

# Detecting Twitter Bots and Twitter Textual Posts Generated with ChatGPT 3.5 Using Machine Learning Models



UNIVERSITY OF  
LINCOLN

Kleybson Albuquerque Rodrigues De Sousa



School of Computer Science  
College of Health & Science  
University of Lincoln

Submitted in partial fulfilment of the requirements for the  
Degree of BSc (Hons) Computer Science

*Supervisor* Dr Vassilis Cutsuridis

May 2024

# Acknowledgements

I am grateful to my supervisor, Dr. Vassilis Cutsuridis, for his invaluable guidance and support during this project. I also appreciate the friendship and support of Dr. Mike Wright, as well as the support of Dr. Craig Green.

I would like to thank all my friends and family for their support throughout the development of this project, especially my mother Maria Da Gloria Albuquerque Silva, whom I have always drawn inspiration from, your love and support have meant a lot to me.

BY Kleybson Sousa

# Abstract

Social media platforms, especially Twitter, have become an essential part of modern communication and information sharing, playing a significant role in shaping public opinion and influencing real-world events. However, the presence of social media bots is a significant challenge, as they can spread misinformation and manipulate narratives for various agendas. In this paper, I propose a solution to detect Twitter Bot accounts and distinguish between ChatGPT-generated text on the platform.

To develop this solution, I analyzed previous methodologies and models aimed at bot detection and text classification. I identified limitations in prior approaches and proposed a novel two-model solution. The first model uses the BERT model for word embeddings and several models such as Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Random Forest for text classification. I compared their results to determine their effectiveness.

Additionally, I implemented other models, such as Random Forest, Decision Trees, and SVM, to detect Twitter bots by exclusively looking at the Accounts' details. I compared their performance and evaluated the impact of the dataset on their classification.

# Table of Contents

Introduction	1
1.1 Introduction	1
1.1.1 The Current Environment	1
1.1.2 Proposed Model	3
1.2 Dissertation Structure	5
1.3 Aims & Objective	6
1.3.1 To create a model for the identification of ChatGPT 3.5 Generated textual posts on Twitter (x.com)	6
1.3.2 Create an ML model for the identification of bot accounts on Twitter.	6
Literature Review	7
2.1 Machine Learning	8
2.2 ChatGPT	9
2.3 Twitter Bot Detection Models	10
Requirements Analysis	18
3.1. Requirement Analysis	18
3.1.1. Functional Requirements	18
3.1.2. Non-Functional Requirements	19
3.2. Risk Analysis	19
3.2.1. Project Scope Risk	19
3.1.1. Data Quality and Bias – Bias and Incomplete data	19
3.1.2. Explainability –Frameworks for Model Interpretation	20
3.1.3. Model Overfitting and Underfitting	20
Design & Methodology	21
4.1. Project Management	21
4.2. Toolset and Machine Environment	24

4.1.	Dataset	25
4.1.1.	Dataset Generation	25
4.1.1.	Dataset Cleaning	28
4.2.	Word Embeddings	33
4.3.	Algorithm/Model Design Selection	34
4.3.1.	KNN	34
4.3.2.	SVM	35
4.3.3.	Decision Trees	35
4.3.4.	Random Forest	36
4.4.	Hyperparameter Tuning	37
4.5.	Train-Test Split Testing Methodology	37
4.6.	K-Fold Stratified Cross Validation	38
4.7.	Performance Metrics	39
	Implementation	42
5.1.	ChatGPT Identification Model	42
5.2.	Bot Account Identification Model	48
	Results & Discussion	53
6.1.	ChatGPT Identification Model Results	53
6.2.	Bot Account Identification Model	63
6.3.	Discussion	67
	Conclusion	68
	References	70
	Appendix A	75

# List of Figures

Figure 1 - Models Pipeline Diagram .....	5
Figure 2 - An Example of Using ChatGPT to Generate A Textual Post.....	10
Figure 3 - (Benevenuto et al., 2010) Model Output Prediction .....	10
Figure 4 - (Davis et al., 2016) Model Output Prediction .....	11
Figure 5 - Bot detection model proposed by (Heidari, Jones and Uzuner, 2022) .....	12
Figure 6 - An Example of How Embeddings are Created (www.cs.cmu.edu, n.d.) .....	14
Figure 7 - Illustration of the BOTLE model developed by (Feng and Uyen Trang Nguyen, 2023). .....	15
Figure 8 - Performance Comparison of BOTLE against similar models using the Cresci 2017 dataset. .....	15
Figure 9 - Model Pipeline Diagram presented (Pinnapureddy Manasa, Malik and Batra, 2024) .	16
Figure 10 - A Small Sample of The Gantt Chart Created .....	21
Figure 11 - Samples of the Note Taken During the Development of The Project .....	22
Figure 12 - A Sample of The Weekly Working Schedule .....	23
Figure 13 - An Example of How the New Dataset Will Be Formed .....	26
Figure 14 - An Example of How-to User ChatGPT API to Generate the Data .....	27
Figure 15 - Code Snippet Showing how the Messages were formatted .....	27
Figure 16 - An Example of How the KNN model works (Anil Gokte, n.d.) .....	34
Figure 17 - An Example of How SVM Models Work .....	35
Figure 18 - An Example of the Decision Trees Model .....	36
Figure 19 - An Example of Random Forest Model (Koehrsen, 2020) .....	36
Figure 20 - An Example of Hyperparameter Tuning with GridSearchCV (Stalfort, 2019) .....	37
Figure 21 - An Example of How Stratified Cross-Validation Folds Are Split .....	39
Figure 22 - An example of Confusion Matrices .....	41
Figure 23 - Code Snippet Showing Libraries imported.....	43
Figure 24 - Example of How to Use the BERT Model .....	43
Figure 25 - Example of preparing the data for training the models .....	43

Figure 26 - Code Snippet Showing the Creation of the Embeddings .....	44
Figure 27 - Code Snippet Showing the Usage of train_test_split .....	45
Figure 28 - An example of how to use GridSearchCV .....	45
Figure 29 - Best Parameters Found .....	45
Figure 30 - Code Snippet of The SVM Training.....	46
Figure 31 - Code Snippet Showing the training of KNN model .....	46
Figure 32 - Code Snippet Showing the Training of the RandomForest Model.....	47
Figure 33 - Creation of Pandas Dataframe capable of storing the results .....	47
Figure 34 - An example of how the models were Cross Validated.....	48
Figure 35 - Code Snippet Showing the libraries used .....	49
Figure 36 - Code Snippet displaying the further processing of the dataset for the second model .....	49
Figure 37 - An Example of how StandardScaler was used .....	49
Figure 38 - An Example of Using GridSearchCV for SVM .....	50
Figure 39 - An Example of Using GridSearchCV for DecisionTree .....	50
Figure 40 - An Example of Using GridSearchCV for RandomForest .....	50
Figure 41 - Code Snippet for the training of the RandomForest Model .....	51
Figure 42 - Code Snippet for the training of the DecisionTree Model .....	51
Figure 43 - Code Snippet for the training of the SVM model.....	51
Figure 44 - An example of how cross-validation was performed on the different models .....	52
Figure 45 - Plot of the Dataset Distribution for 1k subset .....	53
Figure 46 - Plot of the Dataset Distribution for 10k subset .....	54
Figure 47 - Plot of the Dataset Distribution for 100k subset .....	54
Figure 48 - An Example of The decision boundary, using the 1k subset.....	55
Figure 49 - SVM Model's Confusion Matrix .....	55
Figure 50 - KNN Model's Confusion Matrix.....	56
Figure 51 - RandomForest Model's Confusion Matrix .....	56
Figure 52 - 80/20 SVM Model's Confusion Matrix.....	57
Figure 53 - 80/20 KNN Model's Confusion Matrix .....	58
Figure 54 - 80/20 RandomForest Model's Confusion Matrix.....	58

Figure 55 - Random Forest Confusion Matrix .....	65
Figure 56 - Decision Tree Confusion Matrix .....	66
Figure 57 - SVM Confusion Matrix .....	67

BY Kleybson Sousa



# List of Tables

Table 1 - Models' Expected Outcome Predictions .....	4
Table 2 - Model's Performance Using 60/40 Split.....	60
Table 3 - Model's Performance Using 70/30 Split .....	61
Table 4 - Model's Performance Using 80/20 Split .....	62
Table 5 - Model's Performance Using 90/10 Split .....	63
Table 6 - Model 2 Results .....	64

# Chapter 1

## Introduction

This introduction builds on top of the Interim report previously completed as part of the development of this project (Albuquerque Rodrigues De Sousa, 2024a)

### 1.1 Introduction

#### 1.1.1 The Current Environment

Social Media platforms have become fundamental parts of our lives. Their nature of the free exchange of information and ideas, allowed for the development of an environment of quick dissemination of information and opinions. Twitter is one such platform, created in 2006 by Jack Dorsey as a form of microblog for celebrities and news organisations, twitter allows users to post textual posts of up to 280 characters for non-subscribers or up to 25.000 characters for those who pay a monthly subscription, as well as pics, videos and links to external websites. In the UK, 43% of internet users are active on Twitter every month. (Statista, 2022)

Over the past two decades, Twitter has become a platform exclusively for debate and distribution of information, acquired by billionaire Elon Musk for 44 billion dollars (Murphy et al., 2022). Twitter, among other social media platforms, has gained the attention of many researchers and experts for its capability to quickly and easily disseminate real-time information to large audiences. Its new owner has renewed the worries of researchers who wonder how it could be used to manipulate public opinions. As shown by (Ante, 2023), Musk's opinions can be highly influential.

The concerns of researchers and experts regarding social media's influence are legitimate. Over the past decades, these platforms' influences have had a significant impact on real-world events. The Egyptian Revolution of 2011 has been widely studied

as an event that had a great influence derived from social media platforms. (Attia et al., 2011) concluded that “All these social network variables facilitated the formation of a positive attitude towards political change among Egyptian youth, which gave them positive feelings toward participating in the uprising and revolting”. Furthermore, (Tudoroiu, 2014) argues that “social media served as an instrument of domestic and international revolutionary contagion; and, critically, as a means of enhancing pan-Arab consciousness which, in turn, was fertile soil for that contagion”. Most recently, during the COVID-19 pandemic, social media platforms demonstrated once more their power of influence over today’s society. According to the research (Gallotti et al., 2020), those platforms were widely used during the pandemic to propagate unreliable information among users. According to the research (Rocha et al., 2021), we can conclude that the dissemination of fake and false information created, during the pandemic, an environment of distrust in our democratic institutions such as the elected Government as well as in the World Health Organisation and other international health organisations. This has caused a direct impact on people’s lives, from increased levels of anxiety among the population to the death of those most vulnerable to the virus (Morel, 2021).

As it becomes clear the influence social media platforms can have on our day-to-day lives, it also becomes apparent that those platforms can be used as a tool by third-party actors interested in manipulating public opinion in one way or another, being that for financial gains or political gains. Often, this is done through the usage of automated accounts, widely known as Bot accounts, those accounts are generally easy to create and can be used for a variety of purposes, for instance, to generate fake engagement automatically or by a command to like or share someone’s post on Twitter – those bots are known as “like bots”. They can also be used as a tool to disseminate false and unreliable information. According to the research (Ferrara, 2020), during the 2020 US Election, Twitter bots were used to disseminate “hyper-partisan and conspiratorial websites”. This further produces what is known as social bubbles - when individuals are exposed to information or opinions that either align with their own beliefs or are based on their recent activity. This exposure increases the chances of confirmation bias – When an individual favours information that confirms their beliefs, despite contradictory

evidence (Spohr, 2017). Additionally, (Zhang, Ma and Fang, 2024) further demonstrate how bots can be used as an effective tool to influence public opinion.

The rise of Artificial Intelligence, more specifically Natural Processing Language (NLP) models such as OpenAI's ChatGPT, has allowed for the development of newer more advanced bot accounts. These new sophisticated iterations of bots have taken advantage of those technologies to produce highly convincing content, content that is more capable of effectively mimicking human behaviours. This has raised newer challenges to our current bot detection models. (Ferrara, 2023) presents an in-depth review of our current bot detection models and the challenges that these new advanced bots present, based on this research, we can conclude that a new modern approach to bot detection is needed. This new approach needs to be effective at detecting ChatGPT-generated textual posts while also being able to look at the account's available information, this will allow it to be able to detect modern bot accounts with ChatGPT-generated posts accurately.

### **1.1.2 Proposed Model**

As I have demonstrated in the previous section, it's imperative nowadays for any bot detection tool to be able to detect bots that use AI-generated textual posts to avoid our current bot detection approaches.

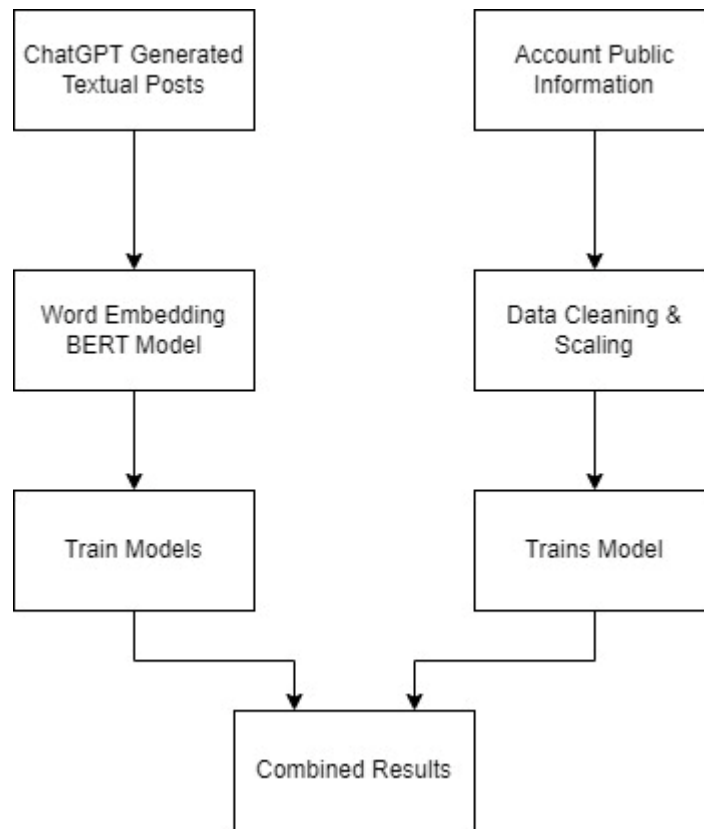
In this project, I propose the creation of a new bot detection method that looks at both the account details as well as the textual posts in order to determine whether an account is automated (bot) or not. I aim to create 2 new machine learning models: One for the identification of Twitter bot accounts based on the account's information such as age, gender, and location, and another for the identification of textual posts generated using ChatGPT 3.5.

A classification model similar to (Heidari, Jones and Uzuner, 2022) will be used to classify the accounts, with the posts. The models will output the results as follows: Human Account and Bot Account with a further 2 subclasses each to define if the post is ChatGPT generated or not, those will be called ChatGPT Generated Post and ChatGPT Not Detected.

	Twitter Account	Twitter Posts
	Bot Account	-
<b>Model 1</b>	Human Account	-
<b>Model 2</b>	-	ChatGPT Generated Post
	-	ChatGPT Not Detected

*Table 1 - Models' Expected Outcome Predictions*

Unlike (Heidari, Jones and Uzuner, 2022), the first model won't divide each information into categories. The second model will focus on identifying if the post was ChatGPT generated. A classification model will be used to classify the different texts, similarly to (Mitrović, Andreoletti and Ayoub, 2023), open-source models and algorithms such as XGBoost could be used and adapted for my solution. There is also the possibility of using Neural Networks for both classification problems. Everything will be created using Python and Python's libraries.



*Figure 1 - Models Pipeline Diagram*

## 1.2 Dissertation Structure

This Dissertation has been structured as follows:

- **Literature Review** – A review of current and relevant academic literature, along with a proposed model
- **Requirement Analysis** – Desired outcomes
- **Design & Methodology** – A description of how the models will be built.
- **Implementation** – How the project has been built.
- **Results & discussion** – A review of the outcomes
- **Conclusion** – A critical analysis of the entire project development
- **References** – A list of references in Harvard style.
- **Appendices** – Additional Content

## **1.3 Aims & Objective**

### **1.3.1 To create a model for the identification of ChatGPT 3.5 Generated textual posts on Twitter (x.com)**

#### **Specific**

- Develop an ML model to identify ChatGPT-generated textual posts on Twitter (x.com)
- Be able to differentiate between human and ChatGPT textual posts.
- Produce a binary classification model; Posts to be classified as ChatGPT-generated or not.

#### **Measurable**

- Achieve an accuracy of at least 90% for correctly classifying texts as ChatGPT generated.

#### **Achievable**

- Create a new dataset for the training and testing of the new model.
- Explore similar open-source models, including pre-trained models.
- Develop the model using Python and its libraries.

#### **Relevant**

- Relevant for content moderation and integrity of social media platforms

#### **Time-Bound**

- Initial model to be developed by March 2024
- Final model and completion of the project by April 2024

### **1.3.2 Create an ML model for the identification of bot accounts on Twitter.**

#### **Specific**

- Develop an ML model to identify bot accounts on Twitter.
- The model needs to be able to identify bot accounts based on their public information.
- The model can be binary; the Accounts are to be classified as a Bot or not Bot.

