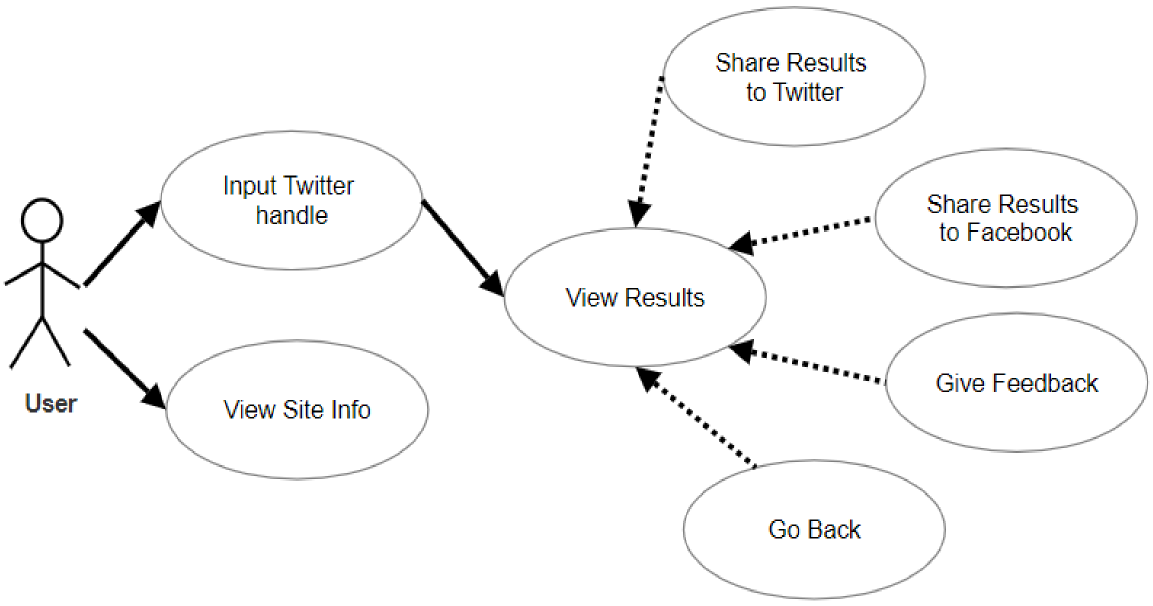


Figure 37 User Use Case Diagram

4.2.2 Flow-Chart

The Flow-chart in Figure 34 below shows the entire process from the user entering in a Twitter handle to all the possible outcomes from this such as not being able to find data on that handle due to it not existing or predicting that handle's account is a bot account.

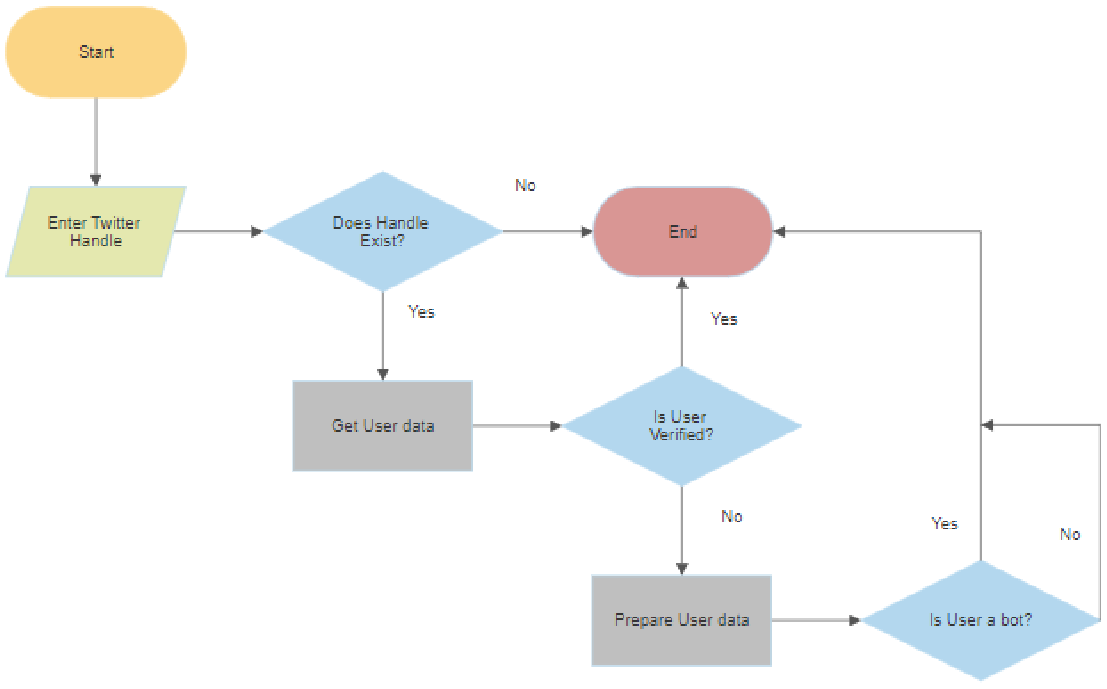


Figure 38 Prediction Flow Chart Diagram

4.2.3 Entity Relationship

All the attributes for the Account table can be seen in Figure 35 below. This table is stored within the local PostgreSQL database as part of the Data Mining aspect of the project. Data is read in from csv files into this table and is detailed later.

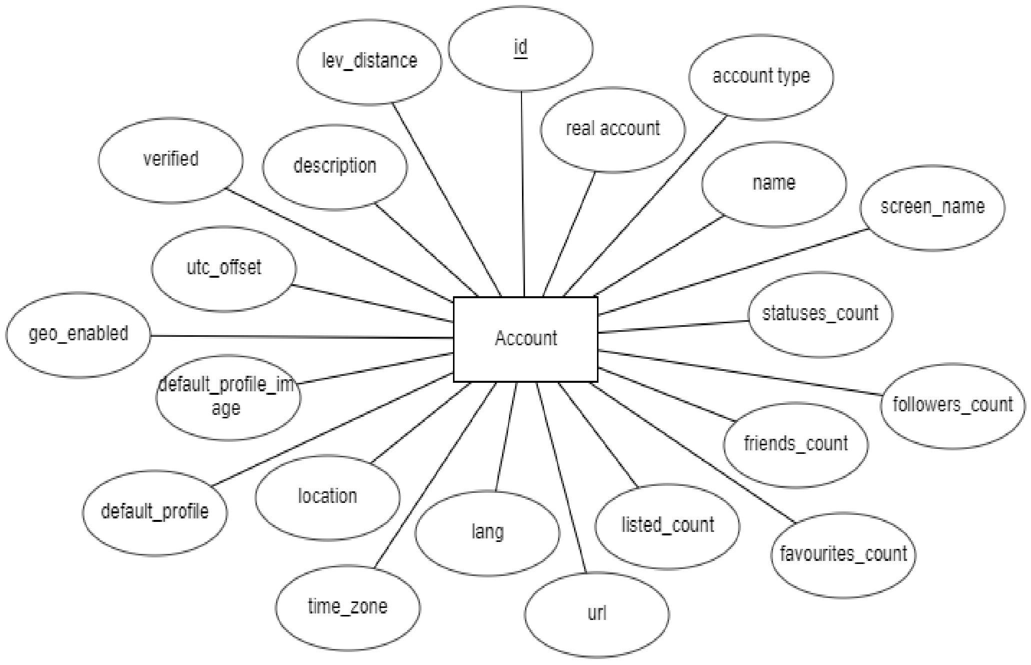


Figure 39 Entity Relationship Diagram

Attributes in Account Table:

id: big integer, unique identifier for each user account within the database.

real_account: boolean, whether this account is denoted real or fake.

account_type: integer, type of account, 1 for real, 2 for fake follower, 3 for traditional spambot and 4 for social spambot.

name: 60-character varying, the accounts display name.

screen_name: 16-character varying, accounts handle.

statuses_count: integer, total amount of statuses this account has posted.

followers_count: integer, total amount of other accounts that follow this account.

friends_count: integer, total amount of accounts this account follows, otherwise known as friends

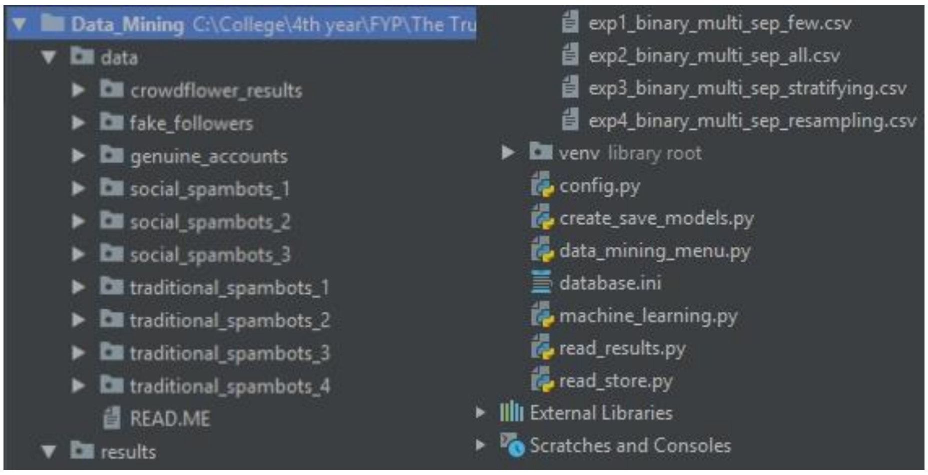
favourites_count: integer, total amount of statuses this account has favourited.

listed_count: integer, total amount of lists this account has been put into by other accounts.
url: 100-character varing, URL for the accounts Twitter page.
lang: 25-character varying, the language the user has chosen for their account.
time_zone: 40-character varying, the time zone the account operates in.
location: 70-character varying, location the user has set for the account.
default_profile: integer, whether the account uses the default profile.
default_profile_image: integer, whether the account uses the default profile image.
geo_enabled: integer, whether geolocation is turned on or not for the account.
utc_offset: integer, difference in time between UTC time and the time of the account's location.
verified: integer, whether this account has been verified by Twitter to be who they say they are.
description: 320-character varying, a public summary about the account.
lev_distance: integer, the minimum Levenshtein distance between tweets by the account, which is the lowest number of single-character edits to change one string to another.

4.2.4 Source Code Layout

4.2.4.1 Data Mining Application

The data mining application consists of a data folder for the dataset that is split up between 9 sub-folders containing user and tweet csv files, a results folder for storing the results of the various experiments done with the data, different classifiers and different account type comparisons and then several python files to process each of the tasks required.

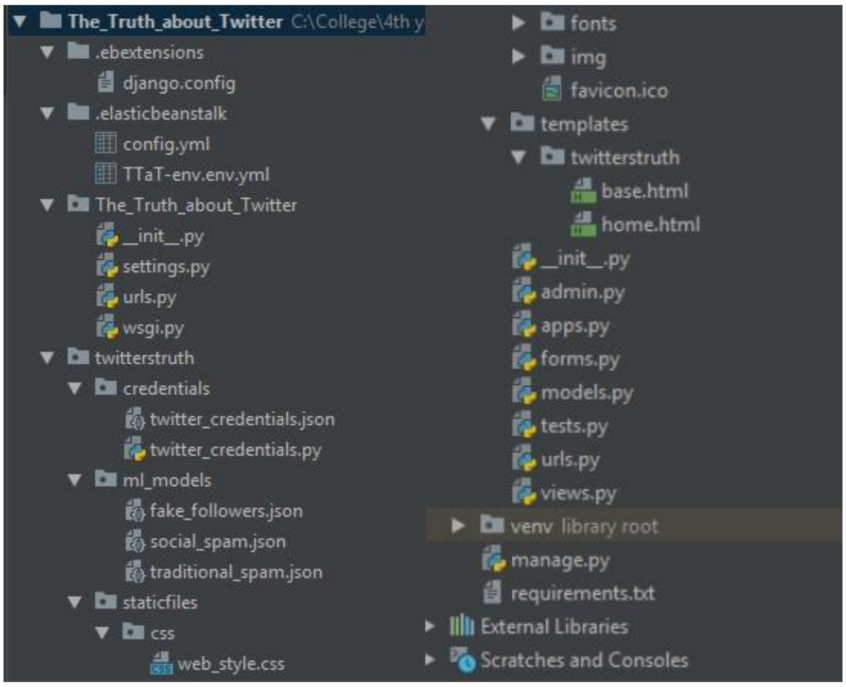


```
▼ Data_Mining  C:\College\4th year\FYP\The Tru
  ▼ data
    ▶ crowdflower_results
    ▶ fake_followers
    ▶ genuine_accounts
    ▶ social_spambots_1
    ▶ social_spambots_2
    ▶ social_spambots_3
    ▶ traditional_spambots_1
    ▶ traditional_spambots_2
    ▶ traditional_spambots_3
    ▶ traditional_spambots_4
      READ.ME
  ▼ results
      exp1_binary_multi_sep_few.csv
      exp2_binary_multi_sep_all.csv
      exp3_binary_multi_sep_stratifying.csv
      exp4_binary_multi_sep_resampling.csv
  ▶ venv  library root
    config.py
    create_save_models.py
    data_mining_menu.py
    database.ini
    machine_learning.py
    read_results.py
    read_store.py
▶ External Libraries
▶ Scratches and Consoles
```

Figure 40 Data Mining Code Layout

4.2.4.2 Web Application

The web application uses the Django framework and builds upon it by adding in both base and home HTML files for rending to screen, staticfiles folder for all CSS, font and image files, credentials folder to save all required keys to make calls to the Twitter API, ml_folders for the machine learning models from the data mining application to be saved to and the requirements.txt and .ebextensions and .elasticbeanstalk folders to facilitate the applications deployment to an AWS Elastic Beanstalk instance.



```
▼ The_Truth_about_Twitter  C:\College\4th y
  ▼ .ebextensions
      django.config
  ▼ .elasticbeanstalk
      config.yml
      TTaT-env.env.yml
  ▼ The_Truth_about_Twitter
      __init__.py
      settings.py
      urls.py
      wsgi.py
  ▼ twitterstruth
    ▼ credentials
        twitter_credentials.json
        twitter_credentials.py
    ▼ ml_models
        fake_followers.json
        social_spam.json
        traditional_spam.json
    ▼ staticfiles
      ▼ css
          web_style.css
      ▶ fonts
      ▶ img
        favicon.ico
    ▼ templates
      ▼ twitterstruth
          base.html
          home.html
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      urls.py
      views.py
  ▶ venv  library root
    manage.py
    requirements.txt
▶ External Libraries
▶ Scratches and Consoles
```

Figure 41 Web Application Code Layout

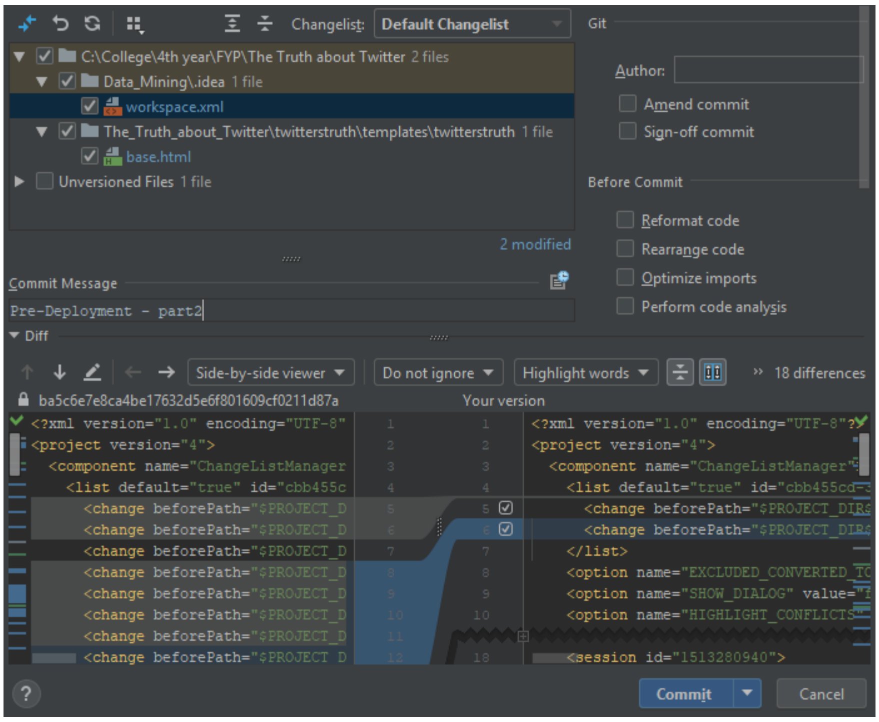
5 Development

This chapter covers how version control was employed, all the development undertaken for both the data mining and web applications and the major challenges that arose during this phase and how they were solved.

5.1 Version Control

A GitHub Repository was used to keep a backup for the project and to act as a remote master for version control and historic purposes. Locally this was managed via PyCharm's Git integration. Using the PyCharm IDE gave a lot of visual control over what new files and changes were being committed.

When a commit was being made, all changes between the previous commit and this one for a file could be viewed as well as deciding what un-versioned files to add. Options surrounding whitespaces, empty lines and how to highlight changes were also available. Then once making a push to the remote repository, all commits to be pushed can be seen and drill downed into to make any last checks before the remote is updated.



```
C:\College\4th year\FYP\The Truth about Twitter 2 files
  Data_Mining\.idea 1 file
    workspace.xml
  The_Truth_about_Twitter\twitterstruth\templates\twitterstruth 1 file
    base.html
Unversioned Files 1 file
2 modified
```

Commit Message: `Pre-Deployment - part2`

```
ba5c6e7e8ca4be17632d5e6f801609cf0211d87a
<?xml version="1.0" encoding="UTF-8"          <?xml version="1.0" encoding="UTF-8"?>
<project version="4">                         <project version="4">
  <component name="ChangeListManager            <component name="ChangeListManager"
    <list default="true" id="cbb455c              <list default="true" id="cbb455cd-
      <change beforePath="$PROJECT_D                <change beforePath="$PROJECT_DIR$
      <change beforePath="$PROJECT_D                <change beforePath="$PROJECT_DIR$
      <change beforePath="$PROJECT_D              </list>
      <change beforePath="$PROJECT_D              <option name="EXCLUDED_CONVERTED_TO
      <change beforePath="$PROJECT_D              <option name="SHOW_DIALOG" value="
      <change beforePath="$PROJECT_D              <option name="HIGHLIGHT_CONFLICTS"
      <change beforePath="$PROJECT_D
      <change beforePath="$PROJECT_D              <session id="1513280940">
```

Figure 42 Making a Git Commit in PyCharm

5.2 Data Mining Application

This section dives into how the CRISP-DM methodology was implemented in this project with the aim of creating accurate models to classify different types of real and bot Twitter accounts and the various pieces of code used to facilitate this. The full details and evaluation of the experiments done are

5.2.1 Business Understanding

The projects domain is that of social media accounts, in particular those belonging to Twitter. A strong understanding of the domain has been cultivated over the last decade due to near-constant interaction with a variety of social media platforms, including Twitter. This meant that a lot of time that would have been spent getting to grips with understanding the domain could be instantly shifted to working on solving the main business goal of the project: *"Is it possible to predict whether a Twitter account is real or fake with a high degree of accuracy?"*

The solution that is undertaken with this project to find a dataset of Twitter users and combine this with the areas of data mining and machine learning to learn the behaviours of various different types of bot accounts and perform experiments to model and classify these different behaviours allowing the final, accurate results to be made use of by the general public to differentiate between real and fake Twitter accounts.

