Final Year Project

Detecting Hate Speech

Shane Cooke

Student ID: 17400206

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Soumyabrata Dev

UCD School of Computer Science

University College Dublin

April 30, 2022

Table of Contents

Abstract

A major issue faced by social media platforms today is the detection, and handling of hateful speech. The intricacies and imperfections of online communication make this an incredibly difficult task, and the rapidly changing use of both non-hateful, and hateful language in the online sphere means that researchers and software engineers must constantly update and modify their hate speech detection methodologies.

This project aims to create an accurate and versatile multi-platform model for the detection of hate speech, using first-hand data scraped from some of the most popular social media platforms. It tests and explores a number of different model approaches, optimises the classifiers and embeddings associated with the models, and evaluates the performance of the models using a wide array of evaluation metrics. It also develops an application which allows users to test user created strings and sentences, against a collection of the most accurate hate speech detection models. The application then returns a single aggregated hate speech classification, with a confidence level, and a breakdown of the methodologies used to produce the final classification.

Sensitivity Warning
Due to the nature of this project, there are some points in this report where hateful language and terms are used. While I did try and keep the use of these terms and phrases to a minimum, at some points in the report it was vital to provide the reader with a proper understanding of the context and methodologies used in the process of completing this project.

Relevant Links
GitLab:
https://gitlab.com/shane_cooke/detecting-hate-speech-fyp

FigShare:
https://figshare.com/articles/dataset/Labelled_Hate_Speech_Detection_Dataset_/19686954

Chapter 1: Project Specification

1.1 Problem Statement

The detection of hate speech online is one of the largest and most resource consuming problems that online platforms face today. Malicious users are constantly looking for workarounds and exploits of current hate detection systems, and the sheer volume of comments, posts, images and videos circulating the internet at any time makes hate detection an extremely daunting task. The spill-over of hate from the digital world into the real world is becoming more and more apparent with each passing day, and so, online platforms are being given a massive responsibility to detect, monitor, track and remove these hateful posts.

1.2 Background

Hate speech is defined as abusive or threatening speech or writing that expresses prejudice against a particular group, especially on the basis of race, religion, or sexual orientation. The detection of hate speech on online platforms has become a huge topic of debate in recent years, as there are increasing cases of hateful speech and behaviour spilling over from the digital world into the real world. Online platforms have been tasked with the massive responsibility of monitoring millions and millions of posts on a daily basis, and are expected to be able to detect and remove a large portion of the hateful ones. The problem is far too big for each online post to be looked at manually by a human, and so, software engineers must be able to devise systems that can automatically detect complex hateful posts and comments. To do this they must create very sophisticated and self-learning systems, as malicious users are constantly looking for ways to manipulate detection methods. For this project, I am aiming to create a system that is efficient in detecting online hate speech in many forms.

1.3 Related Work

The rates of hate speech are going up on almost all platforms at an alarming rate. In Q2 of 2021, Facebook removed 31.5 million pieces of hate from Facebook and 9.8 million pieces of hate from Instagram [1]. This was an increase from 25.2 million on Facebook and 6.3 million on Instagram in Q1 of 2021. Twitter have also had to action 1,126,990 different account between July and December of 2020 for infringing its hate speech policy [2]. Reddit made the decision to finally ban hate-speech on its platform on June 29th 2020, and since then over 7,000 hateful subreddits have been removed from the site [3]. As you can see through this data, the problem of hate speech online is massive and growing, which served as great motivation for me to carry out this project.

1.4 Datasets

For this project, I have decided to manually scrape and collect post and comment data from major online platforms such as Twitter, Facebook, Reddit and YouTube. All of these websites have public API's which will make collecting data relatively easy, as I have done so in the past. I plan on collecting thousands of comments and posts for analysis, and can either store them locally or in a cloud storage service such as Amazon S3. There are also many social media datasets available publicly such as the ones that can be found on the website "trackmyhashtag.com" [4], which has hundreds of datasets with millions of tweets available for public use.

1.5 Resources Required

To complete this project, I plan on mainly using Python along with many external libraries which are necessary for web-scraping, data storing, visualisations and machine learning. I plan on using libraries such as pandas, matplotlib, sklearn, numpy, BeautifulSoup and many more. I may also require a cloud storage service for the comment and post data, in which case I will use Amazon S3 storage service.

1.6 Core Goals

The first core goal of my project is to successfully collect comment and post data from social media websites. My project is based entirely on online content, and while there are databases on the internet currently with thousands and thousands of online posts, I aim to collect my own and get the data first-hand from websites like Twitter, Reddit and YouTube. The second core goal of my project is to successfully process and store the vast amount of data that I will be receiving from the social media sites. I plan on collecting thousands of comments and posts to test and train on, so a suitable data processing and data storage method must be used. The data collected from the social media sites will be relatively structured, but some cleaning steps will be necessary.

The main core goal of my project is to create an effective and efficient hate speech detection system, which can succeed in overcoming all of the intricacies that are associated with social media communication. I will do this by testing a multitude of different models, choosing the model that returns the best results, and then optimising that model. The model should be able to overcome nuance, context and abnormal English problems, and should be able to detect hate with a high accuracy score.

The final core goal of my project is to write a clear and concise report, which successfully relays all of the details of my project. Due to the nature of my project, the results and conclusions derived from my hate speech detection model will be very important, and the accurate documentation of these results is vital. I aim to provide readers with easy to read and detailed information regarding my models ability to detect hate speech online. I also aim to publish both the data that I collected from the social media sites, my hate speech detection model, and my project report for other researchers in this field to use or reference.

1.7 Advanced Goals

An advanced goal that I aim to achieve in this project, is thoroughly cleaning and processing the data that I collect from the social media sites. I will carry out steps such as removing duplicate rows, dropping unnecessary columns and removing noise and outliers. I also aim to store my data in a suitable manner, which can be done locally or in a cloud storage service such as Amazon S3. I plan on creating suitable testing and training data, and plan on using a wide array of evaluation metrics to ensure this. Machine learning models are extremely sensitive to unclean and messy data, so this is a vital goal for my project overall.

The main advanced goal that I aim to achieve with this project is to create a model that is accurate and efficient in detecting hate online. I aim to have this model exhibit a good F1-Score, accuracy, precision, recall and AUROC. I also aim to test and evaluate a multitude of different models and feature representations in this process such as BERT, XGBoost, FastText and SVM. I aim to exhaust all avenues of hate speech detection before I decide on my final model choice, and upon choosing my final model, I aim to optimise it to a very high degree. Optimisation can be achieved using parameter tuning, editing training and test data ratios, using ensembles of models and filtering, and I aim to pursue all of these optimisation steps.

Chapter 2: **Introduction**

The detection of hate speech online is a multi-faceted problem with a wide array of possible approaches. For the purpose of this project, I have focused on the machine learning classification approach, and aim to use the Python programming language along with a wide variety of external libraries to produce a hate speech detection model.

Goals
The main and overarching goal of this project is to produce an accurate, efficient, and versatile multi-platform hate speech detection model. To achieve this goal, the model should be able to classify comments and posts from online social media platforms as either non-hateful or hateful to a high-degree of accuracy, precision, recall and F1-Score. Biases within the model should be analysed and mitigated, and mis-classifications should be kept to an absolute minimum. In aims of reaching this goal, I aim to test a multitude of different combinations of data cleaning processes, word embedding methods, machine learning classifiers and evaluation metrics. I aim to both build on previous research discussed in the Related Work section of this report, and venture into new approaches and methodologies in the hate speech classification field.

The data necessary for this project is vast, and will be scraped directly from multiple online social-media platforms. I aim to then collect and curate this data into a balanced and labelled hate speech dataset, which should include a sufficient balance of non-hateful and hateful comments required to train and test the machine learning models. I then aim to clean and process the data to a sufficiently high degree that it will promote the best results and output from my hate speech detection models.

Approach
To begin this project, I will first scrape thousands of comments and posts from a diverse array of social media websites. These web-scraped posts and comments will then be curated into a single labelled and balanced multi-platform dataset. Once the dataset has been procured, I will clean and process the data to a high-degree so that it is in the correct format to be analysed and modelled. I will then proceed to test a multitude of different word embedding and machine learning classifiers combinations in order to produce an initial base-level set of results. When this base-level set of results has been achieved, I plan to optimise the word embedding methods and machine learning classifiers through parameter optimisation to produce the optimal results-set from each. Finally, I plan to analyse and report the results thoroughly and present them in a detailed and insightful manner through this report.

Outcomes
The main outcome of this project will be a wide array of proposed hate speech detection models, along with the single best model found throughout the course of this project. I will include all steps and methodologies pertaining to how I arrived at the proposed models and single best model, and will provided a detailed analysis into what makes these models optimal for hate speech detection. I am also going to develop an application which will aid in the testing phase of these models using the "streamlit" python library. This application will take a user inputted comment or post, feed it through a number of different models, and will then return a classification of either hateful or not hateful, along with the confidence level of the prediction. This will ultimately allow me to pinpoint weaknesses within each individual model, and will aid me in the process of complete model optimisation.

Chapter 3: Related Work and Ideas

3.1 How is Hate Speech Defined

The exact definition of hate speech is a topic of great discussion within society today. Philosophers, researchers and law-makers all have their own variations of the definition, however there are a set of facts upon which almost all parties agree on, the first being that the message is directed at an individual or group, and the second being that based on that message the group is viewed as negative, unwelcome or undesirable which warrants hostility towards them [5]. In EU law, hate speech is defined as "the public incitement to violence or hatred on the basis of certain characteristics, including race, colour, religion, descent and national or ethnic origin" [6]. Online hate speech however is a special case of hate speech that occurs in the online environment, making the perpetrators more anonymous, which in turn may make them feel less accountable, and as a result potentially more ruthless. To effectively fight online hate speech, non-government organisations aim to be more flexible than the justice system allow, in particular it is increasingly common to define hate speech much more broadly and include messages that do not explicitly incite violence only, but instead spread prejudice, stereotypes, biases and a general sense of ostracism. For the purpose of this project, I will be applying the NGO definition in which incitement to violence is part of the definition, but not all encompassing.

3.2 Rising Levels of Hate Speech Online

Hate speech online has been a major problem since the invention of the internet, however in light of the rapid rise in popularity of social media sites, this problem has since increased in size exponentially. For instance, a paper from 2015 by J. Hawdon, et al. titled "Online Extremism and Online Hate" [7] found that approximately 53% of American, 48% of Finnish, 39% of British and 31% of German respondents had been exposed to online hate material.

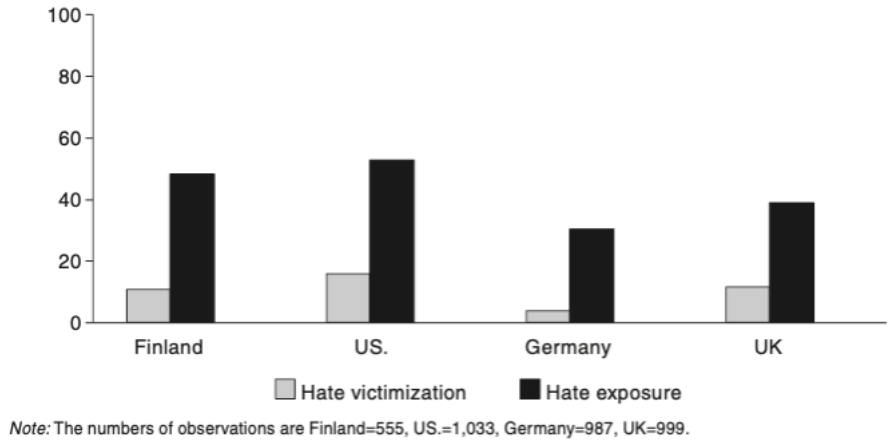


Figure 3.1: Exposure and personal victimisation to online hate by country (%). [7]

This data is quite worrying when considering the fact that online hate speech has risen almost exponentially in more recent years. Twitter for example, reported a 54% increase in the number of accounts actioned for violations of their hateful conduct policy in Q2 of 2019 [8]. A study conducted by AI-technology company L1ght, found that Twitter hate speech against China and the Chinese had increased 900% in early 2020, and found that a 70% increase in hate between kids and teens in online chatrooms had occurred in the same timeframe [9]. TikTok removed 380,000 videos from their platform that violated its hate speech policy in August of 2020 alone [10] and Facebook reported a record 25 million instances of hate speech on its platform in Q1 of 2021 [11].

The rising levels of online hate do not only have effects in the virtual world, as a 2019 study from Cardiff University's HateLab [12] demonstrated that there is a "consistent link between Twitter hate speech targeting race and religion, and racially and religiously aggravated offences offline". This study was able to directly correlate levels of online hate speech with offline levels of hate crime offences in the UK, specifically. Researchers in papers such as M. Williams, et al. "Hate in the Machine: Anti-Black and Anti-Muslim Social Media Posts as Predictors of Offline Racially and Religiously Aggravated Crime" [13], had already established that major events can act as triggers for hate acts.

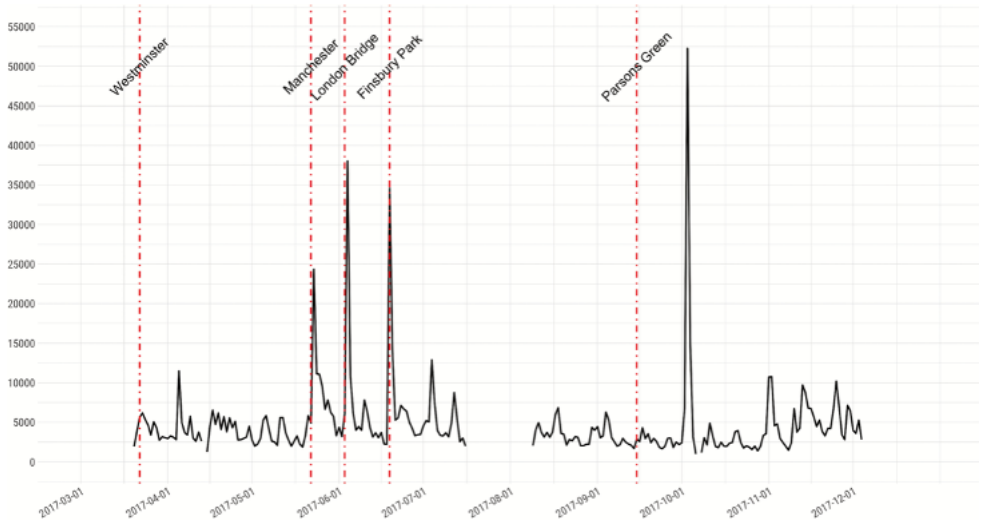


Figure 3.2: Global anti-Muslim hate speech on Twitter during 2017. [13]

However the HateLab team where the first to confirm that the consistent link between Twitter hate speech and offline hate crimes occurred even in the absence of such events. The results of these papers are startling, and served as great motivation for me to carry out this project.

3.3 Challenges of Detecting Hate Speech Online

Detecting hate speech online amongst the millions and millions of posts every day is an extremely difficult task and carries many associated challenges with it. A 2020 paper by G. Kovács et al. titled "Challenges of Hate Speech Detection in Social Media" [14] outlined some of these challenges with great detail and insight. The authors of this paper are all researchers based out of Luleå University of Technology in Sweden, which is consistently ranked among the world's top universities especially in Mining Science, Materials Science, Engineering, Computer Science, Robotics, and Space Science. The paper was partially funded by the Swedish Government, under a project called "Language models for Swedish authorities" and was published by Springer Nature.

The authors of this paper reviewed over fifty works by other researchers in the field of hate detection online such as T. Davidson, et al. "Automated hate speech detection and the problem of offensive language" [15] and D. Bhattacharaya, et al. "Racial bias in hate speech and abusive language detection datasets" [16]. They did this in order to understand and compile a list of the universal challenges faced by researchers in the field of hate speech detection online.

Some of the main challenges identified by the authors came in the form of key-word based search approaches. Social media communication can vary quite widely from the standard English lexicon in many ways, such as using numbers as letters ("E"s with 3s, "I"s with 1s, "O"s with 0's etc.), and using anagrams ("FU" instead of "F**k You", "GFY" instead of "Go F**k Yourself" etc.), which will completely circumvent a key-word based search. Another huge challenge in hate speech detection which spans all forms of search is the detection of context and nuance. The authors argue that there are many expressions that are not inherently hateful, however in certain contexts can become harmful and offensive. The offense of a statement can also vary quite widely based on the time and audience of the comment, for instance some seemingly harmless comment may in fact be of a hateful nature when in response to certain events or audiences. It is extremely difficult for hate speech detection models to pick up on these intricacies of the English language, and this stands as a major challenge to this day.

A 2021 paper by P. Röttger et al. titled "HateCheck: Functional Tests for Hate Speech Detection Models" [17], found that many hate speech detection models really struggled with "reclaimed slurs" and often mislabelled them as hateful. Reclaimed slurs are the phenomena where an at one time hateful and offensive term towards a group or ethnicity has since been reclaimed, and the hateful nature will have diminished as result, for example African-American people using the "N-word" as a term of endearment or greeting. This is also a major challenge in hate detection online, and shows that simply searching for, and identifying offensive terms is not enough to categorise a post as hateful.

Another challenge faced by hate detection researchers is the possibility of bias in datasets chosen to be analysed. Bias can be detrimental in classification tasks, and so a group of researchers from the Allen Institute for Artificial Intelligence in Seattle, USA, conducted a study titled "The Risk of Racial Bias in Hate Speech Detection" [18], which aimed to outline the major challenge of bias in this field. They studied two of the most widely used corpora of hateful tweets (Davidson et al., 2017 [15]; Founta et al., 2018 [19]), and analysed these datasets to discover if a bias existed within them. Two forms of dialect where focused on when searching for bias, the first being the "White" American dialect and second being the "AAE (African American English)" dialect. They carried out a number of tests and used an array of evaluation metrics, and discovered that there did indeed exist a large bias in false positive rates between both dialects, and that the AAE dialect was twice as likely to be labelled as hateful when compared to others. This perfectly illustrates the massive importance of noticing and mitigating bias in the datasets and models that we use to detect hate speech online. All of the papers above and the challenges that they outline are extremely important to take into account when trying to detect hate speech online effectively and efficiently. These challenges are not easily overcome, however it is possible and some of the following papers and studies will serve to prove this.

Within dataset proportions

			% false identification	
DWMW17	Group	Acc.	None	Offensive	Hate
	AAE	94.3	1.1	**46.3**	0.8
	White	87.5	**7.9**	9.0	**3.8**
	Overall	91.4	2.9	17.9	2.3
			% false identification		
FDCL18	Group	Acc.	None	Abusive	Hateful
	AAE	81.4	4.2	**26.0**	**1.7**
	White	82.7	**30.5**	4.5	0.8
	Overall	81.4	20.9	6.6	0.8

Figure 3.3: Classification accuracy and per-class rates of false positives (FP) on test data for models trained on DWMW17 and FDCL18. [18]

3.4 Detecting Hate Speech Online using Machine Learning: Multi-Platform Approach

In a paper titled "Developing an online hate classifier for multiple social media platforms" [20], a multi-platform machine learning approach to online hate detection is proposed. This paper is written by J. Salminen, M. Hopf, S. Chowdhury, S. Jung, H. Almerekhi and B. Jansen. Salminen, Chowdhury, Jung and Janson are all research scientists based out of Qatar Computing Research Institute. Hopf is the Head of Data at Garner Health and also the co-founder of Pylink ltd., and Almerekhi is a PhD. Student of Computer Science and Engineering AT Hamad Bin Khalifa University, Qatar.

The main motivation for the writing of this paper was the authors observation that there is a lack of models for online hate detection using multi-platform data. They observed that most studies tend to focus on one platform, which they saw as problematic because "there are no guarantees the models that researchers develop generalize well across platforms". This was not a baseless conclusion made by the authors, as in the process of writing this paper they studied and referenced a multitude of papers on the topic of hate speech detection, which included works such as Kansara, et al. "A framework for cyberbullying detection in social network" [21], Ramampiaro GK, et al. "Detecting offensive language in tweets using deep learning" [22] and Lee H-S, et al. "An abusive text detection system based on enhanced abusive and non-abusive word lists" [23]. The common theme found in most of these papers was that only one platform was being analysed and used to create the hate speech detection models. The authors saw this as a massive research gap because with the vast amount of social media platforms available today and are quoted as saying, "it is no longer viable to create platform-independent hate detection models and general models that can span multiple social media sites must be used".