#### **Measurable**

- The model is to correctly identify accounts with a minimum accuracy of 90%.

#### **Achievable**

- Use a publicly available dataset for the training and testing of the model.
- Explore different approaches and open-source models available.
- Develop the model using Python and its libraries.

#### **Relevant**

- Relevant for the moderation and integrity of social media networks.

- Making sure that engagement and numbers in the social media platforms are genuine so that advertisers are not misled.

**Time-bound**

- Initial Model to be developed by March 2024
- The Final Project must be completed by April 2024

## **Chapter 2**

### **Literature Review**

This literature review builds on top of the Interim report previously completed as part of the development of this project (Albuquerque Rodrigues De Sousa, 2024b)



## 2.1 Machine Learning

According to IBM, “Machine learning (ML) is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy.” (IBM, 2023)

There are four types of Machine Learning models (Salian, 2018):

- **Supervised Learning** – When a model uses a dataset that has been pre-labelled to allow it to “learn” the predictive function.
- **Unsupervised learning** – When a model uses an unlabelled dataset and has to identify patterns and relationships between the samples.
- **Semi-Supervised Learning**- When a model uses a dataset that contains both labelled and unlabelled data, the goal here is for the model to learn from the labelled data and then apply the function to the unlabelled data.
- **Reinforced learning** – When a model learns an approach of trial & error to achieve an optimal solution.

For this project I have decided to use a supervised model, the reason for this is the necessity of using data to understand the different patterns of bot accounts.

There are two types of Supervised Learning algorithms in Machine Learning:

- **Regression** – When the algorithm predicts a continuous output.
- **Classification** – When the algorithm predicts a discrete value, such as the class/label of the input data.

As the input data to be predicted in this project are labels (Bot or not Bot), a classification model is the most appropriate. Different models such as SVM (Support Vector Machines), Decision Trees and Random Forest are going to be implemented and performance compared in the **Results & Discussion** section, in order to find the best solution to this problem.

## 2.2 ChatGPT

ChatGPT is an Artificial Intelligence chatbot developed by OpenAi. It has gathered a significant amount of attention since its launch back in November 2022. This AI model is capable of producing human-like textual content, including short stories, upon users' requests. It's highly capable of generating responses to user queries, even when they are complex and ambiguous. ChatGPT's strength lies within its ability to understand the context and nuances of the prompts it receives, allowing it to produce highly personalised and engaging responses.

Built based on Generative Pre-Trained Transformers (GPT), ChatGPT is “a type of LLM (Large Language Model) that uses deep learning to generate human-like text” (World Economic Forum, 2023). It was initially developed in 2018 as an AI model designed to predict the next word or complete a sentence in human-generated text (Roumeliotis and Tselikas, 2023). Its current free version, 3.5 along with its premium version (available via paid subscription) has amassed a staggering number of 180.5 million users worldwide (Tong, 2023).

Despite not being an Open-Source model, ChatGPT through its API allows users and businesses to access and deploy it in various manners. An example of this is Microsoft's release of “Copilot” back in November 2023, An AI model based on GPT-4 that allows users to access the newest premium version of ChatGPT through a service that is integrated into a number of Microsoft Services.

As I have mentioned previously, its openness and availability, allow users to produce any type of textual content with any given context, for instance, ChatGPT can be easily used to produce political propaganda with minimum effort.

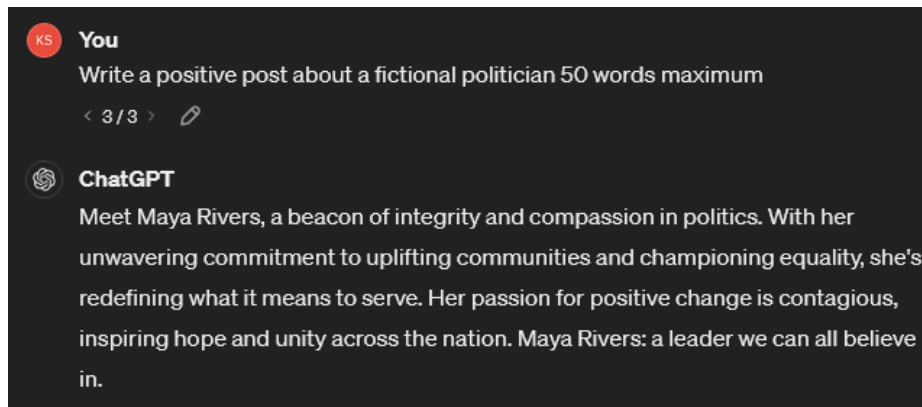


Figure 2 - An Example of Using ChatGPT to Generate A Textual Post

Therefore, it is necessary that, when looking at developing modern bot detection models, these newer models are able to identify AI-generated textual posts on social media in order to be able to differentiate between genuine accounts or bot accounts built with the use of AI tools such as ChatGPT. In this project, I intend on developing such a model, that is able to identify ChatGPT 3.5 generated textual posts on Twitter (X.com).

### 2.3 Twitter Bot Detection Models

Over the years there have been multiple attempts to create Machine Learning models for the detection of Twitter bots. In 2010, (Benevenuto et al., 2010) proposed a model for the identification of Spam Bots on Twitter. The researchers collected a vast dataset of 54 million Twitter profiles, along with their tweets and other information. Then, they manually labelled each account as spammer and non-spammer. They then used the dataset to train a non-linear SVM model. The result was a model that achieved a 70% accuracy in identifying spammer accounts and 96% accuracy in detecting non-spammers.

		Predicted	
		Spammer	Non-spammers
True	Spammer	70.1%	29.9%
	Non-spammer	3.6%	96.4%

Figure 3 - (Benevenuto et al., 2010) Model Output Prediction

The model worked by identifying certain patterns of activity, such as the ratio of tweets with URLs and the average number of tweets with URLs, the average age of the accounts was also taken into account. It was one of the earliest attempts to detect bots

on Twitter, but researchers at the time had to deal with the challenge of no datasets being available. Although the model achieved a good level of accuracy, its effectiveness when dealing with modern bots is uncertain. The reason for this is that while many bots are still spreading URLs, this model would likely struggle, and may even be unable to, identify other types of bots such as engagement bots and bots built with the use of AI, designed specifically to mimic human behaviours.

(Davis et al., 2016) presented a model for the identification of Bots using a classification model, the model works by extracting “more than 1,000 features using available meta-data and information extracted from interaction patterns and content”. These features are then grouped into six categories: Network, User, Friends, Temporal, Content and Sentiment. Then, a Random Forest Classifier is trained on each group and a final Classifier is trained on the overall results of the previous ones, giving a probabilistic score determining if an account is a Bot or not. The model is evaluated using an AUROC (Area Under the Receiver Operating Characteristics Curve) metric, which determines the model as 0.95 accurate.

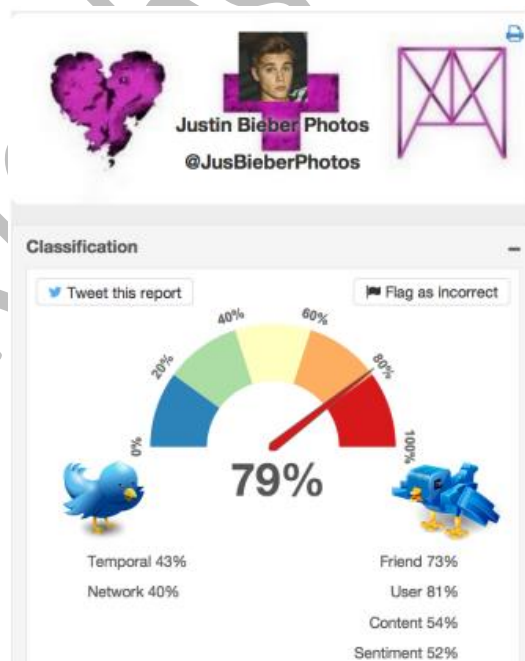


Figure 4 - (Davis et al., 2016) Model Output Prediction

Although this is a great probabilistic model, it may have issues determining tweets generated with ChatGPT or other AI tools, this is because when the model looks at the

content of tweets, it only looks for specific keywords (tags), semantic cues or for specific sentiments through the use of sentiment analyses. An AI-generated account could easily change its behaviour in order to avoid being identified, for instance, if the model is looking for positive sentiments, the bot could instead generate negative tweets to avoid detection. It is therefore unclear how this model would behave with modern bot accounts.

(Heidari, Jones and Uzuner, 2022) proposes a model for the identification of Twitter bot accounts using the user's personal information such as age, gender, personality, and education. The model determines if an account is a bot solely based on the user's personal information, with no regard to the user's posts except for the specific extraction of information such as education and personality. It uses a dual Neural Network structure for each of the user's information. In other words, for every individual information such as age and gender, 2 separate neural networks are trained, dividing them into 2 categories. As a result of this dual approach, the accuracy of the model increased by more than 3%.

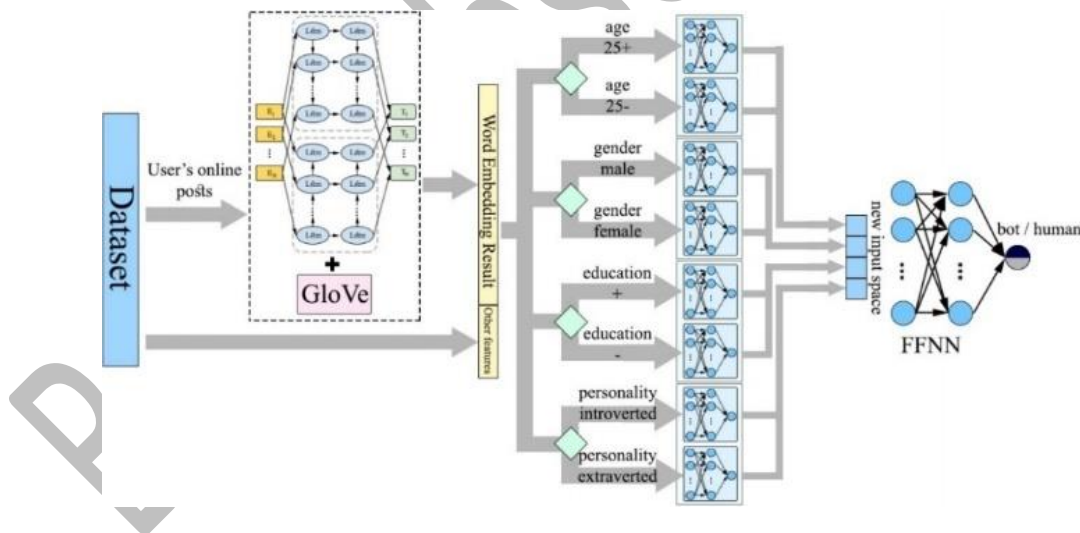


Figure 5 - Bot detection model proposed by (Heidari, Jones and Uzuner, 2022)

The proposed model was found to perform better than other similar models when tested using 2 different test sets; it outperformed in all three metrics used: Accuracy, F-Measure and MCC

Although this is a very good model for the identification of bot accounts on Twitter, it has, in my view, a few delimitations. For instance, as the model only looks at the available user's data, it is likely that the model won't be able to make an accurate prediction if a given user's account doesn't contain the necessary information to feed the model. Many bot accounts contain little to no information, but this is also the case of recently created real user accounts. Therefore, the model won't be able to differentiate between users, and its accuracy will be affected. Another limitation of this model is how it will classify newer bot accounts built using AI tools such as ChatGPT. Those tools allow the bot's posts to look more 'human-like', evading bot detection models that look at the text. Similarly, with the popularisation of those tools among real users, the model likely won't be able to identify when it's a Bot or a real user using ChatGPT to generate their post. I concluded that although this is a good model, it is very unlikely that it will perform well in a real-world scenario where data won't be completely available, and where the user uses an AI tool to generate their posts.

(Feng and Uyen Trang Nguyen, 2023) proposes a model that uses only textual posts from Twitter accounts. The model, called BOTLE, is found to offer the best or the second-best performance among similar models.

BOTLE is developed using a Recurrent Neural Networks (RNN) model that uses BiGRU (bidirectional lightweight gated recurrent unit). BiLGRU consists of two LGRUs, one with input in a forward direction and the other in a backward direction, allowing it to capture information from both the past and future states of a given text. It is used in this paper to capture the textual features of tweets. Researchers also implemented a numerical representation of words, in the field of machine learning, those are called Word embedding. Embeddings are a way of representing words in a numerical form, usually represented as a vector in a multi-dimensional plane, words are encoded by the meaning of the words, although different embedding models can encode words in different manners, such as by the similarity of the words and the context that they are placed in.

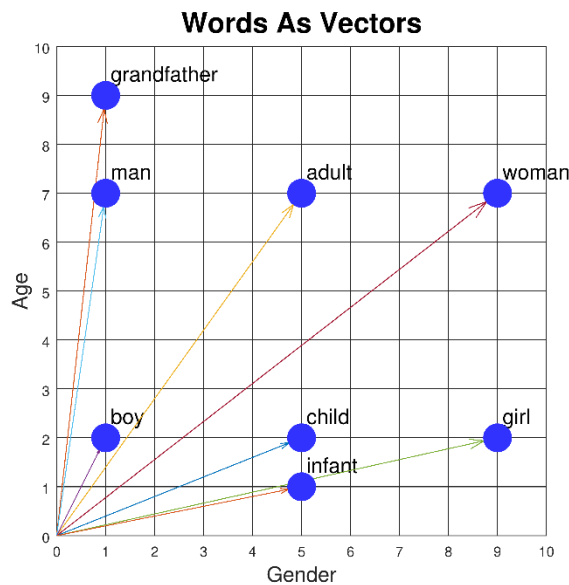
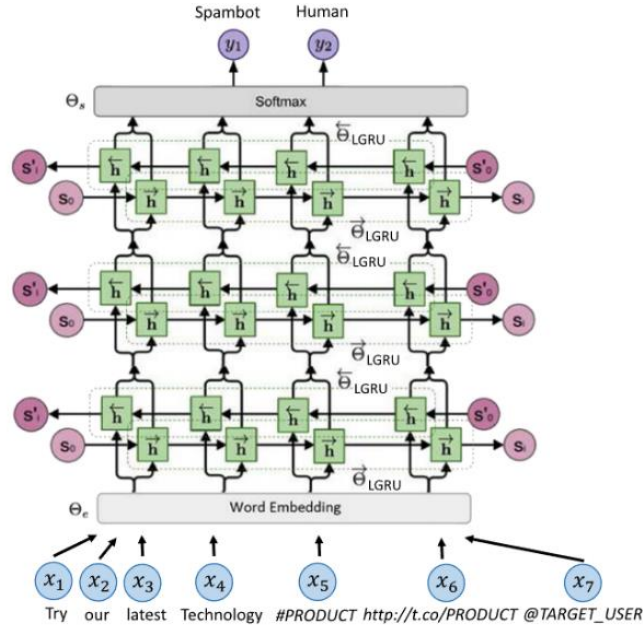


Figure 6 - An Example of How Embeddings are Created ([www.cs.cmu.edu](http://www.cs.cmu.edu), n.d.)

The vector's values and sizes can vary depending on the model chosen as well as the size of the training dataset. For this model, the authors decided to use a pre-trained embedding model of GloVe (Global Vectors), the embedding model chosen was trained on a vocabulary of 1.2 million words.

The (Cresci et al., 2017) is then used to train the model by passing the textual content of tweets. This dataset contains more than 3 million tweets, labelled and divided between genuine human tweets and different bot accounts.

The author, argues that, as this model uses only textual content, contrarily to similar models that require handcrafted features and/or assumptions about Twitter profiles, this model prevents the need for time-consuming tasks and labour-intensive feature engineering, while providing a competitive performance.



**Figure 7** - Illustration of the BOTLE model developed by (Feng and Uyen Trang Nguyen, 2023)

The table below is presented as evidence of the competitiveness of the model when compared to similar models. Note that the model had a better recall performance in the first test set than the other models, and better accuracy and F1-Score in the second test set. Overall, we can conclude as the authors did, that the model offers a competitive performance, outperforming or being the second-best performing model.

	Model	Type	Detection Result			
			Precision	Recall	Accuracy	F1-score
Test Set #1	Human annotators	Manual	0.267	0.080	0.698	0.123
	BotOrNot? [1]	Supervised	0.471	0.208	0.734	0.288
	Yang <i>et al.</i> [2]	Supervised	0.563	0.170	0.506	0.261
	Wei & Nguyen [33]	Supervised	0.940	0.976	0.960	0.958
	BOTLE	Supervised	0.956	0.977	0.969	0.967
	Miller <i>et al.</i> [3]	Unsupervised	0.555	0.358	0.526	0.435
	Ahmed <i>et al.</i> [6]	Unsupervised	0.945	0.944	0.943	0.944
Test Set #2	Cresci <i>et al.</i> [5]	Unsupervised	0.982	0.972	0.976	0.977
	Human annotators	Manual	0.647	0.509	0.829	0.570
	BotOrNot? [1]	Supervised	0.635	0.950	0.922	0.761
	Yang <i>et al.</i> [2]	Supervised	0.727	0.409	0.629	0.524
	Wei & Nguyen [33]	Supervised	0.933	0.919	0.929	0.926
	BOTLE	Supervised	0.938	0.934	0.939	0.936
	Miller <i>et al.</i> [3]	Unsupervised	0.467	0.306	0.481	0.370
	Ahmed <i>et al.</i> [6]	Unsupervised	0.913	0.935	0.923	0.923
	Cresci <i>et al.</i> [5]	Unsupervised	1.000	0.858	0.929	0.923

The Proposed Model BOTLE Uses LGRU and Four Linguistic Embeddings (Word, Character, POS, and NE). The Best and Second Best Results are Highlighted in Green and Blue, Respectively. The results of Human Annotators and of the Baseline Models BotOrNot?[1], Yang *et al.* [2], Miller *et al.* [3], Ahmed *et al.* [6], and Cresci *et al.* [5] are taken from [5]; the Results of Wei & Nguyen [33] are from [33].