Fortunately, a suitable dataset, cresci-2017 [40], was found with a little bit of research as several other groups had attempted to do similar work. While this data is two years old, it will provide a solid base to build a solution from and in the future a means to add to it can be generated, albeit an arduous and time intensive task in and of itself.

5.2.2 Data Understanding

The dataset used is comprised of 13,270 Twitter accounts split between four types:

- 3,474 real human users that had been contacted directly during the creation of the dataset, with the requirement that their response being manual to qualify for inclusion in this group.

- 3,351 fake follower accounts that were used to boost the popularity or influence of real user's accounts.
- 1,533 traditional spambots, some without any focus, some spamming scam URLs and lastly ones spamming out job offers.
- 4,912 social spambots, some which all retweet a political candidate in Italy, some spamming users to download a mobile app and lastly ones spamming products up on sale on Amazon.com.

The data is split between various users and corresponding tweets files, with some tweets files containing millions of tweets.

The columns contained within the users files are:

[*'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'*]

While columns contained within the tweets files are:

[*'id','text','source','user_id','truncated','in_reply_to_status_id','in_reply_to_user_id','in_reply_to_screen_name','retweeted_status_id','geo','place','contributors','retweet_count','reply_count','favorite_count','favorited','retweeted','possibly_sensitive','num_hashtags','num_urls','num_mentions','created_at','timestamp','crawled_at','updated'*]

There are several columns in the former list that are off no value to this project, such as '*profile_background_color*' and '*profile_image_url_https*', and as such are never used. Within the latter list only the 'id' and 'text' columns are of interest, with 'id' being used to link to a user in users and 'text' to potentially draw different forms of knowledge from.

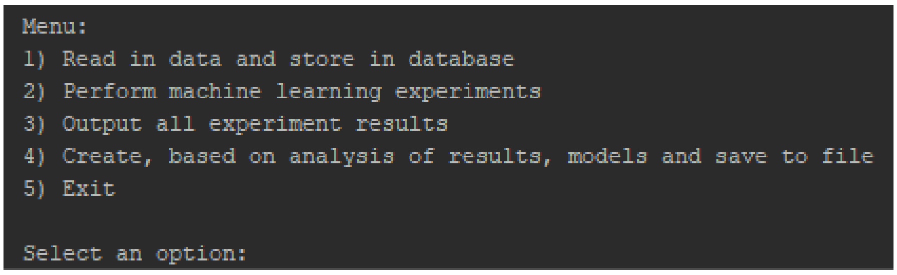
Next comes the task that has taken the most significant investment of time across the project: Understanding the behaviours of the different types of bot accounts and how to represent these as features for use in training models. Various research papers, web articles and similar applications [2, 6, 7, 50, 51, 52] were combed through and insights gained from then combined with personal thoughts and understandings of bot behaviour to arrive at these features for use:

1. Does the account have a name separate to the handle?
2. Is the default profile image being used?
3. Does the account have geolocation turned on?
4. Has the bio/description been filled out?
5. Does the account have less than 30 friends?
6. Does the account have more than 1000 friends?
7. Has the account never tweet before?
8. Has the account tweeted less than 50 times?
9. Is the accounts language set to English?
10. Does the bio/description contain a link?
11. Has the account got 3 times as many friends as followers?
12. Has the account got 50 times as many friends as followers?
13. Has the account got 100 times as many friends as followers?
14. Is the minimum Levenshtein distance between the account's tweets less than 30? This distance is the number of single-character edits to change one sequence of characters to another.

5.2.3 Task Menu

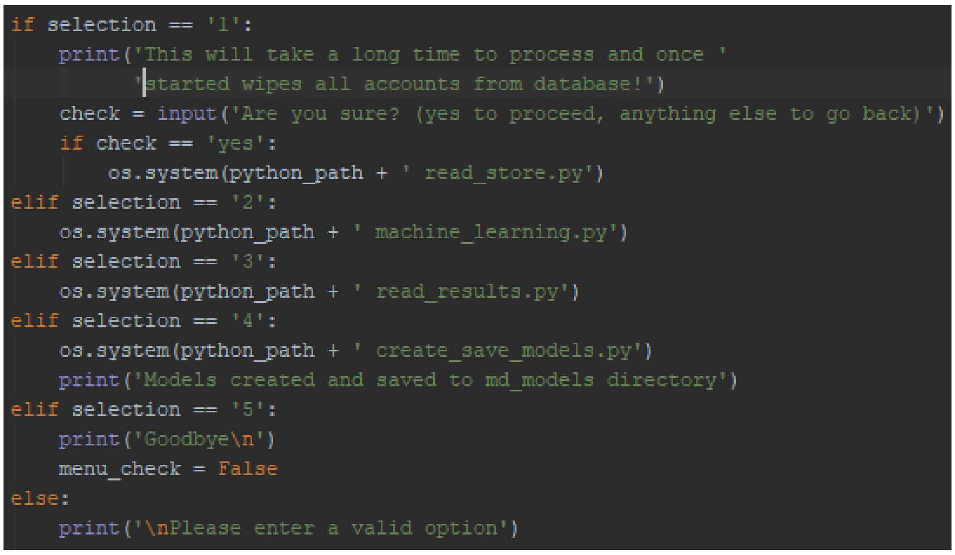
A simple command line menu was created to provide a central control function to run the various processes that comprise this part of the project. This ensured that an error in one of the processes would not stop the running of any of the others.

A warning is given before running the first process as it involves reading in all the user data from CSV files, computing each users minimum Levenshtein distance between their tweets and then storing the data in the PostgreSQL database, a time and processor intensive task.



```
Menu:
1) Read in data and store in database
2) Perform machine learning experiments
3) Output all experiment results
4) Create, based on analysis of results, models and save to file
5) Exit

Select an option:
```

Figure 43 Data Mining Menu



```python
if selection == '1':
    print('This will take a long time to process and once '
          'started wipes all accounts from database!')
    check = input('Are you sure? (yes to proceed, anything else to go back)')
    if check == 'yes':
        os.system(python_path + ' read_store.py')
elif selection == '2':
    os.system(python_path + ' machine_learning.py')
elif selection == '3':
    os.system(python_path + ' read_results.py')
elif selection == '4':
    os.system(python_path + ' create_save_models.py')
    print('Models created and saved to md_models directory')
elif selection == '5':
    print('Goodbye\n')
    menu_check = False
else:
    print('\nPlease enter a valid option')
```

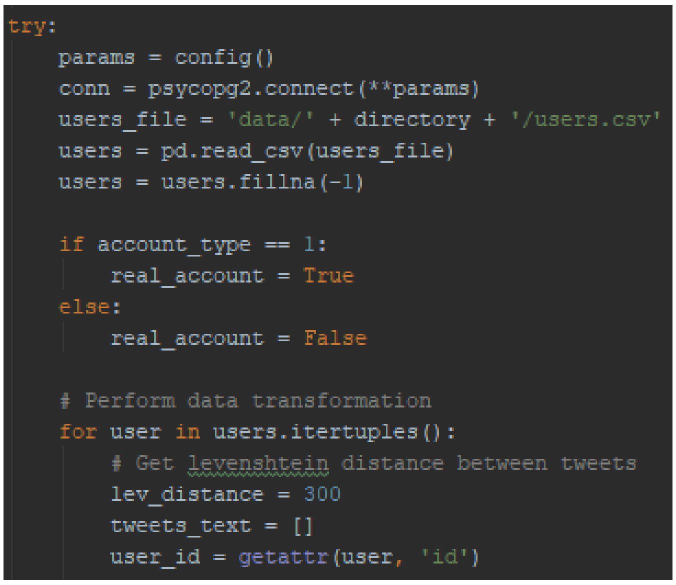
Figure 44 Menu Selection Processing

All database connections, reads and writes, throughout the application, are created using the psycopg2 library with the database parameters being read in from database.ini via a config() function that is imported where needed.

5.2.4 Data Preparation

Before the data is used to create and test models it must first be prepared which can entail various pre-processing task, cleaning, reformatting and aggregation. For this project, this stage comprised of two parts: reading all relevant account information into the database and then creating data instances comprised of the features listed in 5.2.2 by performing checks on specific user attributes.

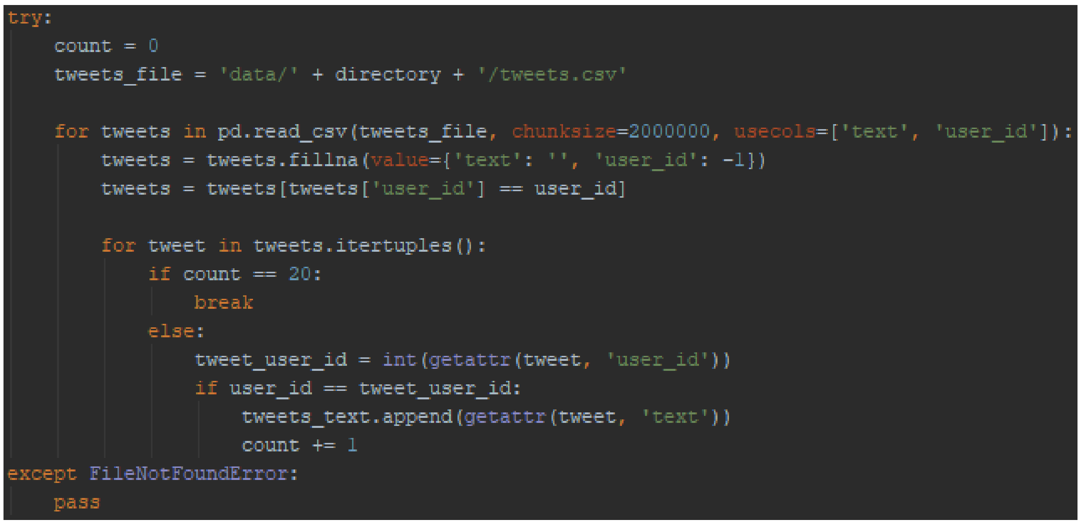
Real, fake follower, social spambots and traditional spambots are labelled 1, 2, 3 and 4 in the same order as it is important to know which sub category an account belongs too. For each sub-directory within the data directory folder, the sub-directory name and integer representing the account type is passed to the read_in_csv() function.



```python
try:
    params = config()
    conn = psycopg2.connect(**params)
    users_file = 'data/' + directory + '/users.csv'
    users = pd.read_csv(users_file)
    users = users.fillna(-1)

    if account_type == 1:
        real_account = True
    else:
        real_account = False

    # Perform data transformation
    for user in users.itertuples():
        # Get levenshtein distance between tweets
        lev_distance = 300
        tweets_text = []
        user_id = getattr(user, 'id')
```

Figure 45 Reading in Data Part 1

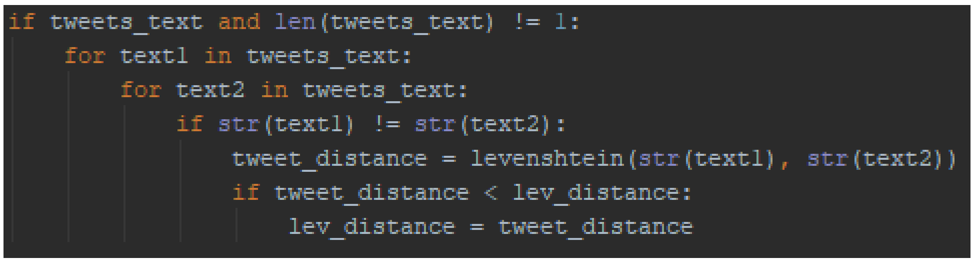
This function reads in the users file contained within that subdirectory into a dataframe, fills any null values in it and sets up the connection to the database. It then iterates over the rows, one user at a time, of the dataframe, setting up to compute that user's minimum Levenshtein distance.



```python
try:
    count = 0
    tweets_file = 'data/' + directory + '/tweets.csv'

    for tweets in pd.read_csv(tweets_file, chunksize=2000000, usecols=['text', 'user_id']):
        tweets = tweets.fillna(value={'text': '', 'user_id': -1})
        tweets = tweets[tweets['user_id'] == user_id]

        for tweet in tweets.itertuples():
            if count == 20:
                break
            else:
                tweet_user_id = int(getattr(tweet, 'user_id'))
                if user_id == tweet_user_id:
                    tweets_text.append(getattr(tweet, 'text'))
                    count += 1
except FileNotFoundError:
    pass
```

Figure 46 Reading in Data Part 2

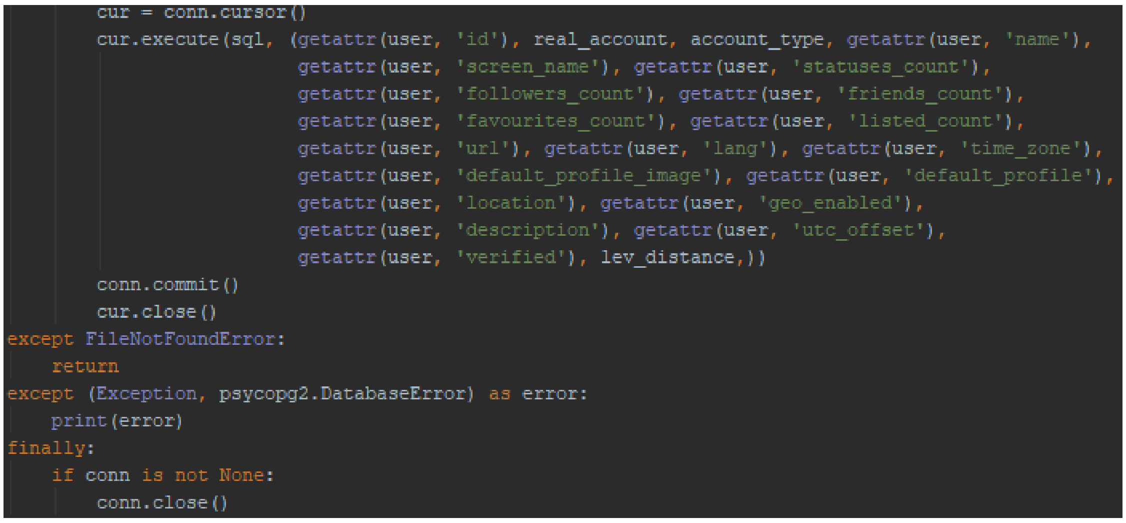
To do this the corresponding tweets file is read into another dataframe, in chunks of 2 million due to size, and iterated over until it has found the first 20 tweets linked to that user, saving them to a list.



```python
if tweets_text and len(tweets_text) != 1:
    for text1 in tweets_text:
        for text2 in tweets_text:
            if str(text1) != str(text2):
                tweet_distance = levenshtein(str(text1), str(text2))
                if tweet_distance < lev_distance:
                    lev_distance = tweet_distance
```

Figure 47 Reading in Data Part 3

Next, as long as at least 2 tweets have been found, the Levenshtein distance is computed between each unique pair of tweets and compared to the minimum to see if it is smaller and if so replaces that value.

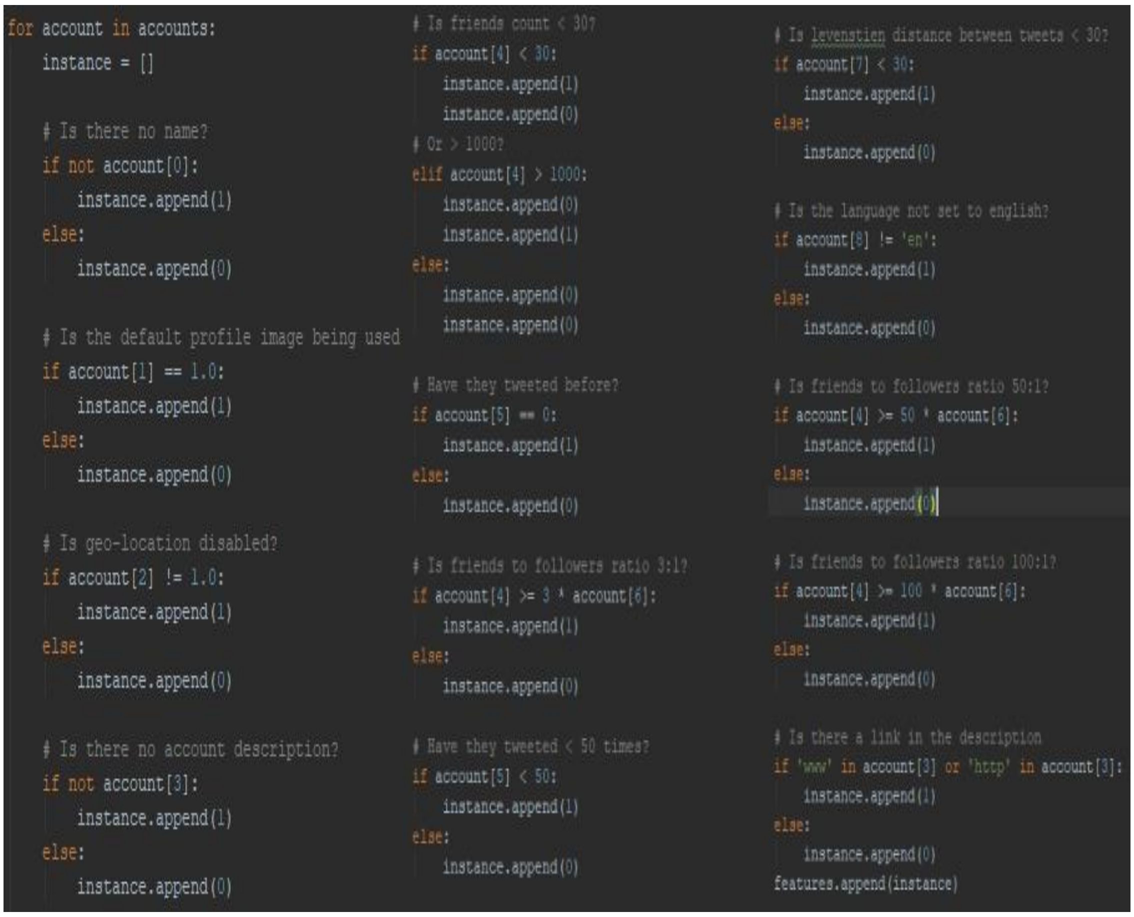


```python
        cur = conn.cursor()
        cur.execute(sql, (getattr(user, 'id'), real_account, account_type, getattr(user, 'name'),
                          getattr(user, 'screen_name'), getattr(user, 'statuses_count'),
                          getattr(user, 'followers_count'), getattr(user, 'friends_count'),
                          getattr(user, 'favourites_count'), getattr(user, 'listed_count'),
                          getattr(user, 'url'), getattr(user, 'lang'), getattr(user, 'time_zone'),
                          getattr(user, 'default_profile_image'), getattr(user, 'default_profile'),
                          getattr(user, 'location'), getattr(user, 'geo_enabled'),
                          getattr(user, 'description'), getattr(user, 'utc_offset'),
                          getattr(user, 'verified'), lev_distance,))
        conn.commit()
        cur.close()
except FileNotFoundError:
    return
except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
```

Figure 48 Reading in Data Part 4

Finally, that user’s data is saved to the database and the next user processed with any errors being outputted to scree and finally once all users processed from that file, the connection is closed.