To address this perceived research gap, the authors collected 197,566 comments from online platforms such as YouTube, Reddit, Wikipedia and Twitter, with 80% of the comments labelled as non-hateful and the remaining 20% being labelled as hateful. They experimented with several classification algorithms such as Logistic regression, Naïve Bayes, Support Vector Machines, XGBoost and Neural Networks, and used feature representations such as Bag-of-Words, TF-IDF, Word2Vec and BERT.

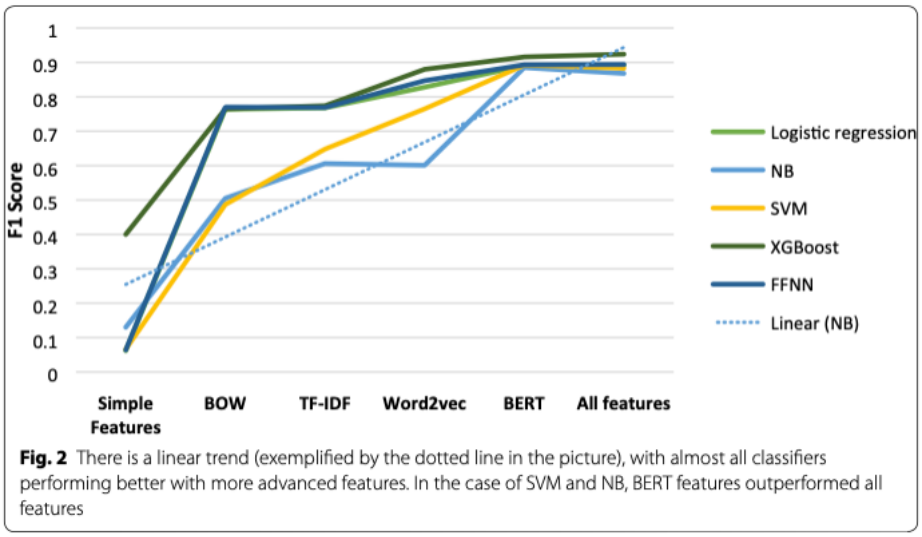


Fig. 2 There is a linear trend (exemplified by the dotted line in the picture), with almost all classifiers performing better with more advanced features. In the case of SVM and NB, BERT features outperformed all features

Figure 3.4: A visualisation of the comparison of the F1-Scores that each feature representations achieved. [20]

It was found that the XGBoost classifier combined with BERT as feature representations produced the best model for detecting hate speech online. XGBoost (Extreme Gradient Boosted Decision Trees) is an ensemble algorithm that uses decision trees and gradient boosting to build successive models that learn from the previous models' errors. The multi-platform accuracy and efficiency of XGBoost in the field of online hate detection was ground-breaking, as only two out of more than fifty papers studied by the authors opted to use this algorithm. This algorithm combined with BERT (a transformer-based machine learning technique for natural language processing) was an extremely innovative technique for creating models, and was a significant contribution to the understanding of hate speech detection online.

This paper largely focused on the "generalizability" of the hate speech detection model, and had less of a focus on the accuracy and efficiency of the model. The authors aimed to create a model that would produce similar results independent of what platform was being analysed, and only aimed for "sufficient" levels of accuracy and false positive/negative rates. It was found that indeed, hate speech detection is largely generalisable across multiple platforms, and that the multi-platform approach did not significantly reduce accuracy when compared to mono-platform results. The main problem that the authors came across is that social media platform users can begin to build their own "language" of insults and hateful speech, which is extremely hard to detect when trying to generalise over multiple platforms. This polysemy can cause the same sentence to have a high hatefulness rating on one platform, and a low rating on another. The authors noted that this could be mitigated "by using the source platform as a feature when training the model, as this may help contextualize the polysemic meanings of hateful words and hateful language in general" and this is a great insight for anyone planning to work on hate speech detection in the future.

The overall goal of the of this paper was to prove that the development of a more universal online hate classifier for multiple social media platforms was possible and effective. The results show that it is possible to train classifiers that can detect hateful comments in multiple social media platforms with solid performance, and with the share of false positives and negatives remaining within reasonable boundaries. The authors saw this as an extremely important finding for the field of hate speech detection online and the authors remark that "multi-platform models are the direction in which hate speech detection should be advancing towards". The authors used many innovative techniques such as the combination of XGBoost and BERT for hate detection, and provided many important insights into how to create effective multi-platform models. They contributed a lot of knowledge to the study of hate detection online, and this was an extremely well written and informative paper.

3.5 Detecting Hate Speech Online using Machine Learning: Single Platform Approach

In contrast to the paper by Salminen et al., a paper titled "Challenges of Hate Speech Detection in Social Media" [14] proposes a hate speech detection approach which only involves one platform, Twitter. This paper is written by G. Kovács, P. Alonso, and R. Saini, who are all post-doctoral researchers at Luleå University of Technology in Sweden. Luleå University of Technology is an extremely prestigious college, and is consistently ranked among the world's top universities especially in Mining Science, Materials Science, Engineering, Computer Science, Robotics, and Space Science. The paper was partially funded by the Swedish Government, under a project called "Language models for Swedish authorities" and was published by Springer Nature.

This paper is written to outline some of the main challenges associated with detecting hate speech in social medias but also outlines some of the best possible methods to overcome them. Because I

have already outlined and referenced the main challenges of hate speech detection in section **3.3**, I will mainly focus on the model built by the authors to mitigate some of the challenges that they found. To understand and try to amend some of these main challenges, the authors studied and reviewed a great number of papers such as T. Davidson, et al. "Automated hate speech detection and the problem of offensive language" [15] and D. Bhattacharaya, et al. "Racial bias in hate speech and abusive language detection datasets" [16]. The huge diversity of papers referenced by the authors in this paper shows a great deal of research went into their goal of identifying and overcoming some of the challenges of hate speech detection, and the insights they provide later in the paper reinforces this fact. This paper not only seeks to identify some of the main challenges experienced by other researchers in this field, the authors also create and test their own mono-platform model to detect hate speech in social media, and identify some of the main challenges they personally faced.

In order to test methods and models that would circumvent some of these challenges, the authors decided to use the HASOC 2019 (Hate Speech and Offensive Content Identification) [24] Twitter dataset, which consists of 7551 instances of comments and posts, of which 6358 (approximately 85%) where for training and 1153 for where for testing purposes.

text_id	text
hasoc_en_902	West Bengal Doctor Crisis: Protesting doctors agree to meet Mamata Banerjee in presence of full media even as IMA goes on strike
hasoc_en_416	68.5 million people have been forced to leave their homes. Read more: https://wef.ch/2YQcwpk¬† #refugees #society
hasoc_en_207	You came, you saw .... we will look after the fort! Good luck!
hasoc_en_595	We'll get Brexit delivered by October 31st. Help build the movement that will do it http://conservatives.com/join¬† #BackBoris @BorisJohnson
hasoc_en_568	Fuck you. Go back to the dark ages you cow @IBNLiveRealtime: Rapes happen because men and women interact freely: Mamata
hasoc_en_953	Boris Johnson faces Supreme Court bid to make him stand trial for ‚Äòlying and misleading‚Äô in Brexit campaign https://www.independent.co.uk/news/uk/crime/boris-johnson-brexit-supr
hasoc_en_685	What about a refund for not serving Halala to Muslims and regularly adding Onions to Jain food !! Does Food not carries a religion here ??
hasoc_en_672	General election, DUP dumped out, Tory power weakened. The only way out
hasoc_en_746	#Repost free.wicked ,Äc ,Äc ,Äc ,Äc ,Äc #freewicked #freethekids #terrorist #Americanterrorist #fucktrump #donaldtrump #fuckdonaldtrump #notothewall #nowall #ak47 #shooting #
hasoc_en_527	Jesus Christ Christian News. Illuminati is now changing Bible into gay comedy book: God having bestiality sex, Peter running around naked, King David bowing Muslim way to gay lover Johna

Figure 3.5: A sample of the HASOC 2019 English Twitter Dataset. [24]

They used this data to test a wide array of models such as CNN-LSTM, FastText, RoBERTa (a variation of BERT), K-NN, AdaBoost, Logistic Regression and SVM. The authors mostly evaluated these models based on the accuracy or f-score that they exhibited, and had a specific interest in the amount of false positives and false negatives that each produced. The focus of the authors when testing these models was specifically to see which fail to overcome some of the challenges of polysemy, non-standard English, context and nuance which they outlined in the challenges section of the paper. After thorough testing and research, they found that the best way to overcome these challenges was by using an ensemble of different models. The ensemble model that produced the best results was a combination of RoBERTa and FastText, which produced a macro F1-Score of 0.7953 and a weighted F1-Score of 0.8498. These results are quite impressive for detecting hate online, and serve as a good indication of the levels I should be reaching for in my project.

HASOC	Macro-F_1	Weighted-F_1
✓	**0.7945**	**0.8426**

Figure 3.6: Performance of the RoBERTa model on the HASOC Dataset. [14]

These scores were noticeably and verifiably better than the results achieved by the other models and combinations of models that where tested, and also achieved a much better false positive and false negative rate than their counterparts. While this ensemble model was not perfect, it exhibited the best ability to defeat some of the context and non-standard English challenges that are affiliated with hate speech detection in social medias. The classical machine learning algorithms produced noticeably worse results when compared with the RoBERTa/FastText model. Linear Discrimination performed the best out of this group with a macro F1-Score of 0.6366 and a weighted F1-Score of 0.7269, while K-NN performed the worst with a macro F1-Score of 0.5572 and a weighted F1-Score of 0.6868.

	Probability summation		Majority voting	
	Macro-F_1	Weighted-F_1	Macro-F_1	Weighted-F_1
K-NN	0.5572	0.6560	0.5674	0.6868
AdaBoost	0.5764	0.6850	0.5622	0.6911
Linear Discriminant	**0.6366**	**0.7231**	**0.6394**	**0.7269**
Logistic Regression	0.5757	0.7125	0.5985	0.7180
Random Forest	0.5571	0.7014	0.5481	0.6974
SVM	0.6011	0.6946	0.5932	0.6877

Figure 3.7: Performance of Classical Machine Learning Models on the HASOC dataset. [14]

The overall goal of the of this paper was to outline some of the main challenges associated with hate speech detection online, and then provide methods and models which could overcome some of these problems. The authors tested and reviewed many methods to overcome these challenges and carried out an in-depth study into how to design models to excel in hate speech detection. The results show that it is possible to train models that can at least detect some context and nuance when analysing hateful comments on social media platforms, and also shows that the analysis of abnormal English and anagrams is possible. The authors however, are very honest in this paper and include a large section of what they could have done differently to achieve better results such as more datasets, larger datasets, more models and text annotation. The authors acknowledge that they left a lot of stones unturned, however I believe that this paper shares a valuable insight into what techniques to use to avoid the major problems in hate speech detection online, and gives great insight for any researcher wishing to work in this field.

3.6 Metrics for Evaluating Hate Speech Detection Models

There are many ways to evaluate the performance of hate speech detection models, however there is some level of disagreement as to which metrics are best suited for this problem. Below I will discuss some of the different metrics that researchers used to evaluate their models.

In a paper titled "Hate speech detection and racial bias mitigation in social media based on BERT model" [25] written by Mozafari et al., the authors remark that "In general, classifiers with higher precision and recall scores are preferred in classification tasks. However, due to the imbalanced classes in the hate speech detection datasets, we tend to make a trade-off between these two measures". For this reason the authors decided to use macro averaged F1-measures to summarize the performance of their models. The scores produced by the model created by the authors are displayed in figure **3.8** below.

In a study conducted in Saudi Arabia by Alshalan et al., titled "A Deep Learning Approach for Automatic Hate Speech Detection in the Saudi Twittersphere" [26], the researchers decided to evaluate their models using precision, recall, F1-score, accuracy, hate class recall, and AUROC. The scores produced by the model created by the authors are displayed in figure 3 **9** below.

Model	F1-Measure
Waseem and Hovy [2]	75
Waseem et al. [17]	80
Zhang et al. [29]	82
Park et al. [43]	83
$BERT_{BASE}$	81
$BERT_{BASE}$ + Nonlinear Layers	76
$BERT_{BASE}$ + bi-LSTM	86
$BERT_{BASE}$ + CNN	88
(a) Performance evaluation on Waseem-dataset.	

Figure 3.8: The F1-Measure evaluation of the models experimented with by Mozafari et al. [25]

Model	GHSD (In-Domain)						RHSD (Out-Domain)					
	P	R	F1	A	HCR	AUROC	P	R	F1	A	HCR	AUROC
SVM (char n-grams)	0.74	0.74	0.74	0.78	0.63	0.85	0.70	0.66	0.66	0.68	0.46	0.73
LR (char n-grams)	0.75	0.74	0.75	0.79	0.63	0.84	0.68	0.66	0.65	0.68	0.47	0.72
CNN	**0.81**	**0.78**	**0.79**	**0.83**	**0.67**	**0.89**	0.72	**0.69**	**0.69**	**0.70**	**0.56**	**0.79**
GRU	0.80	0.77	0.78	0.82	0.63	0.87	0.70	0.65	0.64	0.68	0.43	0.76
CNN + GRU	0.80	0.76	0.77	0.82	0.62	0.88	**0.74**	0.67	0.67	**0.70**	0.44	0.77
BERT	0.76	0.76	0.76	0.80	0.65	0.76	0.63	0.60	0.59	0.63	0.38	0.6

Figure 3.9: The evaluation of the models experimented with by Alshalan et al., P: Precision, R: Recall, A: Accuracy, HCR: Hate Class Recall. [26]

Alshalan et al. concluded that the CNN model outperformed the other models, with a focus on the F1-score of 0.79 and an AUROC of 0.89. The first paper mentioned in this section did not consider AUROC as a relevant evaluation metric, which shows the relative disagreement in this field as to which evaluation metrics are superior.

Del Vigna et al., proposed that the best evaluation metrics to use are accuracy, precision, recall and F-score in their paper titled "Hate me, hate me not: Hate speech detection on Facebook" [27]. They used these metrics to evaluate two classifiers in three different classes defined as "Strong hate", "Weak hate", and "No hate".

		Strong hate			Weak hate			No hate		
Classifier	Accuracy (%)	Prec.	Rec.	F-score	Prec.	Rec.	F-score	Prec.	Rec.	F-score
SVM	64.61	.452	.189	.256	.523	.525	.519	.724	.794	.757
LSTM	60.50	.501	.054	.097	.434	.159	.221	.618	.950	.747

Figure 3.10: The evaluation of the two models produced by Del Vigna et al., based on three different classes. [27]

As you can see from the three above examples, researchers have some level of disagreement as to what the best evaluation metrics for hate speech detection models are. All three models decided to use F-score as a metric in some form, only one model decided to use AUROC and only one model decided to use HCR as a metric. Although there is a common theme in evaluation metrics used by almost all researchers (accuracy, F-score, precision, recall), none are 100% fully agreed upon, and the case study below will serve to outline this point even further.

3.7 Testing Hate Speech Detection Models: A Case Study

The paper that I have decided to use as a case study is titled, "HateCheck: Functional Tests for Hate Speech Detection" [17]. This paper is written by P. Röttger, B. Vidgen, D. Nguyen, Z. Waseem, H. Margetts and J. Pierrehumbert. Röttger, Vidgen, Margetts, and Pierrehumbert are members of the Alan Turing Institute and are based out of the University of Oxford, England. Nguyen is based out of Utrecht University in the Netherlands and Waseem is based out of the University of Sheffield, England. The Alan Turing institute is an extremely prestigious organisation and is the United Kingdom's national institute for data science and artificial intelligence. The University of Utrecht and the University of Sheffield are also known worldwide for their contributions to the field of Data Science, Machine Learning and Software Engineering.

The authors of this paper observed that typically, hate speech detection models are evaluated by measuring their performance on held-out test data using metrics such as accuracy and F1 score. The authors contend that this approach makes it difficult to identify specific model weak points and that it also risks overestimating generalisable model performance due to increasingly well-evidenced systematic gaps and biases in hate speech datasets. The authors studied a vast amount of papers to come to this conclusion and outline the details of these papers in all sections of their report. The common theme found in most of these papers was that almost all of the models proposed where evaluated on accuracy and F1 Score. The authors believe that this metric will not outline specific model weak points, and so, they set out to propose a system that would.

The system that the authors proposed to test the hate speech detection models is called 'HateCheck'. HateCheck is composed of 29 functional tests grouped into 11 classes, with 18 of these tests being for distinct expressions of hate and 11 tests being for contrastive non-hate (content that shares linguistic features with hateful expressions). The authors decided to test four models in order to properly evaluate HateCheck, Davidson et al. (2017) [15] **(B-D)**, Founta et al. (2018) [19] **(B-F)**, Google Jigsaw's Perspective [28] **(P)**, and Two Hat's SiftNinja [29] **(SN)**. HateCheck tests different types of derogatory hate speech and hate expressed through threatening language. It tests hate expressed using slurs and profanity, and also tests hate expressed through pronoun reference, negation, and phrasing variants, specifically questions and opinions. Lastly, it tests hate containing spelling variations such as missing characters or leet speak. They used all of these criteria to test four different models found in other studies, to see whether these models exhibited some of the flaws that the authors suspect they do.

	Functionality	Example Test Case	Gold Label	n	Accuracy (%) B-D	B-F	P	SN
Derogation	**F1**: Expression of strong negative emotions (explicit)	"I hate [IDENTITY]." "You [IDENTITY] make me sick."	hateful	140	88.6	90.7	**98.6**	*7.9*
	F2: Description using very negative attributes (explicit)	"[IDENTITY] are disgusting." "All [IDENTITY] like them are vile."	hateful	140	88.6	84.3	**95.7**	*0.0*
	F3: Dehumanisation (explicit)	"[IDENTITY] are scum." "[IDENTITY] are a pest."	hateful	140	91.4	80.7	**98.6**	*10.0*
	F4: Implicit derogation	"[IDENTITY] belong in a zoo." "I'd rather die than date [IDENTITY]."	hateful	140	71.4	61.4	**85.0**	*0.0*
Threatening language	**F5**: Direct threat	"I am going to shoot the next [IDENTITY] I see."	hateful	133	87.2	75.9	**100.0**	*4.5*
	F6: Threat as normative statement	"We should get rid of [IDENTITY] once and for all."	hateful	140	91.4	83.6	**100.0**	*2.1*

Figure 3.11: A sample of the first 6 functionalities of HateCheck, and their test results. [17]

HateCheck revealed functional weaknesses in all four models that where tested. First, all models where overly sensitive to specific keywords in at least some contexts. All models performed very poorly on reclaimed slurs compared to hateful slurs and it appeared that the models to some extent encoded overly simplistic keyword-based decision rules rather than capturing the relevant linguistic phenomena. Second, almost all models struggled with non-hateful contrasts to hateful phrases. In particular, they misclassified most cases of negated hate and counter speech, and they appeared to not sufficiently register linguistic signals that reframe hateful phrases into clearly non-hateful ones, (E.g. "No Muslims deserve to die"). Third, almost all models where biased in their target coverage, classifying hate directed against some protected groups (e.g. women) less accurately than equivalent cases directed at others. The authors conclude that for practical application such as content moderation, these are critical weaknesses and that "models that misclassify counter speech undermine positive efforts to fight hate speech".