**Figure 8** - Performance Comparison of BOTLE against similar models using the Cresci 2017 dataset.



There are some limitations for the BOTLE model, for instance, it wouldn't be able to correctly identify bots that only like and/or tweet posts as those accounts generally have little to no textual posts. The model also might have issues with more modern bot accounts that generate their posts using AI tools similar to ChatGPT as those bots can effectively mimic real accounts.

(Pinnapureddy Manasa, Malik and Batra, 2024) proposes an approach of using two models. The first model uses the unsupervised GloVe model to extract vocabulary features from textual tweets. Then an LSTM deep learning model is used to detect spam text based on these features. The second model is a CNN (Convolutional Neural Networks) model that takes as an input the dataset containing the tweets, and other details such as the location and number of followers, along with meta-heuristics features extracted from the tweets, including the number of hashtags and tags.

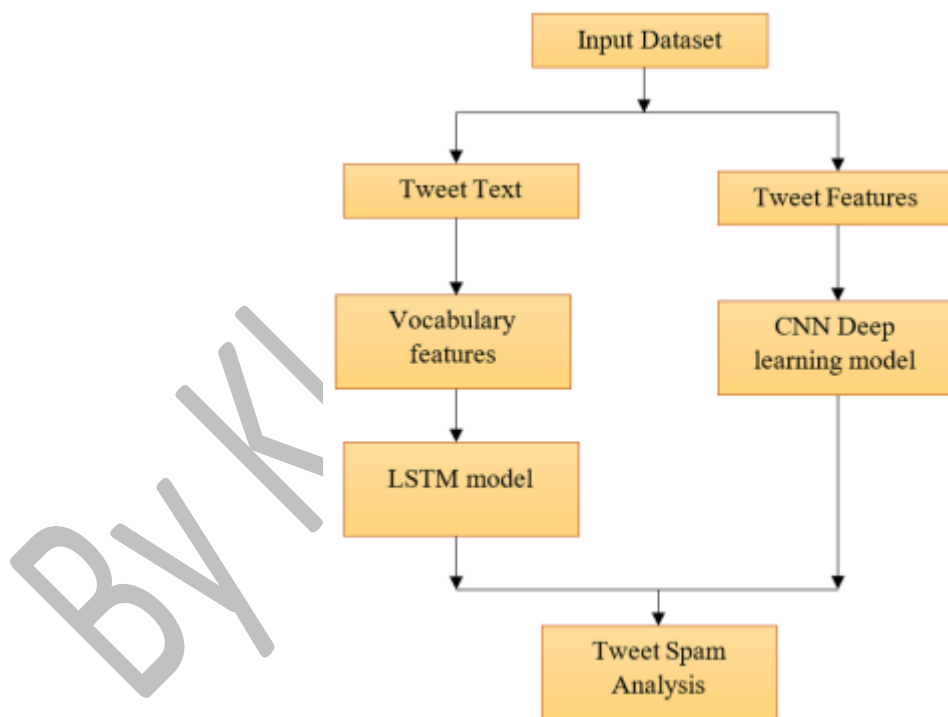


Figure 9 - Model Pipeline Diagram presented on (Pinnapureddy Manasa, Malik and Batra, 2024)

The combination of those achieved a very high accuracy of 99%; however, I believe that the model is likely overfitting the dataset. The reason for this is that the model is unable to understand the entire context of every single tweet, the results in

the paper indicate that the model classified as “hate speech” every tweet that contained any obscene word not taking into account the broad context.

BY Kleybson Sousa

# Chapter 3

## Requirements Analysis

### 3.1. Requirement Analysis

#### 3.1.1. Functional Requirements

In order to work with Python, it is necessary to import various libraries such as Pandas, Sklearn, and Transformers. Additionally, the dataset file needs to be imported into the Jupyter Notebook. The data goes through various stages before it can be used for training models. To achieve this, a script generates the data, while another script formats and splits the dataset into subsets of the main dataset. Finally, another script is utilised to train the models.

The new subsets need to contain the necessary information for each of the models. For instance, for the model that looks at the textual posts, the account number and posts need to be included. Following this, labels are extracted from the dataset and stored in a different array named 'y'. At this point, the function "train\_test\_split" from Sklearn is used to split the dataset into X\_train, X\_test, y\_train, and y\_test.

This function allows the implementation of the Train-Test Split Testing Methodology. It also allows for the easy alteration of the train-test split sizes. For instance, I can set the train test to be 60% while the test set to be 40% of the dataset.

Further steps are taken to ensure the results are valid and that the model is not overfitting the data. For instance, I have implemented a Stratified KFold Cross Validation, and the models are tested on 3 different occasions with this type of cross-validation. Each occasion uses a different number of folds such as 5,10, and 15 folds.

Hyperparameter tuning is also used when training the models on the dataset. This is done in order to find the best parameter for each of the models. I have opted to use

GridSearchCV for this process as it allows me to manually set the parameter to be tested.

Finally, to allow the comparison between algorithms, for each of the models, the results are displayed using pandas and written into a CSV file.

### **3.1.2. Non-Functional Requirements**

In order for this project to be successful, all necessary environments must be set up before coding begins. This includes installing required software such as Visual Studio Code and Python Programming Language. Prior to use, Python libraries must also be installed using the PIP package, which enables the installation and management of these libraries.

Given the project's large dataset and multi-stage process, it was crucial to work in an interruption-free environment. For example, creating word embeddings can take hours to process thousands of textual posts. Therefore, it is essential to ensure that the model is not interrupted during this time. To ensure reproducibility, the embeddings of the 100,000 thousand samples subset were saved in a CSV format, reducing the model's running time by over four hours.

## **3.2. Risk Analysis**

### **3.2.1. Project Scope Risk**

Changes to the project scope caused by unclear objectives during the development stages can cause the project to expand more than originally projected, leading to unmet deadlines, unclear goals and increasing labour. To prevent this from occurring I will be setting up clear goals and aims, as well as producing a clear roadmap in the form of a Gant Chart

### **3.1.1. Data Quality and Bias – Bias and Incomplete data**

The quality of the dataset is one of the most important aspects of the project, incomplete or incorrect data can lead the model to conduct incorrect predictions. To prevent this from happening I have selected a dataset (Cresci et al., 2017) that is used for research by

more than 200 papers of different institutions and research subjects, demonstrating the reliability of the dataset.

### **3.1.2. Explainability –Frameworks for Model Interpretation**

Complex ML models can make it very difficult to understand how their predictions are being made, for those reasons, it's important to be able to try to uncover what's behind the decisions. For this project, I will attempt to implement the LIME framework. LIME is an open-source framework built specifically to explain ML decisions. Although I don't have experience using it before, I had some contact with it in previous projects and believe it's a good choice.

### **3.1.3. Model Overfitting and Underfitting**

There are always the risks that the ML model overfits or underfits our data. To prevent model overfitting and overfitting from happening I will be using a large dataset (Cresci et al., 2017) that contains more than 10.000 Twitter accounts with detailed account information. For the textual content of the bot accounts, I will make sure that the new posts are diverse in subjects, or that are of the same subjects as the human accounts. I will also implement K-fold validation as described in (Frantzidis, 2023) to assess the performance and generalisation of the model.

Feature Engineering can be performed to select only the most relevant features of our dataset and ignore the irrelevant features, by doing this the model dimensionality can be reduced, preventing model overfitting (Brownlee, 2014)

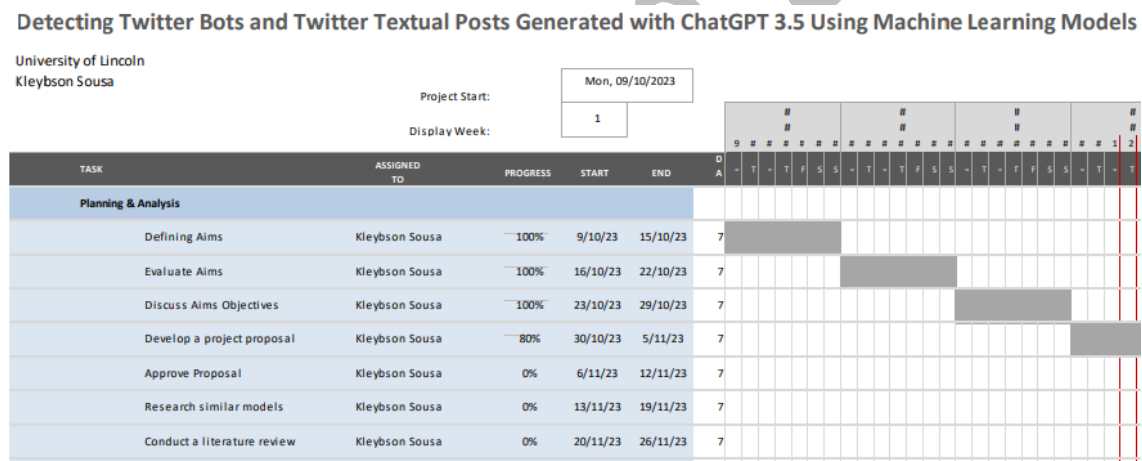
L1 and L2 are 2 Regularization techniques that could also be used to prevent model overfitting (Babitz, 2021)

The model will also constantly be evaluated using metrics such as recall, accuracy, confusion matrix and ROC curve on a separate validation set to check for signs of overfitting/underfitting.

## Design & Methodology

## 4.1. Project Management

For the management of this project, a Gantt was created, this allowed me to set goals and keep track of the progress through the development and research stages.

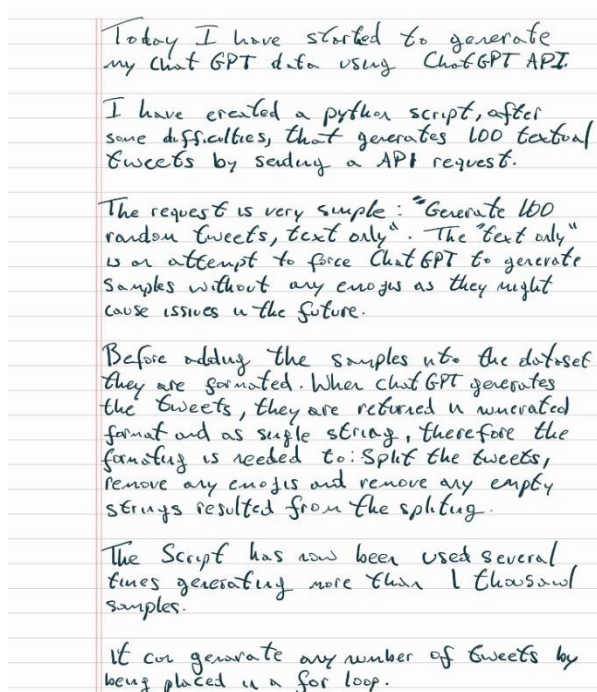


*Figure 10 - A Small Sample of The Gantt Chart Created*

For this project, I chose the Agile methodology. The reason for choosing Agile is that it allows for continuous improvement throughout the development cycle. Agile's greatest advantage is its ability to adapt to challenges and be flexible to changes and improvements. This methodology is designed to understand that flexibility is essential when building a project (Karlesky & Voord, 2008).

To keep track of the development stages, I have maintained a diary where I record the actions taken on a given day. This allows me to reflect on what I did right and where

I could have improved. Instead of using an online Kanban, I utilized the notes on my tablet to create tasks and objectives in the same way as a Kanban board.



Today I have started to generate my Chat GPT data using ChatGPT API.

I have created a python script, after some difficulties, that generates 100 textual tweets by sending a API request.

The request is very simple: "Generate 100 random tweets, text only". The "text only" is an attempt to force ChatGPT to generate samples without any emojis as they might cause issues in the future.

Before adding the samples into the dataset they are formatted. When ChatGPT generates the tweets, they are returned in unformatted format and as single string, therefore the formatting is needed to: Split the tweets, remove any emojis and remove any empty strings resulted from the splitting.

The Script has now been used several times generating more than 1 thousand samples.

It can generate any number of tweets by being placed in a for loop.

Figure 11 - Samples of the Note Taken During the Development of The Project

Furthermore, I developed a working schedule around my university timetable to efficiently work on this project and other projects I had to work on concurrently. This helped me keep track of all my projects and better manage my time for each of them.

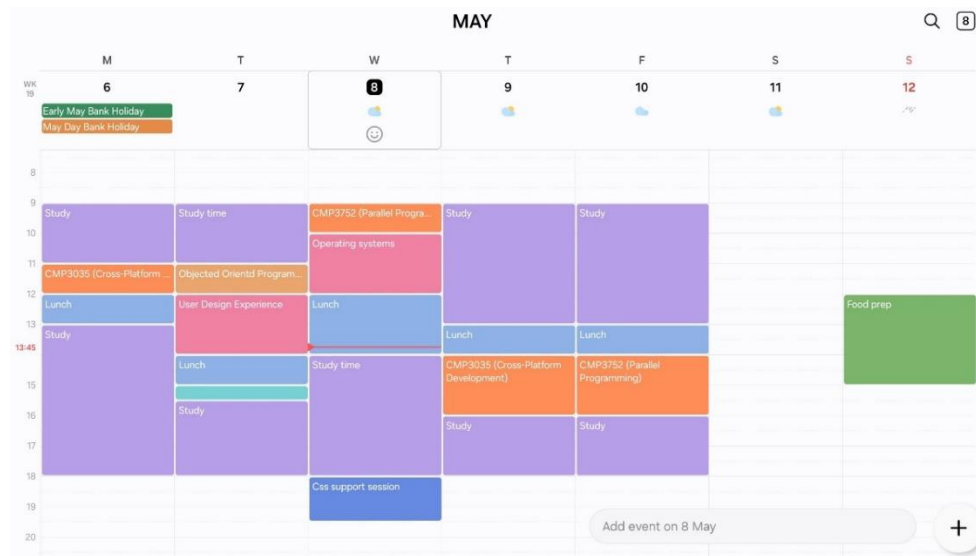


Figure 12 - A Sample of The Weekly Working Schedule

During the initial stages of model development, some tasks were delayed which resulted in missing some of the original Gantt chart deadlines. Unfortunately, these delays were due to the data generation process taking longer than anticipated. For example, generating 10,000 tweets using ChatGPT API would take around 4-5 hours. Later, after all the data had been generated, I had to transform the samples into embeddings for tweet generation. For the 100,000 samples subset, it took around 48 hours to generate embeddings using the BERT model. To save time, I saved the embeddings generated on a CSV file called “100k\_embeddings.CSV” for future use. This reduced the training time for the models to around 4-5 hours for this subset.

During the early stages of the project's research and development, I had regular meetings with my supervisor, Dr. Vassilis. Although I had many ideas about the project's direction, Dr. Vassilis helped me focus on more practical and achievable options. His guidance was essential in making critical decisions throughout the development process. For instance, when discussing how tweets should be formatted, I initially suggested removing hashtags and emojis to help focus on the text. However, Dr. Vassilis helped me understand that these features also convey crucial information and removing them could skew the dataset.

At the beginning of the project, we had weekly meetings, but as the coding stage progressed, we met less frequently and communicated through Teams instead. Dr.



Vassilis was always available to answer any questions or review the results that I had to present.

## 4.2. Toolset and Machine Environment

For this project, I used the IDE (Integrated Development Environment) Visual Studio Code, a free IDE developed and supported by Microsoft. Through Visual Studio Code, I had access to the Integrated Interactive Window, a native way of accessing Jupyter Notebooks through Python code files.

In this way, I could write Python Scripts and execute them in sequence on a Jupyter Notebook with minimum effort, allowing me to take advantage of Jupyter Notebooks by executing code snippets and viewing results in real-time as the code runs.

Jupyter Notebook's biggest benefit is the ability to "blend natural language, media, visualisations and executable code in a single document allowing for an interactive, human-focused computing experience" (Johnson, 2020).

As I have previously mentioned, Visual Studio Code is a free IDE, it's built with the aim of being a "lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux" (Visual Studio Code, 2023). Users can access a variety of code languages and frameworks by simply installing the language they wish to use as an extension. Developers can easily add their programming language and tools to the VS Code Marketplace by building and publishing their own extensions.

I have selected Python for this project for many reasons, one being that it is compatible with Jupyter Notebooks. It is an Open-Source language with a huge community online, with a lot of available resources, that includes various libraries such as the ones I have used in this project. The most important ones are Pandas and Sklearn. "Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labelled" data both easy and intuitive" (pandas.pydata.org, n.d.). Scikit-learn (Science Kit Learn), known as sklearn, is an open-source library that provides access to "a wide range of state-of-the-art machine learning

algorithms for medium-scale supervised and unsupervised problems” (Pedregosa et al., 2011).