To get the data ready for use in modelling, once it’s read out of the database, a number of checks are performed on the data to create the required binary features. Each of these checks adds a 1 to the data instance if bot like behaviour is found and a 0 if it isn’t. A corresponding targets array is used to keep track of each of the data instances output labels.



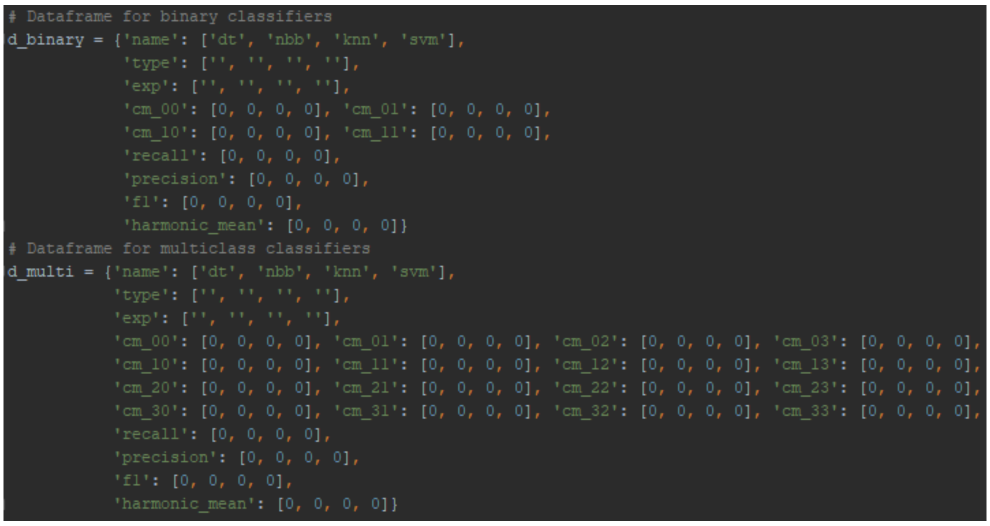
```python
for account in accounts:
    instance = []

    # Is there no name?
    if not account[0]:
        instance.append(1)
    else:
        instance.append(0)

    # Is the default profile image being used
    if account[1] == 1.0:
        instance.append(1)
    else:
        instance.append(0)

    # Is geo-location disabled?
    if account[2] != 1.0:
        instance.append(1)
    else:
        instance.append(0)

    # Is there no account description?
    if not account[3]:
        instance.append(1)
    else:
        instance.append(0)

    # Is friends count < 30?
    if account[4] < 30:
        instance.append(1)
        instance.append(0)
    # Or > 1000?
    elif account[4] > 1000:
        instance.append(0)
        instance.append(1)
    else:
        instance.append(0)
        instance.append(0)

    # Have they tweeted before?
    if account[5] == 0:
        instance.append(1)
    else:
        instance.append(0)

    # Is friends to followers ratio 3:1?
    if account[4] >= 3 * account[6]:
        instance.append(1)
    else:
        instance.append(0)

    # Have they tweeted < 50 times?
    if account[5] < 50:
        instance.append(1)
    else:
        instance.append(0)

    # Is levenstien distance between tweets < 30?
    if account[7] < 30:
        instance.append(1)
    else:
        instance.append(0)

    # Is the language not set to english?
    if account[8] != 'en':
        instance.append(1)
    else:
        instance.append(0)

    # Is friends to followers ratio 50:1?
    if account[4] >= 50 * account[6]:
        instance.append(1)
    else:
        instance.append(0)

    # Is friends to followers ratio 100:1?
    if account[4] >= 100 * account[6]:
        instance.append(1)
    else:
        instance.append(0)

    # Is there a link in the description
    if 'www' in account[3] or 'http' in account[3]:
        instance.append(1)
    else:
        instance.append(0)
    features.append(instance)
```

Figure 49 Feature Creation

5.2.5 Modelling

The modelling stage comprised of creating and running various experiments on the data. For each experiment multiple models were created for use to train using different account types:

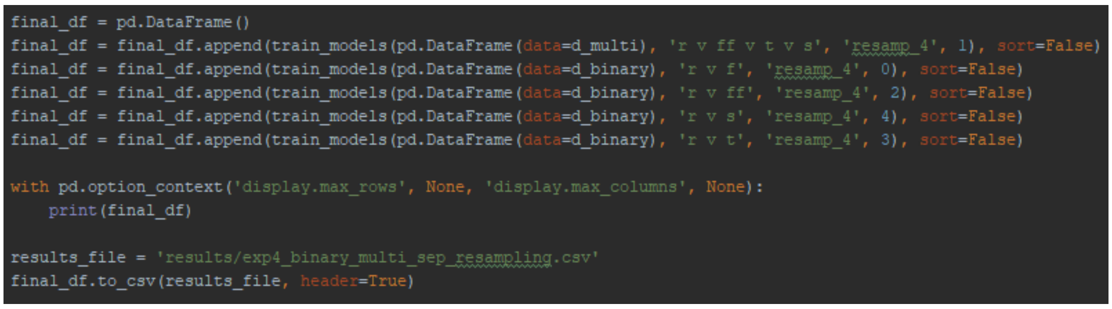
- Real vs Bot accounts that grouped all fake follower, social spambot and traditional spambot accounts together under one category, 13,270 instances split 3,474 to 9767.
- Real vs Fake Follower vs Social Spambot vs Traditional Spambot that used all the accounts but kept them all in their four individual categories, 13,270 instances split 3,474 to 3,351 to 4,912 to 1,533.
- Real vs Fake Follower, 6,825 instances split 3,474 to 3,351.
- Real vs Social Spambot, 8,386 instances split 3,474 to 4,912.
- Real vs Traditional Spambot, 5,007 instances split 3,474 to 1,533.



```
# Dataframe for binary classifiers
d_binary = {'name': ['dt', 'nbb', 'knn', 'svm'],
            'type': ['', '', '', ''],
            'exp': ['', '', '', ''],
            'cm_00': [0, 0, 0, 0], 'cm_01': [0, 0, 0, 0],
            'cm_10': [0, 0, 0, 0], 'cm_11': [0, 0, 0, 0],
            'recall': [0, 0, 0, 0],
            'precision': [0, 0, 0, 0],
            'f1': [0, 0, 0, 0],
            'harmonic_mean': [0, 0, 0, 0]}
# Dataframe for multiclass classifiers
d_multi = {'name': ['dt', 'nbb', 'knn', 'svm'],
           'type': ['', '', '', ''],
           'exp': ['', '', '', ''],
           'cm_00': [0, 0, 0, 0], 'cm_01': [0, 0, 0, 0], 'cm_02': [0, 0, 0, 0], 'cm_03': [0, 0, 0, 0],
           'cm_10': [0, 0, 0, 0], 'cm_11': [0, 0, 0, 0], 'cm_12': [0, 0, 0, 0], 'cm_13': [0, 0, 0, 0],
           'cm_20': [0, 0, 0, 0], 'cm_21': [0, 0, 0, 0], 'cm_22': [0, 0, 0, 0], 'cm_23': [0, 0, 0, 0],
           'cm_30': [0, 0, 0, 0], 'cm_31': [0, 0, 0, 0], 'cm_32': [0, 0, 0, 0], 'cm_33': [0, 0, 0, 0],
           'recall': [0, 0, 0, 0],
           'precision': [0, 0, 0, 0],
           'f1': [0, 0, 0, 0],
           'harmonic_mean': [0, 0, 0, 0]}
```

Figure 50 Dataframes Used for Data Tracking & Saving

Two dataframes, one each for binary and multi-class models, were initialised for use in recording results, in the form of entries in the confusion matrix, and accuracy metrics for each classifier across each model.

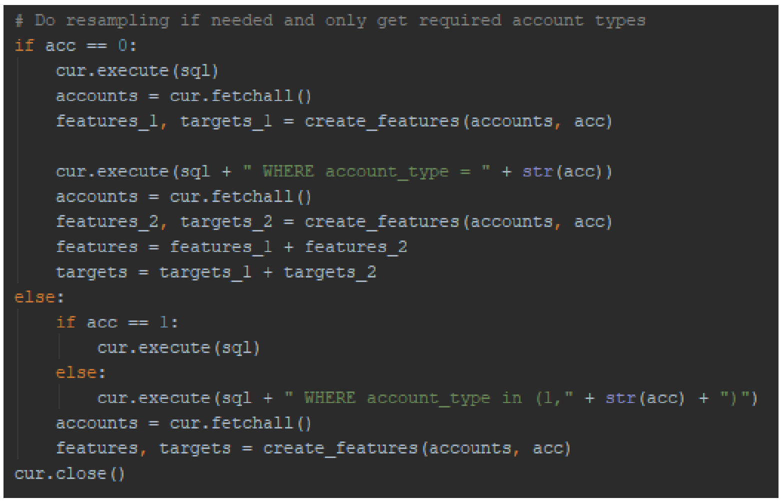


```
final_df = pd.DataFrame()
final_df = final_df.append(train_models(pd.DataFrame(data=d_multi), 'r v ff v t v s', 'resamp_4', 1), sort=False)
final_df = final_df.append(train_models(pd.DataFrame(data=d_binary), 'r v f', 'resamp_4', 0), sort=False)
final_df = final_df.append(train_models(pd.DataFrame(data=d_binary), 'r v ff', 'resamp_4', 2), sort=False)
final_df = final_df.append(train_models(pd.DataFrame(data=d_binary), 'r v s', 'resamp_4', 4), sort=False)
final_df = final_df.append(train_models(pd.DataFrame(data=d_binary), 'r v t', 'resamp_4', 3), sort=False)

with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(final_df)

results_file = 'results/exp4_binary_multi_sep_resampling.csv'
final_df.to_csv(results_file, header=True)
```

Figure 51 Experiment Running & Results Saved

These dataframes are used, one trained model at a time, to create a final, larger dataframe that contains all of the experiments details that is printed to console for immediate exploration and to file for use in comparing against other experiments.

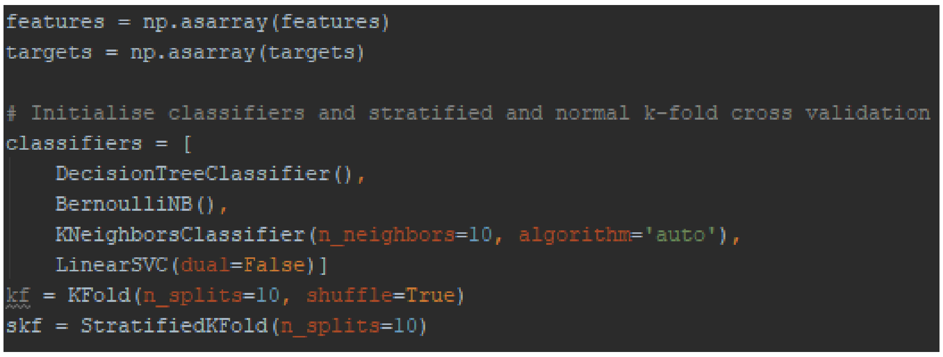
The code for the train_models() function from here on out is as it was at the end of the experiments and will be discussed as such.



```
# Do resampling if needed and only get required account types
if acc == 0:
    cur.execute(sql)
    accounts = cur.fetchall()
    features_1, targets_1 = create_features(accounts, acc)

    cur.execute(sql + " WHERE account_type = " + str(acc))
    accounts = cur.fetchall()
    features_2, targets_2 = create_features(accounts, acc)
    features = features_1 + features_2
    targets = targets_1 + targets_2
else:
    if acc == 1:
        cur.execute(sql)
    else:
        cur.execute(sql + " WHERE account_type in (1," + str(acc) + ")")
    accounts = cur.fetchall()
    features, targets = create_features(accounts, acc)
cur.close()
```

Figure 52 Get Data from Database with Resampling

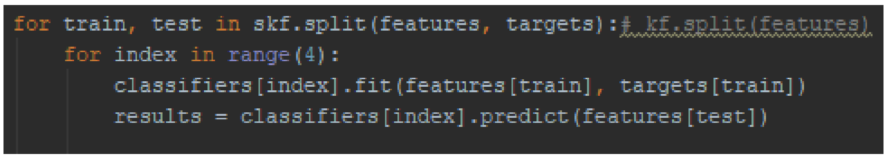
Depending on what type of model is being trained, the required accounts are retrieved from the database and features created using code in Figure 45 with corresponding instances target or account type being processed.



```
features = np.asarray(features)
targets = np.asarray(targets)

# Initialise classifiers and stratified and normal k-fold cross validation
classifiers = [
    DecisionTreeClassifier(),
    BernoulliNB(),
    KNeighborsClassifier(n_neighbors=10, algorithm='auto'),
    LinearSVC(dual=False)]
kf = KFold(n_splits=10, shuffle=True)
skf = StratifiedKFold(n_splits=10)
```

Figure 53 Array Conversion & Classifier initialisation

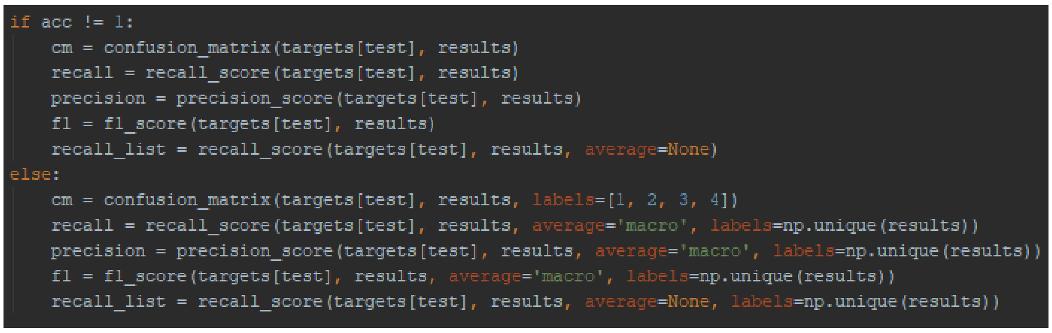
The array of instance features, and corresponding targets are converted into NumPy arrays for use in training the 4 classifiers initialised here: Decision Tree, Bernoulli Naïve Bayes, K-Nearest Neighbour and Linear Support Vector Machine.



```
for train, test in skf.split(features, targets):# kf.split(features)
    for index in range(4):
        classifiers[index].fit(features[train], targets[train])
        results = classifiers[index].predict(features[test])
```

Figure 54 Training and Testing Classifiers

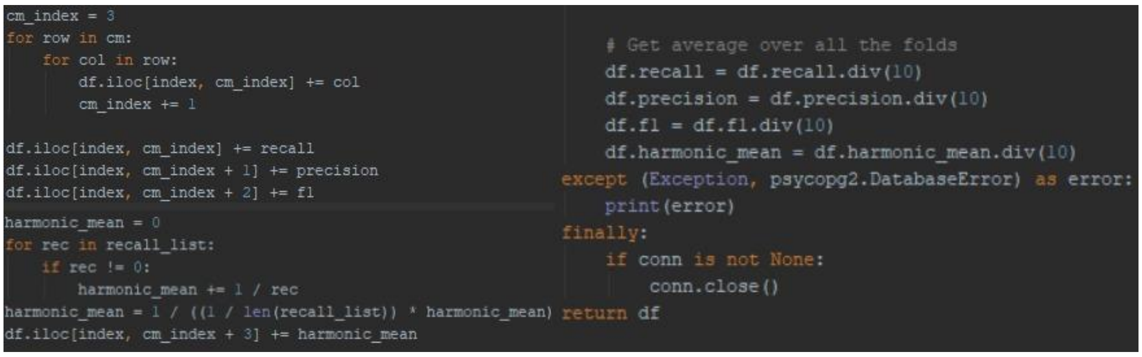
K-fold cross validation and then the Stratified version, see section 6.1, are used to split the data into 10 even folds with each one being used in turn as testing data and the other 9 as training data with each of the 4 classifiers.



```
if acc != 1:
    cm = confusion_matrix(targets[test], results)
    recall = recall_score(targets[test], results)
    precision = precision_score(targets[test], results)
    f1 = f1_score(targets[test], results)
    recall_list = recall_score(targets[test], results, average=None)
else:
    cm = confusion_matrix(targets[test], results, labels=[1, 2, 3, 4])
    recall = recall_score(targets[test], results, average='macro', labels=np.unique(results))
    precision = precision_score(targets[test], results, average='macro', labels=np.unique(results))
    f1 = f1_score(targets[test], results, average='macro', labels=np.unique(results))
    recall_list = recall_score(targets[test], results, average=None, labels=np.unique(results))
```

Figure 55 Result Metrics Computed

Depending on whether the model is binary or multi-class, the result metrics need to be computed slightly differently as the multi-class has more target value labels.



```
cm_index = 3
for row in cm:
    for col in row:
        df.iloc[index, cm_index] += col
        cm_index += 1

df.iloc[index, cm_index] += recall
df.iloc[index, cm_index + 1] += precision
df.iloc[index, cm_index + 2] += f1
harmonic_mean = 0
for rec in recall_list:
    if rec != 0:
        harmonic_mean += 1 / rec
harmonic_mean = 1 / ((1 / len(recall_list)) * harmonic_mean)
df.iloc[index, cm_index + 3] += harmonic_mean
```

```
    # Get average over all the folds
    df.recall = df.recall.div(10)
    df.precision = df.precision.div(10)
    df.f1 = df.f1.div(10)
    df.harmonic_mean = df.harmonic_mean.div(10)
except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
return df
```

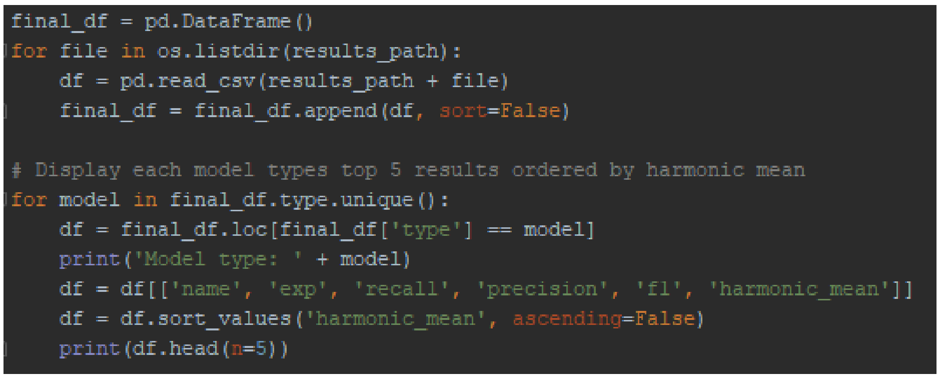
Figure 56 Computing Metrics and Function End

Next the confusion matrix entries and accuracy metrics, including the Average Class Accuracy once it has been computed, are added to the correct entry in the dataframe and once all 4 classifiers have been trained and tested with all 10 folds the accuracy metric's entries are divided by 10 to get an average value across the folds.

Finally, errors, if they happen, are outputted to console so they can be viewed, connection to the database is closed and the dataframe returned to be appended to the larger dataframe.

5.2.6 Evaluation

While the full evaluation of the experiments while modelling is detailed in 6.1 the code used to explore the results is given below.