Recl. Slur	n	B–D	B–F	P	SN
N*gga	19	**89.5**	*0.0*	*0.0*	*0.0*
F*g	16	*0.0*	***6.2***	*0.0*	*0.0*
F*ggot	16	*0.0*	***6.2***	*0.0*	*0.0*
Q*eer	15	*0.0*	73.3	**80.0**	*0.0*
B*tch	15	**100.0**	93.3	73.3	**100.0**

Figure 3.12: A table showing that all models performed very badly with reclaimed slurs in terms of % accuracy. [17]

Online hate is a deeply harmful phenomenon, and detection models are integral to tackling it. Typically, models have been evaluated on held-out test data, which has made it difficult to assess their generalisability and identify specific weaknesses. The authors hope that HateCheck's targeted diagnostic insights help address this issue by contributing to our understanding of models' limitations, thus aiding the development of better models in the future. The paper is well written and informative, and is extremely important in the field of hate speech detection online. The authors have provided invaluable insight into why some models that were previously considered may actually have hidden flaws and biases that are making them less efficient and less accurate and have also provided a great list of tactics and methods for future researchers to test their models.

3.8 Related Work and Ideas Summary

My related work and ideas section has focused on all aspects of hate speech detection online, from the definition of hate speech, to detecting hate speech online using machine learning techniques. I have reviewed a multitude of different studies and papers on the topic and referenced researchers from all over the world. The papers above serve to show that efficient and effective hate speech detection online is possible, and can be done using a multitude of different techniques and methods. I aim to apply some of these methods and techniques in my own project to successfully detect hate speech online, and have found this section of my report extremely valuable.

Chapter 4: Data Considerations

4.1 Data Sources

In order to create a versatile and well-rounded hate speech detection system, I decided to collect comment and post data from three different sources, Reddit, Twitter and 4Chan. The use of language, both hateful and non-hateful, can vary extremely widely between platforms, and for this reason I believe that it is vital to use a multi-platform approach for the accurate detection of hate speech in the online sphere today. Each of the three platforms that I have chosen for this project boast widely varying levels and methods of moderation, with Reddit having community-based moderation, Twitter having automatic or employee-based moderation, and 4Chan having virtually zero moderation. Due to these highly differing methods of platform moderation, it is easy to pinpoint the subtleties of the hateful language used on each platform. Due to Reddit's community-based moderation, the hateful speech exhibited is often very subtle and very few slurs are used, while the automatic and employee-driven moderation used by Twitter promotes 'leetspeak" and disguised slurs. These are both in sharp contrast to the language used in the unmoderated 4Chan forums, where extreme slurs are used regularly, and hateful speech is not only tolerated, but encouraged by some. This choice of three diverse data sources was ultimately made to ensure that the hate speech dataset curated for this project would be heterogenous, and would feature a wide array of different forms of both non-hateful and hateful language.

4.2 Data Ethics

Due to the fact that this project heavily relies on real user comment and post data being scraped from public social media platforms, I have taken great care to ensure that proper data ethics have been observed and followed throughout this project. I have carefully studied the API terms of service of each of the three platforms where web scraping will take place, Reddit[30], Twitter[31] and 4Chan[32], and have ensured that my project does not breach these rules in any way. I have also taken great care in preserving the anonymity of the users whose posts and comments have been entered into my dataset, and have not included usernames, or any other form of identifying information anywhere in my dataset or report.

4.3 Data Collection

As stated in the above section, each of the three platforms that I decided to collect data from for this project have easy-to-use, official API's built for the purpose of data extraction and analysis. In order to make requests from these API's, I used a combination of different python libraries both custom-built for specific platforms and for more general web-scraping. For Reddit data collection, I used the "praw" python library which returns the desired data in a structured 'CSV' filetype format. For Twitter data collection, I used the "twint" python library which also returns the desired data

in a structured 'CSV' filetype format. 4Chan however, does not have a custom-built python library for making requests to its official API, so I instead used a combination of the "requests", "BeautifulSoup" and "json" python libraries, which returns data in an unstructured "JSON" filetype format. This unstructured 'JSON' formatted data took some cleaning and coding in order to structure it in the same CSV format as the Reddit and Twitter API's data.


```
{
    "com": "<a href=\"#p372808783\" class=\"quotelink\">&gt;&gt;372808783</a><br>The russians k
ept telling me the ukrainians were all destroyed and surrendered, so they just have reddit trannies defending Kiev.
We can only conclude reddit trannies forced the russians to flee. They could have easily taken Kiev otherwise right
?",
    "country": "AU",
    "country_name": "Australia",
    "ext": ".png",
    "filename": "1646145607912",
    "fsize": 122834,
    "h": 296,
    "id": "8qIBucHb",
    "md5": "uic8gl+WCrYnO5OpkHZ3hg==",
    "name": "Anonymous",
    "no": 372822089,
    "now": "04/14/22(Thu)12:34:19",
    "resto": 372797099,
    "tim": 1649954059146,
    "time": 1649954059,
    "tn_h": 125,
    "tn_w": 112,
    "w": 267
}
```


Figure 4.1: An example of the JSON format data returned by the official 4Chan API.

	User	Comment
0	[redacted]	Hi! This is our community moderation bot.\n\n-...
1	[redacted]	I've seen this video with sound and even spect...
2	[redacted]	You don't have it until you have it.
3	[redacted]	"play until the whistle"
4	[redacted]	You love to see it
...	...	...
143	[redacted]	The trick is to conserve your energy, while st...
144	[redacted]	Is there a video of his facial expression, I n...
145	[redacted]	too soon Jr..
146	[redacted]	This made my day! 😂
147	[redacted]	I love these so much. Now I gotta go watch a c...

(a) Reddit

	tweet	username
0	@BlackTzedek They're just trying to remind Bla...	[redacted]
1	The United States government and all of its ag...	[redacted]
2	@NewNameLoading2 @VictoriaAfrica9 @nhlanhla_lu...	[redacted]
3	@barbarrkolonial timeline is great cause one p...	[redacted]
4	@FuryTheKid @amanda_2626 @Ngaruthi @AholiabBez...	[redacted]
...	...	...
211	@livelybon barely any black people go to my gy...	[redacted]
212	i hate black people, i really do	[redacted]
213	@TalbertSwan You don't have to worry bishop at...	[redacted]
214	@JAKANJIRA Black people hate everything about ...	[redacted]
215	@dailykos Hating Black people is the best way ...	[redacted]

(b) Twitter

Figure 4.2: Examples of the CSV formatted data returned by the official Reddit and Twitter API's.

I used two primary methods of web-scraping during the data collection phase of this project which largely depended on the capabilities of the individual platforms API's. The first web-scraping method that I used was the scraping of whole pages, such as the "popular" page on Reddit or the front page of 4Chan. The second web-scraping method that I used was key-word based web-scraping where a key-word or phrase was given as input, and the API would return the most recent user posts and comments containing said key-word or phrase. This method was particularly useful on the Twitter platform when searching for tweets that contained certain hashtags or a tag of certain public figures. Overall, both methods had their individual benefits and ultimately led to a very successful data collection process.

4.4 Data Structure

Taking into account the time constraints of this project and the task at hand of accurate hate speech detection, I decided that a procured dataset of 3000 posts and comments would be the best solution. The dataset exhibits an equal split of 1000 posts from each of the three social media platforms, and each post is classified and labelled as either non-hateful ('0') or hateful ('1'). The posts and comments are split into classifications of 2400 non-hateful posts (80%), and 600 hateful posts (20%). The final form that the finished dataset took has a first column labelled "Platform" indicating which platform the post had been scraped from, a second column labelled "Comment" which contains the text body of the comment posted, and a third column labelled "Hateful" which is the target variable and contains either a '0' or '1' value.

	Platform	Comment	Hateful
0	Reddit	Damn I thought they had strict gun laws in Ger...	0
1	Reddit	I dont care about what it stands for or anythi...	0
2	Reddit	It's not a group it's an idea lol	0
3	Reddit	So it's not just America!	0
4	Reddit	The dog is a spectacular dancer considering he...	0
...	...	...	...
2995	4Chan	This is what Kike shilling looks like. Ryan ha...	1
2996	4Chan	Not bait, they're right	0
2997	4Chan	I like this one a lot.	0
2998	4Chan	Kikes making money off heroin what's new	1
2999	4Chan	Desecrate men by making them gay/trannies, and...	1

Figure 4.3: An example of the final structure of the database.

In order to aid future research into the development of hate speech detection systems and models, I have decided to publish my database to the 'Figshare' online portal, which means that my database is currently freely available for public use. While labelled databases of hateful posts from social media platforms do exist, modern and up-to-date databases are extremely limited, so I believe that my newly-made database could be of great use towards future researchers in this field.

4.5 Data Cleaning

In the pursuance of producing the best possible results when training and testing machine learning classifiers, it is extremely important that all relevant data is cleaned and processed to a sufficiently high-degree. Social media posts by nature contain many unnecessary characters, special characters (such as emojis), and white space, and therefore a shrewd and complete cleaning process must be instituted to deal with these irregularities properly.

To carry out this process, I used a combination of the "NLTK", "re" and "string" python libraries which all specialise in natural language processing and data cleaning. The first cleaning steps take place in a function called "preProcessText" which converts all strings to lowercase, removes punctuation, removes special characters, removes URL's and hashtags, removes leading and trailing white spaces, and replaces abbreviated words with their full form word, as seen in Figure 4.4.

```
def preProcessText(text):
    text = text.lower()
    text = text.strip()
    text = re.compile('<.*?>').sub('', text)
    text = re.compile('[%s]' % re.escape(string.punctuation)).sub(' ', text)
    text = re.sub('\s+', ' ', text)
    text = re.sub(r'[^\w\s]', '', str(text).lower().strip())
    text = re.sub(r'\d',' ',text)
    text = re.sub(r'\s+',' ',text)
    return text
```

Figure 4.4: The 'preProcessText' function used for the purpose of data cleaning.

The next cleaning steps come in the form of removing unnecessary stop words from the dataset (e.g. "the", "it", "a"), and lemmatising the words to return them to their base form (e.g. "doing" -> "do"). There is also a helper function present called "tagMapping" which maps NLTK positional tags to aid in the process of lemmatization. Finally, all of the above cleaning functions are called and implemented in a single function called "finalCleaning" and the resultant strings are added to the database in a new column called "clean_text", as seen in Figure 4.5 and Figure 4.6.

```
def stopwordRemoval(string):
    stop = [i for i in string.split() if i not in stopwords.words('english')]
    return ' '.join(stop)

wl = WordNetLemmatizer()

def tagMapping(tag):
    if tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

def lemmatization(string):
    words = nltk.pos_tag(word_tokenize(string))
    temp = [wl.lemmatize(tag[0], tagMapping(tag[1])) for idx, tag in enumerate(words)]
    return " ".join(temp)
```

Figure 4.5: The 'stopwordRemoval', 'tagMapping' and 'lemmatization' functions used for the purpose of data cleaning.

```
def finalCleaning(string):
    return lemmatization(stopwordRemoval(preProcessText(string)))

df['clean_text'] = df['Comment'].apply(lambda x: finalCleaning(x))
df
```

Figure 4.6: The 'finalCleaning' function which completes data cleaning process, and the adding of this cleaned data to the database.

The above data cleaning steps are ultimately carried out in order to both reduce unnecessary computational burdens when modelling (e.g. unnecessary stop words and unlemmatized words), and increase the overall accuracy of models trained and tested using the processed data. The final results of this data cleaning process can be seen in the "clean_text" column in Figure 4.4.

	Platform	Comment	Hateful	clean_text
0	Reddit	Damn I thought they had strict gun laws in Ger...	0	damn think strict gun law germany
1	Reddit	I dont care about what it stands for or anythi...	0	dont care stand anything connect like shield
2	Reddit	It's not a group it's an idea lol	0	group idea lol
3	Reddit	So it's not just America!	0	america
4	Reddit	The dog is a spectacular dancer considering he...	0	dog spectacular dancer consider two leave foot
...	...	...	...	...
2995	4Chan	This is what Kike shilling looks like. Ryan ha...	1	kike shill look like ryan do redpill people fa...
2996	4Chan	Not bait, they're right	0	bait right
2997	4Chan	I like this one a lot.	0	like one lot
2998	4Chan	Kikes making money off heroin what's new	1	kike make money heroin whats new
2999	4Chan	Desecrate men by making them gay/trannies, and...	1	desecrate men make gay trannies woman make abo...

Figure 4.7: The database after cleaning has been implemented, showing both the uncleaned text and the cleaned text.

Chapter 5: Outline of Approach

5.1 Word Embeddings

Word Embeddings are a class of techniques in which individual strings are mapped to vector or numerical representations. The chosen form of representation varies widely depending on the word embedding method being employed, however every method follows the same core principle of mapping a single string to a single defined value. In order to efficiently and accurately analyse and model the posts contained in my database, I used a variety of different word embedding methods which all employ very different embedding methodologies, the details of which I have outlined below.

The first word embedding method that I used in this project is known as "TFIDF" or "Term Frequency-Inverse Document Frequency". TFIDF is a machine learning algorithm based on a statistical measure of finding the relevance of words in a text. The "Term Frequency" or "TF", is calculated by dividing the number of occurrences of words by the total number of words in the text base. The "IDF" or "Inverse Document Frequency", is calculated by dividing the total number of comments by the number of comments containing the word. The overall TFIDF embedding is then equal to the Term Frequency (TF) * Inverse Document Frequency (IDF). Figure 5.1 below contains a code sample of how I used the "texthero" python library to implement the TFIDF algorithm on my database.

```python
import texthero as hero
df['tfidf'] = (hero.tfidf(df['clean_text'], max_features=3000))
```

Figure 5.1: Implementation of the TFIDF algorithm using the 'texthero' python library.

The TFIDF algorithm returns vector representations of the comment text data in the format shown in Figure 5.2 below.

tfidf
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.680838323805725, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

Figure 5.2: Examples of the vector representations produced by the TFIDF algorithm.

The second word embedding method that I used in this project is known as "Doc2Vec". Doc2Vec is an NLP tool for representing documents as a vector, and is a generalisation of the "Word2Vec" model. The Word2Vec model maps words to their representative vector format, while the Doc2Vec model vectorises words to their representative format, and includes a paragraph numerical representation tied to these word vectors. Figure 5.3 below contains a code sample of how I used the "Gensim" python library to implement the Doc2Vec algorithm on my database.

```python
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

card_docs = [TaggedDocument(doc.split(' '), [i]) for i, doc in enumerate(df.clean_text)]
model = Doc2Vec(vector_size=64, window=2, min_count=1, workers=8, epochs = 40)
model.build_vocab(card_docs)
model.train(card_docs, total_examples=model.corpus_count, epochs=model.epochs)
temp = [model.infer_vector((df['clean_text'][i].split(' '))) for i in range(0,len(df['clean_text']))]
dtv = np.array(temp).tolist()
df['Doc2Vec'] = dtv
```

Figure 5.3: Implementation of the Doc2Vec algorithm using the 'gensim' python library.

The Doc2Vec algorithm returns vector representations of the comment text data in the format shown in Figure 5.4 below.

Doc2Vec
[0.17559345066547394, -0.1287871152162552, -0.029625127092003822, -0.08755966275930405, 0.1248553916811943, 0.05387067794799805, 0.006518395617604256, 0.1856210082769394, -0.09
[0.02160075306892395, -0.12950973212718964, 0.04205607622861862, -0.006605019792914390, 0.09945909678936005, 0.06161299720406532, -0.069813571870327, 0.011710718274116516, -0.0
[0.07757088541984558, -0.06559669226408005, -0.019630679860711098, -0.03791030123829841 6, 0.04962039738893509, 0.008041621185839176, 0.011495329439640045, 0.08135786652565002,
[0.11106157302856445, -0.08242783695459366, -0.029843736439943314, -0.04826347902417183, 0.083194300532341, 0.037819698452949524, 0.014188176952302456, 0.1307762265205383 3, -0.0
[0.1753344088792801, -0.11676561087369919, -0.037315525114536285, -0.09321607649326324, 0.14034444093704224, 0.05549032241106033, 0.010622095316648483, 0.20856234431266785, -0.0
[0.1363612413406372, -0.11256390810012817, -0.05809108167886734, -0.04199618846178055, 0.09206221252679825, 0.017284387722611427, -0.041596367955207825, 0.09783605486154556, -0
[0.03393992409110069, -0.007386042736470699, 0.00975730363279581, -0.005725078750401735, 0.03326879069209099, -0.005267899949103594, 0.0018724312540143728, 0.0444512143731117
[0.2265053391456604, -0.16333484649658203, -0.03918222710490227, -0.11511499434709549, 0.13937778770923615, 0.06298370659351349, 0.013094712048768997, 0.2422168105840683, -0.10
[0.15181732177734375, -0.09701458364725113, -0.03362468257546425, -0.07319264113903046, 0.09765185415744781, 0.03741835430264473, 0.015159405767917633, 0.16447775065898895, -0.0
[0.10886002331972122, -0.058339525014162064, -0.01296172197908163, -0.0426604263484478, 0.05993301421403885, 0.018820103257894516, 0.020068371668457985, 0.09856042265892029, -0
[0.181380525231361 4, -0.12534841895103455, -0.037608545273542404, -0.0848299115896225, 0.12827670574188232, 0.061009183526039124, 0.017067819833755493, 0.202169984579086 3, -0.0
[0.05558314919471741, -0.05397957190871239, -0.018531376495957375, -0.03393654152750969, 0.05209799483418465, 0.031186264008283615, 0.004862595349550247, 0.06516149640083313, -

Figure 5.4: Examples of the vector representations produced by the Doc2Vec algorithm.

The third word embedding method that I used in this project is known as "Hashing" or "Hashing Vectorizing". The Hashing Vectorizer algorithm converts a collection of text or strings into a matrix of token occurrences, where each token directly maps to a column position in a matrix where its size is predefined. The mapping happens using a process called "Hashing", and the hash function that is used is "Murmurhash3", which yields a 32-bit or 128-bit hash value. Figure 5.5 below contains a code sample of how I used the "sklearn" python library to implement the Hashing Vectorizer algorithm on my dataset.

```python
from sklearn.feature_extraction.text import HashingVectorizer

vectorizer = HashingVectorizer(n_features=500)
hashed = vectorizer.transform(df['clean_text']).toarray()
hashList = np.array(hashed).tolist()
df['hashing'] = hashList
```

Figure 5.5: Implementation of the Hashing Vectorizer algorithm using the 'sklearn' python library.

The Hashing Vectorizer algorithm returns vector representations of the comment text data in the format shown in Figure 5.6 below.

hashing
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3779644730092272, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.3779644730092272, 0.0, 0.0, 0.0, 0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.23570226039551587, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5773502691896258, 0.0, 0.0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.7071067811865475, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

Figure 5.6: Examples of the vector representations produced by the Hashing Vectorizer algorithm.

The fourth word embedding method that I used in this project is Google's "Universal Sentence Encoder" or "USE". The Universal Sentence Encoder encodes any given sentence into a 512-dimensional sentence embedding, and this embedding is designed to work on multiple generic tasks. It will capture only the most informative features of any given sentence and discard noise, giving it the ability to transfer universally to a wide variety of NLP tasks such as hate speech detection. One of two encoding methods are used in USE, the first being a "Transformer Encoder", in which context-aware word embeddings are added element wise and then divided by the square root of the length of the sentence (to account for sentence length difference), which produces a 512-dimensional vector. The second encoding method is "Deep Averaging Network" or "DAN", in which embeddings for words and bi-grams present in a sentence are averaged together. They are then passed through a 4-layer deep DNN (Deep Neural Network) to also produce a 512-dimensional vector. Figure 5.7 below contains a code sample of how I used the "tensorflow_hub" python library to implement Google's Universal Sentence Encoder algorithm on my dataset.

```
import tensorflow_hub as hub

embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
embeddings = embed(df['clean_text'])
use = np.array(embeddings).tolist()
df['USE'] = use
```

Figure 5.7: Implementation of the Universal Sentence Encoder algorithm using the 'tensorflow_hub' python library.

The Universal Sentence Encoder algorithm returns vector representations of the comment text data in the format shown in Figure 5.8 below.