## **4.1. Dataset**

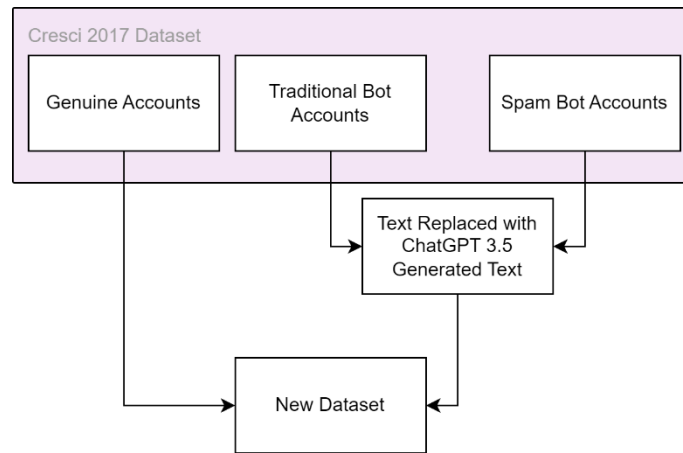
### **4.1.1. Dataset Generation**

One of the biggest issues with the development of this project was the unavailability of suitable datasets. There are currently, to the best of my knowledge, no datasets available with the necessary data for the development of the models. Those datasets would need to have Twitter accounts with details about the accounts as well as ChatGPT-generated textual posts along with human-generated textual posts. There are numerous datasets with Twitter accounts’ details, human-generated Tweets and traditional bot posts but none with ChatGPT-generated textual posts.

In the face of these circumstances, I decided after researching and discussing with my supervisor, to partially obtain the data necessary. The idea here is to use a well-known and validated labelled dataset with Twitter Accounts’ details, Human-generated textual posts and Bot Accounts’ textual posts, then use OpenAI API to produce ChatGPT-generated textual posts to replace the original Traditional Bot Accounts posts with the new data.

I decided to select the Cresci 2017 Dataset (Cresci et al., 2017) as this is a widely used dataset within the research community, which proves its validity as a reliable dataset. The Cresci data contains more than 8 million tweets from more than 10 thousand Twitter accounts. The dataset is split between Human Accounts with real tweets and Bot Accounts of different types such as social spambots, traditional spambots and fake followers. The account’s details include the account’s Name, Number of followers, Data, and time of the account creation, as well as the date and time of each individual tweet and more.

The Diagram below displays how the new dataset is going to be built, using the Cresci 2017 as its base:



*Figure 13 - An Example of How the New Dataset Will Be Formed*

As I have previously mentioned, I used the OpenAI API to generate the tweets for the dataset. OpenAI API gives access to ChatGPT and other tools developed by OpenAI such as Image generation and speech-to-text tools. Here, it is used to generate the tweets using a Python Script.

Figure 14 shows part of the script I created for the data generation. As we can observe, the script is very straightforward. It uses the OpenAI API to access ChatGPT through the method “chat completions”. The model parameter allows for a “GPT” model to be selected, as I have mentioned previously, there are various versions of ChatGPT including GPT 3.5 and 4.0. Although GPT 4.0 is the newest and most powerful model, I have decided to use GPT 3.5-turbo as this is the current version of the free tier of the ChatGPT model (OpenAI, 2022), this is the version most widely used for being free as well as the version most likely used by bot farms due to its cost-effectiveness. The “message” parameter is the prompt to be passed to the model. Note that the structure allows users to take roles, this includes the role of the model itself, meaning that users can assume the role of the model in order to “force” it to behave in certain ways.

```

messages_history = []
client = openai.OpenAI(api_key="API KEY HERE")

def generate_tweets(client):
    completion = client.chat.completions.create(
        model = "gpt-3.5-turbo",
        messages = [{ 'role': 'user', 'content': "generate 100 random tweets, multiple subjects" }]
    )
    return completion

```

*Figure 14 - An Example of How-to User ChatGPT API to Generate the Data*

The prompt I passed into the model generates 100 tweets at every time it is executed. Later, this is placed inside a loop that is run 10 times every time this code is executed, totalling 1000 tweets for each execution of the script. The results are then stored in a Pandas Dataframe that is then saved as a CSV file.

The decision to generate only 100 tweets per completion was taken to the model only being able to produce a maximum of 4,096 tokens at once, this equals to around 164 tweets of 100 characters each. As I don't specify a length and I am unable to predict the length of each tweet, I took the decision of only generating 100 tweets per completion while giving it space for length flexibility. Only 1000 tweets are generated per script execution to ensure that if there is an issue with the generation of tweets, the cost of the lost token is not significant. For instance, if there is an error while generating the tweets, the model will return an error message instead of the data, if any, generated before encountering the error.

The tweets are formatted before being passed into a pandas' dataframe, this is done to remove any unnecessary characters, such as apostrophes marks that sometimes encapsulate the tweets.

```

def format_tweets(messages_history, completion):
    reply = completion.choices[0].message.content
    new_reply = re.sub(r'\d+\. ', '', reply)
    split = new_reply.split('\n')
    split_strings = [s.strip() for s in split if s]
    messages_history.extend(split_strings)

```

*Figure 15 - Code Snippet Showing how the Messages were formatted*

Finally, the data is saved into the CSV file called "tweets\_text.CSV".

#### 4.1.1. Dataset Cleaning

The Cresci 2017 dataset is an enormous dataset with various details and information about the accounts and tweet posts, therefore it was necessary before performing any modifications to clean the data by removing any unnecessary information and removing rows with empty/null values.

Firstly, I ensured to remove all tweet samples with sensitive content. Then, I proceeded to remove rows with empty or null values, as (Kaiser, 2014) demonstrates, there are different methods of dealing with missing values, and as there is no shortage of data samples in this dataset, I decided to remove all the rows with empty or null values.

Before removing columns with unnecessary information, it is important to understand the layout of the Cresci 2017 dataset. The dataset is split into two different CSV files for each type of Twitter account. For example, genuine human accounts have a separate CSV file for all their tweets and another file for all their account details.

In order to create a single dataset file with all the information needed for the training of the models, I decided to merge the dataset files. However, during this process, I had to make some decisions in regard to the columns of each file. The pre-processing of the dataset was done on the Python Script “data\_cleaning\_subset\_creation.py”. The data from both files were loaded into pandas dataframes, from the tweet dataframes I removed the following columns:

- contributors
- truncated
- favorited
- possibly\_sensitive
- updated
- in\_reply\_to\_screen\_name
- geo
- favorited
- retweeted

- created\_at
- source
- place
- timestamp
- crawled\_at
- in\_reply\_to\_status\_id
- in\_reply\_to\_user\_id
- retweeted\_status\_id

Those columns were removed as they did not contain any valuable information for the training of the models. For the accounts' dataframe, I have removed the following columns:

- name
- screen\_name
- 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
- crawled\_at
- updated

The columns were removed as some of them contained null information. Additionally, many of them contained information that was not relevant to the models' training, for instance, "profile\_image\_url" which provided a link to the picture of the Twitter profile. In the end, the dataset was left with the following columns, starting with the tweets dataframe:

- id
- text
- user\_id
- retweet\_count
- reply\_count
- favorite\_count
- num\_hashtags
- num\_urls
- num\_mentions

The accounts' dataframe was left with the following:

- id
- statuses\_count
- followers\_count
- friends\_count
- favourites\_count
- created\_at
- timestamp

At this point, I had to modify the account's dataframe. I noticed that the 'id' column on the account dataframe corresponded to the 'user\_id' on the tweets dataframe, thus I needed to merge them on this value, for this to happen, both columns needed to contain the same name. Therefore, I changed the 'id' column's name to 'user\_id' and then, I proceeded with the merge of the same class data, that is merging the account's details with their respective tweets data.

I have also modified the datasets so that they all have the same Date Time formatting, as some of them contained epoch time while others had human-readable format.

Then before compiling, I had to take into account the amount of ChatGPT-generated text I would be able to produce. As I am generating the data myself, it would be unrealistic to generate the same amount of bot textual posts as there is in this dataset. This is because it could take up to 5 hours to generate 10.000 samples, and significantly more to match the dataset. For this reason, I set the target of generating 100.000 samples. Hence, after merging the datasets, I sampled different subset sizes of 500, 5.000 and 50.000 bot accounts, and matched those sizes with real accounts, creating subsets of balanced data of 1.000, 10.000 and 100.000 samples. Then, I used the data I generated, stored on the "tweets\_texts.CSV" file to replace the textual posts of the bot accounts. I followed the same process of sampling the data from the textual posts, sampling 500, 5000 and 50000 samples and then using those samples to replace the respective textual posts from the 3 subsets created earlier. All samplings used a common random state, this is to ensure that the sampling is consistent with the accounts they are being allocated to.



The last step before the merge of the dataset was to assign labels, as the models I trained are supervised, they necessarily need the dataset to be labelled between Human & ChatGPT-generated/Bot data. This was a simple process, as I have previously mentioned the dataset is divided between genuine accounts data and bot accounts data, therefore all I needed to do before merging was to add an additional column named “label” with a numerical value of ‘0’ on the genuine accounts’ data and ‘1’ on the bot accounts’ data.

With this, I modified subsets of the dataset with all I needed to train my models. Those are saved as ‘subset\_1k.csv’, ‘subset\_10k.csv’ and ‘subset\_100k.csv’.

Further processing of the data is later done to prepare the dataset for the second model that looks at the account’s details. As this model only looks at the account details, there are some unnecessary data in this dataset that need to be removed before continuing. This is included in the “data\_cleaning\_subset\_creation.py” script, please note the comments indicating these further processes.

In order to create a newer subset, I had to go back to the point in time when the accounts were merged with their respective tweets. The reason behind this is that although the Cresci 2017 dataset contains over 8 million tweets, it consists of only more than 10,000 accounts. If I were to include only the accounts that were in my subset, the resulting subset would have a very small number of accounts.

Here, I dropped all the rows that belonged to the same accounts. Then, I proceeded to drop the following columns:

- text
- timestamp
- num\_hashtags
- num\_urls
- num\_mentions
- reply\_count
- retweet\_count

- favorite\_count

Essentially, I kept only the data that were related to the accounts' details, removing the rest of the data related to the tweets of the tweets. Then I labelled the subset by adding a new column called 'label' and assigning the value of '0' to genuine accounts and '1' to bot accounts. Then, after merging the genuine accounts and bot account into a single dataframe, I saved the new subset as "model\_2\_subset\_2k.csv".

## 4.2. Word Embeddings

To accurately classify tweets, I had to convert the text into numerical representations, which is commonly referred to as embedding. Word embeddings are a method of representing words as vectors in a multi-dimensional space. The distance and direction between vectors indicate the similarity and relationships among the corresponding words (Barnard, 2024).

There are various open-source pre-trained models for the generation of word embeddings, including but not limited to GloVe, Word2Vec, BERT and FastText, each model has its advantages, and a lot of research has been done to compare different models. For instance, (Wang et al., 2019) evaluated some of the most popular models, it concluded that embedding models are more efficient when solving specific tasks.

For this project, I have decided to use a pre-trained BERT model. BERT is a model developed by Google in 2018 (Devlin et al., 2019). It is a bidirectional encoder that can contextualise words by looking at the entire textual content when creating embeddings. This means it can capture the true meaning of the words, giving different values for words depending on their inserted context. This is particularly important as I intended to use the entire textual post to create the embeddings that were then used to train the model for identifying ChatGPT-generated text. One more advantage of BERT is that it produces fixed-length vectors for the embeddings, making it easier to use them for training.

The BERT model here is used by importing the Transformers Library. Transformers is a library that allows users to access NLP pre-trained models stored online at HuggingFaces.com, an open-source community-supported repository with hundreds of

pre-trained NLP models, it manages the Trans-formers library as it acts as a channel between the repositories and APIs. As demonstrated by (Wolf et al., 2020b) Transformers are a great facilitating tool to access large-scale pre-trained NLP models.

## 4.3. Algorithm/Model Design Selection

During session 2.1, I discussed the different types of machine learning models available. If I have a labelled dataset, then my only option would be to use a supervised model. Since my primary objective would be to classify data samples, it would be necessary to choose a classification model. The models I have implemented will be discussed in the upcoming sections.

### 4.3.1. KNN

KNN, short for K-Nearest Neighbors, is a machine learning model that falls under the supervised category. It works by calculating the distances between samples in a two-dimensional plane to predict their classes. The model uses a majority voting system to determine the class of a given data point based on the K nearest neighbors. The value of K is predetermined and determines the number of nearest neighbors considered to classify the data points.

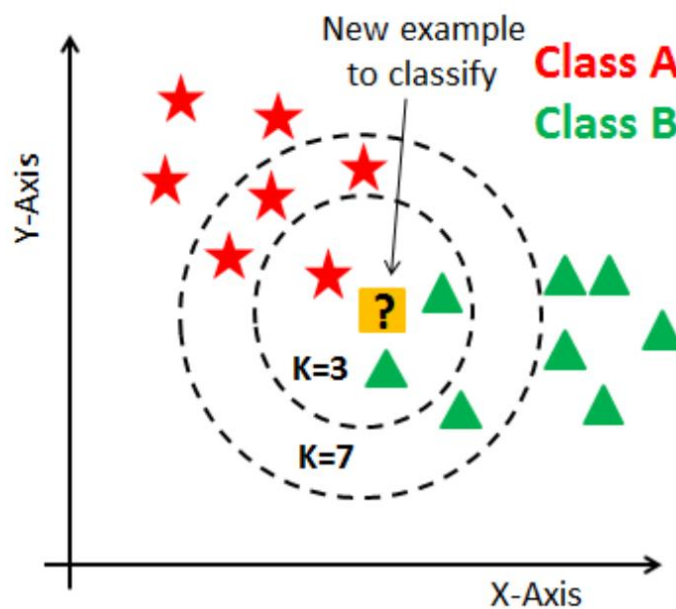


Figure 16 - An Example of How the KNN model works (Anil Gokte, n.d.)

### 4.3.2. SVM

SVM (Support Machine Vectors) is a supervised machine learning model that attempts to classify data by finding the optimal hyperplane, that maximises the distance between two classes (IBM, 2023). It particularly performs well when dealing with binary classification problems. As demonstrated by (Cervantes et al., 2015a). It is also efficient working with high-dimensional and large-scale textual data (Cervantes et al., 2020b).

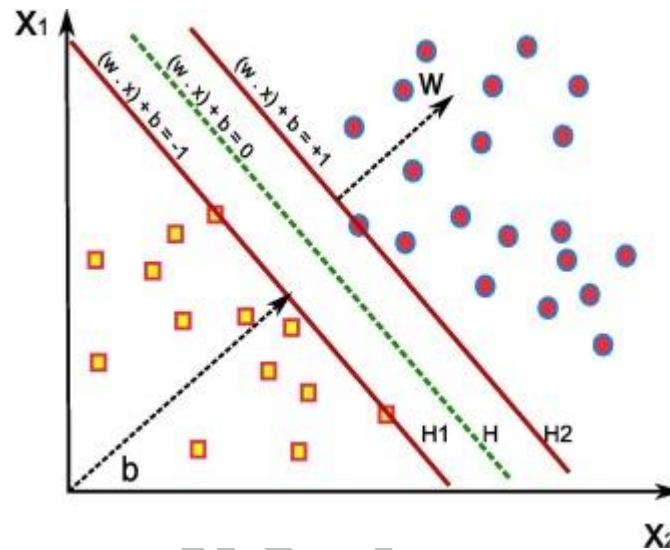


Figure 17 - An Example of How SVM Models Work

### 4.3.3. Decision Trees

Decision Trees is a supervised machine learning model that can be used for both classification and regression problems. It operates in a tree-like system with a root node at the top of the hierarchy that connects into different nodes where each node represents a decision to be taken by the algorithm and subsequent nodes are outcomes of that decision. Each node leads to another node until there are no more decisions to be made. The tree then reaches its final node, usually called 'leaf', as it represents the prediction of the model.

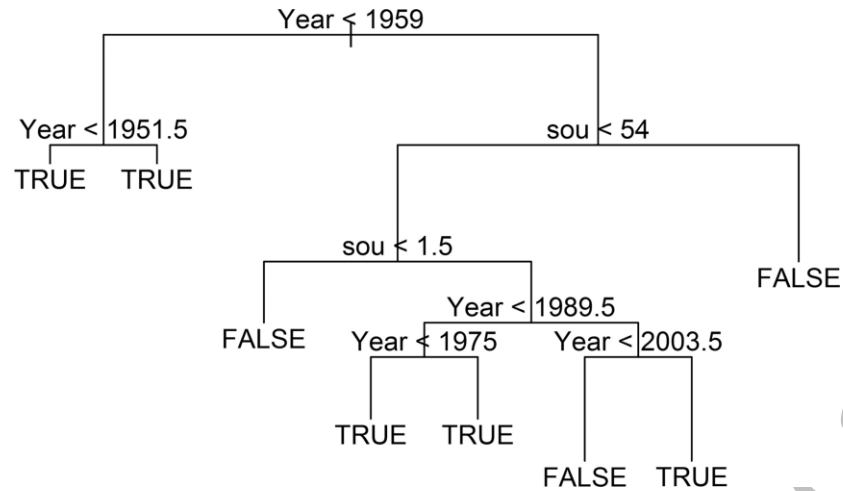


Figure 18 - An Example of the Decision Trees Model

#### 4.3.4. Random Forest

The issue with decision trees, as demonstrated by (Heglich, 2016), is that it suffers from high variance. That means that it can be very sensitive to changes in the dataset, leading to different outcomes. To overcome this issue, we have the Random Forest model.