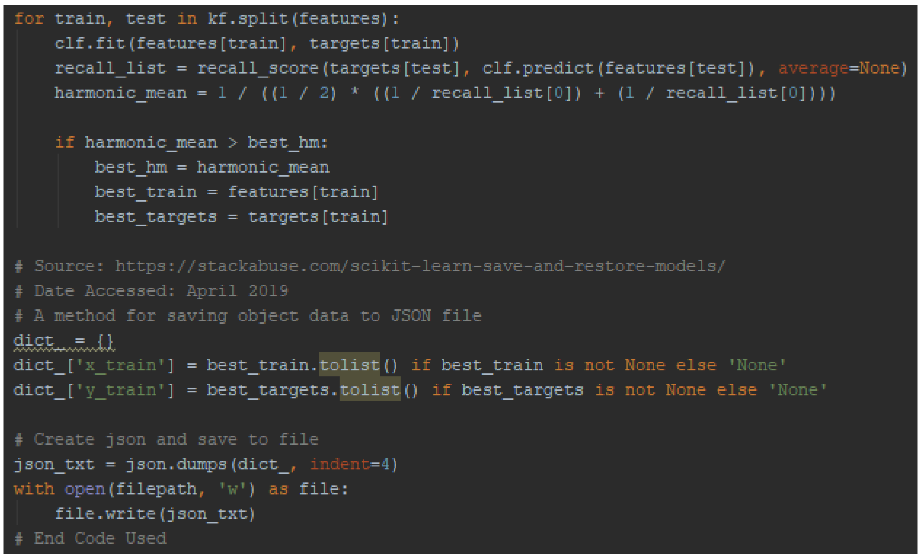
```python
final_df = pd.DataFrame()
for file in os.listdir(results_path):
    df = pd.read_csv(results_path + file)
    final_df = final_df.append(df, sort=False)

# Display each model types top 5 results ordered by harmonic mean
for model in final_df.type.unique():
    df = final_df.loc[final_df['type'] == model]
    print('Model type: ' + model)
    df = df[['name', 'exp', 'recall', 'precision', 'f1', 'harmonic_mean']]
    df = df.sort_values('harmonic_mean', ascending=False)
    print(df.head(n=5))
```

Figure 57 Results Review

This reads in all the CSV files within the results directory into one large dataframe and outputs to screen, sorted by the Average Class Accuracy, the model type, the experiment name and figures for the Recall, Precision, F1 measure and Average Class Accuracy scores so to tell how the models and experiments all lined up against each other with a variety of metrics to view.

5.2.7 Deployment

Once the final models were chosen, see section 6.1.6, a streamlined version of the modelling done in section 5.2.5 was created. Required models were trained and tested using K-fold cross validation to find the best training and test data set, based on the Average Class Accuracy score, to use for each.



```python
for train, test in kf.split(features):
    clf.fit(features[train], targets[train])
    recall_list = recall_score(targets[test], clf.predict(features[test]), average=None)
    harmonic_mean = 1 / ((1 / 2) * ((1 / recall_list[0]) + (1 / recall_list[0])))

    if harmonic_mean > best_hm:
        best_hm = harmonic_mean
        best_train = features[train]
        best_targets = targets[train]

# Source: https://stackabuse.com/scikit-learn-save-and-restore-models/
# Date Accessed: April 2019
# A method for saving object data to JSON file
dict_ = {}
dict_['x_train'] = best_train.tolist() if best_train is not None else 'None'
dict_['y_train'] = best_targets.tolist() if best_targets is not None else 'None'

# Create json and save to file
json_txt = json.dumps(dict_, indent=4)
with open(filepath, 'w') as file:
    file.write(json_txt)
# End Code Used
```

Figure 58 Model Data being Saved to Web App

For each model, once this data has been found, it is turned into a dictionary and saved to a JSON file with the file location being within the web applications directory structure, so it can be accessed by it.

5.3 Web Application

This section details how the web application was created and then deployed, including the initial setup of the Django framework, the logic used behind the scenes, the presentation layer and finally how it was deployed to an AWS Elastic Beanstalk instance for use by the public.

5.3.1 Getting Setup

The first two tasks for this section of the project was to create the basic Django framework and to get set up as a Twitter developer [53].

The former proved relatively simple due to the fruits of some early research done as it was found out that PyCharm Professional comes with several frameworks, including Django, to use out of the box when starting a new PyCharm project. PyCharm Professional requires a paid subscription but a free one-year subscription can be obtained using a valid student email address. Once this was obtained it was a simple matter of starting a new project that utilised the Django framework.

The latter meant logging into the Twitter developers site using a personal Twitter account and following the steps to get setup. On the site a Twitter application had to be created with a description of it and intent of use. Access and secret keys were created as part of this process for use in making calls to the Twitter API from within the application. These keys were saved locally to use in the next section.

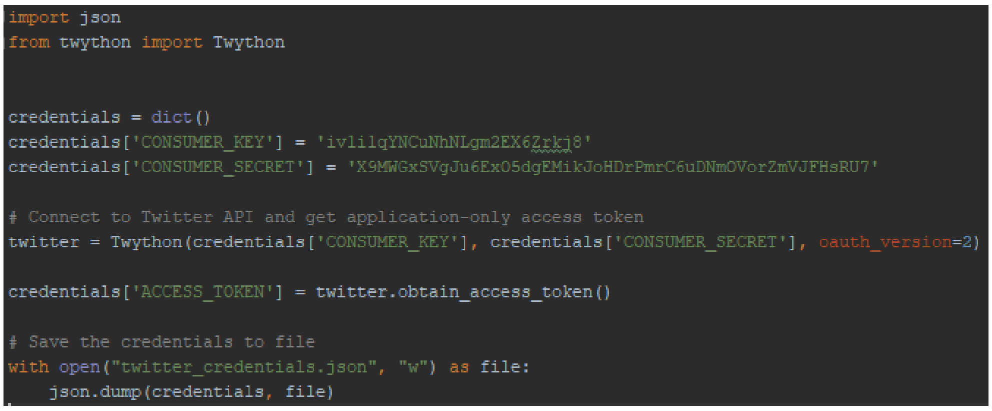
5.3.2 OAuth2 Authorisation

One feature that sets this application apart from others is the fact that it does not need the user to login to Twitter and give permission to the application to make API calls on their behalf. This gives it the distinct advantage that it can be used by anybody and at any time.

Implementing this feature meant using the OAuth2 two-legged authentication protocol as opposed to OAuth2 three-legged and OAuth1 authentication protocols as they both require a user to login so that the application can act on their behalf.

At first the Tweepy library was used to make calls to the Twitter API from within the web application, but this required using a personal access and secret key to do so. To implement OAuth2 two-legged properly it was required to swap to the Twython library [54].

From here it was a simple case of using the applications access and secret keys to request an access token through a Twitter API call via the Twython library. The keys and token were saved to a json file for any further usage in retrieving user data. Read-only API calls can be made using this method meaning the security of the keys and token was of minimal concern.



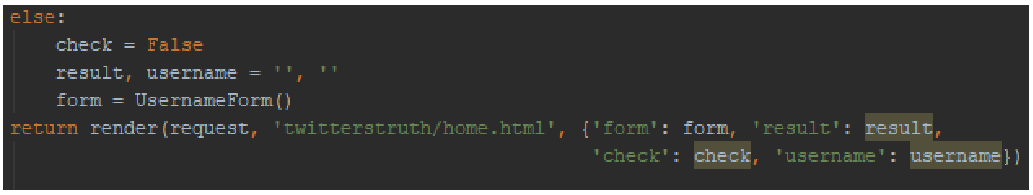
```
import json
from twython import Twython


credentials = dict()
credentials['CONSUMER_KEY'] = 'ivlilqYNCuNhNLgm2EX6Zrkj8'
credentials['CONSUMER_SECRET'] = 'X9MWGxSVgJu6ExO5dgEMikJoHDrPmrC6uDNmOVorZmVJFHsRU7'

# Connect to Twitter API and get application-only access token
twitter = Twython(credentials['CONSUMER_KEY'], credentials['CONSUMER_SECRET'], oauth_version=2)

credentials['ACCESS_TOKEN'] = twitter.obtain_access_token()

# Save the credentials to file
with open("twitter_credentials.json", "w") as file:
    json.dump(credentials, file)
```

Figure 59 Requesting & Saving Keys & Token

5.3.3 Views

View.py file processes all requests from the user, computes all the business logic needed and then returns a specified template or HTML file to be rendered to the screen.

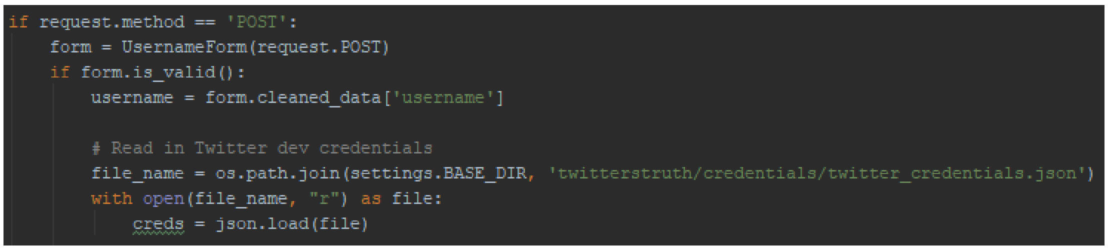
If the request method isn't POST, then it simply returns check as false and a new form instance along with home.html to be rendered to screen. This is the main page or part of home.html the users see when they visit the website and it allows them to input a Twitter handle to make a prediction on.



```
else:
    check = False
    result, username = '', ''
    form = UsernameForm()
return render(request, 'twitterstruth/home.html', {'form': form, 'result': result,
                                                    'check': check, 'username': username})
```

Figure 60 Home Page logic of view.py

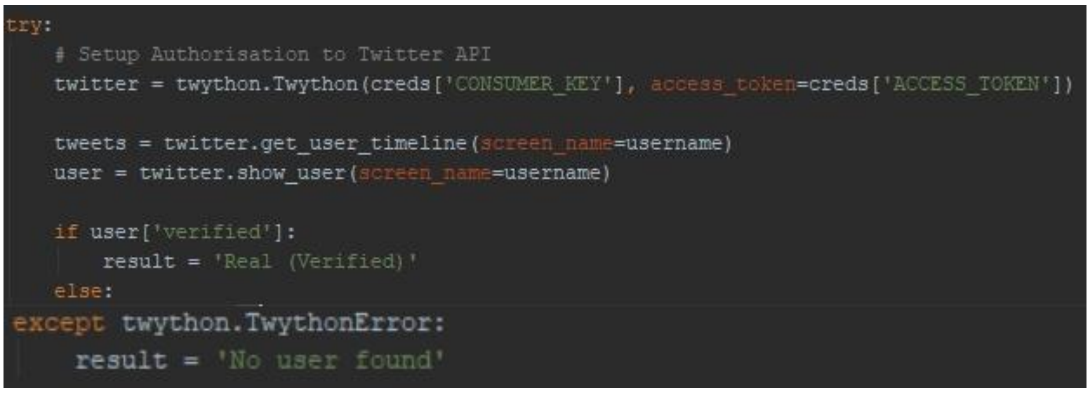
Once the user fills out the form and submits the form, the method changes to POST and the main section of business logic is run through. The form is processed and if valid the handle entered is cleaned and extracted. The saved credentials for making calls to the Twitter API are loaded from file.



```
if request.method == 'POST':
    form = UsernameForm(request.POST)
    if form.is_valid():
        username = form.cleaned_data['username']

        # Read in Twitter dev credentials
        file_name = os.path.join(settings.BASE_DIR, 'twitterstruth/credentials/twitter_credentials.json')
        with open(file_name, "r") as file:
            creds = json.load(file)
```

Figure 61 Results Page logic of view.py Part 1

Using these credentials, two calls to the Twitter API are made, one to get user account data and the other to get the twenty most recent tweets by that user both based on the handle passed through. If these calls fail, due to not being able to find an account with that handle, then the exception is processed with an appropriate error passed to the user. If the calls are successful there is then a check to see if the account is verified as nothing more needs to be done if this is the case bar returning that result.

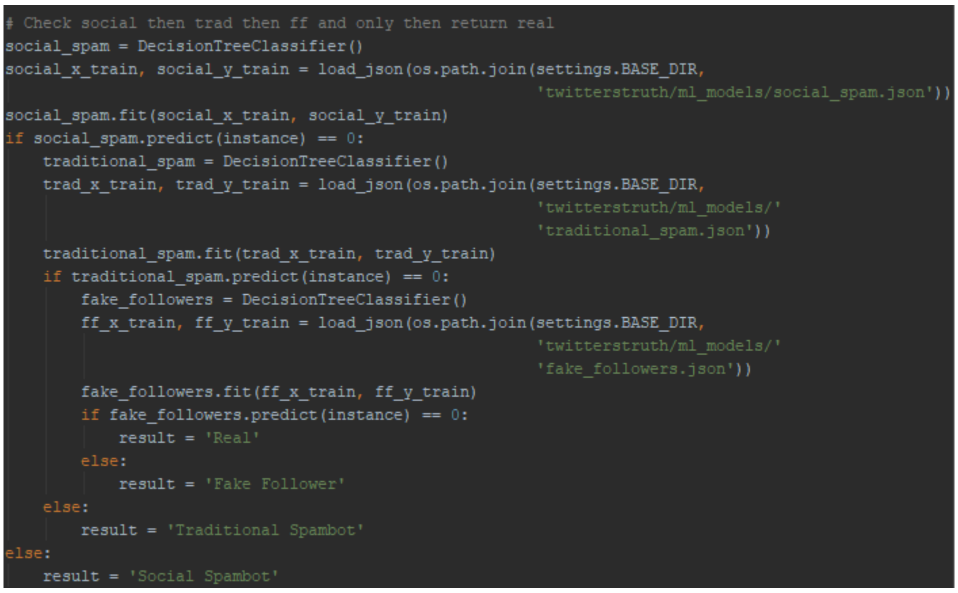


```
try:
    # Setup Authorisation to Twitter API
    twitter = twython.Twython(creds['CONSUMER_KEY'], access_token=creds['ACCESS_TOKEN'])

    tweets = twitter.get_user_timeline(screen_name=username)
    user = twitter.show_user(screen_name=username)

    if user['verified']:
        result = 'Real (Verified)'
    else:
except twython.TwythonError:
    result = 'No user found'
```

Figure 62 Results Page logic of view.py Part 2

If not, then the same logic used within the data mining application to compute the features from the account details including the minimum Levenshtein distance between the accounts twenty most recent tweets is processed.

Once this is done, the training data JSON files, saved over from the data mining application, is used to train new decision tree classifiers with predictions made in order of Social Spambot -> Traditional Spambot -> Fake Follower. If at any stage the prediction isn't real, then that result is returned with real only being returned if all predict the account to be.

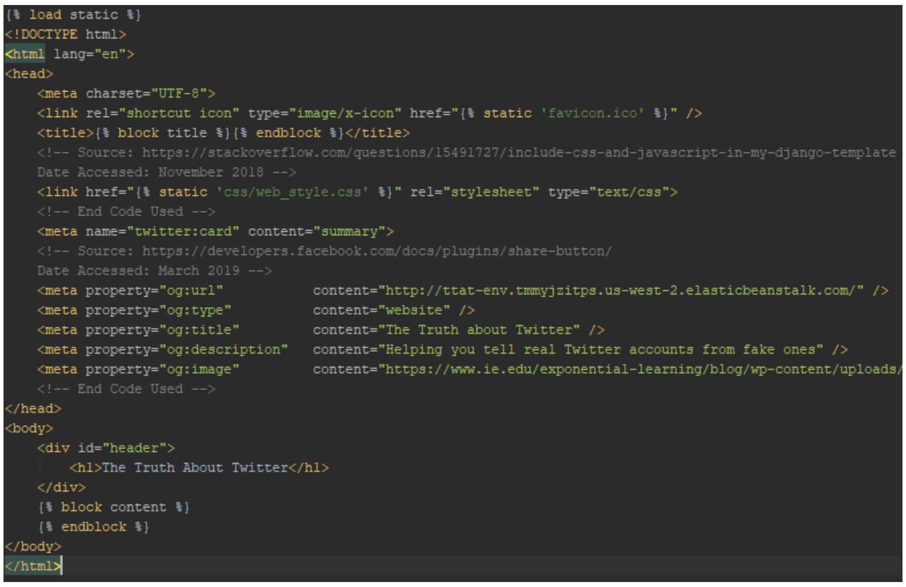


```
# Check social then trad then ff and only then return real
social_spam = DecisionTreeClassifier()
social_x_train, social_y_train = load_json(os.path.join(settings.BASE_DIR,
                                                        'twitterstruth/ml_models/social_spam.json'))
social_spam.fit(social_x_train, social_y_train)
if social_spam.predict(instance) == 0:
    traditional_spam = DecisionTreeClassifier()
    trad_x_train, trad_y_train = load_json(os.path.join(settings.BASE_DIR,
                                                        'twitterstruth/ml_models/'
                                                        'traditional_spam.json'))
    traditional_spam.fit(trad_x_train, trad_y_train)
    if traditional_spam.predict(instance) == 0:
        fake_followers = DecisionTreeClassifier()
        ff_x_train, ff_y_train = load_json(os.path.join(settings.BASE_DIR,
                                                        'twitterstruth/ml_models/'
                                                        'fake_followers.json'))
        fake_followers.fit(ff_x_train, ff_y_train)
        if fake_followers.predict(instance) == 0:
            result = 'Real'
        else:
            result = 'Fake Follower'
    else:
        result = 'Traditional Spambot'
else:
    result = 'Social Spambot'
```

Figure 63 Results Page logic of view.py Part 3

5.3.4 Templates

Contained within the templates directory are the HTML files base and home that are called on and rendered to the screen when someone accesses the website. When a request is made to the application and all relevant business logic computed, the called-on function within view.py returns the required HTML file and all required variables.