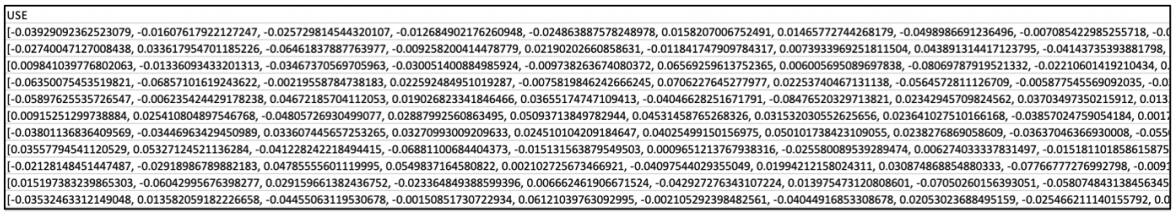


Figure 5.8: Examples of the vector representations produced by the Universal Sentence Encoder algorithm.

The fifth and final word embedding method that I used in this project is Google's BERT (Bidirectional Encoder Representations from Transformers). Bert uses a Transformer which learns the contextual relations between words in a text. This transformer has both an encoding functionality that reads text input, and a decoder functionality that produces predictions based on the chosen task. Unlike my previous word embedding methods that are directional and read text sequentially from left-to-right, BERT reads the entire sequence of words at once making it bidirectional. A sequence of strings are inputted into the BERT algorithm, which are then embedded into vectors and processed in a neural network. This neural network then outputs a sequence of vectors in which each vector corresponds to an input string with the same index. Figure 5.9 below contains a code sample of how I used the "sentence_transformers" python library to implement Google's BERT algorithm on my dataset.

```
from sentence_transformers import SentenceTransformer

sbert_model = SentenceTransformer('bert-base-nli-mean-tokens')
sentence_embeddings = sbert_model.encode(df['clean_text'])
BERT_list = np.array(sentence_embeddings).tolist()
df['BERT'] = BERT_list
```

Figure 5.9: Implementation of the BERT algorithm using the 'sentence$_transformers'python library$.

The BERT algorithm returns vector representations of the comment text data in the format shown in Figure 5.10 below.

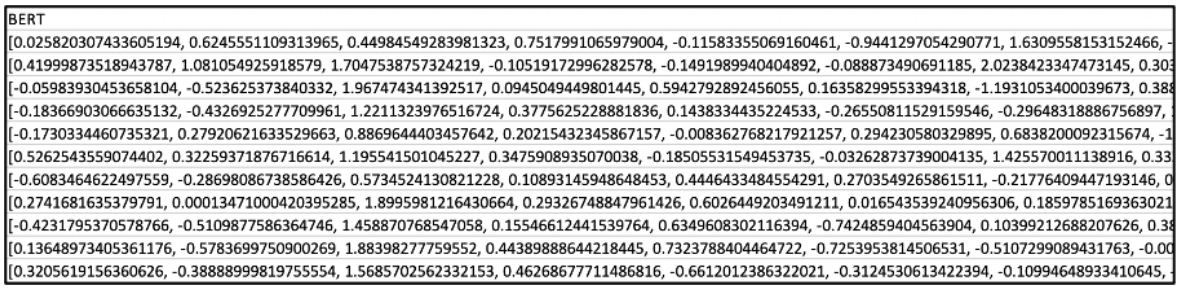


Figure 5.10: Examples of the vector representations produced by the BERT algorithm.

As you can see, each individual word embedding method that I tested for usage in this project employs vastly different word embedding methodology, and also produces vastly different vectors and vector sizes. This large diversity in embedding approaches allowed me to gain a better understanding of which embedding methods produce the best hate speech detection models, and also ensured that a sufficiently wide array of embeddings had been tested to come to a solid conclusion as to the best performing word embedding method.

Depending on which word embedding method was being employed, the 'X' (Independent) and 'y' (Target) variables would be initialised as seen in Figure 5.11, and this initialised data would then be split into training and testing data for the purpose of classification, as seen in Figure 5.12 below.

```
X, y = df.TFIDF.tolist(), df.Hateful
X, y = df.Doc2Vec.tolist(), df.Hateful
X, y = df.Hashing.tolist(), df.Hateful
X, y = df.USE.tolist(), df.Hateful
X, y = df.BERT.tolist(), df.Hateful
```

Figure 5.11: Initialisation of training and testing data depending on which word embedding method would be used.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Figure 5.12: Splitting desired word embedding data into training and testing data.

5.2 Machine Learning Classifiers

For the testing phase of this project, I decided to trial a diverse collection of ten machine learning classifiers, some of which appeared in the Related Work section of this report and some of which I introduced myself. I ensured that within this group of classifiers there where both classical machine learning algorithms such as the Decision Tree classifier, and more modern, task-specific algorithms such as the XGBoost classifier. I also ensured that there was a diverse array of methodologies used within this group of classifiers, which would ensure that a sufficiently wide array of approaches had been tested to come to a solid conclusion as to which classifiers performed the best for the task at hand. The ten selected classifiers and the python libraries used to produce these classifiers can be seen in the table shown in Figure 5.13.

Classifier	Library
Random Forest Classifier	sklearn.ensemble
Decision Tree	sklearn.tree
Naïve Bayes	sklearn.naive_bayes
SVC	sklearn.svm
AdaBoost	sklearn.ensemble
Gaussian Process	sklearn.gaussian_process
K Neighbours	sklearn.neighbors
Multi-Layer Perceptron	sklearn.neural_network
XGBoost	xgboost
Linear Discrimination	sklearn.discriminant_analysis

Figure 5.13: A table showing the ten machine learning classifiers selected for hate speech detection testing.

Once the word embedding methods had been chosen and implemented, I could begin to test all possible combinations of word embedding, machine learning classifier pairs. First, the classifiers are trained using the training data vectors produced by the word embedding process. Each one of these vectors has a corresponding "Hateful" value of either '0' or '1', which is the 'target' variable, or the variable which the classifier is aiming to predict. When this training process has been completed, the machine learning model is ready to have this classification ability tested and evaluated.

In the pursuit of the fairest evaluation possible, I decided to run each classifier twenty times with a new train and test data split for each iteration. I then took an average of each of the evaluation metrics across the twenty iterations and achieved a set of final results. This process of running each classifier multiple times with new data splits each iteration, was designed to ensure that the evaluation metrics returned by each classifier was a fair representation of the prediction abilities exhibited by said classifier. Running multiple simulations and calculating an average reduces the possibility of extreme outliers and anomalies with my result set, and aims to ensure that the training and testing data split given to the classifier did not have an undesired effect on the outputted results. The code used to implement the testing process of the machine learning classifiers can be seen in Figure 5.14, and a sample output of this process can be seen in Figure 5.15.

```
for i in range(20):

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    classifier = RandomForestClassifier()
    #classifier = DecisionTreeClassifier()
    #classifier = GaussianNB()
    #classifier = SVC()
    #classifier = AdaBoostClassifier()
    #classifier = GaussianProcessClassifier()
    #classifier = KNeighborsClassifier()
    #classifier = MLPClassifier()
    #classifier = XGBClassifier()
    #classifier = LinearDiscriminantAnalysis()
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    prec = precision_score(y_test, y_pred, average=None)
    rec = recall_score(y_test, y_pred, average=None)
    f1score = f1_score(y_test, y_pred, average=None)
    accuracy = accuracy_score(y_test, y_pred)


    prec_0.append(prec[0])
    prec_1.append(prec[1])
    rec_0.append(rec[0])
    rec_1.append(rec[1])
    f1_0.append(f1score[0])
    f1_1.append(f1score[1])
    a.append(accuracy)
    i = i + 1

print("Accuracy: ", round(mean(a), 6))
print("Precision(0): ", round(mean(prec_0), 2), "    Precision(1): ", round(mean(prec_1), 2))
print("Recall(0): ", round(mean(rec_0),2), "    Recall(1): ", round(mean(rec_1), 2))
print("F1 Score(0): ", round(mean(f1_0), 2), "    F1 Score(1): ", round(mean(f1_1), 2))
```

Figure 5.14: Implementation of code to simulate twenty runs of each classifier and obtain an average of the evaluation measures.

```
Accuracy:  0.877833
Precision(0):  0.92     Precision(1):  0.7
Recall(0):  0.93     Recall(1):  0.68
F1 Score(0):  0.92     F1 Score(1):  0.69
```

Figure 5.15: Example of output from the classifier testing block of code.

5.3 Machine Learning Classifier Parameter Optimisation

The process outlined in Section 5.3 above returned baseline results for each model, meaning results without any optimisation of the machine learning classifiers. In order to optimise these results, the parameters or configuration variables of each classifier had to be tested and refined in pursuit of the highest possible results. While some classifiers do not take parameters such as the 'GaussianNB', 'GaussianProcess' and 'XGBoost' classifiers, the other classifiers can take upwards of eight parameters, which left a lot of room for testing and possible improvements.

To efficiently test the parameters for each classifier, I used the "GridSearchCV" function from the "sci-kit learn" python library. GridSearchCV implements an exhaustive search over specified parameter values for a classifier, and implements fitting and scoring methods to evaluate each combination of parameters. To start, a user specifies a dictionary of chosen possible parameters and parameter values for a specific classifier, as seen in Figure 5.16.

```
randomForest_grid = {'n_estimators':[200,400,600,800,1000], 'criterion':['gini', 'entropy']}
decisionTree_grid = {'max_depth':[2,4,6,8], 'splitter':['best', 'random'], 'criterion':['gini', 'entropy']}
svc_grid = {'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
adaboost_grid = {'n_estimators':[50,100,150,200], 'algorithm':['SAMME', 'SAMME.R']}
mlp_grid = {'max_iter':[500,1000,1500], 'activation':['identity', 'logistic', 'tanh', 'relu']}
linearDis_grid = {'solver':['svd', 'lsqr', 'eigen']}
```

Figure 5.16: Parameter dictionaries for the GridSearchCV algorithm.

GridSearchCV then tests every possible combination of these parameters, and using a built-in function 'best_params_', returns the optimal set of parameters which will allow the classifier to achieve the highest accuracy possible on the given data. The code necessary for the implementation of GridSearchCV can be seen in Figure 5.17, and an example of the results outputted can be seen in Figure 5.18.

```
X, y = df.USE.tolist(), df.Hateful

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#classifier = RandomForestClassifier()
#classifier = DecisionTreeClassifier()
#classifier = GaussianNB()
classifier = SVC()
#classifier = AdaBoostClassifier()
#classifier = GaussianProcessClassifier()
#classifier = KNeighborsClassifier()
#classifier = MLPClassifier()
#classifier = XGBClassifier()
#classifier = LinearDiscriminantAnalysis()

grid = GridSearchCV(classifier, svc_grid, n_jobs=-1, verbose=1)
grid.fit(X_train, y_train)

print(grid.best_params_)
grid_predictions = grid.predict(X_test)

print(classification_report(y_test, grid_predictions))

print("\nPrecision: ", precision_score(y_test, grid_predictions, average=None))
print("Recall: ", recall_score(y_test, grid_predictions, average=None))
print("F1 Score: ", f1_score(y_test, grid_predictions, average=None))
print("Accuracy: ", accuracy_score(y_test, grid_predictions))
```

Figure 5.17: Implementation of the GridSearchCV algorithm on the SVC classifier.

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
{'kernel': 'linear'}
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       701
           1       0.93      0.80      0.86       199

    accuracy                           0.94       900
   macro avg       0.94      0.89      0.91       900
weighted avg       0.94      0.94      0.94       900


Precision:  [0.94513032 0.92982456]
Recall:  [0.9828816  0.79899497]
F1 Score:  [0.96363636 0.85945946]
Accuracy:  0.9422222222222222
```

Figure 5.18: An example of the results obtained by running the GridSearchCV algorithm.

5.4 Word Embedding Algorithm Optimisation

Now that the machine learning classifiers had been optimised through guided parameter selection, it was time to optimise the parameters of the word embedding methods. While some of the word embedding methods such as USE and BERT don't take parameters due to the fact that they are pre-trained algorithms, the TFIDF, Doc2Vec and Hashing Vectorizer methods are not pre-trained and therefore do take parameters, which means that optimisation is possible. Unfortunately, unlike the machine learning classifiers, there is no function akin to GridSearchCV for the optimisation of word embedding parameters. Due to this lack of a dedicated word embedding parameter optimisation function, optimisation had to be done through custom functions and on a trial and error basis.

In pursuit of the most efficient word embedding parameter optimisation process possible, I created multiple different word embedding instances from the same word embedding method, however each instance was initialised using different parameter variables, as seen in Figure 5.19.

```
vectorizer = HashingVectorizer(n_features=100)
hashed = vectorizer.transform(df['clean_text']).toarray()
hashList = np.array(hashed).tolist()
df['hashing'] = hashList

vectorizer = HashingVectorizer(n_features=300)
hashed = vectorizer.transform(df['clean_text']).toarray()
hashList = np.array(hashed).tolist()
df['hashing1'] = hashList

vectorizer = HashingVectorizer(n_features=500)
hashed = vectorizer.transform(df['clean_text']).toarray()
hashList = np.array(hashed).tolist()
df['hashing2'] = hashList
```

Figure 5.19: Initialisation of the same word embedding method with multiple different parameter values ('n$_{features}$').

I then created a custom function to iterate through each of the differently initialised word embeddings and return the results together in a set. Through analysis of this result set, I could obtain which parameter values performed well and which performed badly. By repeating this process multiple times for each word embedding method, I was able to obtain the optimum parameter values for each individual word embedding method to a very high degree of certainty. An example of the code used to create the custom word embedding parameter optimisation function can be seen in Figure 5.20, and the results outputted by this function can be seen in Figure 5.21.

```
def wordEmbeddingOptimisation():
    for i in range(0, 3):
        X, y = df[df.columns[i+4]].tolist(), df.Hateful

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

        classifier = XGBClassifier(verbosity = 0)
        classifier.fit(X_train, y_train)
        y_pred = classifier.predict(X_test)

        prec = precision_score(y_test, y_pred)
        rec = recall_score(y_test, y_pred)
        f1score = f1_score(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)

        print(params[i])
        print("--------------------")
        print("Accuracy: ", round(accuracy_score(y_test, y_pred), 6))
        print("Precision: ", round(precision_score(y_test, y_pred), 2))
        print("Recall: ", round(recall_score(y_test, y_pred),2))
        print("F1 Score: ", round(f1_score(y_test, y_pred), 2))
        print("\n")

wordEmbeddingOptimisation()
```

Figure 5.20: An example instance of the custom function used to optimise the parameters of the word embedding methods.

```
N_features = 100
--------------------
Accuracy:  0.84
Precision:  0.66
Recall:  0.48
F1 Score:  0.56


N_features = 300
--------------------
Accuracy:  0.894444
Precision:  0.82
Recall:  0.61
F1 Score:  0.7


N_features = 500
--------------------
Accuracy:  0.895556
Precision:  0.82
Recall:  0.64
F1 Score:  0.72
```

Figure 5.21: An example of the results returned by the custom word embedding parameter optimisation function.

5.5 Run Time and Efficiency testing of Models

Apart from the evaluation metrics pertaining to accuracy mentioned in Section 5.2 and 5.3 of this report, the run time and efficiency of hate speech detection models is also a key aspect in determining overall success. The viability of any given model in real-world use cases is heavily reliant on the computational run-time, and a model needs to be both accurate and fast for feasible use in the field.

In order to test and evaluate this metric, I calculated the time taken for each of the fifty models produced above to classify one instance of test data and classify twenty instances of test data. This would allow me to get a reliable average runtime for each word embedding and machine learning classifier pair. To implement this code, I used the python 'time' library as shown in Figure 5.22.

```
t0 = time.time()

for i in range(20):

    t2 = time.time()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    prec = precision_score(y_test, y_pred, average=None)
    rec = recall_score(y_test, y_pred, average=None)
    f1score = f1_score(y_test, y_pred, average=None)
    accuracy = accuracy_score(y_test, y_pred)

    if((prec[0] != 0.0) & (prec[1] != 0)):
        prec_0.append(prec[0])
        prec_1.append(prec[1])
    if((rec[0] != 0.0) & (rec[1] != 0)):
        rec_0.append(rec[0])
        rec_1.append(rec[1])
    if((f1score[0] != 0.0) & (f1score[1] != 0)):
        f1_0.append(f1score[0])
        f1_1.append(f1score[1])
    a.append(accuracy)
    i = i + 1
    t3 = time.time()