Random Forest is a supervised model that uses multiple decision trees algorithm in order to reach a prediction. Each Tree is trained on a random subset of the dataset, a voting system is then used to determine the class of the sample.

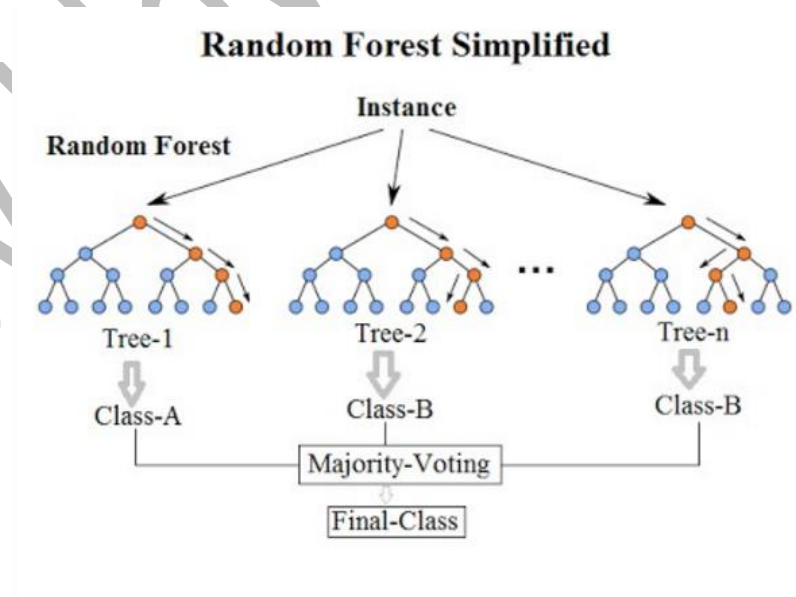


Figure 19 - An Example of Random Forest Model (Koehrsen, 2020)

## 4.4. Hyperparameter Tuning

Hyperparameter tuning is a method used in machine learning to optimise the parameters of a model. It involves trying out a set of predetermined parameters in a "search grid" until the most optimal solution is found (Ghawi and Pfeffer, 2019).

In this project, hyperparameter tuning is done using the GridSearchCV class of the sklearn library. This class evaluates the performance of the model on every combination of pre-set parameters.

To demonstrate how it works, consider the example of the image below, starting with the combination [z, a]. Grid search starts with this combination and tries every combination of Z, such as [z, b] and [z, c], for each hyperparameter. The method then returns the best parameters based on a specific metric, such as accuracy or precision.

### Grid Search

Pseudocode

```
Hyperparameter_One = [a, b, c]  
Hyperparameter_Two = [x, y, z]
```

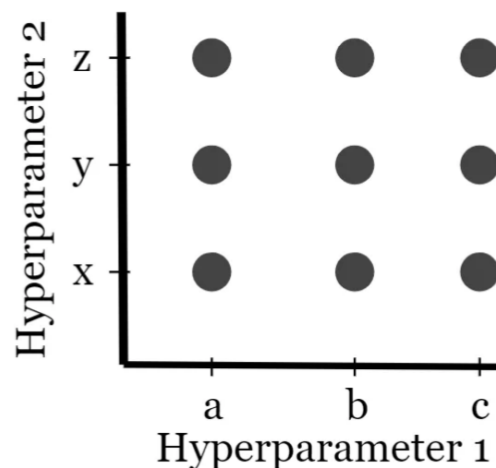


Figure 20 - An Example of Hyperparameter Tuning with GridSearchCV (Stalfort, 2019)

## 4.5. Train-Test Split Testing Methodology

The train-test methodology is essential to evaluate different machine learning models. Different splits ratios can significantly impact the performance of the models. (RÁCZ,

Bajusz and Héberger, 2021) compares the impact of different split sizes on five different machine learning models, the results demonstrate a significant impact between splits ratio.

The method involves systematically altering the split ratio and comparing how a smaller or larger dataset for training and testing can affect the model's ability to predict correct classes.

In this project, I have used various split ratios to train my model. For instance, I have used an 80/20 split, which means 80% of the dataset was used for training and 20% for testing. Additionally, I have also tested and compared other splits such as 60/40, 70/30, and 90/10. This step is crucial as it not only affects the model's performance but also helps to identify any overfitting issues with the data.

## **4.6. K-Fold Stratified Cross Validation**

K-Fold Stratified cross-validation is a technique in machine learning that assesses how well a model performs on new, unseen data. The "stratified" part of the technique ensures that there is an equal representation of all classes in the dataset. This approach is especially useful for imbalanced datasets, where one class is more prevalent than others.

The process works by dividing a given dataset into a pre-determined number of equal folds. Each fold is then used as the testing set, while the remaining folds are used to train the classifier. The trained classifier is then evaluated against the test set. This process is repeated until all the folds have been used as the testing set. (Zeng and Martinez, 2000)



Figure 21 - An Example of How Stratified Cross-Validation Folds Are Split

## 4.7. Performance Metrics

Performance metrics are crucial for measuring the ability of models to accurately predict the correct class of data samples. It is essential to showcase how well a model can perform with unseen data. Additionally, they are an effective tool for comparing the performance of different machine learning models.

Accuracy and F1 scores are widely used evaluation metrics in machine learning (Chicco and Jurman, 2020). Recall and precision are also important metrics. These metrics are derived from a Confusion Matrix, which is a method of displaying the predicted class of a model and the actual class of the data predicted.

For this project, I decided to use Accuracy, F1 score, Precision, Recall and a Confusion matrix as metrics of performance evaluation and model comparison.

Accuracy is defined as the total number of correct predictions divided by the total number of predictions. It can also be calculated, for binary classification, True Positives and True Negatives divided by the number of True Prediction, True Negatives, False Positives and False Negatives:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$



Precision is defined as the proportion of true positive prediction predictions among all positive predictions, that is the proportion of actual correct prediction. It is calculated by dividing the number of True Positives by True Positives and False Positives:

$$Precision = \frac{TP}{TP + FP}$$

Recall is defined as the proportion of actual positive predictions that were identified by the model. It is calculated by dividing the True Positives by True Positives and False Negatives:

$$Recall = \frac{TP}{TP + FN}$$

F1 Score is defined as the harmonic mean of the precision and recall of the model. It can be considered a more balanced way of evaluating the models:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

As I have previously mentioned, a confusion matrix is a method of displaying the model's prediction alongside the actual class of the predicted data:

		True Class	
		Positive	Negative
Predicated Class	Positive	TP	FP
	Negative	FN	TN

Figure 22 - An example of Confusion Matrices

# Chapter 5

## Implementation

This project required me to create two machine-learning models. I have decided that for this implementation, I will split the two models into sections, as I believe this approach is more beneficial for the understanding of the processes undertaken.

### 5.1. ChatGPT Identification Model

The implementation of this model can be found on the script “model\_with\_bert.py”, included in this project’s supporting files.

The code here is modified to use the “subset\_1k.CSV” file that contains the 1.000 samples subset. However, an adapted version of the code to each of the other subsets such as the 10.000 and 100.000 samples subsets is included. Those are named: “model\_with\_bert\_10k.py” and “model\_with\_bert\_100k.py”.

Please note, that to facilitate the reproducibility of this project, the script “model\_with\_bert\_10k.py” doesn’t use the BERT model to generate its embeddings, those were generated before the submission and saved in the file “100k\_embeddings.CSV” included in the supporting files. Nevertheless, the implementation of the BERT model is included in the script, but those are commented out.

The first step in this script is to import all the necessary libraries, which are: sklearn, torch, pandas, matplotlib and transformers, although transformers is only used to give access to the BERT model.

```

from sklearn.decomposition import PCA
import torch
from transformers import BertTokenizer, BertModel
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

```

Figure 23 - Code Snippet Showing Libraries imported

The next step is to initiate an instance of the Bert Tokenizer. A tokenizer essentially prepares the data for a model, in this case, the Bert Tokenizer prepares the data input for the Bert model. The model here defined as “bert-base-uncased” is a pre-trained version of the BERT that doesn’t differentiate between lowercase and uppercase letters. For instance, the sentence: “Hello World!”, would be the same as “hello world!”.

```

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

```

Figure 24 - Example of How to Use the BERT Model

The dataset after being cleaned in the data generation stages and split into different subsets (see section **4.1.1 Dataset Cleaning**) is read into a pandas dataframe. In here I slice the subset a few times, first I take out the “text” and “label” columns as they are the columns used to create the model. Then, I drop any empty rows from the dataframe. The variable ‘y’ is assigned the values of the ‘label’ column contained in the subset. The column is then dropped.

```

dataset = pd.read_csv('subset_1k.csv')
subset = dataset[['text', 'label']]
subset = subset.dropna()

y = subset['label']
subset = subset.drop(axis=1, columns=['label'])
subset = subset.reset_index(drop=True)

```

Figure 25 - Example of preparing the data for training the models

The next step is to use the data to create the word embeddings, for that I created a new pandas dataframe called 'X'. The code below demonstrates how the embeddings are created, here the BERT tokenizer is used to encode the text for the BERT model. After this step is completed the embedding is extracted from one of the layers of the BERT model. For every word, a vector of 768 Columns is generated, torch is then used to average all the vectors from each textual post, the embeddings are turned into a NumPy format and added into the 'X' dataframe. This process continues until all the textual posts in the subset are processed.

```
X = pd.DataFrame()

def get_embeddings(tokenizer, model, X, tweet):
    if tweet.startswith('') or tweet.endswith(''):
        tweet = tweet[1:-1]

    encoding = tokenizer.encode_plus(tweet,
                                     padding=True,
                                     truncation=True,
                                     return_tensors='pt',
                                     add_special_tokens=True,)

    input_ids = encoding['input_ids']
    attention_mask = encoding['attention_mask']

    with torch.no_grad():
        output = model(input_ids, attention_mask=attention_mask)
        word_embeddings = output.last_hidden_state
        sentence_embedding = torch.mean(word_embeddings, dim=1)
        sentence_embedding = sentence_embedding.numpy()
        X = pd.concat([X, pd.DataFrame(sentence_embedding)])
    return X

for i in range(len(subset)):
    tweet = subset['text'][i]
    X = get_embeddings(tokenizer, model, X, tweet)

X = X.reset_index(drop=True)
```

Figure 26 - Code Snippet Showing the Creation of the Embeddings

The next step is to split the dataset into train and test sets, to achieve this I have used the 'train\_test\_split' from the sklearn library. The image below displays how I

implemented the split 60/40. In this way, I can easily change the dataset splits by only changing the values of the variables “train\_size” and “test\_size”. The split

```
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
train_size = 0.60
test_size = 0.40
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=train_size, test_size=test_size, stratify=y)
```

Figure 27 - Code Snippet Showing the Usage of train\_test\_split

At this point I implemented hyperparameter tuning (as discussed in session 4.4), this is done with the help of the method GridSearchCV from sklearn. It takes a set of parameters and tries every combination, using cross-validation and the metric ‘accuracy’ to determine the most efficient. Here, I am trying to find the best parameters for an SVM model, the parameter to be attempted are stored in the dictionary “parameters”

```
parameters = {'kernel': ['poly', 'linear', 'rbf', 'sigmoid'], 'C': [0.001, 0.01, 0.1, 1, 10]}
svc = svm.SVC()
grid = GridSearchCV(svc, parameters, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
grid.score(X_test, y_test)
params = grid.best_params_
```

Figure 28 - An example of how to use GridSearchCV

The best parameters to fit this model are stored in params. As we can see in the image below, those are the best parameters found with GridSearchCV.

```
params
✓ 0.0s
{'C': 1, 'kernel': 'poly'}
```

Figure 29 - Best Parameters Found

C is equal to the regularisation parameter, it controls how sensitive the model is to the data points. While the kernel is essentially the function that separates the classes on a hyperplane according to the sklearn library’s documentation (scikit-learn, n.d.). Although GridSearchCV intends to find the best parameters, different dataset sizes can affect how it performs, it can for instance find different values that are considered the best, for this reason, it is always good to remember that the intention of a machine

learning solution is not to find the perfect solution but the solution that generalises the best. As demonstrated by (Althnian et al., 2021) a limited dataset can affect the performance of the model.

Those best parameters can then be used to train the model by passing straight into the SVM model, however I decided to manually change those parameters. As I previously mentioned, this example is using the 1k dataset, when using a 100k subset, the best parameters found are actually different. It finds that the kernel “rbf” is the most accurate. For this reason, I have used the parameter C equals 1 and kernel equal “rbf” to train of the SVM models independently of the size of the dataset. This is crucial so that results can be compared and valid.

To train the SVM model I simply create an instance of the classifier, used the parameters I have mentioned above. Then, after training I used the method score, to get an accuracy of the given model.

```
clf = svm.SVC(C=1,kernel='rbf')  
clf.fit(X_train,y_train)  
clf.score(X_test,y_test)
```

*Figure 30 - Code Snippet of The SVM Training*

I then followed the same method for all different models, although GridSearchCV was only done for the SVM model due to it taking extremely long with 100k samples, for instance, I estimated that it could have taken a week of training to find the best parameters for the RandomForest model. As I was time and resources constrained I was only able to run a GridSearchCV using the SVM model. The other models were trained as follows:

```
neigh = KNeighborsClassifier(n_neighbors=7)  
neigh.fit(X_train,y_train)  
neigh.score(X_test,y_test)
```

*Figure 31 - Code Snippet Showing the training of KNN model*

The choice for the 7 neighbours' parameters was once more based on the 100k subset, this choice produced the best accuracy for this model. Following the RandomForestModel was training using 100 estimators:

```
classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(X_train,y_train)
classifier.score(X_test,y_test)
```

Figure 32 - Code Snippet Showing the Training of the RandomForest Model

Next, I performed cross-validation on all the models, but before doing that I created a pandas dataframe to store all the results together, allowing me to compare the results of different split sizes

```
pd.options.display.float_format = "{:.16f}".format
cols = pd.MultiIndex.from_tuples([(subset_size, "SVM"),(subset_size, "KNN"),(subset_size, "RandomForest")
                                ])

rows = pd.MultiIndex.from_tuples([(split_text,"CV 5", "Accuracy"),
                                (split_text,"CV 5", "F1"),
                                (split_text,"CV 5", "Recall"),
                                (split_text,"CV 5", "Precision"),
                                (split_text,"CV 10", "Accuracy"),
                                (split_text,"CV 10", "F1"),
                                (split_text,"CV 10", "Recall"),
                                (split_text,"CV 10", "Precision"),
                                (split_text,"CV 15", "Accuracy"),
                                (split_text,"CV 15", "F1"),
                                (split_text,"CV 15", "Recall"),
                                (split_text,"CV 15", "Precision"),
                                ])

final_results = pd.DataFrame(columns=cols,index=rows)
```

Figure 33 - Creation of Pandas Dataframe capable of storing the results

All the cross-validations are done inside their respective functions. The format for functions and cross-validation of the models is as follows:



```

cv_results = cross_validate(clf,X_train , y_train, cv=5,scoring=('accuracy', 'f1_weighted', 'recall_weighted', 'precision_weighted'))

final_results.loc[(split_text,"CV 5","Accuracy"),(subset_size, "SVM")] = cv_results['test_accuracy'].mean()
final_results.loc[(split_text,"CV 5","F1"),(subset_size, "SVM")] = cv_results['test_f1_weighted'].mean()
final_results.loc[(split_text,"CV 5","Recall"),(subset_size, "SVM")] = cv_results['test_recall_weighted'].mean()
final_results.loc[(split_text,"CV 5","Precision"),(subset_size, "SVM")] = cv_results['test_precision_weighted'].mean()

cv_results = cross_validate(clf,X_train , y_train, cv=10,scoring=('accuracy', 'f1_weighted', 'recall_weighted', 'precision_weighted'))

final_results.loc[(split_text,"CV 10","Accuracy"),(subset_size, "SVM")] = cv_results['test_accuracy'].mean()
final_results.loc[(split_text,"CV 10","F1"),(subset_size, "SVM")] = cv_results['test_f1_weighted'].mean()
final_results.loc[(split_text,"CV 10","Recall"),(subset_size, "SVM")] = cv_results['test_recall_weighted'].mean()
final_results.loc[(split_text,"CV 10","Precision"),(subset_size, "SVM")] = cv_results['test_precision_weighted'].mean()

cv_results = cross_validate(clf,X_train , y_train, cv=15,scoring=('accuracy', 'f1_weighted', 'recall_weighted', 'precision_weighted'))

final_results.loc[(split_text,"CV 15","Accuracy"),(subset_size, "SVM")] = cv_results['test_accuracy'].mean()
final_results.loc[(split_text,"CV 15","F1"),(subset_size, "SVM")] = cv_results['test_f1_weighted'].mean()
final_results.loc[(split_text,"CV 15","Recall"),(subset_size, "SVM")] = cv_results['test_recall_weighted'].mean()
final_results.loc[(split_text,"CV 15","Precision"),(subset_size, "SVM")] = cv_results['test_precision_weighted'].mean()

```

*Figure 34 - An example of how the models were Cross Validated*

Each model is cross-validated three times, and each time the number of folds is increased by 5. The metrics stored, following the discussion in session 4.7 are Accuracy, F1, Recall and Precision. The results are also stored inside a CSV file.