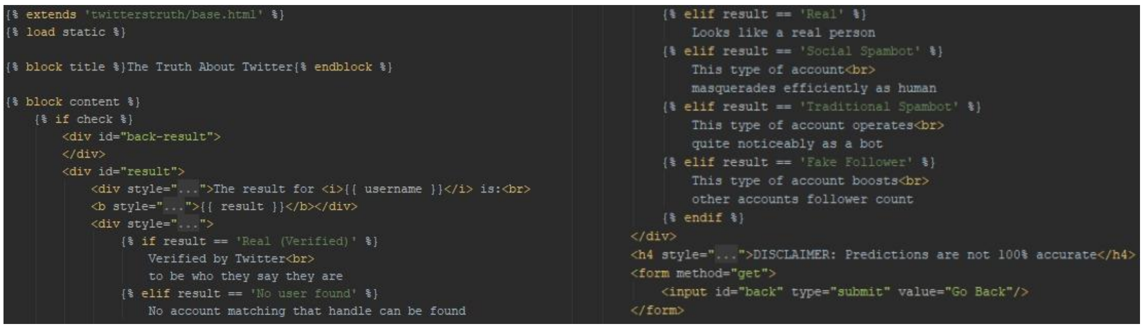


```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="shortcut icon" type="image/x-icon" href="{% static 'favicon.ico' %}" />
    <title>{% block title %}{% endblock %}</title>
    <!-- Source: https://stackoverflow.com/questions/15491727/include-css-and-javascript-in-my-django-template
    Date Accessed: November 2018 -->
    <link href="{% static 'css/web_style.css' %}" rel="stylesheet" type="text/css">
    <!-- End Code Used -->
    <meta name="twitter:card" content="summary">
    <!-- Source: https://developers.facebook.com/docs/plugins/share-button/
    Date Accessed: March 2019 -->
    <meta property="og:url"           content="http://ttat-env.tmmyjzitps.us-west-2.elasticbeanstalk.com/" />
    <meta property="og:type"          content="website" />
    <meta property="og:title"         content="The Truth about Twitter" />
    <meta property="og:description"   content="Helping you tell real Twitter accounts from fake ones" />
    <meta property="og:image"         content="https://www.ie.edu/exponential-learning/blog/wp-content/uploads/
    <!-- End Code Used -->
</head>
<body>
    <div id="header">
        <h1>The Truth About Twitter</h1>
    </div>
    {% block content %}
    {% endblock %}
</body>
</html>
```

Figure 64 Contents of base.html

Base.html acts as a foundation file, holding the head and body tags for the site. Within the head tag lies the icon and title for the website, the CSS file used to format the page and meta data used for site previews on other websites such as when sharing to Twitter or Facebook. The body holds the header for the site as well as a block content. This block content allows other HTML files to gracefully extend this file. In this way the main content of a website's page can be held independent of the underlying structure of the website, reducing the amount of reused code.

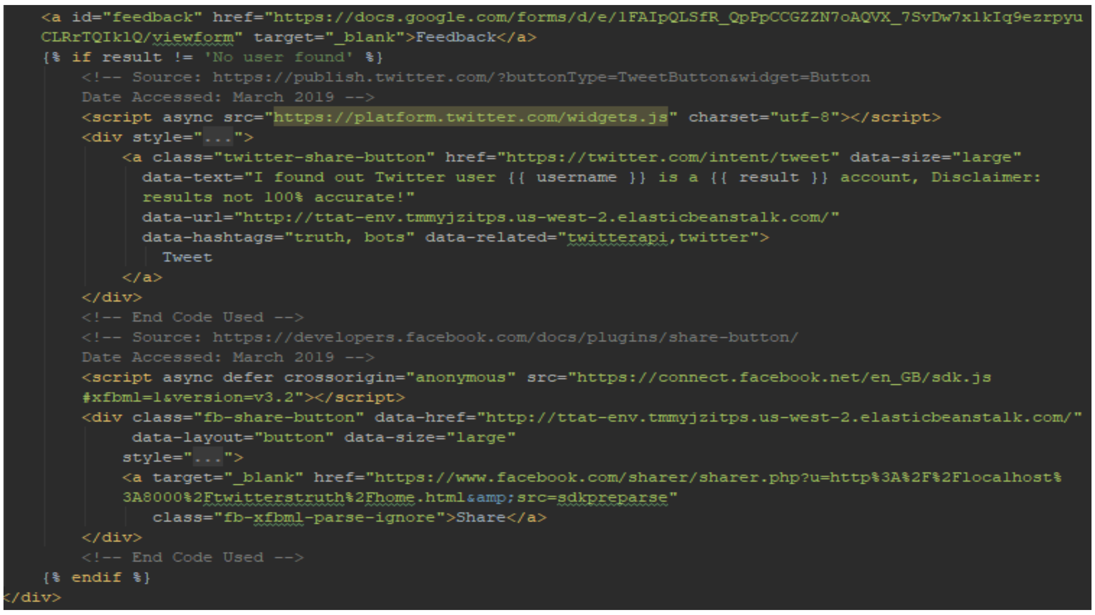
Home.html extends base.html and contains the simple form used to take user input on what Twitter account they wish to check as well as displaying the results to the user. The two parts are kept separate, with only one section being rendered at a time, using a Boolean variable that gets passed from view.py.



```
{% extends 'twitterstruth/base.html' %}
{% load static %}

{% block title %}The Truth About Twitter{% endblock %}

{% block content %}
    {% if check %}
        <div id="back-result">
        </div>
        <div id="result">
            <div style="...">The result for <i>{{ username }}</i> is:<br>
            <b style="...">{{ result }}</b></div>
            <div style="...">
                {% if result == 'Real (Verified)' %}
                    Verified by Twitter<br>
                    to be who they say they are
                {% elif result == 'No user found' %}
                    No account matching that handle can be found
                {% elif result == 'Real' %}
                    Looks like a real person
                {% elif result == 'Social Spambot' %}
                    This type of account<br>
                    masquerades efficiently as human
                {% elif result == 'Traditional Spambot' %}
                    This type of account operates<br>
                    quite noticeably as a bot
                {% elif result == 'Fake Follower' %}
                    This type of account boosts<br>
                    other accounts follower count
                {% endif %}
            </div>
            <h4 style="...">DISCLAIMER: Predictions are not 100% accurate</h4>
            <form method="get">
                <input id="back" type="submit" value="Go Back"/>
            </form>
```

Figure 65 Content of home.html Part 1

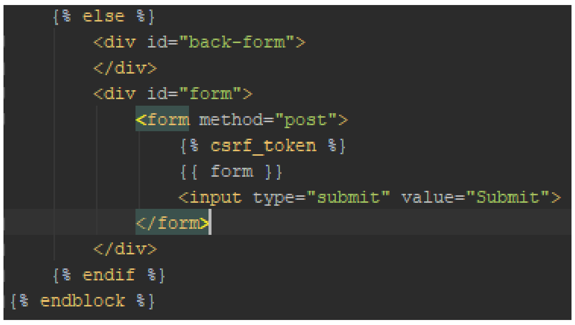
The results section displays the result variable and depending on what that is a corresponding description of what that type of account is. It also gives a disclaimer due to the predictions not being 100% accurate and various buttons, one to return back to try a different account, one which takes the user to a google form to give feedback on the site and their experience and as long as the user entered in a valid Twitter handle buttons to share the site and the prediction made to Twitter and Facebook.



```
    <a id="feedback" href="https://docs.google.com/forms/d/e/1FAIpQLSfR_QpPpCCGZZN7oAQVX_7SvDw7xlkIq9ezrpyu
    CLRrTQIklQ/viewform" target="_blank">Feedback</a>
    {% if result != 'No user found' %}
        <!-- Source: https://publish.twitter.com/?buttonType=TweetButton&widget=Button
        Date Accessed: March 2019 -->
        <script async src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>
        <div style="...">
            <a class="twitter-share-button" href="https://twitter.com/intent/tweet" data-size="large"
              data-text="I found out Twitter user {{ username }} is a {{ result }} account, Disclaimer:
              results not 100% accurate!"
              data-url="http://ttat-env.tmmyjzitps.us-west-2.elasticbeanstalk.com/"
              data-hashtags="truth, bots" data-related="twitterapi,twitter">
                Tweet
            </a>
        </div>
        <!-- End Code Used -->
        <!-- Source: https://developers.facebook.com/docs/plugins/share-button/
        Date Accessed: March 2019 -->
        <script async defer crossorigin="anonymous" src="https://connect.facebook.net/en_GB/sdk.js
        #xfbml=1&version=v3.2"></script>
        <div class="fb-share-button" data-href="http://ttat-env.tmmyjzitps.us-west-2.elasticbeanstalk.com/"
             data-layout="button" data-size="large"
            style="...">
            <a target="_blank" href="https://www.facebook.com/sharer/sharer.php?u=http%3A%2F%2Flocalhost%
            3A8000%2Ftwitterstruth%2Fhome.html&amp;src=sdkpreparse"
               class="fb-xfbml-parse-ignore">Share</a>
        </div>
        <!-- End Code Used -->
    {% endif %}
</div>
```

Figure 66 Contents of home.html Part 2

The initial section contains a simple form for the user to entry a Twitter handle along with a submit button and two div tags acting as the containers and background for the form.



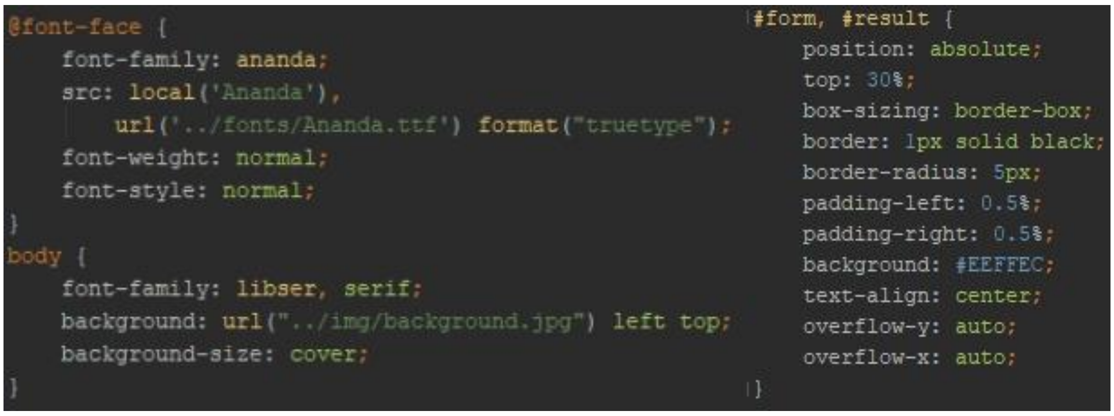
```
    {% else %}
        <div id="back-form">
        </div>
        <div id="form">
            <form method="post">
                {% csrf_token %}
                {{ form }}
                <input type="submit" value="Submit">
            </form>
        </div>
    {% endif %}
{% endblock %}
```

Figure 67 Contents of home.html Part 3

Screenshots of both templates rendered to screen can be seen in section 5.3.6.

5.3.5 Static Files

The web application makes use of several different static files as part of its presentation layer. The main static file is the web_style.css file that contains most of the formatting for the web application. This included everything from loading in and using other static files such as the various font files or the JPG image used as the background to the positioning and borders of the different sections.



```
@font-face {
    font-family: ananda;
    src: local('Ananda'),
        url('../fonts/Ananda.ttf') format("truetype");
    font-weight: normal;
    font-style: normal;
}
body {
    font-family: libser, serif;
    background: url("../img/background.jpg") left top;
    background-size: cover;
}
```

```
#form, #result {
    position: absolute;
    top: 30%;
    box-sizing: border-box;
    border: 1px solid black;
    border-radius: 5px;
    padding-left: 0.5%;
    padding-right: 0.5%;
    background: #EEFFEC;
    text-align: center;
    overflow-y: auto;
    overflow-x: auto;
}
```

Figure 68 Extract from CSS file

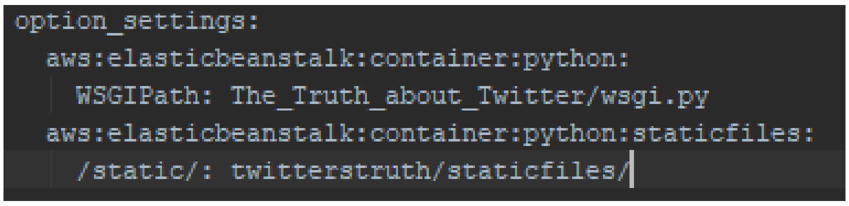
The other static file used was the favicon.ico file that was created through online tools and is used as the icon next to the websites title in a web browser.

Figure 69 Web Icon

5.3.6 Deployment & Demonstration

Several small adjustments were needed to be made for the application to be deployed onto an AWS Elastic Beanstalk instance with the only large one being explained in section 5.4.5.

The server needs to know where to look for certain files within the directory hierarchy such as where all the static files and the wsgi.py file were being held. The Web Server Gateway Interface is how a web server forwards requests to web applications and as such the server needs to know the path to the wsgi.py as this is where all requests are forwarded too. Without this the server cannot communicate with the web application.



```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: The_Truth_about_Twitter/wsgi.py
  aws:elasticbeanstalk:container:python:staticfiles:
    /static/: twitterstruth/staticfiles/
```

Figure 70 Path Settings for AWS EB

Next all the modules used within the application and their versions needed to be saved to file within the root directory to be picked up when deploying so that the server could tell what versions of what modules it needed to have installed. This was done through the command "pip freeze -> requirements.txt".

Once this was done it was time to use the Elastic Beanstalk Command Line Interface [55] to deploy the application by following these steps:

1. Initialising an EB CLI repository: "eb init -p python-3.6 ttat".
2. Create an environment: "eb create ttat-env".
3. Check the status: "eb status".
4. Note the CNAME setting and adding this to the ALLOWED_HOSTS setting within settings.py file
5. Lastly deploy: "eb deploy"

After a few minutes the application was up and running on the server and available for use by the public. Below are a few screenshots of the application in action, including the main page, the results page after a user has inputted an account to check and what comes up when sharing to Facebook.

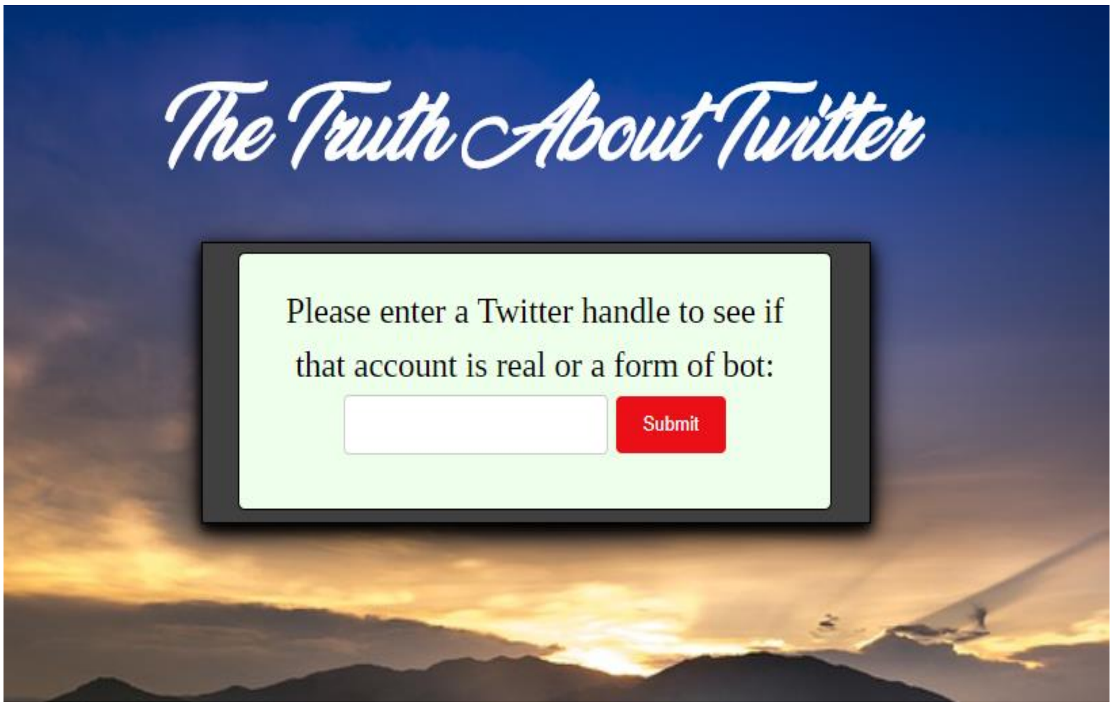
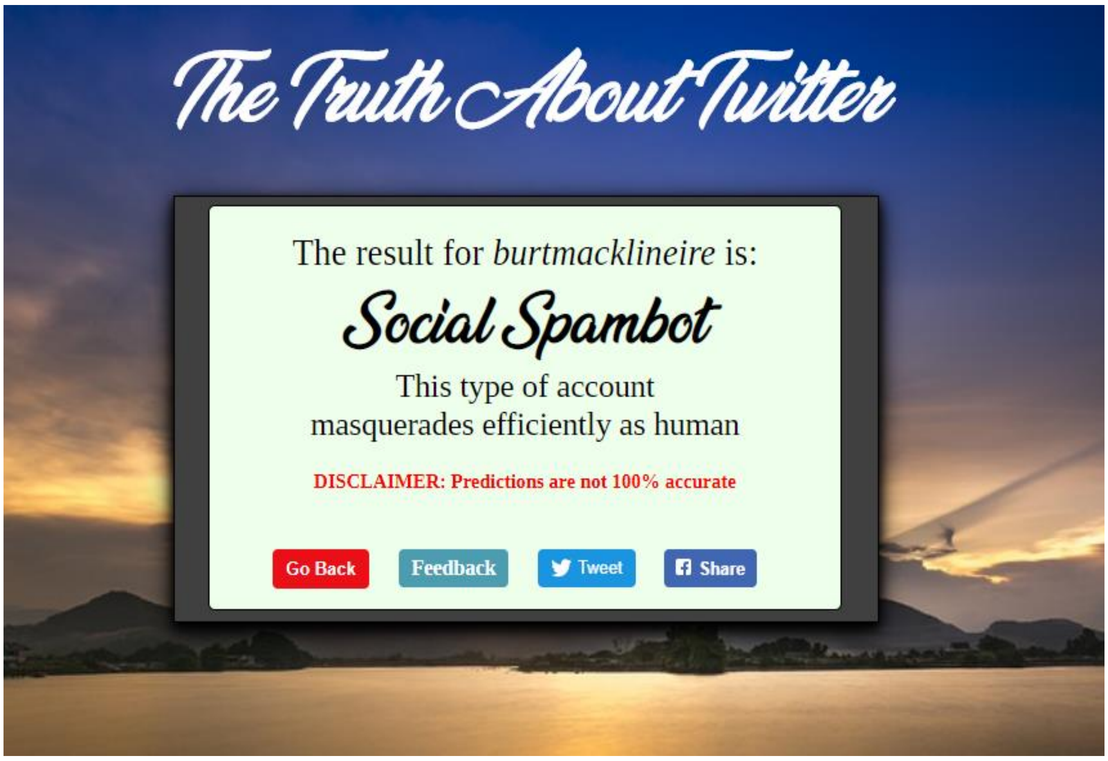

The Truth About Twitter
Please enter a Twitter handle to see if that account is real or a form of bot:
Submit

Figure 71 Screenshot of Main Web Page



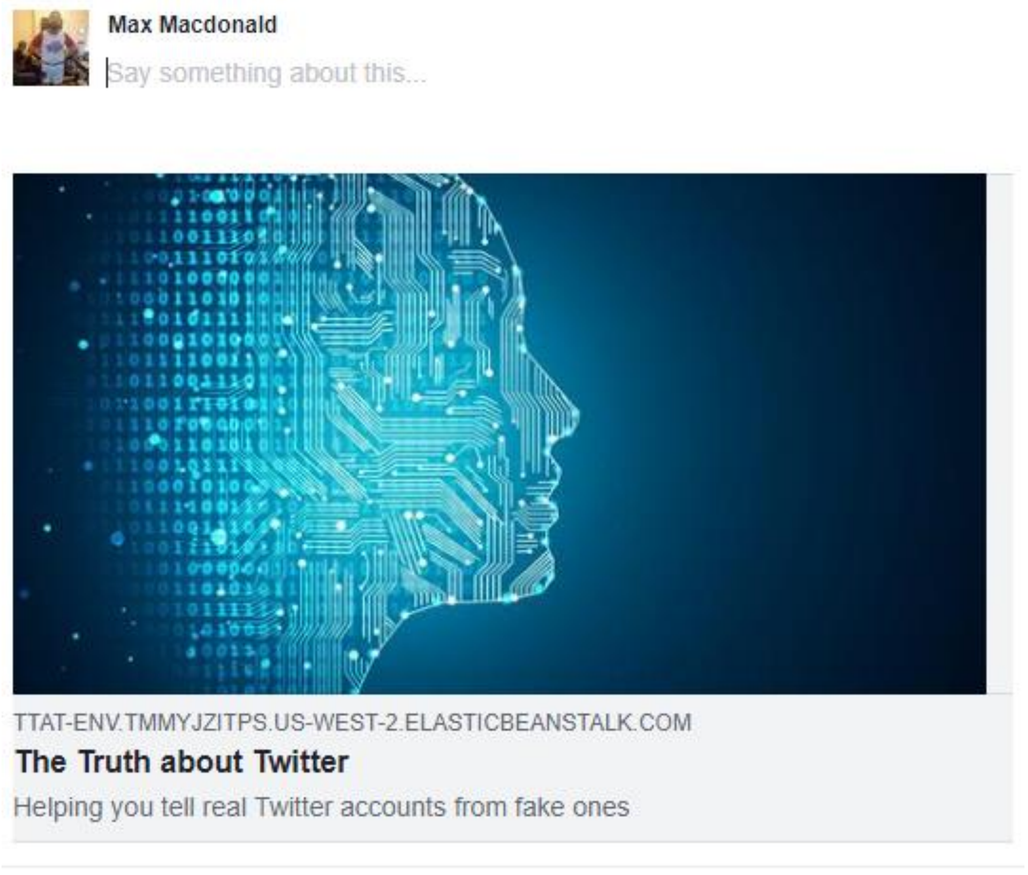
The result for *burtmacklineire* is:

Social Spambot

This type of account masquerades efficiently as human

DISCLAIMER: Predictions are not 100% accurate

Figure 72 Screenshot of Results Page



TTAT-ENV.TMMYJZITPS.US-WEST-2.ELASTICBEANSTALK.COM

The Truth about Twitter

Helping you tell real Twitter accounts from fake ones

Figure 73 Screenshot of Sharing to Facebook

5.4 Challenges Encountered

While there were a wide range of challenges encountered at nearly every stage across the entire project life, only the major ones are explained here and how they were overcome.