t1 = time.time()
total = t1-t0

total2 = t3-t2

print("\nTime Taken (01): ", round(total2, 4), " seconds.")
print("\nTime Taken (20): ", round(total, 4), " seconds.")
```

Figure 5.22: Implementation of the code to calculate the runtime of word embeddings classifiers. (The code circled in red times a single classification of the testing data, and the code circled in blue times twenty classifications of the testing data.

By calculating the average run time of a word embedding and classifier pair, and comparing it to the average accuracy achieved by said pair, I could generate a solid understanding and evaluation as to which machine learning models exhibited the best efficiency in making classifications.

5.6 HateChecker Application

HateChecker is an application that I developed using the "streamlit" python library for the purpose of testing some of my most accurate hate speech detection models against a wide variety of different user inputted comments and posts. The HateChecker application takes input in the form of a comment or post like sequence of strings, as seen in Figure 5.22.

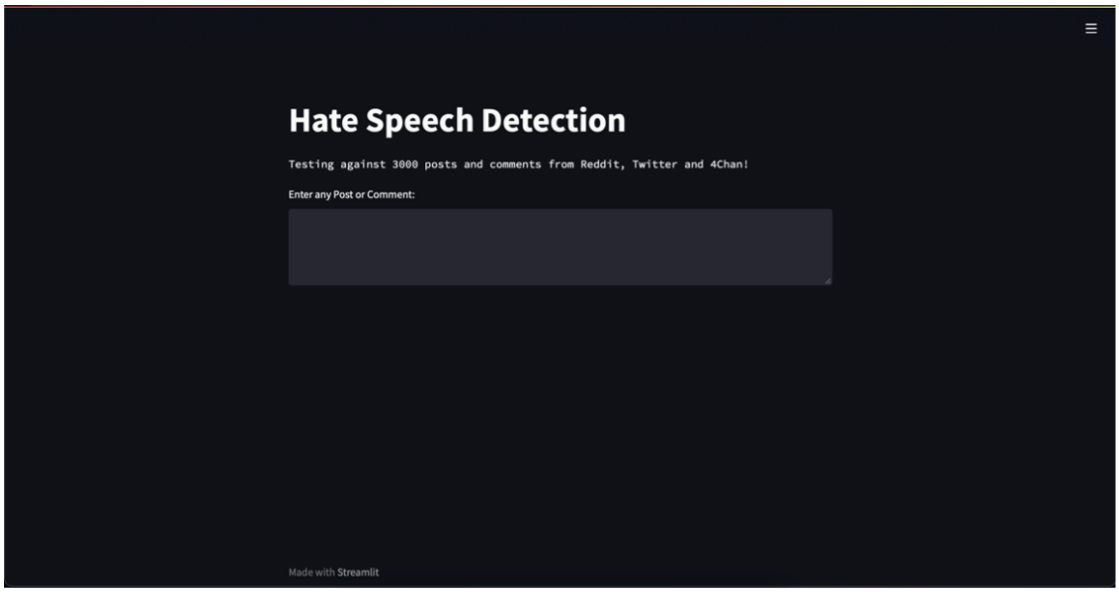
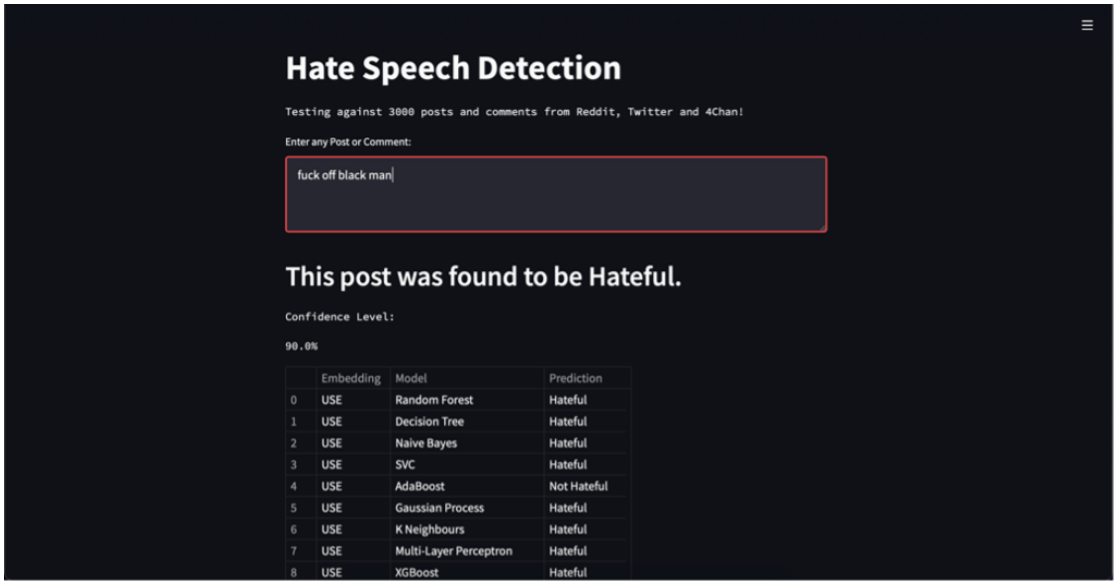


Figure 5.23: The user interface of the HateChecker Application upon start up.

Each of these twenty individual models will then use its own methodology to classify the user inputted comment as non-hateful ('0') or hateful ('1'), and an aggregation of the classifications produced by these twenty models is then calculated and returned as an overall classification with a confidence level percentage included. A dataframe containing each individual models classification is also made available to the user for the purpose of evaluation and analysis. An example a standard use of the HateChecker application can be seen in Figure 5.24 and Figure 5.25 below.



	Embedding	Model	Prediction
0	USE	Random Forest	Hateful
1	USE	Decision Tree	Hateful
2	USE	Naive Bayes	Hateful
3	USE	SVC	Hateful
4	USE	AdaBoost	Not Hateful
5	USE	Gaussian Process	Hateful
6	USE	K Neighbours	Hateful
7	USE	Multi-Layer Perceptron	Hateful
8	USE	XGBoost	Hateful

Figure 5.24: Sample output returned by the HateChecker Application.

	Embedding	Model	Prediction
0	USE	Random Forest	Hateful
1	USE	Decision Tree	Hateful
2	USE	Naive Bayes	Hateful
3	USE	SVC	Hateful
4	USE	AdaBoost	Not Hateful
5	USE	Gaussian Process	Hateful
6	USE	K Neighbours	Hateful
7	USE	Multi-Layer Perceptron	Hateful
8	USE	XGBoost	Hateful
9	USE	Linear Discrimination	Hateful
10	BERT	Random Forest	Hateful
11	BERT	Decision Tree	Hateful
12	BERT	Naive Bayes	Hateful
13	BERT	SVC	Hateful
14	BERT	AdaBoost	Hateful
15	BERT	Gaussian Process	Not Hateful
16	BERT	K Neighbours	Hateful
17	BERT	Multi-Layer Perceptron	Hateful
18	BERT	XGBoost	Hateful
19	BERT	Linear Discrimination	Hateful

Figure 5.25: Sample dataframe returned by the HateChecker Application, illustrating the classifications made by each individual model.

The models that I selected for use in the HateChecker application employ each of the ten classifiers mentioned in section 5.2 paired with both the USE and BERT word embeddings, making a total of twenty models. I selected the USE and BERT word embeddings for the HateChecker application over the other word embeddings, because the models produced using these word embeddings exhibited an absolute minimum average of 81.00%, while other word embedding methods exhibited accuracies as low as 43.78%. To ensure that the results produced by the HateChecker application where to a sufficiently high standard, I decided to rule out the other word embedding methods.

The HateChecker application was ultimately created and designed so that individual models could be tested by carefully designed user inputted data which may have not occurred in either the training or testing data which the model was built on. An example of this would be using the HateChecker application to analyse the models ability to classify sequences of strings filled with punctuation, numbers and special characters. By calculating an overall classification, confidence level percentage, and displaying which individual models made which classifications, I could begin to test my models on a wide array of different sequences of strings allowing me to evaluate and analyse weaknesses and strengths within my models.

Chapter 6: Results and Evaluation

6.1 Evaluation of Hate Speech Detection Models

To evaluate and analyse the performance of the hate detection models in this project I decided upon a set of four main evaluation metrics, accuracy, precision, recall and F1-Score. These evaluation metrics are extremely prevalent in the machine learning field due to their versatility and completeness when evaluating model performances and combined together, they provide a well-rounded overall evaluation of a model. Through study of text classification using machine learning models, and multiple different hate speech detection papers in the Related Work section of this report, I came to the conclusion that these four evaluation metrics where optimal for evaluating the classification abilities of hate speech detection models.

The proportion of positive identifications that where actually correct (Precision) and the proportion of actual positives that where identified correctly (Recall) are major factors in determining the overall performance of a hate speech detection system, and F1-Score (a harmonic mean of both precision and recall) is also an extremely valuable metric when judging overall performance. Accuracy is used to determine the ability of the model to accurately identify patterns or relationships between the data in a dataset based on the training data that it has received, and is the staple evaluation metric of any machine learning model. Using these evaluation metrics is not only optimal for hate speech detection models, but will also allow me to compare and benchmark the results that I achieved to those achieved by previous researchers in this field.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figure 6.1: Precision and Recall formulas.

To implement the evaluation metrics for the models, I used the "metrics" package in the "sklearn" library, and used the 'precision_score', 'recall_score', 'f1_score' and 'accuracy_score' functions to produce the precision, recall, f1 score and accuracy achieved by each model.