The results of these implementations are discussed in the next chapter.

## 5.2. Bot Account Identification Model

This model uses a different subset than the previous one, as a quick recap, this model's aim is to correctly identify bot accounts looking exclusively at the details of the accounts. This implementation can be found in the script "model\_2.py"

The first step is to import all the necessary libraries. It is important to note that this model requires a different set of libraries compared to the previous one. In addition, we also imported the StandardScaler, which is a scaling method used in machine learning models. It removes the mean and scales the data to unit variance. This method is essential when dealing with features that contain different value ranges. Essentially, it standardises every feature in the dataset to an equal range, even when they initially have different ranges.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
```

Figure 35 - Code Snippet Showing the libraries used

After slicing the labels into a variable called *y*, similar to the previous model. Then I perform some specific dataset operations. As there are some DateTime type values in this dataset, it is important to split it into day, month, year, hour, minutes and seconds to be able to use those values in our model. The script below demonstrates how this process was done:

```
Run Cell | Run Above | Debug Cell | Go to [4]
# %%
dataset['created_at'] = pd.to_datetime(dataset['created_at'],format='%c')
dataset['Created_Day'] = dataset['created_at'].dt.day
dataset['Created_Month'] = dataset['created_at'].dt.month
dataset['Created_Year'] = dataset['created_at'].dt.year
dataset['Created_Hour'] = dataset['created_at'].dt.hour
dataset['Created_Minute'] = dataset['created_at'].dt.minute
dataset['Created_Second'] = dataset['created_at'].dt.second
X = dataset.drop(columns=['label','id','user_id','created_at'])
```

Figure 36 - Code Snippet displaying the further processing of the dataset for the second model

After preparing the dataset, I can then split it into train-test splits. As I have previously mentioned, I have used the train-test split methodology of using different splits and comparing the results. However, for a better experience for the reader, this implementation displays the code and results of using an 80/20 test split. Following the splits, the standard scaler is used to standardise the data:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, stratify=y, random_state=42)

scalar = StandardScaler()
X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)
```

Figure 37 - An Example of how StandardScaler was used

GridSearch is once more used, in this case, I created an individual function for each of the models, returning the best parameters, these are then used to train the model. I have decided once again, to manually enter the parameters to ensure that the same parameters are used across different train-test set splits as sometimes a slight variance in the data can affect the results of the hyperparameter tuning. The functions are as following:

```
def GridSearch_RandomForest(X_train, X_test, y_train, y_test):
    grid = {'n_estimators': [100, 200, 300, 1000],
            'max_depth': [None, 5, 10, 20, 30],
            'max_features': ['log2', 'sqrt'],
            'min_samples_split': [2, 4, 6],
            'min_samples_leaf': [1, 2, 4]}
    gvc = GridSearchCV(RandomForestClassifier(random_state=42), grid, cv=10, n_jobs=-1, scoring='accuracy')
    gvc.fit(X_train, y_train)
    print(gvc.score(X_test, y_test))
    return gvc.best_params_
```

*Figure 38 - An Example of Using GridSearchCV for SVM*

```
def GridSearch_DecisionTree(X_train, X_test, y_train, y_test):
    decisionTreeGrid = {
        'criterion': ['gini', 'entropy'],
        'splitter': ['best', 'random'],
        'max_depth': [None, 5, 10, 20, 30],
        'min_samples_split': [2, 4, 6],
        'min_samples_leaf': [1, 2, 4],
        'max_features': ['log2', 'sqrt'] }
    gvc = GridSearchCV(DecisionTreeClassifier(random_state=42), decisionTreeGrid, cv=10, n_jobs=-1, scoring='accuracy')
    gvc.fit(X_train, y_train)
    print(gvc.score(X_test, y_test))
    return gvc.best_params_
```

*Figure 39 - An Example of Using GridSearchCV for DecisionTree*

```
def GridSearch_SVM(X_train, X_test, y_train, y_test):
    svmGrid = { 'C': [0.1, 1, 10],
                'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
    gvc = GridSearchCV(SVC(random_state=42), svmGrid, cv=10, n_jobs=-1, scoring='accuracy')
    gvc.fit(X_train, y_train)
    print(gvc.score(X_test, y_test))
    return gvc.best_params_
```

*Figure 40 - An Example of Using GridSearchCV for RandomForest*

Please note that I have left the best parameter as a comment on the script for guidance only. As I mentioned, those are manually entered into the models. Every model is trained in its own function as following:

```

# Random Forest Classifier
def RandomForestModel(X_train, X_test, y_train, y_test):
    #{'max_depth': None, 'max_features': 'log2',
    # 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
    rf_classifier = RandomForestClassifier(max_depth=None,
                                         max_features='log2',
                                         min_samples_leaf=1,
                                         min_samples_split=2,
                                         n_estimators=100,
                                         random_state=42)

    # Fit the model
    rf_classifier.fit(X_train, y_train)
    # Calculate the accuracy
    accuracy = rf_classifier.score(X_test, y_test)
    print("Accuracy: ", accuracy)
    return rf_classifier

```

Figure 41 - Code Snippet for the training of the RandomForest Model

```

def DecisionTreeModel(X_train, X_test, y_train, y_test):
    #{'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2',
    # 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
    decisionTree = DecisionTreeClassifier(criterion='entropy',
                                         max_depth=5,
                                         max_features='log2',
                                         min_samples_leaf=2,
                                         min_samples_split=2,
                                         splitter='best',
                                         random_state=42)

    decisionTree.fit(X_train, y_train)
    decisionTree.score(X_test, y_test)
    return decisionTree

```

Figure 42 - Code Snippet for the training of the DecisionTree Model

```

def SVM_Model(X_train, X_test, y_train, y_test):
    # {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
    svm = SVC(C=10, gamma=0.1, kernel='rbf', random_state=42)
    svm.fit(X_train, y_train)
    svm.score(X_test, y_test)
    return svm

```

Figure 43 - Code Snippet for the training of the SVM model

To finalise the process, I am using stratified k-fold cross-validation. To implement this, I have used the `cross_validation` function provided by `sklearn`. Each model is cross-

validated using a specific function and the results are displayed on the screen using Pandas. All the functions are implemented in a similar manner to the one shown below:

```
def RandomForest_CV(X_train, y_train, rf_classifier):  
    cv = cross_validate(rf_classifier, X_train, y_train, cv=5,  
                        scoring=('accuracy', 'f1_weighted', 'recall_weighted', 'precision_weighted'))  
    print("Random Forest - Cross Validation Results")  
    print("Accuracy: ", cv['test_accuracy'].mean())  
    print("F1: ", cv['test_f1_weighted'].mean())  
    print("Recall: ", cv['test_recall_weighted'].mean())  
    print("Precision: ", cv['test_precision_weighted'].mean())
```

*Figure 44 - An example of how cross-validation was performed on the different models*

The results and discussion of these results are completed in the next chapter.

# Chapter 6

## Results & Discussion

### 6.1. ChatGPT Identification Model Results

Before discussing the results of the models, it is important to understand the shape of our data. I have used PCA (Principal Component Analysis), PCA works by extracting important information from the data and expensing in a set of new orthogonal variables called components (Abdi and Williams, 2010). It essentially reduces the dimensionality of a given data. In this case, it was used to reduce the dimensionality of our embedding data, reducing from 768 dimensions to 2. This was done in order to generate the following plots ordered from the 1k subset to the 100k subset:

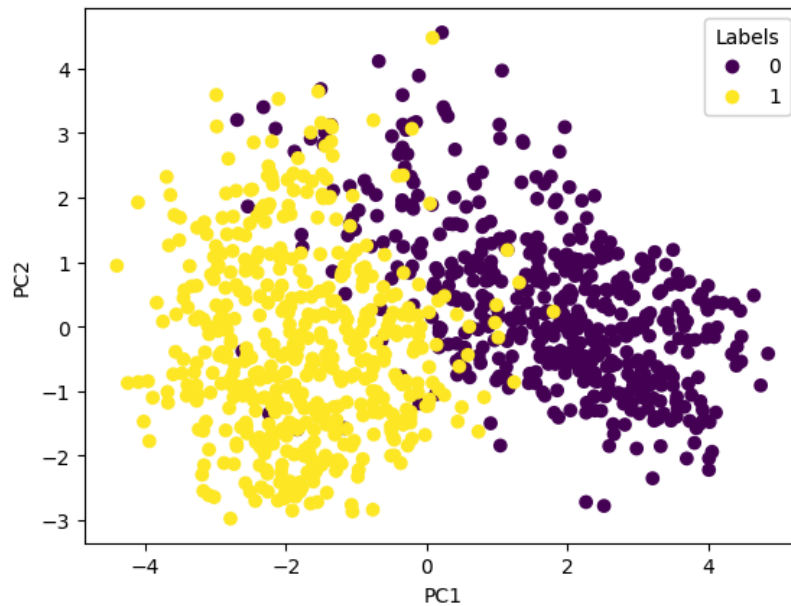


Figure 45 - Plot of the Dataset Distribution for 1k subset

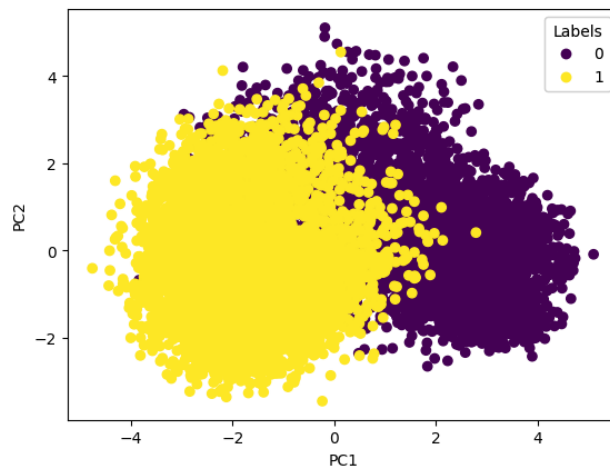


Figure 46 - Plot of the Dataset Distribution for 10k subset

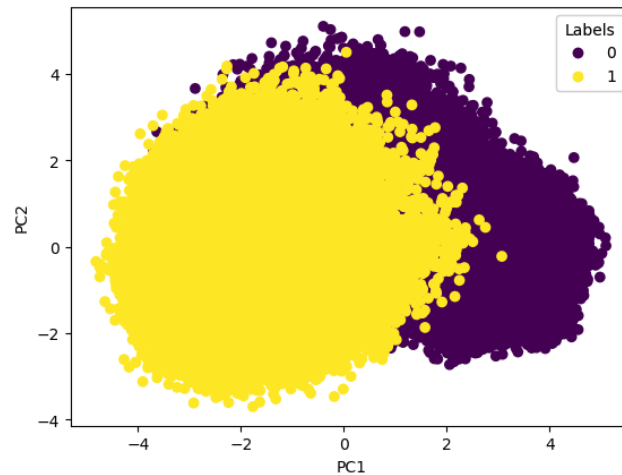


Figure 47 - Plot of the Dataset Distribution for 100k subset

As we can observe in those plots, the data is almost linear, the real tweets represented with the label 0 and the ChatGPT tweets represented with the label 1, demonstrate that there is not much variance from the 1k subset values samples values to the 100k subset values. There can be many reasons for this result, for instance, the ChatGPT model of best-fit words can likely contribute to those results.

Another interesting plot for the SVM model is to plot a decision boundary, recapping that parameters for this model were as follows: C equal to 1 and kernel equal to rbf.

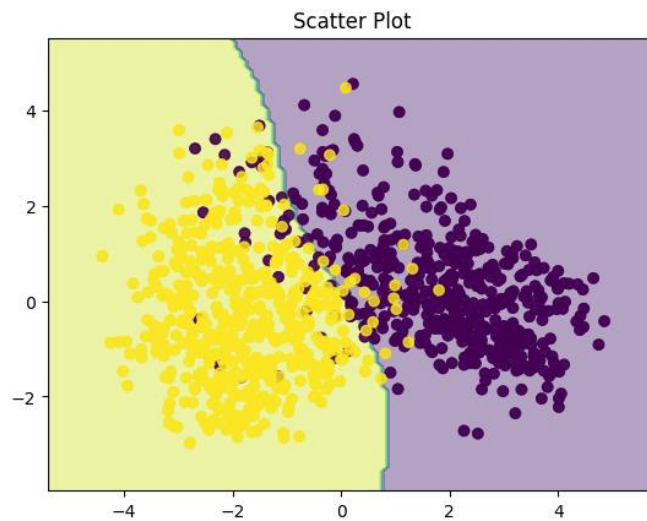


Figure 48 - An Example of The decision boundary, using the 1k subset

As we can see in the decision boundary plot, the model is able to separate most of the data into the correct classes, some plots are still incorrectly classified, although there is a strong indication of a good generalisation capacity.

Below we can visualise the performance of the model using a confusion matrix, as we can observe, the great majority of the data is correctly classified, with a small minority of samples being incorrectly classified

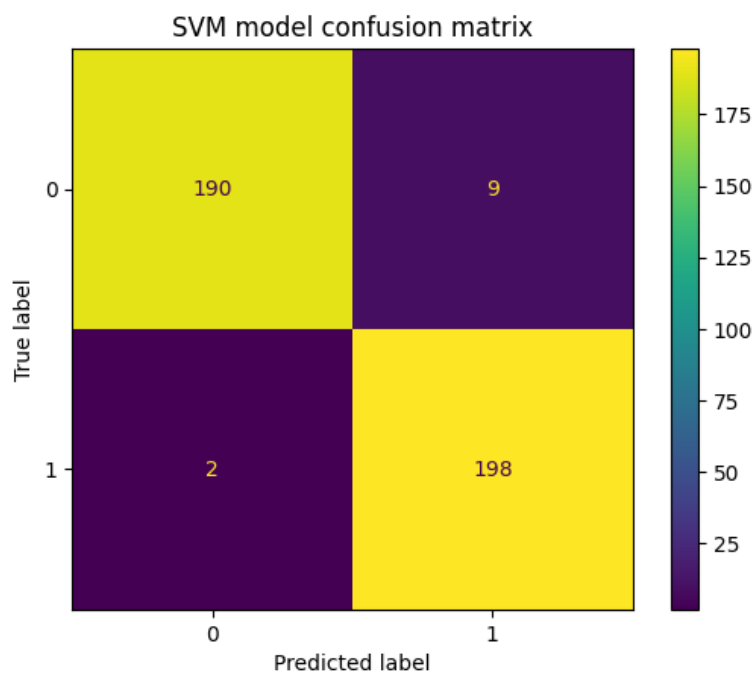


Figure 49 - SVM Model's Confusion Matrix



The KNN model still classifies most of the data correctly, although slightly lower than the SVM model. It is better to classify the ChatGPT-generated tweets than the real genuine tweets.

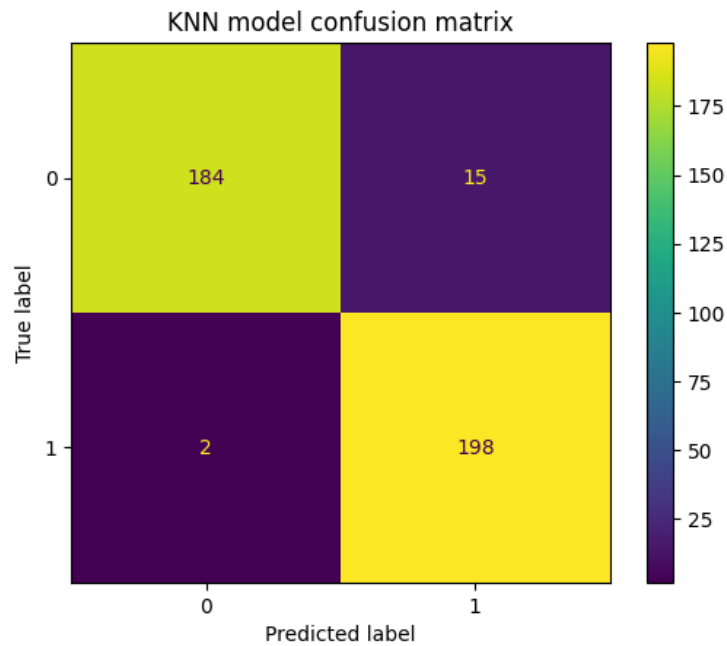
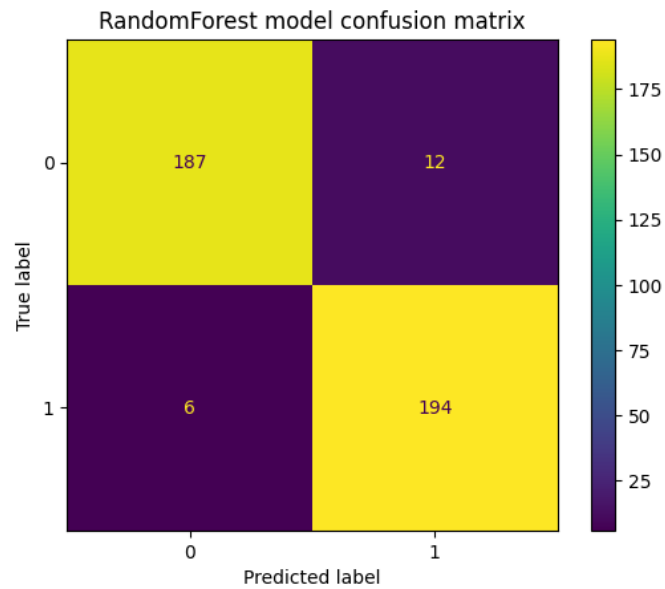


Figure 50 - KNN Model's Confusion Matrix

When it comes to the Random Forest model, it maintains an almost even capacity for classification for both classes, just a very small high ability to classify ChatGPT-generated tweets.