5.4.1 Time Management

This issue was touched on as a potential challenge at the start of the project and it most certainly proved to be. Everything from other college work to family commitments and slight mental health issues worked to derail tasks and extend the time it took to complete them ensuring that there were constant revisions to self-set task deadlines.

There was no easy solution to this issue and while the use of Trello boards certainly helped to keep track of things and try to keep the daily motivation going, they could be quite conveniently ignored if so desired.

What did work though was a combination of parental support, where they would check in and see how everything is going, offering an ear to talk to and words of encouragement, weekly meetings with the project supervisor as progress was expected to be made from week to week and finally seeing the progress others in the year were making with their projects as this proved quite the personal motivator.

5.4.2 User & Tweet Data Storing

As all the account data was split up between various users and tweets CSV files, this needed to be stored for use in the various parts of the data mining application. The format to store it in went through various iterations in the first half of this project as there were various ways of going about it.

Should all the data just be stored raw, some form of it stored with the parts that were not needed left out or features be derived from the data first and then stored. Initially the last option was chosen but upon the realisation that to change any of those features would require that the data within the database be changed partially or completely, a time intensive task due to either the volume of data being read in from the CSV files or implementing SQL to make the changes, other options had to be explored.

Finally, after trying a few different approaches, it was decided that the best way to proceed was to read in the user's account data in its normal form, while dropping a variety of columns that held no value and adding three columns, including one to hold what type of account it is and one to hold the minimum Levenshtein distance between that users tweets thereby eliminating the need to store anything from the tweets files.

5.4.3 Model Accuracy

Another challenge that was brought up at the start of the project was gaining a high level of accuracy in the machine learning models and turned out to be quite a time consuming and vexing challenge to overcome.

Vast quantities of time were devoted to researching what were the characteristics of the various types of bot accounts, various machine learning classifiers, how to make full use of their parameters and other techniques that can be employed on a dataset to gain a higher-level accuracy such as K-fold cross validation and resampling.

Five large scale experiments were undertaken parallel to a lot of this research and incorporating all the knowledge learned. This investment of time and effort payed off as the final models created had approx. 90% accuracy in their classification of a Twitter account.

5.4.4 Web UI Design

Designing and creating the UI for the web application proved to be rather troublesome due to a lack of confidence and ability in this domain. Coming up with a fresh and trendy design and implementing all the CSS that went along with it ended proved a bridge too far for the time span off this project without going out and either completely ripping off another website or purchasing a website framework design.

The solution to this proved rather boring and it was decided to keep all design elements of the web application as simple as possible. This is one challenge that is passed onto future iterations of the project to complete as it would require a substantial amount of time to complete and priorities at that stage lay elsewhere.

5.4.5 Saving Models for Deployment

The only connections between the data mining and web application are the final models used to predict the classification of Twitter accounts. Initially to create and save these models across from one application to the other, they were serialised using the pickle Python library and then saved within the web application.

When the web application was run locally, there was no issue with using the pickle library to then de-serialise the models for use in making predictions. The issue only arose once the application was deployed onto an AWS Elastic Beanstalk instance when trying to load the models from their SAV files to make a prediction with the error "Buffer dtype mismatch, expected 'SIZE_t' but got 'int'" being given.

After investigation, it was found that there was no easy fix for this error as this error had to do either with a mismatch between Python versions or environments, Windows and Linux, locally and on the server instance. The solution found was to change what was being saved across so instead of the models, the training data used to create them was saved across by writing the data to dictionaries and then to their respective JSON files. Within the web application, these files are then loaded up and the data extracted and used to recreate the models once more for use.

6 Testing & Evaluation

This chapter covers the evaluation of the various model experiments undertaken in pursuit of the most accurate way to identify the different types of accounts as well as all forms of testing done across the project, including usability testing of the web application, ad-hoc testing and unit testing.

6.1 Model Experiments

This is the section where the main business goal of *"Is it possible to predict whether a Twitter account is real or fake with a high degree of accuracy?"* is answered. This was done through the running of different experiments using a carefully constructed feature set and different account types as target labels to train various classifiers.

6.1.1 Getting Ready

The different output types used with every classifier within each experiment were:

- Real vs Bot with 13,270 instances split 3,474 to 9,769.
- Real vs Fake Followers vs Social Spambots vs Traditional Spambots with 13,270 instances split 3,474 to 3,351 to 4,912 to 1,533.
- Real vs Fake Followers with 6,825 instances split 3,474 to 3,351
- Real vs Social Spambots with 8,386 instances split 3,474 to 4,912.
- Real vs Traditional Spambots with 5,007 instances split 3,474 to 1,533.

6.1.1.1 Data

The feature set was generated by answering the list of questions in 5.2.2 in a binary fashion. All positive answers pertained to bot like behaviour and the feature was assigned 1 with a 0 for a negative answer.

no_screen_name	default_profile_image	no_geolocation
no_description	friends_less_30	friends_more_1000
never_tweeted	tweeted_more_50	levenshtein_less_30
lang_not_eng	link_in_description	3friends_for_1follower

50friends_for_1follower	100friends_for_1follower

Figure 74 Feature Set

6.1.1.2 Classifiers

The four classifiers that were used during the entire modelling process were Decision Tree, Bernoulli Naïve Bayes, K-Nearest Neighbour and Linear Support Vector Machines as these all were either well round classifiers that could work with numerous different types of datasets or worked well with the binary nature of the features and the binary and multiclass nature of the targets.

6.1.1.3 Training & Testing

The splitting of data into training and testing data was done through the implementation of K-fold cross validation with K=10. This meant the data, for each classifier, was split into 10 even folds, with 1-fold being taken at a time as test data and the remaining folds taken as training data for each of the classifiers. This meant that all the data was used for testing and training purposes across each classifier and model type.

For the 3rd experiment onwards this was modified to Stratified k-fold cross validation where the ratio of account types was preserved across the folds.

6.1.1.4 Evaluation

All models and classifiers created during the experiments were evaluated using the Precisions, Recall, F1 measure and the Average Class Accuracy (HM) with the final metric being the most important. All metrics are an average over the 10 folds that the classifiers were trained over and are given to 2 decimal points. All formulas can be found in 2.4.3.3.

6.1.2 Experiment 1: Minimal Features

This first experiment used 8 of the 14 features to kickstart the whole process and get an indication of how each of the model types and classifiers perform:

[no_screen_name, default_profile_image, no_geolocation, no_description, friends_less_30, friends_more_1000, never_tweeted, 3friends_for_1follower]

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	64.45	48.89	41.44	13.16
B. Naïve Bayes	37.43	45.17	38.71	35.98
K-Nearest Neighbours(10)	60.75	37.47	27.22	1.91
Linear SVM	43.89	46.2	37.37	23.36

Figure 75 Exp 1: Real vs Bot

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	95.81	81.62	88.15	85.06
B. Naïve Bayes	86.76	89.59	88.15	69.67
K-Nearest Neighbours(10)	75.99	99.87	86.31	21.99
Linear SVM	86.67	89.61	88.11	69.42

Figure 76 Exp 1: Real vs Fake Followers vs Social Spambots vs Traditional Spambots

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	94.49	95.49	94.98	95.04
B. Naïve Bayes	94.48	95.73	95.09	95.14
K-Nearest Neighbours(10)	94.75	86.70	87.05	87.10
Linear SVM	92.36	97.33	94.72	94.61

Figure 77 Exp 1: Real vs Fake Followers

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	74.30	42.04	53.65	57.03
B. Naïve Bayes	66.54	42.26	51.66	56.14
K-Nearest Neighbours(10)	44.38	99.62	61.39	10.30
Linear SVM	67.25	32.85	44.10	47.76

Figure 78 Exp 1: Real vs Social Spambots

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	92.27	93.11	92.68	90.99
B. Naïve Bayes	78.70	99.33	87.81	76.37
K-Nearest Neighbours(10)	59.29	99.88	74.39	5.75
Linear SVM	91.68	93.15	92.41	90.52

Figure 79 Exp 1: Real vs Traditional Spambots

Some of the results were quite high already, implying that all the time researching and understanding bot behaviour and the decisions made around what features to create and use has paid off. This will become clear as more experiments are done.

All the KNN scores, bar one, were downright horrendous. This gives very little confidence going forward that this will change drastically but will be kept in anyways as it will be interesting to note how much they can improve if at all.

6.1.3 Experiment 2: Maximum Features

This experiment uses the entire feature set, including the 6 features that were left out of the previous one. The hope here is to see how powerful the full feature set is with classifying bot accounts. If the results are not adequate, there are a few techniques that can be employed to try and raise it otherwise the feature set itself will need to be reviewed.

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	77.33	76.33	76.01	75.01
B. Naïve Bayes	68.70	68.30	67.67	65.17
K-Nearest Neighbours(10)	73.06	54.36	49.49	31.72
Linear SVM	70.69	70.29	69.96	68.68

Figure 80 Exp 2: Real vs Bot

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	95.92	92.32	94.08	89.94
B. Naïve Bayes	93.90	89.89	91.85	85.59
K-Nearest Neighbours(10)	78.46	99.47	87.72	24.69
Linear SVM	92.55	94.20	93.37	84.26

Figure 81 Exp 2: Real vs Fake Followers vs Social Spambots vs Traditional Spambots

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	97.73	98.36	98.04	98.07
B. Naïve Bayes	97.72	97.29	97.50	97.54
K-Nearest Neighbours(10)	98.25	96.28	97.19	97.22
Linear SVM	97.11	99.02	98.05	98.06

Figure 82 Exp 2: Real vs Fake Followers

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	91.54	77.11	8.368	84.94
B. Naïve Bayes	88.86	71.82	79.42	81.09

K-Nearest Neighbours(10)	57.81	98.70	72.90	62.16
Linear SVM	84.51	72.09	77.78	80.01

Figure 83 Exp 2: Real vs Social Spambots

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	94.18	95.11	94.64	93.35
B. Naïve Bayes	90.11	91.13	90.60	88.38
K-Nearest Neighbours(10)	74.73	99.47	85.32	68.29
Linear SVM	93.28	92.77	93.00	91.54

Figure 84 Exp 2: Real vs Traditional Spambots

Improvements right across the board here now that the full dataset is being used. The worry that the results were going to go the other way has been laid to rest and many of these results will have a strong influence on the final model chosen at the end of these experiments.

6.1.4 Experiment 3: Stratifying

While K-fold cross validation is an immensely powerful way of splitting up the dataset into training and testing folds, in some cases this can be made even stronger by combining it with stratifying. This technique ensures that when you split a dataset up into equal size sets each of these sets maintain the same ratio with regards to the output labels instances as the original dataset. This is employed here to see does it affect the results in any way.

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	76.03	73.49	72.52	69.14
B. Naïve Bayes	70.22	67.34	66.55	59.69
K-Nearest Neighbours(10)	72.16	54.14	48.37	24.99
Linear SVM	68.57	67.32	66.47	68.64

Figure 85 Exp 3: Real vs Bot

Classifier	**Precision**	**Recall**	**F1 Measure**	**ACA (HM)**
Decision Tree	95.90	88.53	91.39	87.70
B. Naïve Bayes	92.28	84.40	86.01	80.27
K-Nearest Neighbours(10)	78.61	98.55	87.44	26.42
Linear SVM	92.53	92.12	91.75	83.26

Figure 86 Exp 3: Real vs Fake Followers vs Social Spambots vs Traditional Spambots

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	97.68	97.82	97.74	97.77
B. Naïve Bayes	97.72	97.11	97.38	97.43
K-Nearest Neighbours(10)	98.87	86.69	91.33	91.41
Linear SVM	97.07	98.96	98.00	98.03

Figure 87 Exp 3: Real vs Fake Followers

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	90.63	70.46	77.37	78.55
B. Naïve Bayes	86.87	65.60	73.14	74.80
K-Nearest Neighbours(10)	56.90	98.90	72.23	60.11
Linear SVM	82.12	63.93	70.66	72.96

Figure 88 Exp 3: Real vs Social Spambots

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	93.99	90.37	91.21	90.00
B. Naïve Bayes	89.50	88.48	88.25	86.21
K-Nearest Neighbours(10)	76.08	97.63	85.25	69.65
Linear SVM	92.69	91.59	91.39	89.68

Figure 89 Exp 3: Real vs Traditional Spambots

Interestingly, most of the results are slightly worse then they are in the previous experiment. This pretty much rules out the use of this technique in any future experiments or work unless in the future it can be shown that employing this technique still holds some benefit due to a change in project requirements.

6.1.5 Experiment 4: Resampling

One issue that could be affecting the results for Real vs Bot is the fact that the 3,474 real accounts are being drowned out by 9,769 bot ones as classifiers need to be able to accurately predict accounts, not just guess the majority target. The solution is to use oversampling of real accounts to combat this imbalance. Every real account is taken twice to create 6,948 real accounts to use with the 9,769 bot accounts making a total and more balanced dataset of 16,717 instances.

Classifier	Precision	Recall	F1 Measure	ACA (HM)
Decision Tree	95.90	88.57	91.42	87.76
B. Naïve Bayes	92.28	84.40	86.01	80.27
K-Nearest Neighbours(10)	78.61	98.55	87.44	26.42
Linear SVM	92.53	92.12	91.75	83.27

Figure 90 Exp 4: Real vs Bot

The results, bar the KNN classifier, are much improved from previous experiments meaning that using resampling, the Real vs Bot model type is in contention for use as the final model imported into the web application.

6.1.6 Conclusions

When looking to see which model types and classifiers for those were most accurate it was a case of comparing the scores in a three-way face-off between Real vs Bot, Real vs Fake Followers vs Social Spambots vs Traditional Spambots and the remaining three counting as one whole.

Looking at things with this perspective it was found that the results from experiment 2 for the combination of Real vs Fake Followers, Real vs Social Spambots and Real vs Traditional spambots, with the Decision Tree classifier for all three, proved to be the best with scores of 98.07%, 84.94% and 93.35 respectively. The Decision Tree classifier will then be used with the full feature set to create models for use in the web application.

This also fits into the thought process that Social Spambots are harder to label then Traditional Spambots who again in turn are harder to label then Fake Followers.

6.2 Usability Testing

Usability testing is an important aspect of any system that contains user interaction. Gaining feedback from the user can help in a myriad of ways, such as finding any compatibility issues across browsers or operating systems, weeding out bugs that may have been missed during other phases of testing and to help understand if the system is truly intuitive.

The target audience for the web application consists of anyone who uses the internet as this issue affects not just people with Twitter accounts, but also anyone else who uses other social media platforms, reads news articles online or visits any imaging sharing site, such as Imgur or Reddit, as tweets or screenshots of them are shared throughout all of these.

Once the application was deployed, it was shared via Facebook, asking people to try it out and provide feedback. Eighteen friends took up this opportunity and used the application to test their personal Twitter accounts and various others and provide feedback. Once a prediction was made, there was a button on the results page that the user could press to bring them to a Google form to provide feedback on their experience. Below are the questions that were asked and their responses.