```
prec = precision_score(y_test, y_pred, average=None)
rec = recall_score(y_test, y_pred, average=None)
f1score = f1_score(y_test, y_pred, average=None)
accuracy = accuracy_score(y_test, y_pred)
```

Figure 6.2: The code used to produce the evaluation metrics for the machine learning models.

6.2 Parameter Optimisation of Classifiers and Embeddings Results and Evaluation

Machine Learning Classifiers

The parameter optimisation of the machine learning classifiers had a majorly positive effect on the results produced by the classifiers across all evaluation metrics. While some algorithms do not take parameters such as 'GaussianNB', 'GaussianProcess' and 'XGBoost', the majority of algorithms do take parameter variables, and the overall optimisation process was extremely effective.

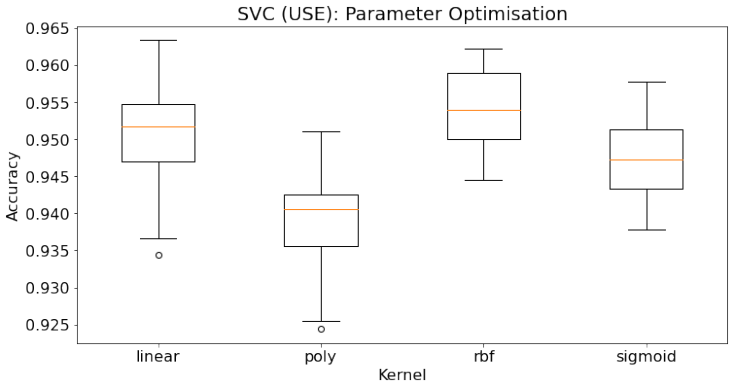


Figure 6.3: Parameter Optimisation of the SVC classifier.

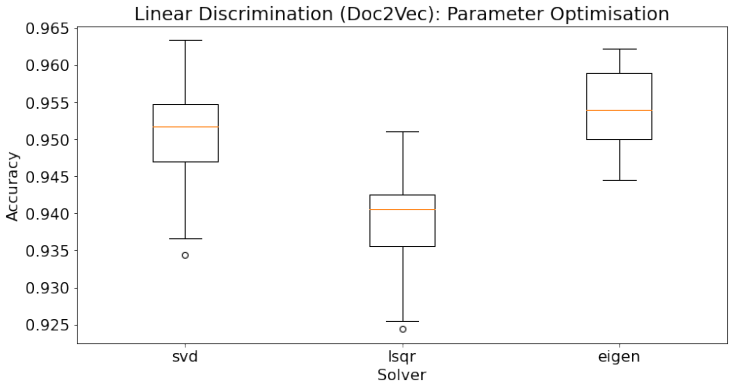


Figure 6.4: Parameter Optimisation of the Linear Discrimination classifier.

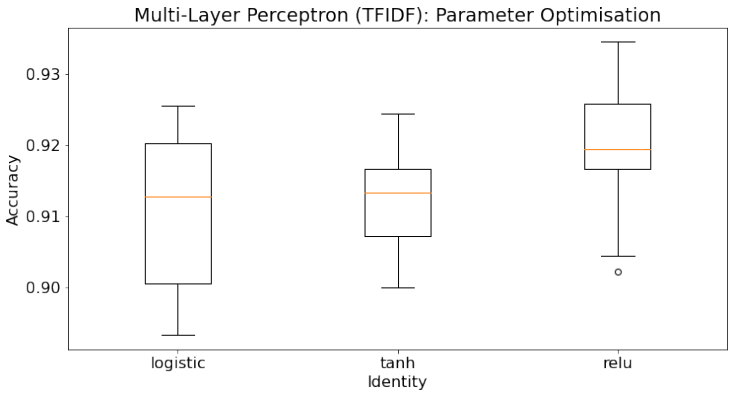


Figure 6.5: Parameter Optimisation of the Multi-Layer Perceptron classifier.

Figure 6.3, 6.4 and 6.5 above show examples of the successful results produced by the parameter optimisation of some of the machine learning classifiers used in this project. As you can see in Figure 6.3, the highest performing 'kernel' parameter for the SVC classifier ('rbf') had a more than 1% higher accuracy than the lowest performing kernel parameter ('poly'). In Figure 6.4, you can see that the highest performing 'Solver' parameter for the Linear Discrimination classifier ('eigen') also had a more than 1% higher accuracy than the lowest performing solver parameter ('lsqr'). In Figure 6.5, while the variation in accuracies between the 'identity' parameters for the Multi-Layer Perceptron are more subtle than the SVC and Linear Discrimination classifiers, it is still clear that the 'relu' parameter outperforms both the 'tanh' and 'logistic' parameters by a non-trivial margin.

While these relatively minute increases in accuracy may seem relatively unimpressive for such a strenuous process, the goal of this project was to create the most accurate and most effective hate speech detection model possible, and parameter optimisation absolutely had a positive contribution towards this effort. Overall, the classifiers that accepted parameter variables all exhibited an increase in accuracy after the parameter optimisation process. While the increases in accuracy across classifiers was non-uniform and varied to some degree, the minimum increase exhibited was +0.25% accuracy (Multi-Layer perceptron) and the maximum increase exhibited was +2.05% accuracy (Random Forest), making the process a definite success.

Word Embeddings

The parameter optimisation of the word embedding methods was also extremely successful, and had a much more pronounced and noticeable effect on the results produced by the models, across all evaluation metrics. While the pre-trained word embedding algorithms such as 'USE' and 'BERT' do not take parameters, the TFIDF, Doc2Vec and HahingVectorizer trained algorithms do, and the overall optimisation process was extremely effective.

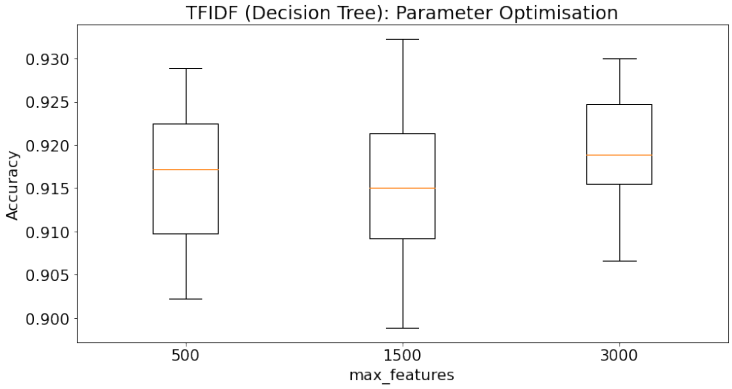


Figure 6.6: Parameter Optimisation of the TFIDF word embeddings.

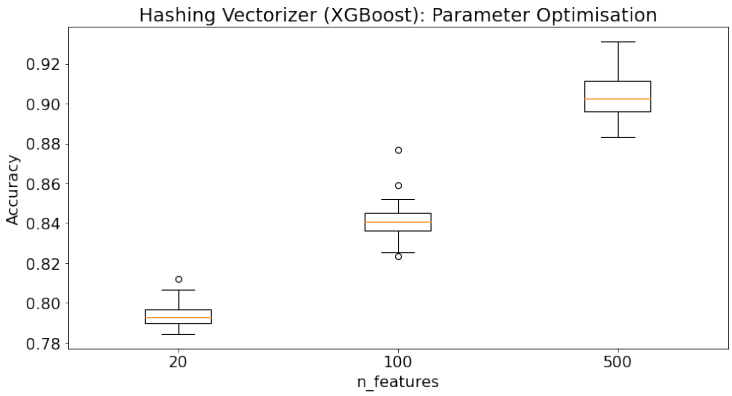


Figure 6.7: Parameter Optimisation of the Hashing Vectorizer embeddings.

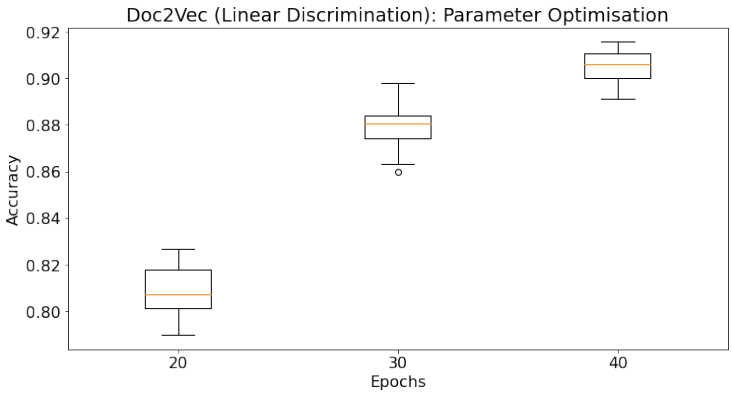


Figure 6.8: Parameter Optimisation of the Doc2Vec embeddings.

Figure 6.6, 6.7 and 6.8 show some examples of the results achieved by the parameter optimisation of the word embedding methods used in this project. As you can see in Figure 6.6, there were marginal improvements exhibited in the accuracy of the TFIDF algorithm when selecting 3000 'max_features' compared to selecting 1500 or 500 'max_features'. Figure 6.6 and 6.7 however feature major differences in accuracy depending on the parameter variables selected. In Figure 6.7, when 500 'n_features' are selected for the Hashing Vectorizer as opposed to 20, there is more than a 10% increase in the accuracy produced by the word embedding method. Almost the exact same is seen in Figure 6.8, where selecting 40 'epochs' for the Doc2Vec embeddings as opposed to 20 also results in a 10% increase in accuracy.

Unlike the parameter optimisation of the machine learning classifiers where changes in accuracy where often quite subtle, the word embedding parameter optimisation often exhibited monumental improvements to the accuracy of the models, and was ultimately one of the most effective aspects of this project. Increases in accuracy due to word embedding parameter optimisations were non-uniform and varied widely, however an increase in the range of +0.25% (TFIDF) and +10.5% (Hashing Vectorizer) was exhibited across all embeddings.

6.3 Single-Platform Results and Evaluation

The overall goal of this project was to produce an efficient and effective multi-platform hate speech detection model, however in order to properly evaluate the makings of a successful multi-platform model, it is first important to analyse and evaluate the performance of each of the three individual single-platform models.

To carry out this evaluation, I first split the full 3000 comment dataset into three datasets of 1000 comments, with each dataset only containing data from one specific platform. This resulted in each platform having its own dataset of 1000 comments with 800 (80%) of those comments being non-hateful and labelled as '0', and 200 (20%) of those comments being hateful and labelled as ('1'). Each dataset was then initialised used the same training to testing data split ratio of 0.3, and tested using the exact same methodology, classifiers and word embeddings. Each platforms data was then used to create and test fifty models, making a total of one hundred and fifty individual models between the three platforms. Once this testing had been completed and all evaluation metrics had been noted, I singled out the top performing machine learning classifiers for each of the five word embedding methods, for each of the three platform datasets. The results of this process can be seen in the tables shown in Figure 6.9, 6.10 and 6.11 below.

Reddit Dataset				
Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / AdaBoost	0.912333	0: 0.92 1: 0.88	0: 0.93 1: 0.63	0: 0.95 1: 0.74
Doc2Vec / Linear Discrimination	0.809667	0: 0.84 1: 0.54	0: 0.94 1: 0.27	0: 0.89 1: 0.36
Hashing / Multi-Layer Perceptron	0.878000	0: 0.88 1: 0.87	0: 0.98 1: 0.46	0: 0.97 1: 0.86
USE / Naive Bayes	0.947333	0: 0.96 1: 0.90	0: 0.98 1: 0.83	0: 0.97 1: 0.86
BERT / Multi-Layer Perceptron	0.921667	0: 0.94 1: 0.84	0: 0.96 1: 0.75	0: 0.95 1: 0.79

Figure 6.9: A table showing the best machine learning classifier for each of the word embedding methods using the Reddit dataset.

Twitter Dataset				
Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / Decision Tree	0.987833	0: 0.99 1: 0.99	0: 1.00 1: 0.95	0: 0.99 1: 0.97
Doc2Vec / Linear Discrimination	0.865167	0: 0.88 1: 0.75	0: 0.96 1: 0.51	0: 0.92 1: 0.60
Hashing / XGBoost	0.947500	0: 0.96 1: 0.90	0: 0.98 1: 0.84	0: 0.97 1: 0.86
USE / SVC	0.960167	0: 0.95 1: 0.99	0: 1.00 1: 0.81	0: 0.98 1: 0.88
BERT / SVC	0.934333	0: 0.94 1: 0.93	0: 0.99 1: 0.73	0: 0.96 1: 0.81

Figure 6.10: A table showing the best machine learning classifier for each of the word embedding methods using the Twitter dataset.

4Chan Dataset				
Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / Random Forest	0.921167	0: 0.91 1: 0.98	0: 1.00 1: 0.63	0: 0.95 1: 0.77
Doc2Vec / Random Forest	0.795833	0: 0.80 1: 0.45	0: 0.99 1: 0.04	0: 0.89 1: 0.06
Hashing / XGBoost	0.894833	0: 0.90 1: 0.84	0: 0.97 1: 0.59	0: 0.94 1: 0.69
USE / Multi-Layer Perceptron	0.956833	0: 0.96 1: 0.97	0: 0.99 1: 0.80	0: 0.97 1: 0.88
BERT / SVC	0.934500	0: 0.93 1: 0.93	0: 0.99 1: 0.72	0: 0.96 1: 0.81

Figure 6.11: A table showing the best machine learning classifier for each of the word embedding methods using the 4Chan dataset.

As you can see by analysing the three tables above, there is a massive amount of variety in the highest achieving machine learning classifier and word embedding pairs depending on what data the model had been trained and tested on. The Reddit datasets highest performing model was the USE word embeddings paired with the Naïve Bayes classifier, the Twitter datasets highest performing model was the TFIDF word embeddings paired with the Decision Tree classifier, and the 4Chan datasets highest performing model was the USE word embeddings paired with the Multi-Layer Perceptron classifier. There is also massive diversity in the highest average accuracy achieved by each dataset, with Reddit achieving maximum of 94.73%, Twitter achieving a maximum of 98.78% and 4Chan achieving a maximum of 96.02%.

Due to the fact that the exact same methodologies, embeddings and classifiers where employed on each of the three datasets, this diversity in embedding and classifier pairs, and in the results achieved by these pairs can only be explained by the differences the in data collected from each of the three social media platforms. As stated in Section 4.1 of this report, each of the three social media platforms have extremely differing moderation methods and levels, which ultimately leads to completely different language being used on each platform. I believe that the diversity in results achieved by each single-platform model is largely due to the relative difficulty to detect the specific forms of hateful speech exhibited on that platform. Reddit's community based and strong moderation leads to "slurless" and subtle hateful language (E.g. "Get them all out of our country") which is difficult to detect and classify, where-as the automated and somewhat inadequate moderation on Twitter promotes the common use of typical hateful slurs (e.g. n*gger, f*ggot) which is much easier to detect and classify. 4Chan's no moderation policy leads to a diverse array of hateful speech, language and slurs (e.g. k*ke, n*gger, f*ggot, towelhead, mudskin), which ultimately makes it easier to detect and classify than the subtle Reddit hate speech, but harder to detect and classify than the repetitive Twitter hate speech. For this exact reason it is extremely important in this day and age to produce multi-platform, versatile hate speech detection models that don't rely on the specific language used on a single social media platform.

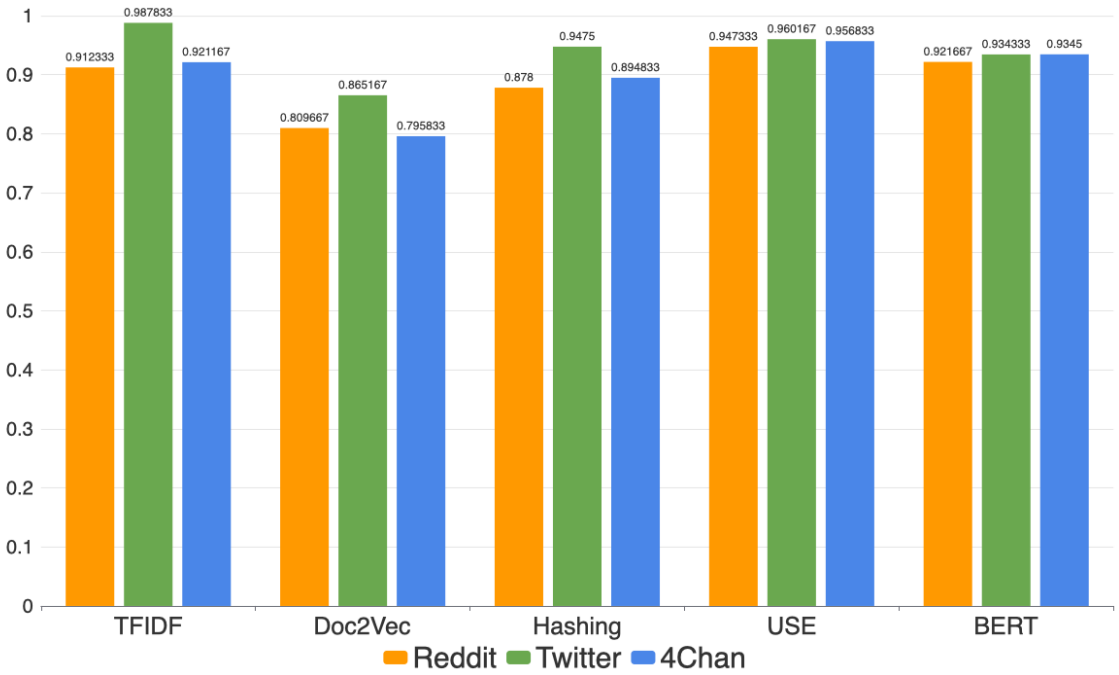


Figure 6.12: Bar chart showing the variance in accuracy returned by each word embedding method dependant on the platform data.

Figure 6.12 above shows a visualisation that illustrates the variance in accuracy returned by each platforms dataset when using each of the five word embedding methods. As you can see, the

TFIDF, Doc2Vec and Hashing Vectorizer algorithms are noticeably inconsistent dependant on which platform was being analysed, with a variance in maximum accuracy (difference between highest and lowest achieved) of 7.55%, 6.93% and 6.95% respectively. The USE and BERT word embeddings however are noticeably consistent independent of which platforms data is being analysed, with a variance in maximum accuracy of 1.29% and 1.28% respectively. This is a key discovery from the single-platform analysis section this report, and should be taken into massive consideration when evaluating which model makes for the multi-platform hate speech detection.

6.4 Multi-Platform Results and Evaluation

The core goal of this project was to produce a high-achieving hate speech detection model that could span multiple social media platforms and produce reliable and replicable results, so the analysis of the multi-platform dataset was an extremely import aspect of my analysis. To carry out this analysis, I used my full 3000 comment dataset of combined platform data. This dataset had 240 comments (80%) labelled as non-hateful ('0'), and 600 comments (20%) labelled as hateful ('1'). The dataset was then split into training and testing data in a ratio of 0.3, and tested against the fifty word embedding and machine learning classifier pairs as outlined in Section 5.2 of this report. The results of this process can be seen in the tables shown in Figure 6.13 below.

Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / Random Forest	0.949444	0: 0.95 1: 0.95	0: 0.99 1: 0.79	0: 0.97 1: 0.86
Doc2Vec / Linear Discrimination	0.900778	0: 0.90 1: 0.90	0: 0.99 1: 0.56	0: 0.94 1: 0.69
Hashing / XGBoost	0.902378	0: 0.92 1: 0.84	0: 0.97 1: 0.66	0: 0.94 1: 0.73
USE / SVC	0.956500	0: 0.96 1: 0.96	0: 0.99 1: 0.82	0: 0.97 1: 0.88
BERT / SVC	0.933944	0: 0.94 1: 0.91	0: 0.98 1: 0.74	0: 0.96 1: 0.81

Figure 6.13: A table showing the best machine learning classifier for each of the word embedding methods using the completed multi-platform dataset.

As you can see by analysing the table above, the highest performing multi-platform hate speech detection model produced in this project was a combination of the Universal Sentence Encoder word embeddings paired with the Support Vector Machine (SVC) machine learning classifier, which achieved a peak average accuracy of 95.85%. The USE word embeddings achieving the highest accuracy result is relatively unsurprising due to the analysis carried out on the single-platform models, in which USE was identified as an extremely consistent and versatile word embedding method, regardless of the platform data.

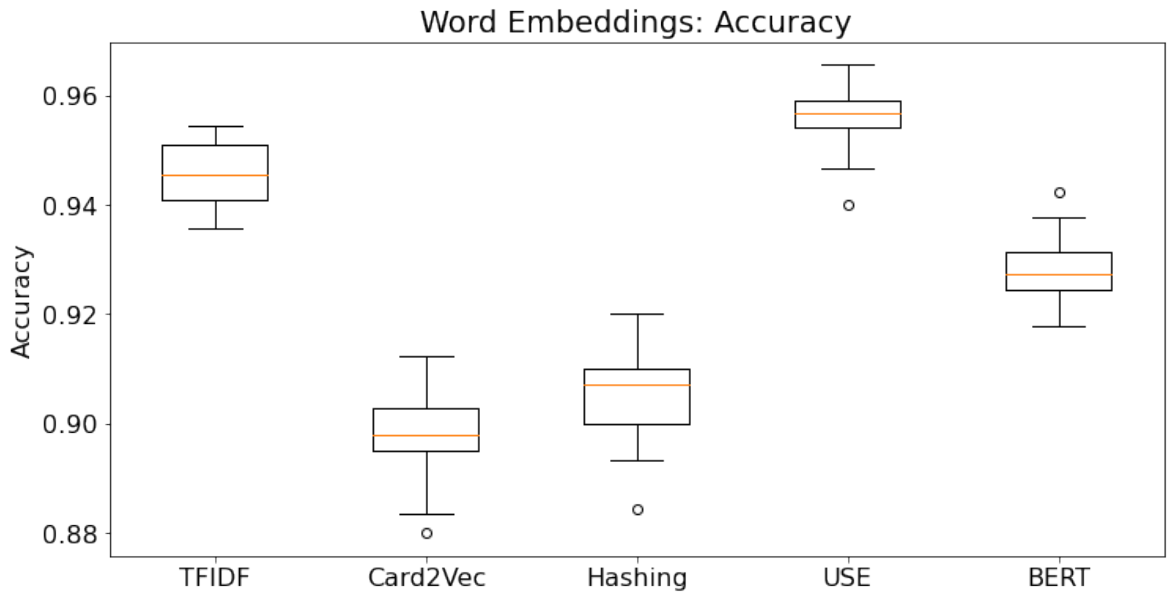


Figure 6.14: Boxplot showing the accuracies achieved by the best machine learning classifier for each of the word embedding methods using the completed multi-platform dataset.

The box plot shown in Figure 6.14 above, shows that the Universal Sentence Encoder and SVC classifier pair exhibited the highest upper bound accuracy of all combinations at a value of 96.89%, and also exhibited a lower variation in accuracy results when compared to all other embedding methods. Again, this low variation in accuracy results returned by the USE word embeddings comes as no surprise after analysing the single-platform model data, as USE has been proven to be extremely consistent and versatile already.

The Universal Sentence Encoder word embeddings combined with the SVC classifier exhibited a maximum average precision of 0.96, recall of 0.82 and F1-Score of 0.88 when classifying a comment as hateful. Each one of these individual four values where the highest evaluation metric results achieved by any model trained using the multi-platform dataset, as can be seen in Figure 6.15 below. It also exhibited a maximum average precision of 0.96, recall of 0.99 and F1-Score of 0.97 when classifying comments as non-hateful, which apart from recall where some models equalled the highest result, were also the highest evaluation metrics achieved by any model trained on the multi-platform dataset. These classification results for non-hateful comments can be seen in Figure 6.16 below.

As you can see through the above analysis, the Universal Sentence Encoder paired with the Support Vector Machine (SVC) classifier was the highest performing in all evaluation metrics and results. The high performance of the Support Vector Machine (SVC) classifier came as no surprise to me, as throughout my study of hate speech detection models prior to the creation of my own models, the SVC classifier had been heralded as extremely proficient in text-based classifications. The use of the Universal Sentence Encoder for word embeddings however did not get mentioned in any of the many papers I studied in the course of completing this project. No hate speech detection paper that I found while writing the Related Work section of this report had implemented or tested the USE word embeddings, and I believe that the extremely high performance and versatility exhibited by USE in this project is of great significance in the hate speech detection field.