Figure 51 - RandomForest Model's Confusion Matrix



Before continuing, I must remind you that those results were obtained using a 60/40 train-test split. A random state was introduced here in order to ensure the reproducibility of those results, additionally to ensure the results are on the same data. An 80/20 train-test split would yield the following confusion matrices:

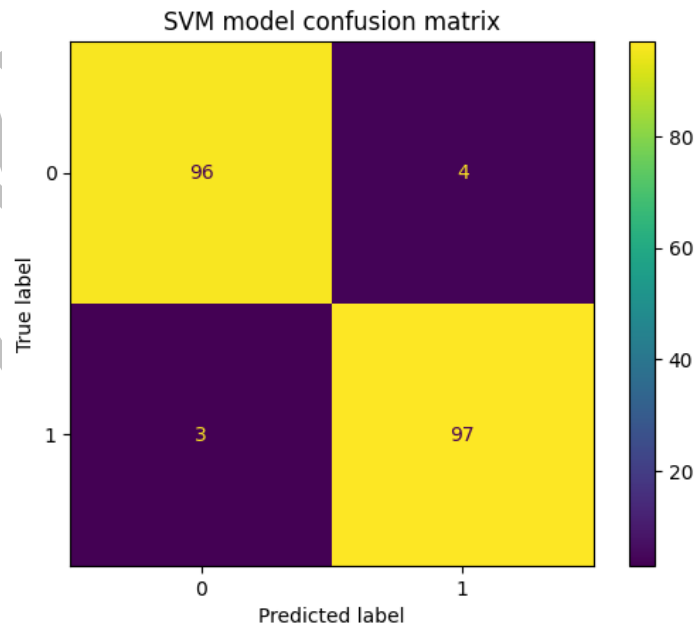


Figure 52 - 80/20 SVM Model's Confusion Matrix

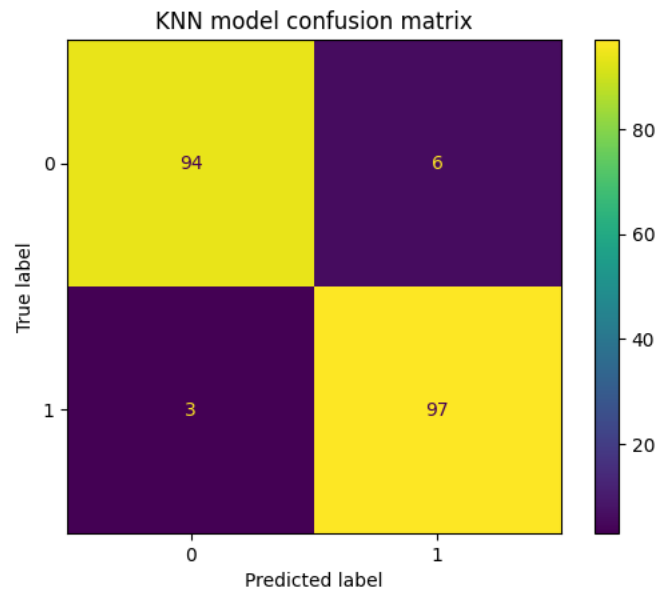


Figure 53 - 80/20 KNN Model's Confusion Matrix

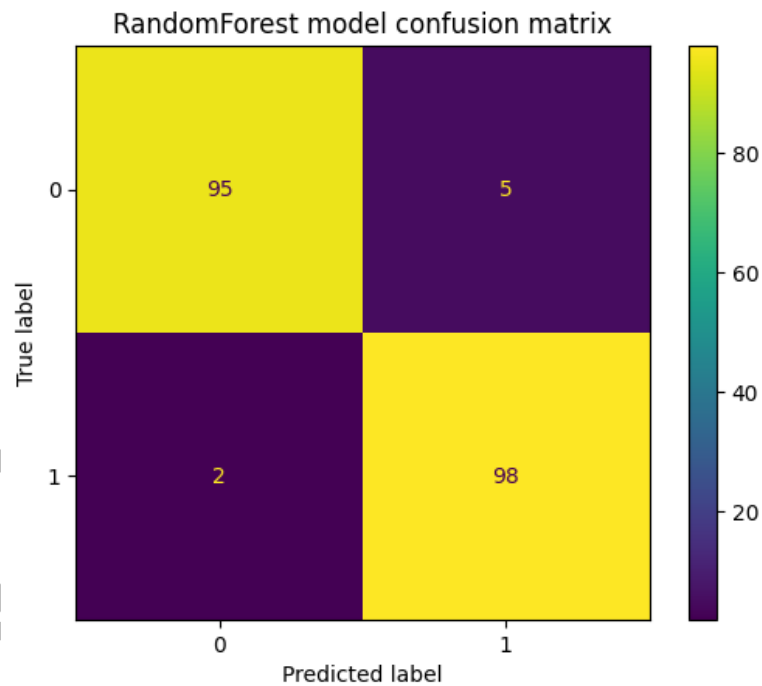


Figure 54 - 80/20 RandomForest Model's Confusion Matrix

Now let's have a look at the metrics derived from those confusion matrices, those metrics are generated using Accuracy, Precision, Recall and F1 score discussed in session 4.7. Those results are from using the 1k subset of our dataset.

Further results including the Confusion Matrices for splits 70/30 and 90/10 are included in Appendix A. A CSV file with the results of each split is also included in the support documents, those are named “subset\_1k\_metrics”.

The results of using a 60/40 train-test split are displayed in Table number 2. The metrics show that SVM is the best model overall, outperforming KNN and Random Forest models. This is probably because the embeddings have a high dimensionality of 768. Even when cross-validated into 15 folds, SVM remains superior to the other models.

When looking at table number 3, there is a slight increase in the performance of the models, with KNN reaching a precision of 97% when cross-validated to 10 and 15 folds. The SVM model still outperforms all the other with above 97 in all of the metrics being measured. The slightly more data for the training is likely the cause of this increase in performance for the models, as there is more information for the models to learn from.

			1K Subset		
			SVM	KNN	Random Forest
60/40	CV 5	Accuracy	0.968193	0.966527	0.948081
		F1	0.968179	0.966488	0.94804
		Recall	0.968193	0.966527	0.948081
		Precision	0.968672	0.967727	0.949189
	CV 10	Accuracy	0.968192	0.964887	0.948164
		F1	0.968142	0.964825	0.948083
		Recall	0.968192	0.964887	0.948164
		Precision	0.969854	0.966621	0.950589

	CV 15	Accuracy	0.969872	0.964872	0.949701
		F1	0.969799	0.964793	0.949594
		Recall	0.969872	0.964872	0.949701
		Precision	0.972306	0.967036	0.952347

Table 2 - Model's Performance Using 60/40 Split

Table number 4 shows a different type of results, instead of increasing the performance, the 80/20 train-test split indicates a slight change in the performance of the algorithms. At first, this can be strange, however, there are other reasons that can impact the results of the models, for instance, it can be due to the smaller test set available, indicating there is not enough data in the test set, it can also be that as the model learned more from the training set, and is now able to generalise more, rather than fitting to the data.

			1K Subset		
			SVM	KNN	RandomForest
70/30	CV 5	Accuracy	0.974193	0.969877	0.958417
		F1	0.974178	0.969851	0.958397
		Recall	0.974193	0.969877	0.958417
		Precision	0.974832	0.971163	0.959307
	CV 10	Accuracy	0.975631	0.969896	0.958468
		F1	0.975613	0.969869	0.958441
		Recall	0.975631	0.969896	0.958468

	CV 15	Precision	0.976435	0.971158	0.959392
		Accuracy	0.977151	0.968455	0.951218
		F1	0.977129	0.968389	0.951154
		Recall	0.977151	0.968455	0.951218
		Precision	0.977945	0.970467	0.952856

Table 3 - Model's Performance Using 70/30 Split

			1K Subset		
			SVM	KNN	RandomForest
80/20	CV 5	Accuracy	0.973601	0.969835	0.955998
		F1	0.973589	0.969803	0.955986
		Recall	0.973601	0.969835	0.955998
		Precision	0.973989	0.971053	0.95638
	CV 10	Accuracy	0.969858	0.969826	0.95481
		F1	0.969813	0.969775	0.954779

		Recall	0.969858	0.969826	0.95481
		Precision	0.971016	0.971543	0.955703
	CV 15	Accuracy	0.97235	0.967319	0.958514
		F1	0.972293	0.967241	0.958455
		Recall	0.97235	0.967319	0.958514
		Precision	0.973859	0.969513	0.960063

Table 4 - Model's Performance Using 80/20 Split

The 90/10 split showcases the most significant improvements in the accuracy, precision, recall, and F1 Score of the SVM models thus far. Achieving an incredible accuracy of 97.5% in cross-validation with 15 folds. However, the number of folds (15 in total) may indicate a potential overfitting, given the relatively small size of each fold, approximately 66 data samples per fold. I believe that this allocation doesn't provide a sufficient number of samples per fold for robust generalisation.

			1K Subset		
			SVM	KNN	RandomForest
90/10	CV 5	Accuracy	0.974333	0.967647	0.956474
		F1	0.974323	0.967618	0.956467
		Recall	0.974333	0.967647	0.956474
		Precision	0.974979	0.968923	0.956761
	CV 10	Accuracy	0.973221	0.970999	0.95985

		F1	0.973199	0.970949	0.959837
		Recall	0.973221	0.970999	0.95985
		Precision	0.974142	0.972715	0.960394
	CV 15	Accuracy	0.975461	0.967646	0.953145
		F1	0.975434	0.967538	0.953104
		Recall	0.975461	0.967646	0.953145
		Precision	0.976565	0.970316	0.954418

*Table 5 - Model's Performance Using 90/10 Split*

In contrast to the SVM model, the Random Forest model experiences a decline in accuracy during cross-validation with 15 folds. Conversely, marginal accuracy improvements are observed with 5 and 10 folds. The KNN model exhibits only slight performance improvements with this particular split

## 6.2. Bot Account Identification Model

This model works on a smaller subset comprising only 2,000 samples, exclusively containing data related to accounts and excluding any textual post data. I have selected three models: SVM, Decision Trees, and Random Forest. The reason behind selecting Random Forest is its ability to handle diverse data types and discern the most relevant features within the dataset.

I have implemented a GridSearchCV for all of those models, as the smaller dataset provides me with the capability of running these models. For each model, the results of the Hyperparameter Tuning are as follows:

- Random Forest:
  - 'max\_depth': None,



- 'max\_features': 'log2',
- 'min\_samples\_leaf': 1,
- 'min\_samples\_split': 2
- , 'n\_estimators': 100
- Decision Tree:
  - 'criterion': 'entropy',
  - 'max\_depth': 5,
  - 'max\_features': 'log2',
  - 'min\_samples\_leaf': 2,
  - 'min\_samples\_split': 2,
  - 'splitter': 'best'
- SVM
  - 'C': 10,
  - 'gamma': 0.1,
  - 'kernel': 'rbf'

Those parameters are manually entered into the models are parameters, the results are as follows:

	Random Forest	Decision Trees	SVM
Accuracy	0.9793750000000001	0.9737500000000001	0.9324999999999999
F1	0.9793735166309142	0.973747252721443	0.932452433066125
Recall	0.9793750000000001	0.9737500000000001	0.9324999999999999
Recall	0.9795058613262257	0.9739579755822009	0.9337140767692343

Table 6 - Model 2 Results

As we can observe in Table 6 the Random Forest model alongside the Decision Tree Model has virtually the same accuracy, while the SVM model still has a significant performance but falls very short of the other ones.

To better analyse the Random Forest and Decision Trees models I have plotted a confusion matrix.

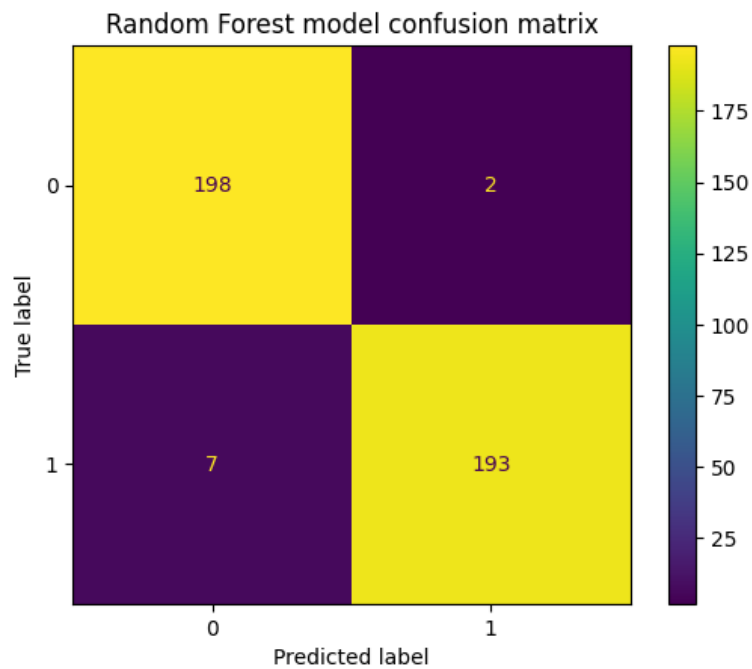


Figure 55 - Random Forest Confusion Matrix

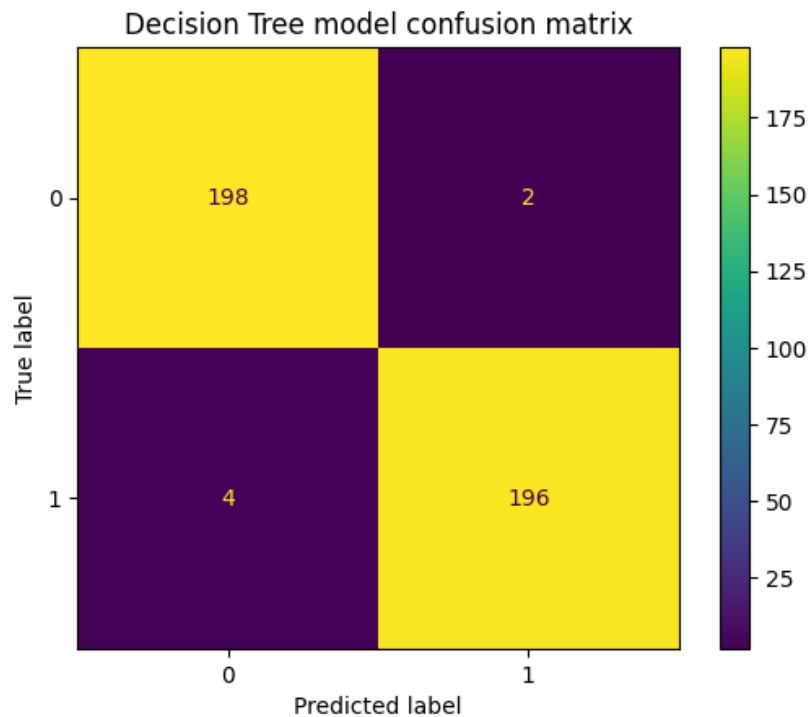
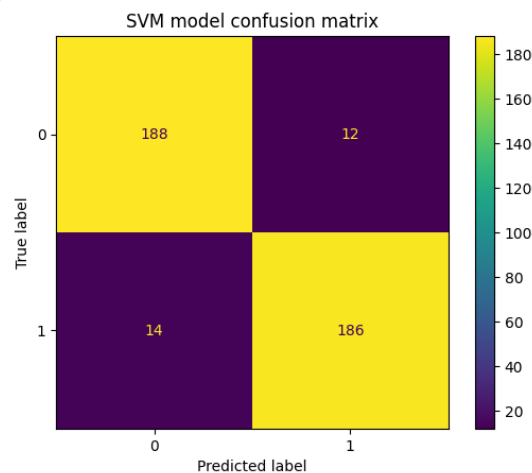


Figure 56 - Decision tree Confusion Matrix

After analysing the confusion matrices, it is apparent that there exists a important difference between the Random Forest and Decision Trees models. Although both models show a high degree of accuracy in their classifications, the Random Forest model seems to face some challenges when it comes to correctly identifying tweets generated by ChatGPT. Furthermore, the SVM model encounters some difficulty in classifying ChatGPT tweets and registers a lower overall performance compared to the other models.



### 6.3. Discussion

Different machine learning algorithms were attempted for each model, and hyperparameter tuning was used wherever possible. The first model aimed to correctly classify tweets generated with ChatGPT. The best parameters for this model were an SVM with a parameter C equal to 1 and a kernel equal to 'rbf', which achieved an average accuracy of 97% between different train-test splits. The train-test split 70/30 has also proved to be the most appropriate, giving enough information for the models while not overfitting.