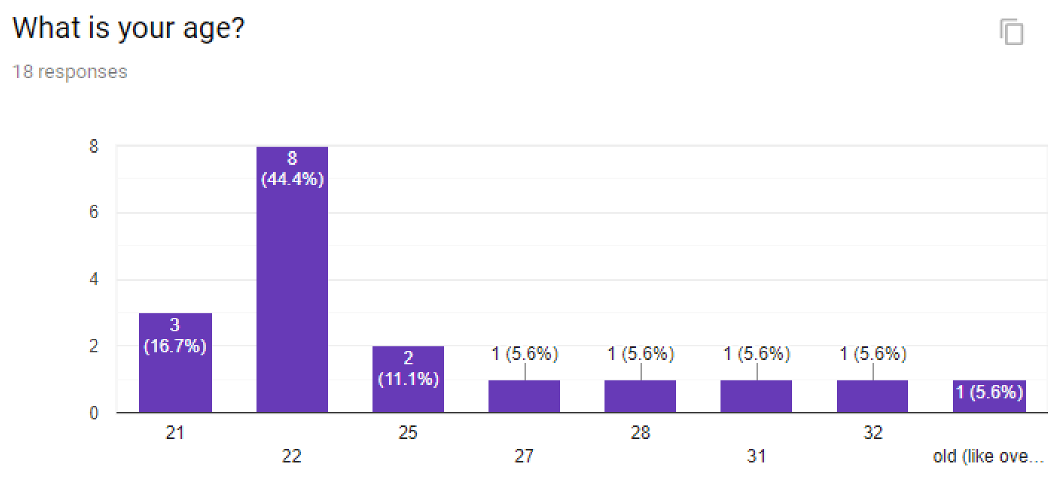


Figure 91 Feedback Question 1 Response

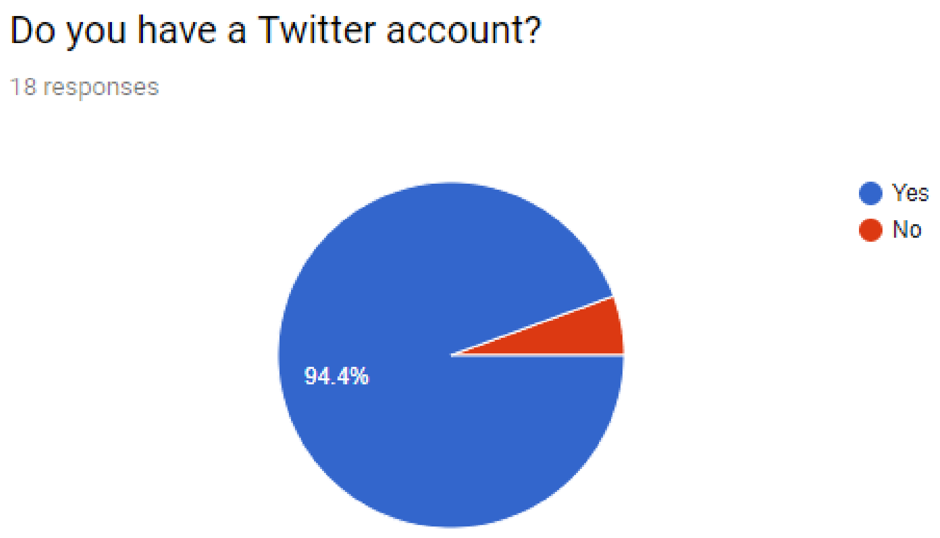


Figure 92 Feedback Question 2

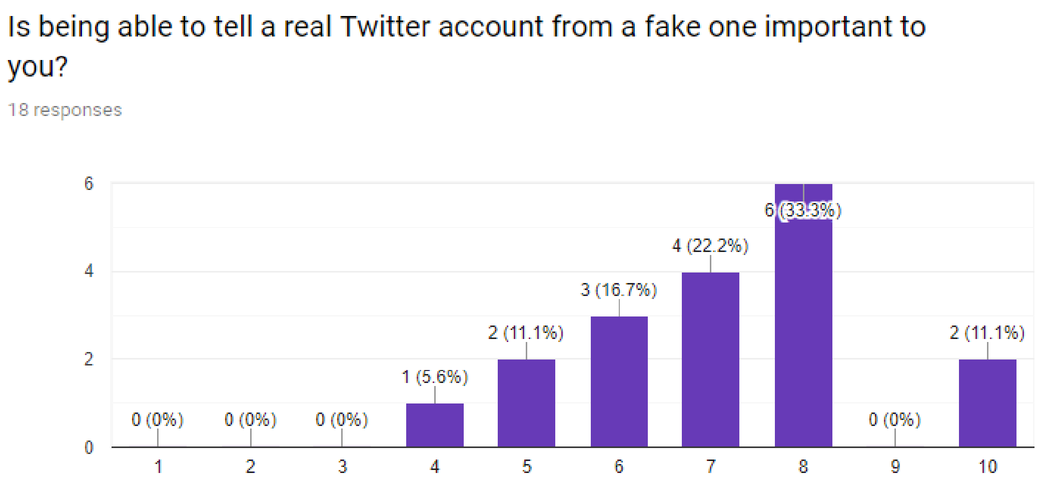


Figure 93 Feedback Question 3

The responses to these three questions show that out of the random sample of friends that used the application and gave feedback, all but one had a Twitter account, showing the platforms popularity, nearly all were in their twenties with a few hitting into the thirties and that the majority of them believe being able to tell whether a Twitter account is real or fake is an important ability showing the merit in undertaking a project on this issue.

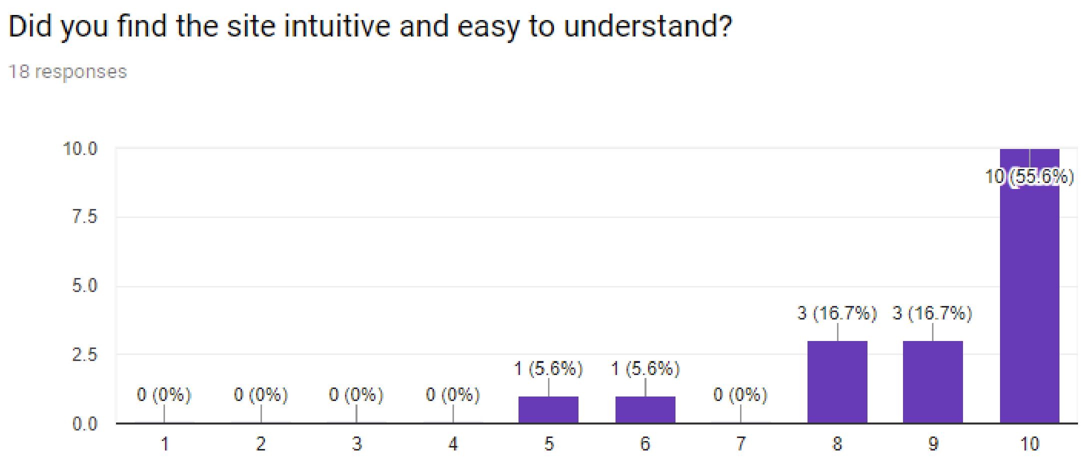


Figure 94 Feedback Question 4

An overwhelming majority of the users found the site intuitive and easy to understand which hits one of the key objectives of the design element of the web application. The two mid-range responses bear noting, as people's abilities and experiences with a system do differ and exploring all of these can lead to the evolution of any system into a better and more accessible version of itself.

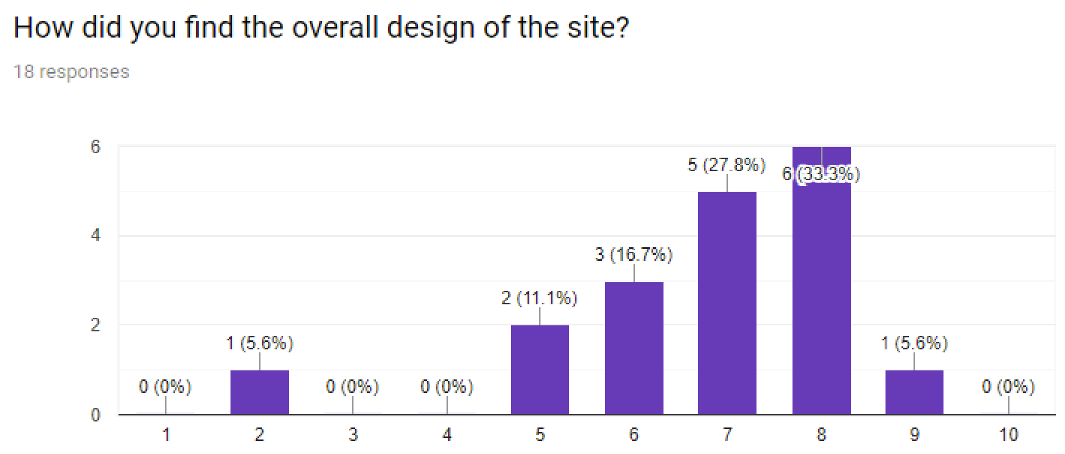


Figure 95 Feedback Question 5

The responses to the overall design, while not as overwhelmingly positive as the intuitiveness of the site, was still quite good with most of the users responding with either 7 or 8 out of 10. One user clearly was not impressed by the design of the site with some others only giving a 5 or 6. This is incredibly important to investigate as, while you can't make a design that absolutely everyone will like, you can do your utmost to design a site that aims to minimise the number of people who find it completely off-putting.

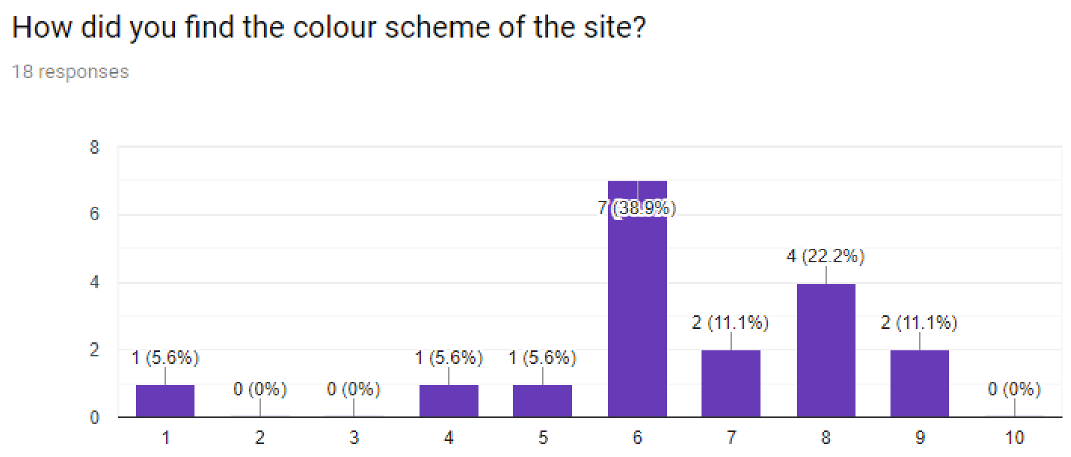


Figure 96 Feedback Question 6

The responses to the colour scheme of the site were again even more subdued, though still slightly positive, with most of them being in the 6-8 range, with most users giving a 6 out of 10. This meant that while people were satisfied with it there was definite room for experimentation and improvement.

When asked what they liked most about the site, the majority response was the simplicity and ease of use of the site with functionality and quick results being the remaining responses.

When asked what they liked least about the site the responses revolved around the lack of stats and information about what went into making a prediction, the size of icons, colour scheme and fonts used and lastly the overall UI design.

Next users were asked for suggestions to improve the website and while several responses understandably dealt with their answer to the previous question, others came out with some really interesting responses such as creating a Goggle Chrome extension version of the site that would run in conjunction with a user browsing through Twitter so any handle that comes up on screen would be checked and including a confidence level and more information with the result.

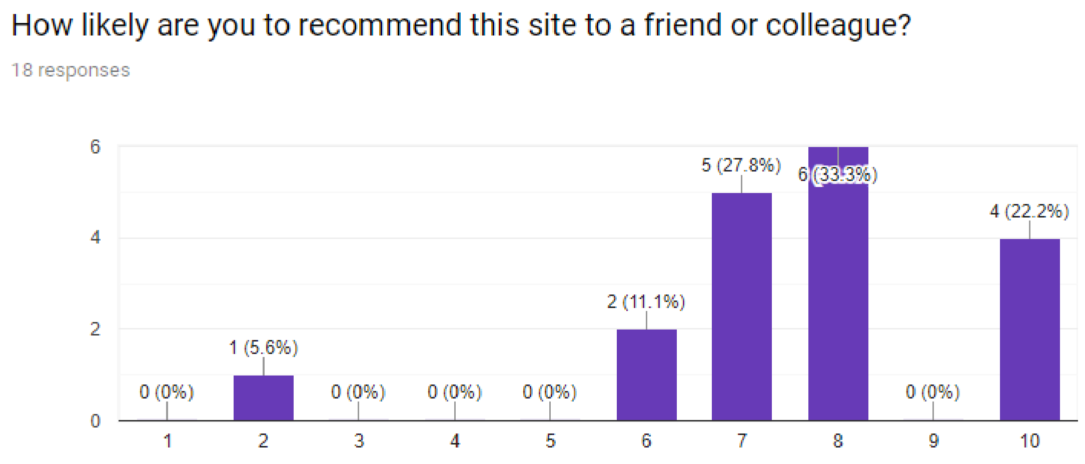


Figure 97 Feedback Question 10

Even with users finding certain parts lacking, most would recommend the site to a friend or colleague meaning that overall the site can be counted as a success!

This phase of testing proved extremely fruitful with plenty of informative and constructive feedback given that can be used to improve the site in a variety of ways although due to time constraints, the issues raised, and additional features requested will be dealt with in future iterations of the application.

6.3 Ad-Hoc Testing

Ad-hoc testing is a form of manual testing "*without any planning and documentation. The tests are conducted informally and randomly without any formal procedure or expected results*"[56]. Tests are run randomly and are expected to only be run once, unless bugs arise. While this quite an informal way of testing, it can be incredibly powerful in finding important bugs.

Due to detailed knowledge of the system is required due to the lack of documentation, the role of tester is nearly always taken up by a developer. They will normally perform this type of testing relatively soon after a piece of code has been written, to test the main functionality of that piece is working correctly before moving on to the next part.

This form of testing was used in both parts of the project. Within the data mining side, this was done to check values being read in from CSV files were of the correct type, rows were being correctly selected and returned from the database, features were being created properly to form an instance used in training and testing the machine learning models and the minimum Levenshtein distance was being correctly computed for each user account.

For the web application, the testing took the form of checking that the Twitter API calls were returning the right type of data, all the buttons that the user can interact with redirected correctly, that the correct information was shown on-screen when redirected to Facebook and Twitter, results were displayed correctly and once deployed, the site was accessible form other devices.

7 Project Plan

This chapter covers the creation of the initial project plan, what the final one looked and explanations for the differences and how Kanban boards were implemented to manage the project and all tasks and objectives.

7.1 Initial Plan

The initial plan can be seen below in Figure 98 in the form of a Gantt chart and includes all deadlines for the project including the vertical prototype and the dissertation. Analysis was done on each major task within the project, with estimated times of completion made and filled out in the chart with one block corresponds to one week. Background reading for example nearly reaches across the entire timespan of the project as there is always different subject matter to be researched or solutions looked up. The time estimates for each task were liberally made to try and consider unexpected delays, inexperience in certain domains and balancing the rest of the years' workload.

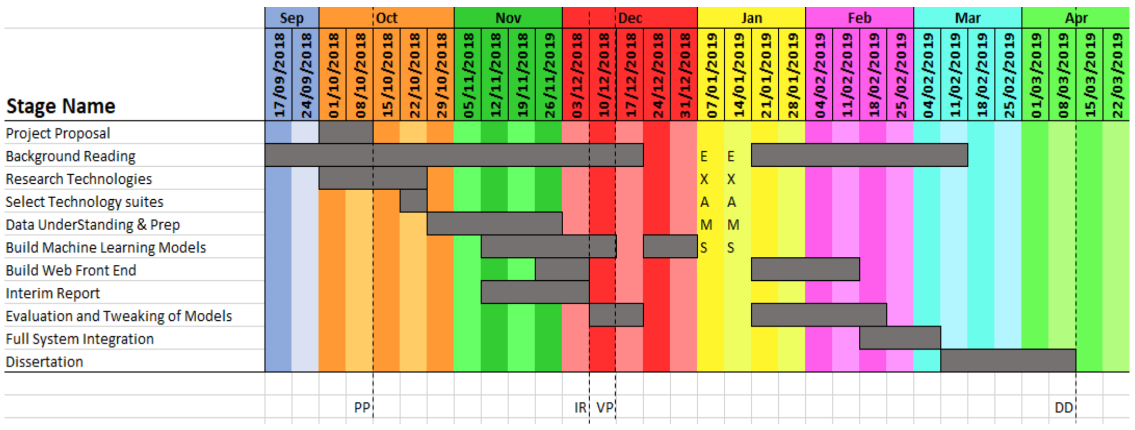


Figure 98 Initial Project Gantt Chart

7.2 Final Plan

The Final plan can be seen below in Figure 99 and shows a great deal of change in the second half of the timespan. This indicates that no matter how well you plan out something things can still go awry due to a variety of reasons. The main one in this instance is the fallout after finishing winter exams and how draining they can be.

It took a lot longer than originally anticipated to recuperate from them and once again have the mental drive to continue onwards with the project and even when this happened it was at a much slower pace. This meant that not only was there a delay in work but there was a decrease in the speed at which that work was getting done.

The evaluation and tweaking or experimentation of the machine learning models took longer to complete and work on the web front end didn't resume until the start of March. This added some additional pressure knowing that the final stages of the project were being entered but through support from certain family members and friends, the rate of work got not just back on track but increased past the normal rate to make up for the few weeks in limbo. This meant that everything was in high gear when it came time to close out the last few objectives of the project.

"Starting strong is good. Finishing strong is epic." [57]

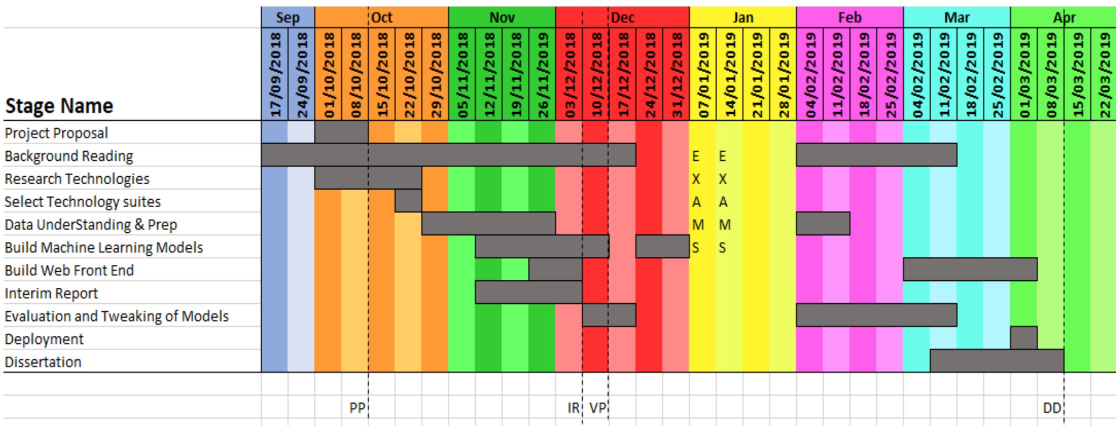
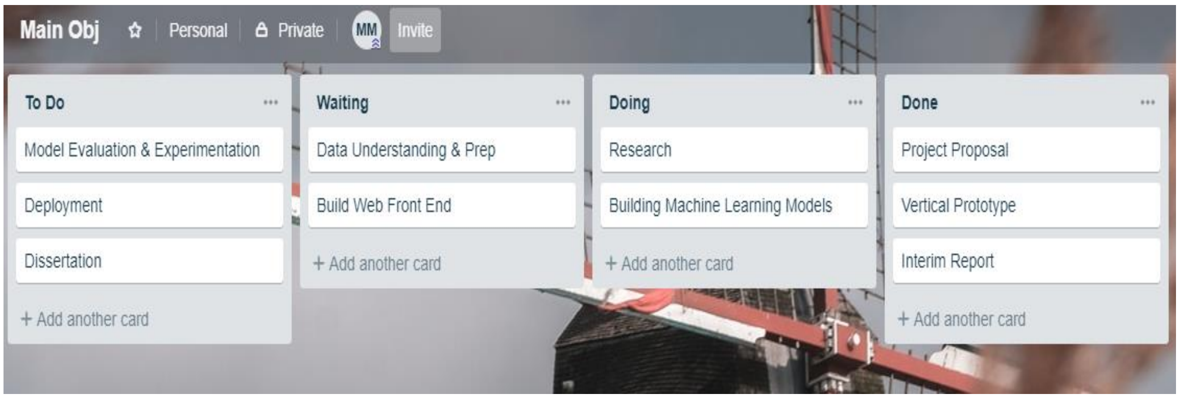


Figure 99 Final Project Gantt Chart

7.3 Kanban & Trello

As mentioned previously in chapter 3, the Kanban subdiscipline of the Agile methodology was chosen to provide day to day management of the project while keeping track of the overarching objectives. A Trello account was made and through that two Kanban boards created through it.