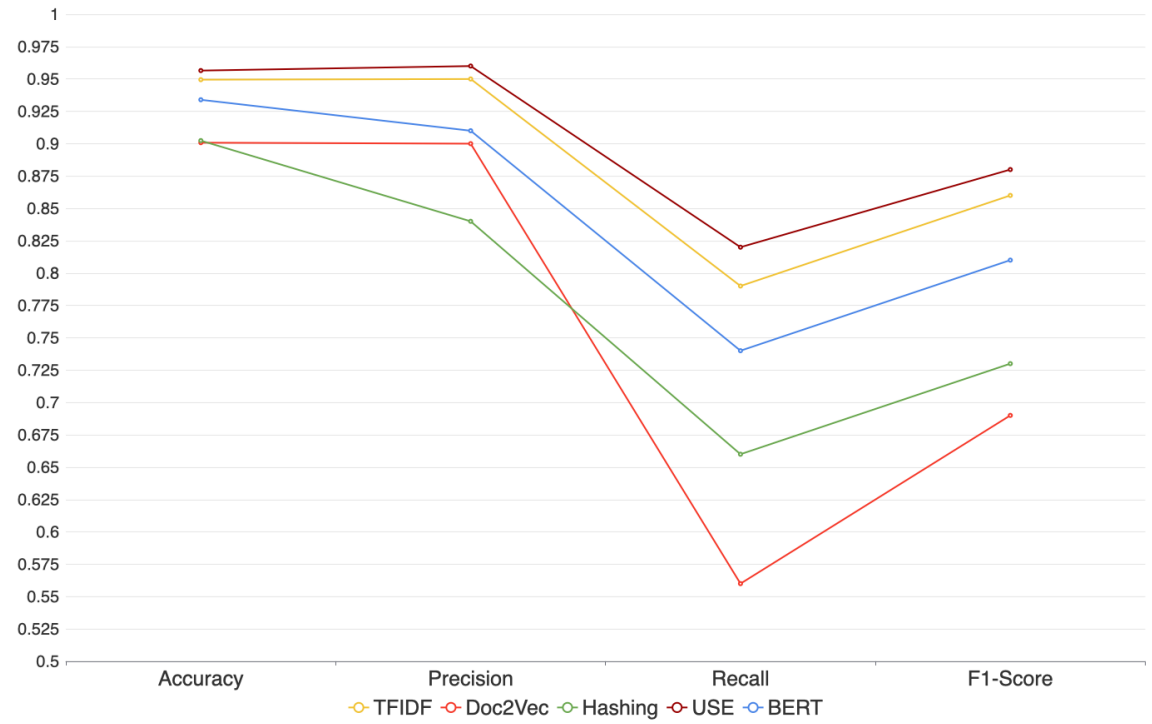


Figure 6.15: A line chart showing the evaluation metrics achieved by each of the best word embedding and classifier pairs when classifying data as hateful (Note that the X-Axis begins at 0.5).

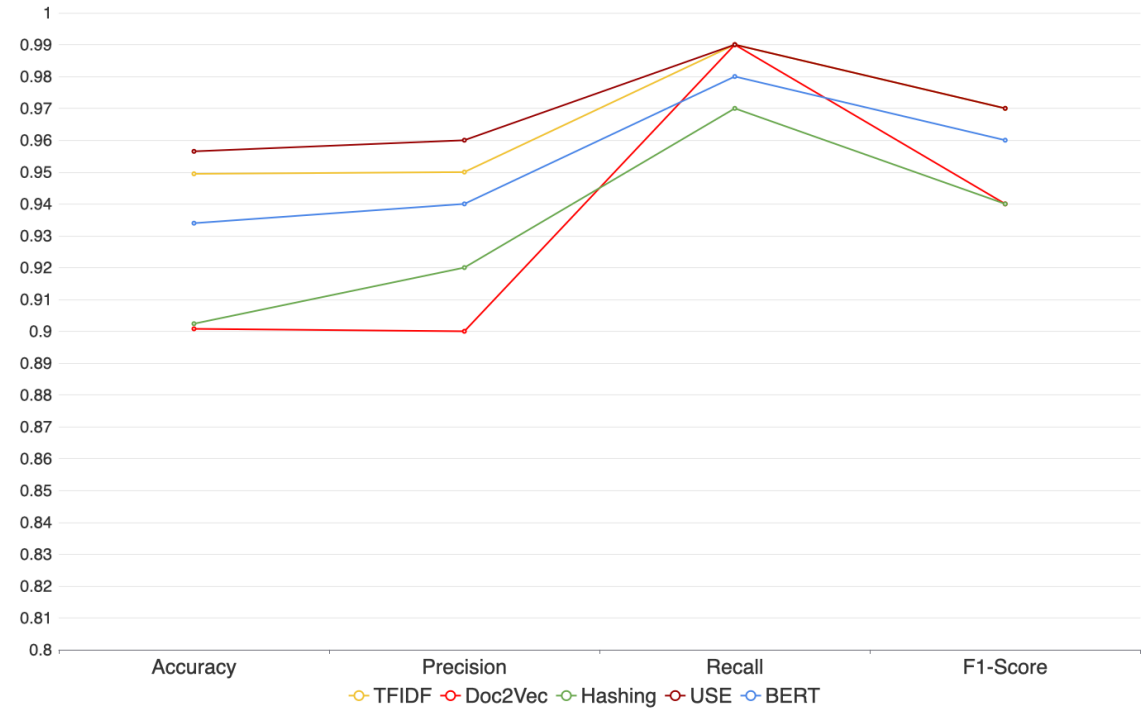


Figure 6.16: A line chart showing the evaluation metrics achieved by each of the best word embedding and classifier pairs when classifying data as non-hateful (Note that the X-Axis begins at 0.8).

6.5 Run Time and Efficiency Evaluation

The run time and efficiency of hate speech detection models are another extremely important factor in determining the success and viability of a model in practice. In the course of this project I measured both the single run time and twenty run time average of the word embeddings and machine learning classifiers. The twenty runtime average for all classifiers and embeddings came out to exactly 20x the signle run time, so I decided to only focus on the single runtime metric. In order to fairly evaluate this metric, I calculated both the average runtime of each machine learning classifier across all word embeddings, and the average run time of each word embedding method across all classifiers and noted the results. This process produced the graphs seen in Figure 6.17 and 6.18 below.

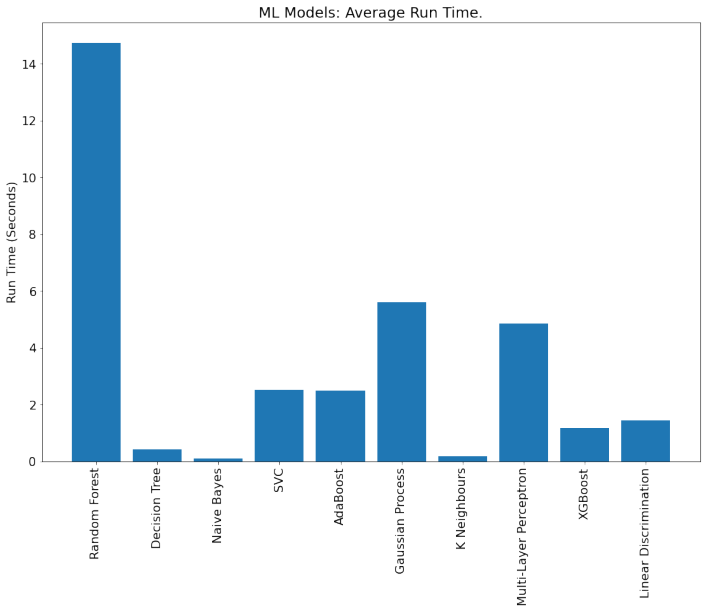


Figure 6.17: A chart illustrating the comparative average run times of the machine learning classifiers on the multi-platform dataset.

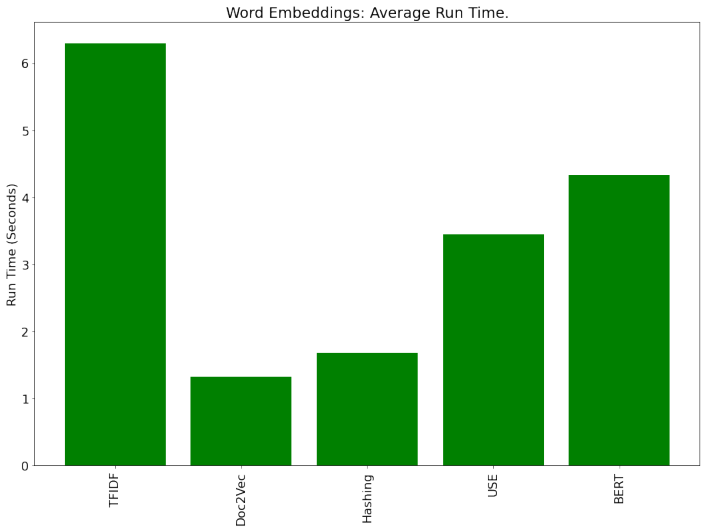


Figure 6.18: A chart illustrating the comparative average run times of the word embedding methods on the multi-platform dataset.

As you can see in Figure 6.17, the classifier with the highest average run time by quite a large margin was the Random Forest Classifier (14.7235s), and the classifier with the lowest average run time was the Naïve Bayes classifier (0.1063s). The Gaussian Process and Multi-Layer Perceptron classifiers also had notably high average run times (5.6062s and 4.8512s respectively), while the K-Neighbours and Decision Tree classifiers had notably low average run times (0.1991s and 0.4270s respectively). In Figure 6.18, you can see that the word embedding with the highest average run-time was TFIDF (6.2958s), with BERT having the second highest average run-time of 4.3350s. Doc2Vec was the fasted performing word embedding method with an average run-time of 1.3284s and the Hashing word embeddings also performed well with a 1.6762s average run time.

Now that the run-time of each model had been calculated and noted, it was time to determine which models exhibited the highest levels of efficiency. Efficiency refers to the ability of a machine learning model to produce accurate results, while also exhibiting a very short relative run-time. In order to come to a conclusion as to the most efficient models, I first created a table which contained all of the run-times of the machine learning models tested during this project, as seen in Figure 6.19 below. I then used this run-time table along with the table of accuracies produced by each model in order to come to a conclusion as to the most efficient models.

	TFIDF	Doc2Vec	Hashing	USE	BERT	Average
Random Forest	15.5070	6.9077	5.9888	18.0517	27.1622	14.7235
Decision Tree	0.4148	0.0747	0.0620	0.6194	0.9643	0.4270
Naive Bayes	0.3136	0.0101	0.0574	0.0570	0.0933	0.1063
SVC	9.6417	0.1189	1.1423	0.7407	0.9411	2.5170
AdaBoost	2.0077	0.4864	0.3037	3.8224	5.8252	2.4891
Gaussian Process	10.4660	4.1397	5.1421	5.7909	2.4921	5.6062
K Neighbours	0.5193	0.0549	0.1422	0.1472	0.1317	0.1991
Multi-Layer Perceptron	14.2263	0.9683	2.7452	3.4606	2.8556	4.8512
XGBoost	4.2010	0.4059	0.7215	1.3560	2.3949	1.8159
Linear Discrimination	5.6602	0.1172	0.4563	0.4749	0.4895	1.4396
Average	6.2958	1.3284	1.6762	3.4521	4.3350	

Figure 6.19: A table illustrating the comparative average run times of each model tested on the multi-platform dataset.

Ultimately, I determined that the three models circled in red in the above table exhibited the highest levels of efficiency out of all tested models. The Doc2Vec and Naïve Bayes model, the Doc2Vec and K-Neighbours model, and the USE and Naïve Bayes model all exhibited an average run time of less than one second (0.0101s, 0.0549s and 0.0570s respectively), and also all exhibited a notably high degree of accuracy when compared to other models. The Doc2Vec and Naïve Bayes model achieved an overall accuracy of 83.02%, the Doc2Vec and K-Neighbours model achieved an overall accuracy of 88.55%, and the USE and Naïve Bayes model achieved an overall accuracy of 93.61%. All three of these models exhibited extremely fast run-times in comparison to other models, and also achieved an extremely high accuracy in comparison to other models, which makes them the most highly efficient and economical models that I produced during the course of this project.

6.6 HateChecker Application Results and Evaluation

The HateChecker application proved to be an invaluable tool for my project, and was extremely successful overall. The main and most useful functionality exhibited by the HateChecker application was its ability to identify problems and errors within my machine learning models through the feedback returned by the application. By inputting a wide array of different sample posts and comments into the user input field, analysing the results, and reviewing the methodology of my models, it was possible to identify and mitigate many biases, mis-classifications and failures that where inherent within my hate speech detection models.

A prime example of this benefit, was that through the testing of strings such as "n1gg3r" and "f4gg0t" in the HateChecker application, it was identified that my models where completely failing when a comment had numerals in place of letters. When I first began inputting these numerical strings into the application, an output of "Non-Hateful, Confidence Level: 100%" was being returned. I then traced this fault back to my data cleaning process and removed a line of code which neutralised and removed all occurrences of numbers within the database. Upon this small data cleaning process change, the models instantly gained a much better ability to identify hateful language disguised through the use of numerals in place of letters, and increased the overall versatility and accuracy of my models. The predictions made for strings containing numerals instead of certain characters by the HateChecker application Is much more accurate now, as can be seen in Figure 6.20.

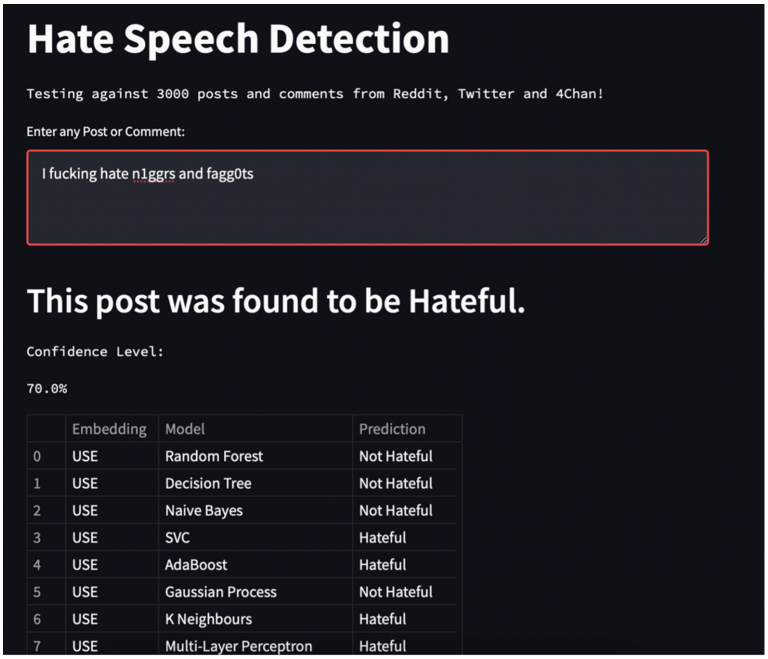


Figure 6.20: An example of the output from the HateChecker application when the input contains numerals substituted for letters.

This same process of testing through the HateChecker application also allowed me to discover fringe cases where my models where extremely successful and accurate. An example of this would be the success of my models when dealing with a diverse array of different punctuation characters which are used to attempt to obfuscate hateful language. Posts that contained strings such as "f_a_g_g_o_t" or "n!gger" had a very high detection rate, and where found to be hateful with a high confidence level almost always, as seen in Figure 6.21.

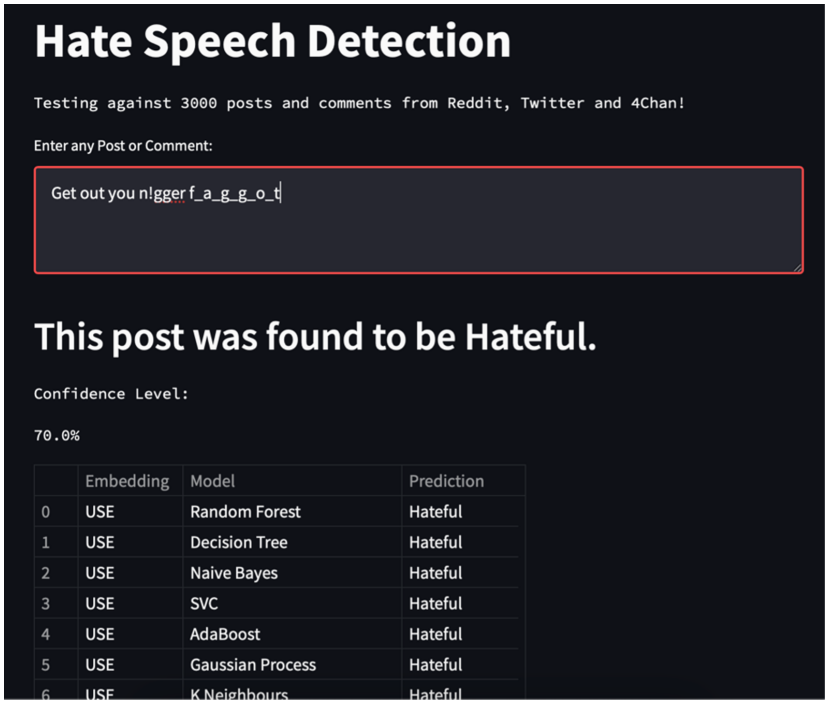


Figure 6.21: An example of the output from the HateChecker application when punctuation is substituted for letters, and used to obfuscate hateful speech.

While the HateChecker application was overall an extremely useful addition to this project it does feature downsides and potential for future work. The major weakness of the HateChecker application in its current form is its extremely lengthy run time. Due to the fact that many multiple machine learning classifiers need to be trained on a relatively large dataset in order to produce a result set, the application is very slow and uses an excessive amount of computational power. In order to mitigate this long run time somewhat, I used the "pandas" python library to implement a pre-cleaned, and pre-embedded database using a '.pkl' file and the 'read_pickle" function. This file type preserves data types in a pandas DataFrame, and allowed me to remove the burden of reading the data, cleaning the data and embedding the data from the HateChecker application. Another possible improvement for the HateChecker application in terms of run time and computational speed would be to pre-save all models used within the application. This would remove the need for the application to train each of the twenty models upon start-up, and would ultimately lead to a much faster and more efficient application output.

Chapter 7: Project Workplan

Below is an outline of how I am going to approach my project illustrated using a Gantt Chart. I have broken the project into five separate stages, which I will discuss further below. Due to the time constraints of the final year project, it is extremely important to have a detailed plan and create deadlines for the various stages of the project.

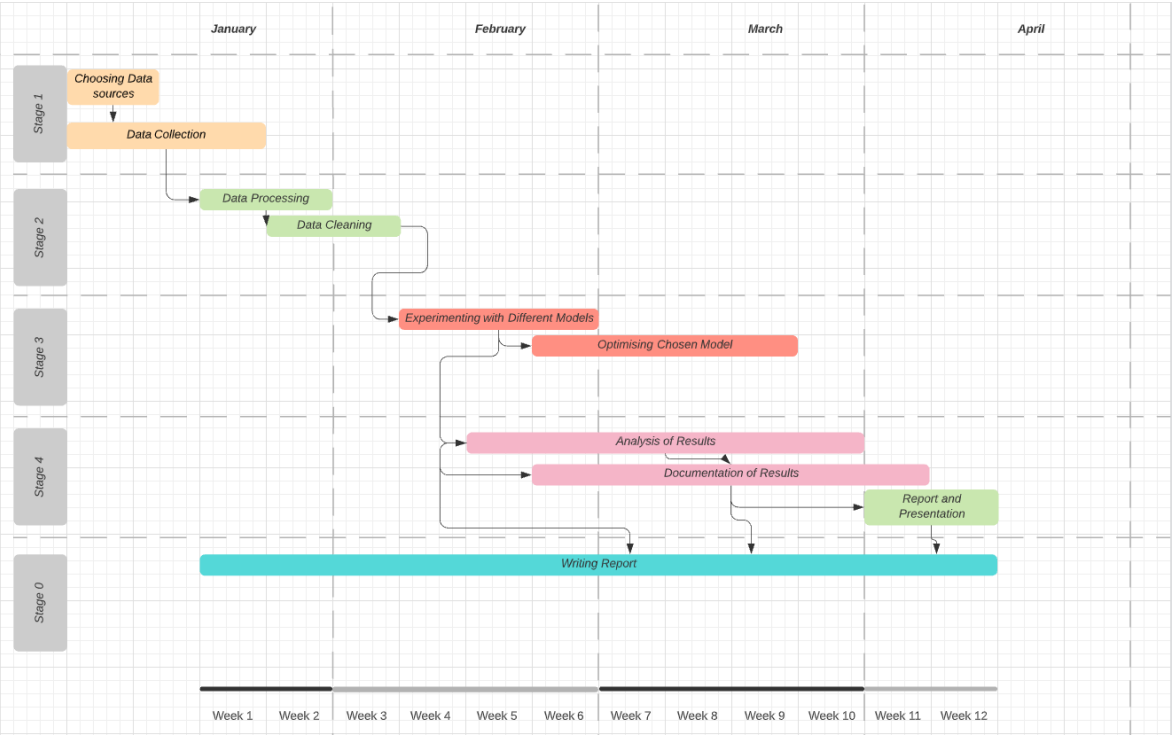


Figure 7.1: A Gantt chart representing the project workplan which I plan to abide by.

Stage 1 (January Week 1 – January Week 3):
The first stage of my project involves deciding what sources I would like to collect data from, and the collection of this data. I will decide which sites will serve me best in the goal of detecting hate speech online (Twitter, Reddit, YouTube etc.), and will then begin to collect this data. I plan on using API's to collect comments and posts from online social media sites and then using this data for the rest of my project.

Stage 2 (January Week 2 – February Week 1):
Stage two of my project involves processing, and cleaning the data collected in stage one. The data collected in stage one will need to be stored in an efficient manner, and the selection of a suitable data structure is vital for the next stages of my project. The data collected from the social media sites will be relatively structured, as it will be collected through an API, however there will be cleaning steps necessary for the next stages of the project. Unnecessary columns, duplicate rows and null/invalid values will need to be removed, and further cleaning may be required such as conversion to binary variables.

Stage 3 (February Week 2 – March Week 3):
Once stage one and two are complete and I have my cleaned, processed and stored data, it will be time to start experimenting with different hate speech detection models. I have many models and feature representations in mind from the 'Related Work and Ideas' section of my report such as

XGBoost, BERT, SVM, FastText and Linear Discrimination. I will test these models and compare them to one another in order to determine which produces the best results, and which is best suited to the goal of my project. Once I have determined the best suited model for my project, I will begin to optimise and fine-tune this model in order to produce the best results. This will involve choosing parameters, training/test data, ratio of training/test data and the amount of data I choose to feed the model.

Stage 4 (February Week 2 – April Week 2):
Stage four of my project involves the documenting and analysis of the results obtained in stage three. During the testing of the various hate detection models, it will be vital to document the results received in order to decide which model I will opt to continue with and optimise. This step is also extremely important for my overall project, as I must be able to justify why I chose to use one model over the many other possible options. The documentation of these results will ensure that the proper choice is made, and that this choice is properly informed.

Next, the results must be analysed. This will involve a number of different evaluation metrics such as F-Score, accuracy, precision, recall and AUROC. These results must be analysed and justified, and if results are not as good as expected further optimisation can occur. I will also create many visualisations in this step for the reader to be able to easily interpret the results and conclusions that I produce.

I have also dedicated one week at the end of this stage to finalise my report and presentation. It will be extremely important to properly report my results and conclusions, and time must be set aside to do this properly.

Stage 0 (January Week 2 – April Week 2):
Stage zero of my project involves the writing of my project report. This stage spans almost the entirety of my project, as I believe it is extremely important to write the report as I go along. This will provide valuable insights from each step of my project which may be forgotten if I leave the entire report to the very end of my project. This will also allow ample time to produce a readable, concise and explanatory report which will serve as the backbone of the project.

The above stages outline the project workplan that I am following for this project. I believe that the deadlines are reachable and fair, and that I can complete all aspects in the periods outlined by the Gantt chart. I have a clear and defined map to complete every stage of my project, and if I follow this map I expect to complete all areas of this project in good time and before the project is finally due. I have included another visualisation that I have created which outlines the main steps of my project, and gives an idea of the overall project architecture in which I will follow, the first stage being the social media collection and the final stage being analysis.

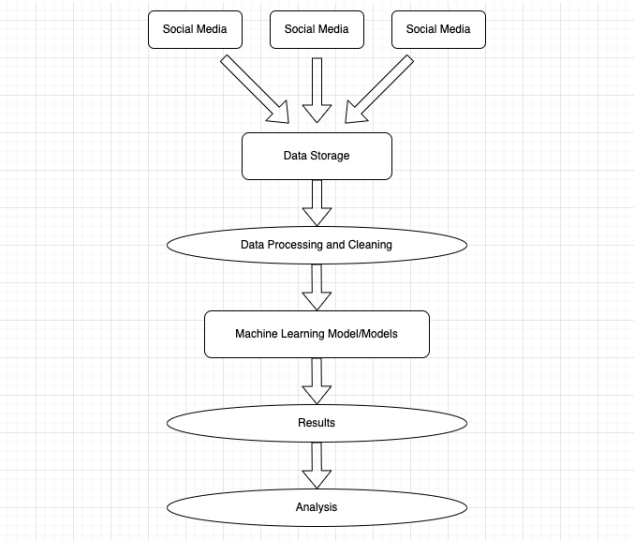


Figure 7.2: A figure that I created outlining the steps of my project in a top to bottom fashion.

Chapter 8: Summary and Conclusions

Goals

Overall, I am extremely happy with the results of this project. While the overall goal was to produce a successful multi-platform hate speech detection model, there were many significant sub-goals that where reached in the process of completing this project, the effects of which cannot be understated. The web-scraping and data collection aspects of this project for example, served as a backbone to all future processes and modelling, and had a massive impact on the overall results returned by the models. I feel as though I picked three sufficiently diverse platforms in terms of moderation standards and use of language (both non-hateful and hateful) and I believe that this had a large part to play in the overall versatility of the future models. The data cleaning processes also had a massive impact on the overall results and accuracy of the hate speech detection models. Through many hours of testing and analysis, and the use of the HateChecker application, I believe that I arrived at an extremely proficient data cleaning process for the specific task of hate speech detection.

While many sub-goals exhibited large success and positive results, the ultimate goal was to create a highly accurate and versatile, multi-platform hate speech detection model, and in my opinion this goal was achieved. Through vigorous testing of a wide variety of different data cleaning processes, word embedding methods and machine learning classifiers, I believe that a very high-performing and accurate hate speech detection model was found. In total, fifty different models where tested in order to discover which word embedding and classifier pair would exhibit the best ability to detect and predict hate speech, and I believe that through this testing process I produced a model that did just that.

Results