For the bot account identification model, decision trees performed the best, correctly classifying almost 98% of the accounts, even better than the random forest model. It used an 80/20 test split which proved to be the best choice.

The project's aims were successfully achieved, exceeding initial expectations due to the difficulty of identifying ChatGPT text online. The various cross-validation stages and metrics give confidence that these models could be successfully used in a real-world scenario.

Furthermore, these findings provide a basis for future research opportunities.

# Chapter 7

## Conclusion

I found this project to be very challenging, but it also gave me the opportunity to explore a topic that I deeply admire. The use of AI provides numerous opportunities for businesses and consumers in various aspects of our current world. However, it also opens the door to malicious uses of such technologies. Therefore, my motivation for this project was to use AI, specifically Machine Learning, to prevent the malicious use of such technologies.

I am pleased to say that this project has successfully achieved its goals. In fact, the models performed better than I had anticipated at the beginning of the development process.

The literature review was conducted in a way that allows readers to understand the different models being created by researchers. The introduction presents my motivation and makes a compelling case for the urgent need for tools that can identify when AI tools are being used for malicious purposes.

The requirements section presents to the reader how I fulfilled the function and non-function requirements, along with an analysis of the risks associated with such a project.

My Design and Methodology section serves as an introduction to the reader for the different tools I used to build these models and what they can be used for. In the Implementation section, I demonstrated how these technologies were used in the code.

Finally, the Results and Discussion section presents the results of building these models, and a comparison of their performance is provided.

By Kleybson Sousa

# References

- Abdi, H. and Williams, L.J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4), pp.433–459.  
doi:<https://doi.org/10.1002/wics.101>.
- Albuquerque Rodrigues De Sousa, K. (2024a). *Detecting Twitter Bots and Twitter Textual Posts Generated with ChatGPT 3.5 Using Machine Learning Models*.
- Albuquerque Rodrigues De Sousa, K. (2024b). *Detecting Twitter Bots and Twitter Textual Posts Generated with ChatGPT 3.5 Using Machine Learning Models*.
- Anil Gokte, S. (n.d.). *Most Popular Distance Metrics Used in KNN and When to Use Them*. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html> [Accessed 9 May 2024].
- Ante, L. (2023). How Elon Musk’s Twitter activity moves cryptocurrency markets. *Technological Forecasting and Social Change*, 186, p.122112.  
doi:<https://doi.org/10.1016/j.techfore.2022.122112>.
- Barnard, J. (2024). *What Are Word Embeddings? / IBM*. [online] [www.ibm.com](https://www.ibm.com). Available at: <https://www.ibm.com/topics/word-embeddings> [Accessed 6 May 2024].
- Benevenuto, F., Magno, G., Rodrigues, T. and Almeida, V. (2010). *Detecting Spammers on Twitter*. [online] Seventh Annual Collaboration, Electronic messaging, Anti-. Abuse and Spam Conference. Available at: [https://gmagno.net/papers/ceas2010\\_benevenuto\\_twitterspam.pdf](https://gmagno.net/papers/ceas2010_benevenuto_twitterspam.pdf) [Accessed 31 Mar. 2024].
- Cervantes, J., García Lamont, F., López-Chau, A., Rodríguez Mazahua, L. and Sergio Ruíz, J. (2015a). Data Selection Based on Decision Tree for SVM Classification on Large Data Sets. *Applied Soft Computing*, [online] 37, pp.787–798.  
doi:<https://doi.org/10.1016/j.asoc.2015.08.048>.

- Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L. and Lopez, A. (2020b). A Comprehensive Survey on Support Vector Machine classification: Applications, Challenges and Trends. *Neurocomputing*, [online] 408(1), pp.189–215. doi:<https://doi.org/10.1016/j.neucom.2019.10.118>.
- Chicco, D. and Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, [online] 21(1). doi:<https://doi.org/10.1186/s12864-019-6413-7>.
- Davis, C.A., Varol, O., Ferrara, E., Flammini, A. and Menczer, F. (2016). BotOrNot. *Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion*. [online] doi:<https://doi.org/10.1145/2872518.2889302>.
- Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.1810.04805>.
- Dixon, S.J. (2023). *UK: most active social networks 2022*. [online] Statista. Available at: <https://www.statista.com/statistics/284506/united-kingdom-social-network-penetration/> [Accessed 31 Mar. 2024].
- Ferrara, E. (2023). Social bot detection in the age of ChatGPT: Challenges and opportunities. *First Monday*. doi:<https://doi.org/10.5210/fm.v28i6.13185>.
- Gallotti, R., Valle, F., Castaldo, N., Sacco, P. and De Domenico, M. (2020). Assessing the risks of ‘infodemics’ in response to COVID-19 epidemics. *Nature Human Behaviour*, 4. doi:<https://doi.org/10.1038/s41562-020-00994-6>.
- Ghawi, R. and Pfeffer, J. (2019). Efficient Hyperparameter Tuning with Grid Search for Text Categorization Using kNN Approach with BM25 Similarity. *Open Computer Science*, [online] 9(1), pp.160–180. doi:<https://doi.org/10.1515/comp-2019-0011>
- Hegelich, S. (2016). Decision Trees and Random Forests: Machine Learning Techniques to Classify Rare Events. *European Policy Analysis*, 2(1). doi:<https://doi.org/10.18278/epa.2.1.7>.



IBM (2023a). *What is Machine Learning?* [online] IBM. Available at: <https://www.ibm.com/topics/machine-learning>.

IBM (2023b). *What is support vector machine? / IBM*. [online] [www.ibm.com](https://www.ibm.com). Available at: <https://www.ibm.com/topics/support-vector-machine> [Accessed 6 May 2024].

Johnson, J.W. (2020). Benefits and Pitfalls of Jupyter Notebooks in the Classroom. *Proceedings of the 21st Annual Conference on Information Technology Education*. [online] doi:<https://doi.org/10.1145/3368308.3415397>.

Kaiser, J. (2014). Dealing with Missing Values in Data. *Journal of Systems Integration*, pp.42–51. doi:<https://doi.org/10.20470/jsi.v5i1.178>.

Karlesky, M. and Vander Voord, M., 2008. Agile project management. *ESC*, 247(267), p.4.

Koehrsen, W. (2020). *Random Forest Simple Explanation*. [online] Medium. Available at: <https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d> [Accessed 9 May 2024].

Morel, A.P.M. (2021). Negacionismo da Covid-19 e educação popular em saúde: para além da necropolítica. *Trabalho, Educação e Saúde*, 19. doi:<https://doi.org/10.1590/1981-7746-sol00315> (Portuguese).

Murphy, H., Aliaj, O., Fontanella-Khan, J. and Bradshaw, T. (2022). Elon Musk closes \$44bn deal to buy Twitter. *Financial Times*. [online] 28 Oct. Available at: <https://www.ft.com/content/b429b624-bf82-4ccd-bf69-b75055403952> [Accessed 31 Mar. 2024].

OpenAI. (2022). *Introducing ChatGPT*. [online] Available at: <https://openai.com/index/chatgpt> [Accessed 4 May 2024].

pandas.pydata.org. (n.d.). *Package Overview — Pandas 1.1.5 Documentation*. [online] Available at: [https://pandas.pydata.org/docs/getting\\_started/overview.html](https://pandas.pydata.org/docs/getting_started/overview.html) [Accessed 5 May 2024].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, [online] 12(85), pp.2825–2830. Available at: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html> [Accessed 5 May 2024].

Rácz, A., Bajusz, D. and Héberger, K. (2021). Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification. *Molecules*, 26(4), p.1111. doi:<https://doi.org/10.3390/molecules26041111>.

Rocha, Y.M., de Moura, G.A., Desidério, G.A., de Oliveira, C.H., Lourenço, F.D. and de Figueiredo Nicolete, L.D. (2021). The impact of fake news on social media and its influence on health during the COVID-19 pandemic: a systematic review. *Journal of Public Health*, [online] 1(10). doi:<https://doi.org/10.1007/s10389-021-01658-z>.

Roumeliotis, K.I. and Tselikas, N.D. (2023). ChatGPT and Open-AI Models: A Preliminary Review. *Future Internet*, [online] 15(6), p.192. doi:<https://doi.org/10.3390/fi15060192>.

Salian, I. (2018). *NVIDIA Blog: Supervised Vs. Unsupervised Learning*. [online] NVIDIA Blog. Available at: <https://blogs.nvidia.com/blog/supervised-unsupervised-learning/>.

Spohr, D. (2017). Fake news and ideological polarization: Filter bubbles and selective exposure on social media. *Business Information Review*, 34(3), pp.150–160. doi:<https://doi.org/10.1177/0266382117722446>.

Stalfort, J. (2019). *Hyperparameter tuning using Grid search and Random search*. [online] Medium. Available at: <https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35> [Accessed 9 May 2024].

Tong, A. (2023). Exclusive: ChatGPT Traffic Slips Again for Third Month in a Row. Reuters. [online] 7 Sep. Available at: <https://www.reuters.com/technology/chatgpt-traffic-slips-again-third-month-row-2023-09-07/>.

Tudoroiu, T. (2014). Social Media and Revolutionary Waves: The Case of the Arab Spring. *New Political Science*, 36(3), pp.346–365.  
doi:<https://doi.org/10.1080/07393148.2014.913841>.

Visual Studio Code (2023). *Documentation for Visual Studio Code*. [online] [code.visualstudio.com](https://code.visualstudio.com/docs). Available at: <https://code.visualstudio.com/docs> [Accessed 5 May 2024].

Wang, B., Wang, A., Chen, F., Wang, Y. and Kuo, C.-C. . J. (2019). Evaluating word embedding models: methods and experimental results. *APSIPA Transactions on Signal and Information Processing*, [online] 8, p.19.  
doi:<https://doi.org/10.1017/ATSIP.2019.12>.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S. and Drame, M. (2020b). Transformers: State-of-the-Art Natural Language Processing. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.  
doi:<https://doi.org/10.18653/v1/2020.emnlp-demos.6>.

World Economic Forum. (2023). Generative AI: a game-changer Society Needs to Be Ready for. [online] Available at: <https://www.weforum.org/agenda/2023/01/davos23-generative-ai-a-game-changer-industries-and-society-code-developers/>.

[www.cs.cmu.edu](http://www.cs.cmu.edu). (n.d.). *Word Embedding Demo: Tutorial*. [online] Available at: <https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html> [Accessed 9 May 2024].

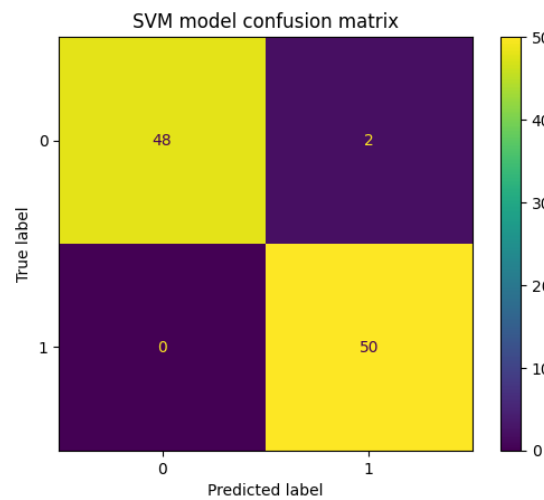
Zeng, X. and Martinez, T.R. (2000). Distribution-balanced Stratified cross-validation for Accuracy Estimation. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1), pp.1–12. doi:<https://doi.org/10.1080/095281300146272>.

# Appendix A

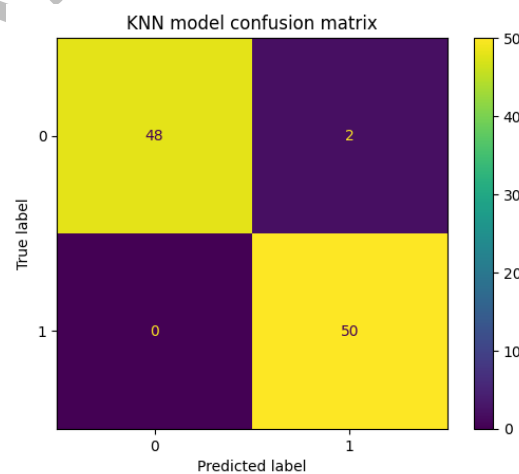
This Appendix presents the results that were no included in the main sessions.

ChatGPT identification model:

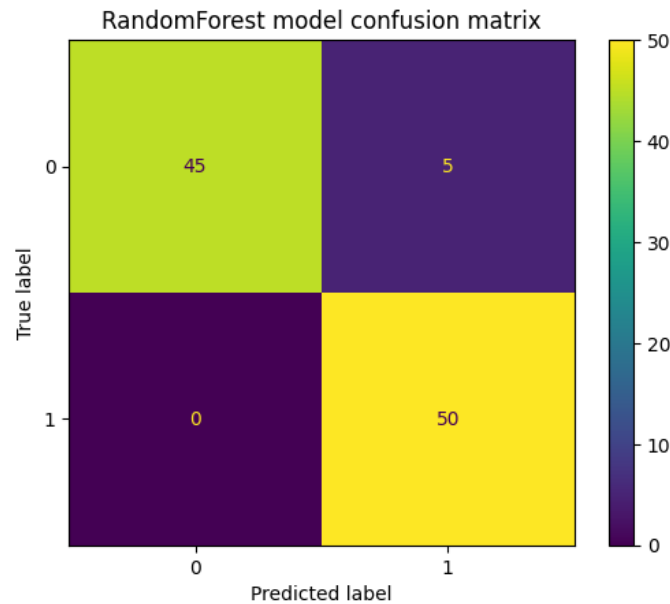
- For the 90/10 Split the confusion matrix is as follows:



*Appendix Figure 1 - 90/10 Split SVM Model Confusion Matrix*



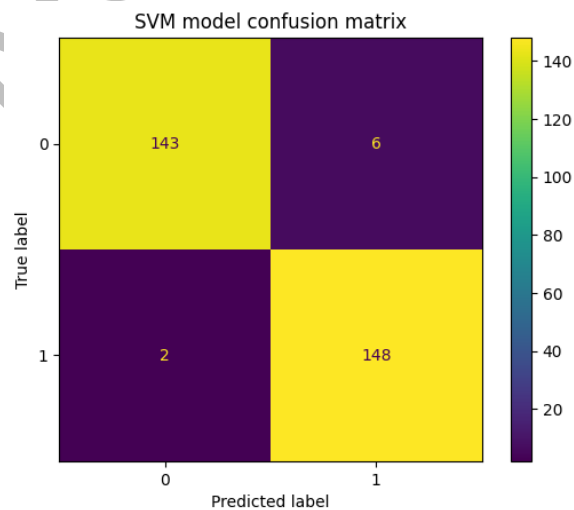
*Appendix Figure 2 - 90/10 Split KNN Model Confusion Matrix*



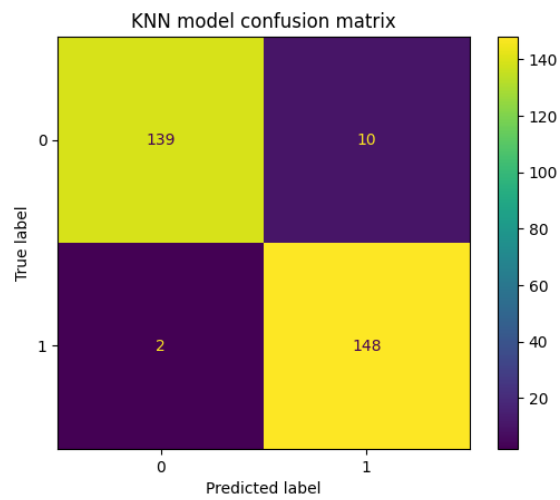
*Appendix Figure 3 - 90/10 Split RandomForest Model Confusion Matrix*

Those results indicate a possible overfitting of the data in the folds when using the 90/10 split. With those splits the model has enough data to almost perfectly fit the data, however would poorly perform when generalising.

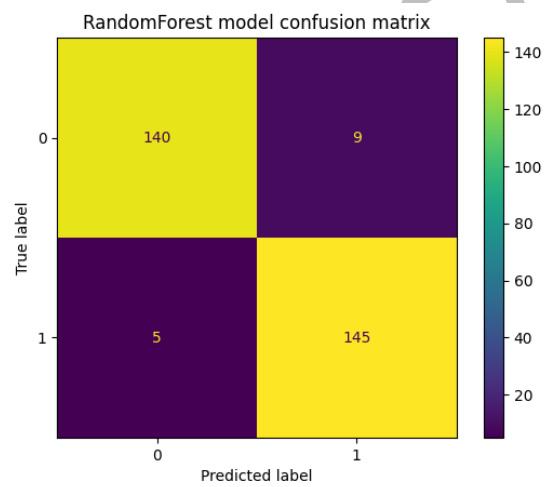
The 70/30 Split presents different results, the models still perform extremely well while not overfitting the data and containing enough data for testing.



*Appendix Figure 4 - 70/30 Split SVM Model Confusion Matrix*



*Appendix Figure 5 - 70/30 Split KNN Model Confusion Matrix*



*Appendix Figure 6 - 70/30 Split RandomForest Model Confusion Matrix*