The first board, shown in Figure 100, was used to keep track of the major objectives of the project, helping to keep the overall project on track without focusing on the individual tasks needing completion. This would be used at the start of the day to reflect on what stage the project was at and help figure out what tasks to work on in the second board.


Main Obj | Personal | Private | MM | Invite

To Do
- Model Evaluation & Experimentation
- Deployment
- Dissertation
- + Add another card

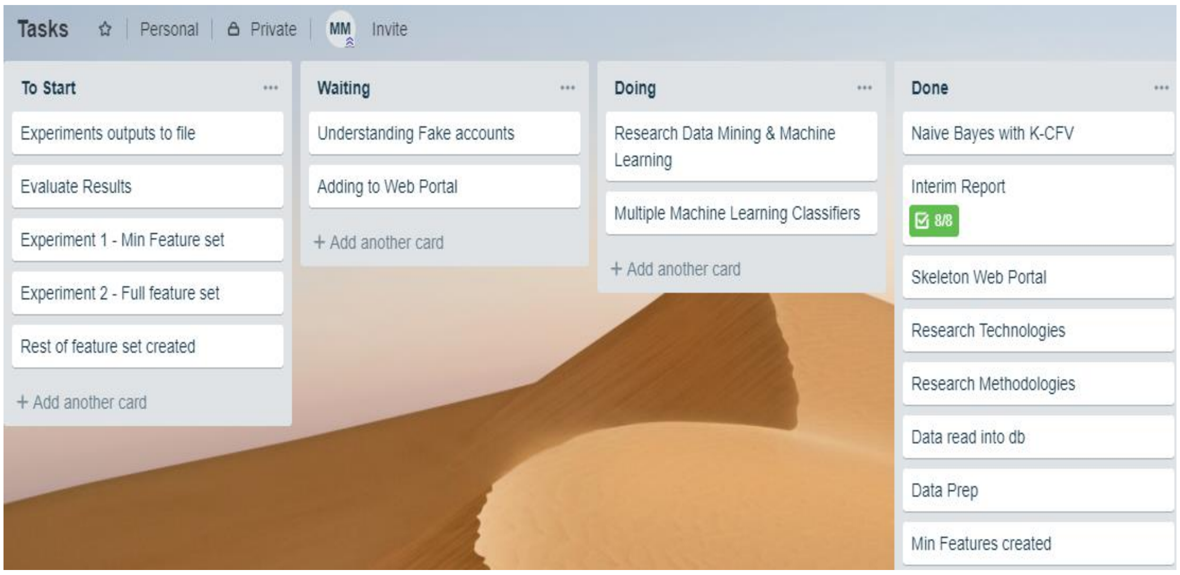
Waiting
- Data Understanding & Prep
- Build Web Front End
- + Add another card

Doing
- Research
- Building Machine Learning Models
- + Add another card

Done
- Project Proposal
- Vertical Prototype
- Interim Report
- + Add another card

Figure 100 Main Objectives Trello Board

The second board, shown in Figure 101, was used to for the day to day management of the project and had all the major objectives broken down into all their respective sub-tasks with some again broken down if they were too big. Every day the priority of all the tasks in the To-Do and Waiting columns were reassessed with new tasks being added throughout. Only two tasks were worked on at most at any one time, with tasks sometimes being moved back to Waiting so that other tasks could be completed. As tasks were completed, they were moved into the Done column and after a month archived from there to remove clutter on the board. This proved an incredibly effective way of managing the workflow for this project and even helped get everything back on track as mentioned previously as the focus was always on one or two tasks instead of the entire project looming over.


Tasks | Personal | Private | MM | Invite

To Start
- Experiments outputs to file
- Evaluate Results
- Experiment 1 - Min Feature set
- Experiment 2 - Full feature set
- Rest of feature set created
- + Add another card

Waiting
- Understanding Fake accounts
- Adding to Web Portal
- + Add another card

Doing
- Research Data Mining & Machine Learning
- Multiple Machine Learning Classifiers
- + Add another card

Done
- Naive Bayes with K-CFV
- Interim Report ✓ 8/8
- Skeleton Web Portal
- Research Technologies
- Research Methodologies
- Data read into db
- Data Prep
- Min Features created

Figure 101 Tasks Trello Board

8 Conclusions & Future Work

This chapter wraps the project up, giving reflections on the project and the personal growth achieved and lessons learned from start to finish and detailing several ideas for future iterations of the project, both for the data mining application and the web application.

8.1 Conclusions

8.1.1 Personal

This project has had its many highs and lows spread across the entire timespan. Some of the highs have directly followed some of the lows as a wall is hit during the completion of a specific task and attempt after attempt fails to solve the issue, leading to incredible frustration and a slight sense of hopelessness until finally a solution is found or thought of leading to a complete swing in mood and outlook as one feels nothing can stop you now.

Other highs have been simply making weekly progress with certain stages of the project and the feeling this consistent output of work gives. One of the main lows, as discussed when evaluating the project plan, was how long it took from the end of winter exams to get back fully into working on the project although this experience in the end will no doubt prove invaluable moving forward.

So much has been learned from every obstacle overcome, every topic researched, and every stage of the project completed that it seems weird to think back to the start of the project and compare how personal knowledge levels from then to now. All of this has contributed to a massive surge in personal growth that when combined with the experiences of work placement ensure that everything is in close to perfect order to make the leap from college to the workforce in a smooth and quick manner.

8.1.2 Project

The project itself has shown how numerous different disciplines, methodologies and technologies can come together to achieve nearly any realistic, even some not so realistic, goal a person or group can come up with, albeit also reliant on that person or groups various skills too.

For example, while Python was used for both the data mining application and the web application, it could have been substituted with R for the former and Java for the latter if someone preferred or needed to work with those languages. Next the CRISP-DM and Kanban project management methodologies could have been replaced with the SEMMA and Scrum methodologies with little disruption to the overall project.

The overall outcome of the project was rather satisfactory, as while some parts could have been implemented better, all the main objectives of the project were hit. The data mining application proved incredibly fruitful with a high level of accuracy achieved when making predictions about Twitter accounts. The web application did everything that was required of it although looking back, this is one part of the project that should have been expanded upon with more features implemented.

There was plenty of good feedback given during testing and self-analysis done once development was finished that means there are various ideas for the project in the future past college if so inclined and these are explored next.

8.2 Future Work

8.2.1 Data Mining Application

8.2.1.1 Find More Data

One of the most important and fundamental parts of the entire project was the procurement of relevant Twitter account data to be used to design accurate machine learning models in the classification of bot accounts. While the dataset used proved extremely helpful and up to the task, within the scope of the project, it is two years old and only contains data on approx. twelve thousand and five hundred accounts.

Being able to add to this dataset with more labelled Twitter accounts, both real and bot, would help with identifying them as some patterns may have been missed within the original dataset. Unless others have already done this and made their dataset available for public use, this would require a significant amount of work, as accounts would need to be chosen, labelled correctly and have their details and tweets saved.

The main issue is labelling them correctly as while the models built and used in this project are largely accurate, any inaccurate predictions made and used in an extended dataset would lead to further inaccurate models being made. The method used to label the original dataset would need to be investigated for use here.

8.2.1.2 Identify New Types of Bots

Completing the previous task would then potentially allow for the identification of new types of bot accounts that were either not identified back when the original dataset was being created or have arisen in the meantime.

These could be the same types of bot accounts that have had their parameters changed in a way such that their behaviour has changed significantly and as such can be classified as their own type or completely brand-new types of bot accounts that have been engineered with various new techniques to masquerade even more efficiently as human and as such pose even more of a threat to society as a whole.

Due to the ever-changing nature of technology and in this instance rise of new types of bot accounts, this is an especially important task as being able to at least keep up somewhat with this pace and identify them allows for Twitter users to be able to know more often than not when they are interacting with these new bot types.

8.2.1.3 Changes to Feature set

One of the main areas that would help with being able to better identify a real person, the known types of bots or any potential new types of bots identified in the previous task, would be to look at the set of features derived from user data and used to train and test the machine learning models.

Through further investigation and new insights gained from the previous two tasks, the feature set could be modified by either adding, removing or changing some of the features within it. These could range from simple tweaks to some of the ratios used between friend and follower counts of a user to more in-depth text mining on the users tweets and even looking at how if at all various bot accounts have interacted with each other and is there anything with those relationships that could be used to help better identify them.

8.2.2 Web Application

8.2.2.1 Better UI

One of the main points taken away from the usability testing was that the UI was a bit plain and that the colour scheme could do with a slight change. Moving forward, the UI could be redesigned to take this and a few other minor response into consideration, leading to a much better layout and user experience. As UI design is not a strong point, a course on it would need to be taken to be able to rectify this area of inadequate skill and fully implement the changes in a way that will satisfy the userbase.

One part of the redesign would be to give an in-depth explanation of the predictions made around an account. This would take the form of what features are used in making the prediction, how they relate to the type of account that is predicted and which ones the predicted account satisfies.

8.2.2.2 Chrome Extension

One brilliant suggestion made be a user during their feedback on the application was to create a Google Chrome extension version of the application that would run in parallel with a user browsing Twitter.

This would mean that as a user was browsing either their own or somebody else's Twitter feed, the extension would make predictions about all the account handles on screen, displaying the results in small pop-ups beside them that could be opened to see more information.

Several interesting settings could be implemented so that the user can tailor their experience, such as the ability to ignore real account results to avoid some possible clutter or to only run on the handles of authors of a tweet, avoiding any handles contained in the tweet.

8.2.2.3 Look at Friend and Follower accounts

A feature that could be added to website would be the ability to not just make a prediction on an account but also look at all accounts that account follows and is followed by and have predictions made on them. The results for these could then be presented via various types of charts such as a Pie chart or a Bar chart showing the totals of the different types of accounts predicted and then the user can drilldown and see the result for each individual account.

This would prefer immensely helpful for a person checking their own account as it would illuminate whether they interact with any bots or for someone checking an account to see has it paid for several follower boosting accounts to see if they really are as influential as they appear to be.

8.2.2.4 Mobile Application

One task that would be done in the future is to create a simple mobile application version of the web application that would also incorporate any changes made to the latter in the previous two task. This would allow mobile users to quickly check a Twitter account by opening the app on their phone instead of having to open a browser's app and then navigate to the web page before being able to check an account.

It would have some inherent design differences from the web application as there are different design systems to follow when creating a mobile app compared with a web application. The mobile app would also be created independently of the web application as Python is not suited for this so Android Studio would be used. How the results from the data mining application is passed to and used by the mobile app would also need to be taken into consideration.

9 Bibliography

[1] Statista; (October 2018), Twitter: number of active users 2010-2018, www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/, Date Accessed: November 2018

[2] Varol, Onur; Ferrara, Emilio; Davis, Clayton A.; Menczer, Filippo; Flammini, Alessandro; (2018) "*Online Human-Bot Interactions: Detection, Estimation, and Characterization*", ICWSM'17, Page 1-9.

[3] Perlroth, Nicole; (April 2013), Fake Twitter Followers Become Multimillion-Dollar Business, bits.blogs.nytimes.com/2013/04/05/fake-twitter-followers-becomes-multimillion-dollar-business, Date Accessed: October 2018

[4] Perlroth, Nicole; (April 2013), Researchers Call Out Twitter Celebrities With Suspicious Followings, bits.blogs.nytimes.com/2013/04/25/researchers-call-out-twitter-celebrities-with-suspicious-followings, Date Accessed: October 2018

[5] Ferrara, Emilio; (November 2016), How Twitter bots affected the US presidential campaign, theconversation.com/how-twitter-bots-affected-the-us-presidential-campaign-68406, Date Accessed: October 2018

[6] Nimmo, Ben; (August 2017), #BotSpot: Twelve Ways to Spot a Bot, medium.com/dfrlab/botspot-twelve-ways-to-spot-a-bot-aedc7d9c110c, Date Accessed: October 2018

[7] Efthimion, Phillip George; Payne, Scott; Proferes, Nicholas; (2018), "*Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots*", SMU Data Science Review, 1(2), Article 5.

[8] Rushe, Dominic; (October 2017), Twitter bans ads from RT and Sputnik over election interference, www.theguardian.com/technology/2017/oct/26/twitter-bans-ads-from-russia-today-and-sputnik-over-election-interference, Date Accessed: October 2018

[9] BBC news; (July 2018), Twitter 'shuts down millions of fake accounts', www.bbc.com/news/technology-44682354, Date Accessed: October 2018

[10] Leskin, Paige; (October 2018), Twitter shuts down bots pushing pro-Saudi reports on missing columnist, uk.businessinsider.com/twitter-shuts-down-pro-saudi-bots-missing-columnist-2018-10?r=US&IR=T, Date Accessed: November 2018

[11] Netimperative; (November 2018), US elections: Twitter shuts down 10,000 bot accounts 'discouraging voting', www.netimperative.com/2018/11/us-elections-

twitter-shuts-down-10000-bot-accounts-discouraging-voting/, Date Accessed: November 2018

[12] Sattler, Jason; (September 2017), blog.f-secure.com/4-reasons-so-hard-for-twitter-to-shut-down-bots/, Date Accessed: October 2018

[13] OSoMe; (-), Botometer by OSoMe, botometer.iuni.iu.edu/#!/faq, Date Accessed: October 2018

[14] Brady, Alex; (2018), "Fantasy Premier League Predictive Analytics", Dublin: D.I.T.

[15] Jennings, Sean, 2017, "CrimAnalytics: A Crime Prediction Web Application", Dublin: D.I.T.

[16] O'Neill, Shane; (2017), "Anti-Bullying with Machine Learning", Dublin: D.I.T.

[17] R; (-), R: The R Project for Statistical Computing, www.r-project.org, Date Accessed: October 2018

[18] Python; (-), Welcome to Python.org, www.python.org, Date Accessed: October 2018

[19] JetBrains; (-), PyCharm: the Python IDE for Professional Developers by JetBrains, www.jetbrains.com/pycharm, Date Accessed: October 2018

[20] Scikit-learn; (-), scikit-learn: machine learning in Python, scikit-learn.org/stable/, Date Accessed: October 2018

[21] Pandas; (-), Python Data Analysis Library, pandas.pydata.org, Date Accessed: October 2018

[22] NumPy; (-), NumPy -NumPy, www.numpy.org, Date Accessed: October 2018

[23] Ronacher, Armin; (-), Welcome | Flask (A Python Microframework), flask.pocoo.org, Date Accessed: October 2018

[24] The Django Software Foundation; (-), The Web framework for perfectionists with deadlines, www.djangoproject.com, Date Accessed: October 2018

[25] Tweepy; (-), Tweepy Documentation, tweepy.readthedocs.io/en/v3.5.0/index.html, Date Accessed: October 2018

[26] The Apache Software Foundation; (-), Welcome! – The Apache HTTP Server Project, httpd.apache.org, Date Accessed: November 2018

[27] Heroku; (-), Cloud Application Platform, www.heroku.com, Date Accessed: November 2018

[28] Amazon; (-), Amazon Web Services (AWS) – Cloud Computing Services, aws.amazon.com, Date Accessed: November 2018

[29] Git; (-), Git, git-scm.com, Date Accessed: November 2018

[30] Mercurial; (-), Mercurial SCM, www.mercurial-scm.org, Date Accessed: November 2018

[31] Oracle; (-), MySQL, www.mysql.com, Date Accessed: October 2018

[32] PostgreSQL; (-), PostgreSQL: The world's most advanced open source database, www.postgresql.org, Date Accessed: October 2018

[33] MongoDB; (-), Open Source Document Database, www.mongodb.com, Date Accessed: October 2018

[34] Mauro, Andrea De; Greco, Marco; Grimaldi, Michele; (2015), "*What is big data? A consensual definition and a review of key research topics*", AIP conference proceedings, 1644(1), pp. 97–104.

[35] Bramer, Max; (2016), "*Principles of Data Mining*", Salmon Tower Building New York City: Springer.

[36] Kelleher, John; Mac Namee, Brian; D'Arcy, Aoife; (2015), "*Fundamentals of machine learning for predictive data analysis*", Cambridge Massachusetts: MIT Press.

[37] Agatonovic-Kustrin, S; Beresford, R; (2000), "*Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research*", J Pharm Biomed Anal., 22(5), pp.717-27.

[38] Reingold, Eyal; (-), Training an Artificial Neural Network, www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural3.html, Date Accessed: January 2019

[39] Choudhary, Pramit; (Feb 2017), Introduction to Anomaly Detection, www.datascience.com/blog/python-anomaly-detection, Date Accessed: October 2018

[40] Botometer; (-), Bot Repository, botometer.iuni.iu.edu/bot-repository/datasets.html, Date Accessed: October 2018

[41] Martin, Robert C.; (2013) "*Agile Software Development, Principles, Patterns, and Practices*", London, Pearson.

[42] Manifesto Authors; (2001), Manifesto for Agile Software Development, agilemanifesto.org, Date Accessed: October 2018

[43] Anderson, David J.; (2010), "*Kanban: Successful Evolutionary Change for Your Technology Business*", Seattle, Blue Hole Press

[44] Toyota; (2004), Toyota Traditions, www.toyota-global.com/company/toyota_traditions/quality/mar_apr_2004.html, Date Accessed: October 2018

[45] Atlassian; (-), Trello, trello.com/en, Date Accessed: October 2018

[46] Wirth, Rüdiger; Hipp, Jochen; (2000), "*CRISP-DM: Towards a Standard Process Model for Data Mining*".

[47] SAS; (-), Data Mining Using SAS® Enterprise Miner™: A Case Study Approach, Third Edition, support.sas.com/documentation/cdl/en/emcs/66392/HTML/default/viewer.htm, Date Accessed: October 2018

[48] Wernigerode, Alex; (-), S4 – Data Mining Flash Cards, quizlet.com/256689355/s4-data-mining-flash-cards/, Date Accessed: October 2018

[49] The Django Book; (2018), The Model-View-Controller Design Pattern, djangobook.com/model-view-controller-design-pattern/, Date Accessed: October 2018

[50] Cresi, S.; Di Pietro, R.; Petrocchi, M.; Spognardi, A.; & Tesconi, M.; (2015), "*Fame for sale: Efficient detection of fake Twitter followers",* Decision Support Systems, 80, 56-71.

[51] Gramlich, John; (April 2018), How we identified bots on Twitter - Pew Research Center, www.pewresearch.org/fact-tank/2018/04/19/qa-how-pew-research-center-identified-bots-on-twitter/, Date Accessed: October 2018

[52] Symantec Security Team; (Oct 2018), How to Spot a Twitter Bot, www.symantec.com/blogs/election-security/spot-twitter-bot, Date Accessed: October 2018

[53] Twitter Developers; (2018), Twitter Developer Platform, developer.twitter.com/content/developer-twitter/en.html, Date Accessed: October 2018

[54] Twython; (2019), Twython – Documentation, twython.readthedocs.io/en/latest/, Date Accessed: March 2019

[55] AWS; (2019), The Elastic Beanstalk Command Line Interface (EB CLI), docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html, Date Accessed: March 2019

[56] Software Testing Fundamentals; (-), Ad hoc Testing, softwaretestingfundamentals.com/ad-hoc-testing/, Date Accessed: January 2019

[57] Sharma, Robin; (Apr 2014), Robin Sharma on Twitter, twitter.com/RobinSharma/status/455646692453281792, Date Accessed: April 2019