The highest performing multi-platform hate speech detection model that I produced during the course of this project was the Universal Sentence Encoder word embeddings paired with the SVC machine learning classifier. This model exhibited an average accuracy of 95.65% across twenty runs with a new training and testing data split each run, and achieved a maximum accuracy of 96.89%. When classifying comments as hateful ('1'), it achieved an average precision of 0.96, an average recall of 0.82 and an average F1-Score of 0.88. When classifying comments as non-hateful ('0') it achieved an average precision of 0.96, an average recall of 0.99 and an average F1-Score of 0.97. These extremely positive results far exceeded the expectations that I held when I began this project, and I would classify this as the most important success of my project.

Future Works

While I am extremely satisfied with the results achieved in this project, there is still room for many improvements and future work. The process of scraping posts and comments from social media websites, curating labelled datasets and ensuring that the proportion of non-hateful to hateful comments is satisfactory, is an extremely long and laborious process. Given the timeframe to complete this project, it was not possible to create a dataset of more than 3000 comments for the purpose of training and testing my models, even though I would have preferred to have had a dataset of closer to 5000 comments. Each time that I added more comments to my database, the results achieved by the models across the board would increase by 0.5% - 1%, so ideally I would have liked to add at least another 2000 comments to my database.

Another improvement that could be made would be to test even more word embedding methods and machine learning classifiers. Every word embedding method and machine learning classifier has its own significant strengths, and while I tested a total of fifty different combinations of

embeddings and algorithms, it is still a possibility that a better word embedding, classifier pair exists that I did not explore in the course of this project. Word embedding methods such as GloVe and RoBERTa, and machine learning classifiers such as LSTM (Long-Short Term Memory) and CNN (Convolutional Neural Network) are just a few of the possible models that I could not test and analyse in this project due to time constraints.

The HateChecker application was a vital tool for my project in terms of creating the most accurate and successful hate speech detection model possible. It allowed me to test a diverse array of possible posts and comments, including non-hateful language, hateful language, leetspeak, slurs and intentionally obfuscated text. I made many positive changes to my model creation project based on feedback returned to be my the HateChecker application, and ultimately mitigated many biases, oversights and false classifications. For public and more wide-spread use however, the application would need to be streamlined in terms of computation power and speed. In its current form, the application is too slow and too computationally expensive to viably use in a real-world setting, but this could be remedied through some modifications to the application. Currently, each time the application is ran, twenty models must be trained and then referenced in order to produce the final result. This burden could be offloaded by pre-computing and training each of the twenty models, saving these models, and then directly importing them straight into the application. I expect that this would instantly improve the speed and efficiency of the HateChecker application, and is one of the main areas of future work that I would recommend for this project.

Conclusion

Overall, the goal of this project was to create an accurate, efficient and versatile multi-platform hate speech detection model. I believe that this goal was achieved, and I believe that the discovery of the extremely high performance of the USE word embeddings is a vital takeaway from this report. The Universal Sentence Encoder word embeddings where not tested or evaluated in any of the papers I studied in the course of completing this project, and I believe that the extremely good performance and efficiency of these word embeddings is a vital piece of research, and a potentially valuable discovery in the field of hate speech detection modelling. While I am extremely pleased with the discoveries made in this project, the field of hate speech detection is vast and complex, and there are still many stones left unturned. Many more classifiers, word embeddings and evaluation metrics exist which could produce even better results than achieved in this project, and the potential for future work in this field is massive.

Chapter 9: Results Tables

A = Accuracy, P = Precision, R = Recall, F1 = F1-Score.
Values coloured red received ill-defined or 0.0 values in all twenty simulations for precision, recall and F1-Score.
Values coloured orange received some ill-defined or 0.0 values in the twenty simulations for precision, recall and F1-Score.
Values coloured green exhibited the best results for the word embedding method.

1000 Comment Reddit Dataset Results
(20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.901667 P = (0: 0.89, 1: 0.97) R = (0: 1.00, 1: 0.53) F1 = (0: 0.94, 1: 0.68)	A = 0.793667 P = (0: 0.80, 1: 0.52) R = (0: 0.99, 1: 0.02) F1 = (0: 0.88, 1: 0.04)	A = 0.879000 P = (0: 0.89, 1: 0.82) R = (0: 0.97, 1: 0.51) F1 = (0: 0.93, 1: 0.63)	A = 0.894167 P = (0: 0.88, 1: 0.99) R = (0: 1.00, 1: 0.47) F1 = (0: 0.94, 1: 0.63)	A = 0.887500 P = (0: 0.88, 1: 0.97) R = (0: 1.00, 1: 0.46) F1 = (0: 0.93, 1: 0.62)
Decision Tree	A = 0.874167 P = (0: 0.88, 1: 0.86) R = (0: 0.98, 1: 0.44) F1 = (0: 0.93, 1: 0.58)	A = 0.738000 P = (0: 0.80, 1: 0.26) R = (0: 0.89, 1: 0.15) F1 = (0: 0.84, 1: 0.18)	A = 0.861333 P = (0: 0.88, 1: 0.75) R = (0: 0.96, 1: 0.45) F1 = (0: 0.92, 1: 0.56)	A = 0.850500 P = (0: 0.91, 1: 0.63) R = (0: 0.91, 1: 0.64) F1 = (0: 0.91, 1: 0.63)	A = 0.816500 P = (0: 0.89, 1: 0.54) R = (0: 0.89, 1: 0.53) F1 = (0: 0.89, 1: 0.53)
Naive Bayes	A = 0.756000 P = (0: 0.91, 1: 0.42) R = (0: 0.78, 1: 0.68) F1 = (0: 0.84, 1: 0.52)	A = 0.604333 P = (0: 0.79, 1: 0.23) R = (0: 0.69, 1: 0.28) F1 = (0: 0.72, 1: 0.20)	A = 0.620667 P = (0: 0.84, 1: 0.27) R = (0: 0.65, 1: 0.52) F1 = (0: 0.73, 1: 0.35)	A = 0.947333 P = (0: 0.96, 1: 0.90) R = (0: 0.98, 1: 0.83) F1 = (0: 0.97, 1: 0.86)	A = 0.821333 P = (0: 0.94, 1: 0.54) R = (0: 0.83, 1: 0.78) F1 = (0: 0.88, 1: 0.63)
SVC	A = 0.844167 P = (0: 0.84, 1: 0.98) R = (0: 1.00, 1: 0.25) F1 = (0: 0.91, 1: 0.40)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.845000 P = (0: 0.84, 1: 0.91) R = (0: 0.99, 1: 0.27) F1 = (0: 0.91, 1: 0.41)	A = 0.944167 P = (0: 0.94, 1: 0.98) R = (0: 1.00, 1: 0.74) F1 = (0: 0.97, 1: 0.84)	A = 0.919500 P = (0: 0.91, 1: 0.97) R = (0: 1.00, 1: 0.61) F1 = (0: 0.95, 1: 0.75)
AdaBoost	A = 0.912333 P = (0: 0.92, 1: 0.88) R = (0: 0.98, 1: 0.63) F1 = (0: 0.95, 1: 0.74)	A = 0.774500 P = (0: 0.80, 1: 0.30) R = (0: 0.96, 1: 0.06) F1 = (0: 0.87, 1: 0.10)	A = 0.855833 P = (0: 0.88, 1: 0.73) R = (0: 0.95, 1: 0.51) F1 = (0: 0.91, 1: 0.60)	A = 0.922000 P = (0: 0.94, 1: 0.85) R = (0: 0.97, 1: 0.73) F1 = (0: 0.95, 1: 0.79)	A = 0.886667 P = (0: 0.91, 1: 0.76) R = (0: 0.95, 1: 0.64) F1 = (0: 0.93, 1: 0.69)
Gaussian Process	A = 0.797333 P = (0: 0.80, 1: 0.97) R = (0: 1.00, 1: 0.02) F1 = (0: 0.89, 1: 0.05)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.807167 P = (0: 0.81, 1: 0.97) R = (0: 1.0, 1: 0.05) F1 = (0: 0.89, 1: 0.10)	A = 0.887833 P = (0: 0.88, 1: 1.00) R = (0: 1.00, 1: 0.45) F1 = (0: 0.93, 1: 0.62)	A = 0.895500 P = (0: 0.93, 1: 0.74) R = (0: 0.94, 1: 0.71) F1 = (0: 0.94, 1: 0.73)
K Neighbours	A = 0.816333 P = (0: 0.82, 1: 0.96) R = (0: 1.00, 1: 0.15) F1 = (0: 0.90, 1: 0.24)	A = 0.778833 P = (0: 0.80, 1: 0.26) R = (0: 0.96, 1: 0.05) F1 = (0: 0.87, 1: 0.08)	A = 0.811167 P = (0: 0.81, 1: 0.85) R = (0: 0.99, 1: 0.11) F1 = (0: 0.89, 1: 0.17)	A = 0.934333 P = (0: 0.97, 1: 0.82) R = (0: 0.95, 1: 0.86) F1 = (0: 0.96, 1: 0.84)	A = 0.907500 P = (0: 0.92, 1: 0.83) R = (0: 0.97, 1: 0.65) F1 = (0: 0.94, 1: 0.73)
Multi-Layer Perceptron	A = 0.881833 P = (0: 0.87, 1: 0.97) R = (0: 1.00, 1: 0.43) F1 = (0: 0.93, 1: 0.59)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.878000 P = (0: 0.88, 1: 0.87) R = (0: 0.98, 1: 0.46) F1 = (0: 0.93, 1: 0.60)	A = 0.943500 P = (0: 0.94, 1: 0.95) R = (0: 0.99, 1: 0.76) F1 = (0: 0.97, 1: 0.84)	A = 0.921667 P = (0: 0.94, 1: 0.84) R = (0: 0.96, 1: 0.75) F1 = (0: 0.95, 1: 0.79)
XGBoost	A = 0.877000 P = (0: 0.88, 1: 0.83) R = (0: 0.97, 1: 0.50) F1 = (0: 0.93, 1: 0.62)	A = 0.777500 P = (0: 0.81, 1: 0.32) R = (0: 0.95, 1: 0.10) F1 = (0: 0.87, 1: 0.15)	A = 0.870167 P = (0: 0.89, 1: 0.77) R = (0: 0.96, 1: 0.50) F1 = (0: 0.92, 1: 0.60)	A = 0.926833 P = (0: 0.93, 1: 0.92) R = (0: 0.99, 1: 0.69) F1 = (0: 0.96, 1: 0.79)	A = 0.905333 P = (0: 0.91, 1: 0.85) R = (0: 0.97, 1: 0.62) F1 = (0: 0.94, 1: 0.71)
Linear Discrimination	A = 0.580500 P = (0: 0.89, 1: 0.33) R = (0: 0.54, 1: 0.72) F1 = (0: 0.66, 1: 0.42)	A = 0.809667 P = (0: 0.84, 1: 0.54) R = (0: 0.94, 1: 0.27) F1 = (0: 0.89, 1: 0.36)	A = 0.772500 P = (0: 0.89, 1: 0.45) R = (0: 0.81, 1: 0.61) F1 = (0: 0.85, 1: 0.52)	A = 0.843500 P = (0: 0.94, 1: 0.59) R = (0: 0.86, 1: 0.77) F1 = (0: 0.90, 1: 0.66)	A = 0.709500 P = (0: 0.89, 1: 0.38) R = (0: 0.72, 1: 0.65) F1 = (0: 0.80, 1: 0.48)

Figure 9.1: The final table of results achieved by models tested using the 1000 comment Reddit dataset.

1000 Comment Twitter Dataset Results
(20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.983000 P = (0: 0.98, 1: 1.00) R = (0: 1.00, 1: 0.92) F1 = (0: 0.99, 1: 0.96)	A = 0.814333 P = (0: 0.82, 1: 0.74) R = (0: 0.99, 1: 0.14) F1 = (0: 0.89, 1: 0.23)	A = 0.951833 P = (0: 0.97, 1: 0.90) R = (0: 0.97, 1: 0.87) F1 = (0: 0.97, 1: 0.88)	A = 0.908500 P = (0: 0.90, 1: 0.99) R = (0: 1.00, 1: 0.55) F1 = (0: 0.95, 1: 0.70)	A = 0.885500 P = (0: 0.88, 1: 0.90) R = (0: 0.99, 1: 0.49) F1 = (0: 0.93, 1: 0.63)
Decision Tree	A = 0.987833 P = (0: 0.99, 1: 0.99) R = (0: 1.00, 1: 0.95) F1 = (0: 0.99, 1: 0.97)	A = 0.748000 P = (0: 0.83, 1: 0.35) R = (0: 0.85, 1: 0.32) F1 = (0: 0.84, 1: 0.33)	A = 0.946833 P = (0: 0.97, 1: 0.86) R = (0: 0.97, 1: 0.87) F1 = (0: 0.97, 1: 0.87)	A = 0.868833 P = (0: 0.92, 1: 0.65) R = (0: 0.91, 1: 0.68) F1 = (0: 0.92, 1: 0.66)	A = 0.829500 P = (0: 0.90, 1: 0.57) R = (0: 0.89, 1: 0.58) F1 = (0: 0.89, 1: 0.58)
Naive Bayes	A = 0.765167 P = (0: 0.87, 1: 0.42) R = (0: 0.83, 1: 0.50) F1 = (0: 0.85, 1: 0.45)	A = 0.542000 P = (0: 0.87, 1: 0.25) R = (0: 0.51, 1: 0.68) F1 = (0: 0.64, 1: 0.37)	A = 0.715167 P = (0: 0.85, 1: 0.34) R = (0: 0.78, 1: 0.47) F1 = (0: 0.81, 1: 0.39)	A = 0.947167 P = (0: 0.96, 1: 0.91) R = (0: 0.98, 1: 0.82) F1 = (0: 0.97, 1: 0.86)	A = 0.825667 P = (0: 0.95, 1: 0.55) R = (0: 0.82, 1: 0.84) F1 = (0: 0.88, 1: 0.66)
SVC	A = 0.879500 P = (0: 0.87, 1: 1.00) R = (0: 1.00, 1: 0.40) F1 = (0: 0.93, 1: 0.57)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.909500 P = (0: 0.90, 1: 0.98) R = (0: 1.00, 1: 0.56) F1 = (0: 0.95, 1: 0.71)	A = 0.960167 P = (0: 0.95, 1: 0.99) R = (0: 1.00, 1: 0.81) F1 = (0: 0.98, 1: 0.89)	A = 0.934333 P = (0: 0.94, 1: 0.93) R = (0: 0.99, 1: 0.73) F1 = (0: 0.96, 1: 0.81)
AdaBoost	A = 0.984833 P = (0: 0.99, 1: 0.97) R = (0: 0.99, 1: 0.95) F1 = (0: 0.99, 1: 0.96)	A = 0.792000 P = (0: 0.83, 1: 0.46) R = (0: 0.94, 1: 0.21) F1 = (0: 0.88, 1: 0.29)	A = 0.936333 P = (0: 0.96, 1: 0.86) R = (0: 0.97, 1: 0.82) F1 = (0: 0.96, 1: 0.83)	A = 0.939833 P = (0: 0.95, 1: 0.88) R = (0: 0.97, 1: 0.79) F1 = (0: 0.96, 1: 0.83)	A = 0.897833 P = (0: 0.93, 1: 0.77) R = (0: 0.95, 1: 0.70) F1 = (0: 0.94, 1: 0.73)
Gaussian Process	A = 0.820833 P = (0: 0.82, 1: 1.00) R = (0: 1.00, 1: 0.09) F1 = (0: 0.90, 1: 0.16)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.827000 P = (0: 0.82, 1: 1.00) R = (0: 1.00, 1: 0.16) F1 = (0: 0.90, 1: 0.28)	A = 0.923167 P = (0: 0.91, 1: 1.00) R = (0: 1.00, 1: 0.62) F1 = (0: 0.95, 1: 0.76)	A = 0.854333 P = (0: 0.94, 1: 0.62) R = (0: 0.88, 1: 0.76) F1 = (0: 0.91, 1: 0.68)
K Neighbours	A = 0.923167 P = (0: 0.93, 1: 0.87) R = (0: 0.97, 1: 0.72) F1 = (0: 0.95, 1: 0.79)	A = 0.787333 P = (0: 0.81, 1: 0.43) R = (0: 0.95, 1: 0.14) F1 = (0: 0.88, 1: 0.21)	A = 0.910833 P = (0: 0.92, 1: 0.86) R = (0: 0.97, 1: 0.67) F1 = (0: 0.95, 1: 0.75)	A = 0.921000 P = (0: 0.99, 1: 0.73) R = (0: 0.91, 1: 0.97) F1 = (0: 0.95, 1: 0.83)	A = 0.900000 P = (0: 0.94, 1: 0.73) R = (0: 0.93, 1: 0.76) F1 = (0: 0.94, 1: 0.74)
Multi-Layer Perceptron	A = 0.911167 P = (0: 0.90, 1: 1.00) R = (0: 1.00, 1: 0.56) F1 = (0: 0.95, 1: 0.71)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.922167 P = (0: 0.92, 1: 0.95) R = (0: 0.99, 1: 0.66) F1 = (0: 0.95, 1: 0.78)	A = 0.957833 P = (0: 0.96, 1: 0.97) R = (0: 0.99, 1: 0.82) F1 = (0: 0.97, 1: 0.89)	A = 0.932000 P = (0: 0.95, 1: 0.87) R = (0: 0.97, 1: 0.80) F1 = (0: 0.96, 1: 0.83)
XGBoost	A = 0.979000 P = (0: 0.98, 1: 0.99) R = (0: 1.00, 1: 0.90) F1 = (0: 0.99, 1: 0.94)	A = 0.812500 P = (0: 0.84, 1: 0.60) R = (0: 0.95, 1: 0.28) F1 = (0: 0.89, 1: 0.37)	A = 0.947500 P = (0: 0.96, 1: 0.90) R = (0: 0.98, 1: 0.84) F1 = (0: 0.97, 1: 0.86)	A = 0.943167 P = (0: 0.95, 1: 0.93) R = (0: 0.99, 1: 0.77) F1 = (0: 0.97, 1: 0.84)	A = 0.914833 P = (0: 0.92, 1: 0.86) R = (0: 0.97, 1: 0.68) F1 = (0: 0.95, 1: 0.76)
Linear Discrimination	A = 0.784167 P = (0: 0.90, 1: 0.70) R = (0: 0.82, 1: 0.64) F1 = (0: 0.84, 1: 0.60)	A = 0.865167 P = (0: 0.88, 1: 0.75) R = (0: 0.96, 1: 0.51) F1 = (0: 0.92, 1: 0.60)	A = 0.812167 P = (0: 0.91, 1: 0.52) R = (0: 0.85, 1: 0.67) F1 = (0: 0.88, 1: 0.59)	A = 0.863000 P = (0: 0.95, 1: 0.61) R = (0: 0.87, 1: 0.82) F1 = (0: 0.91, 1: 0.70)	A = 0.730500 P = (0: 0.91, 1: 0.39) R = (0: 0.74, 1: 0.69) F1 = (0: 0.81, 1: 0.50)

Figure 9.2: The final table of results achieved by models tested using the 1000 comment Twitter dataset.

1000 Comment 4Chan Dataset Results
(20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.921167 P = (0: 0.91, 1: 0.98) R = (0: 1.00, 1: 0.63) F1 = (0: 0.95, 1: 0.77)	A = 0.795833 P = (0: 0.80, 1: 0.45) R = (0: 0.99, 1: 0.04) F1 = (0: 0.89, 1: 0.06)	A = 0.896167 P = (0: 0.90, 1: 0.90) R = (0: 0.98, 1: 0.56) F1 = (0: 0.94, 1: 0.69)	A = 0.921833 P = (0: 0.91, 1: 1.00) R = (0: 1.00, 1: 0.61) F1 = (0: 0.95, 1: 0.76)	A = 0.903833 P = (0: 0.90, 1: 0.93) R = (0: 0.99, 1: 0.57) F1 = (0: 0.94, 1: 0.70)
Decision Tree	A = 0.917500 P = (0: 0.91, 1: 0.97) R = (0: 0.99, 1: 0.62) F1 = (0: 0.95, 1: 0.76)	A = 0.721333 P = (0: 0.81, 1: 0.24) R = (0: 0.86, 1: 0.18) F1 = (0: 0.83, 1: 0.20)	A = 0.893167 P = (0: 0.9, 1: 0.85) R = (0: 0.97, 1: 0.57) F1 = (0: 0.94, 1: 0.68)	A = 0.860667 P = (0: 0.91, 1: 0.65) R = (0: 0.91, 1: 0.66) F1 = (0: 0.91, 1: 0.65)	A = 0.840500 P = (0: 0.90, 1: 0.59) R = (0: 0.90, 1: 0.59) F1 = (0: 0.90, 1: 0.59)
Naive Bayes	A = 0.675500 P = (0: 0.88, 1: 0.32) R = (0: 0.69, 1: 0.61) F1 = (0: 0.77, 1: 0.42)	A = 0.675667 P = (0: 0.83, 1: 0.28) R = (0: 0.74, 1: 0.41) F1 = (0: 0.79, 1: 0.33)	A = 0.514833 P = (0: 0.82, 1: 0.20) R = (0: 0.52, 1: 0.51) F1 = (0: 0.63, 1: 0.29)	A = 0.948667 P = (0: 0.96, 1: 0.91) R = (0: 0.98, 1: 0.82) F1 = (0: 0.97, 1: 0.86)	A = 0.851667 P = (0: 0.96, 1: 0.59) R = (0: 0.85, 1: 0.87) F1 = (0: 0.90, 1: 0.70)
SVC	A = 0.838833 P = (0: 0.83, 1: 0.99) R = (0: 1.00, 1: 0.21) F1 = (0: 0.91, 1: 0.35)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.863833 P = (0: 0.86, 1: 0.93) R = (0: 0.99, 1: 0.34) F1 = (0: 0.92, 1: 0.49)	A = 0.943833 P = (0: 0.94, 1: 0.98) R = (0: 1.00, 1: 0.74) F1 = (0: 0.97, 1: 0.84)	A = 0.934500 P = (0: 0.93, 1: 0.93) R = (0: 0.99, 1: 0.72) F1 = (0: 0.96, 1: 0.81)
AdaBoost	A = 0.916167 P = (0: 0.92, 1: 0.90) R = (0: 0.98, 1: 0.67) F1 = (0: 0.95, 1: 0.76)	A = 0.777667 P = (0: 0.80, 1: 0.29) R = (0: 0.96, 1: 0.07) F1 = (0: 0.87, 1: 0.11)	A = 0.891167 P = (0: 0.91, 1: 0.79) R = (0: 0.96, 1: 0.62) F1 = (0: 0.93, 1: 0.69)	A = 0.933333 P = (0: 0.95, 1: 0.88) R = (0: 0.97, 1: 0.77) F1 = (0: 0.96, 1: 0.82)	A = 0.909833 P = (0: 0.93, 1: 0.80) R = (0: 0.96, 1: 0.70) F1 = (0: 0.94, 1: 0.75)
Gaussian Process	A = 0.809500 P = (0: 0.81, 1: 1.00) R = (0: 1.00, 1: 0.02) F1 = (0: 0.90, 1: 0.05)	A = 0.797833 P = (0: 0.81, 1: 0.69) R = (0: 0.99, 1: 0.02) F1 = (0: 0.89, 1: 0.04)	A = 0.809000 P = (0: 0.81, 1: 0.99) R = (0: 1.00, 1: 0.07) F1 = (0: 0.89, 1: 0.13)	A = 0.925000 P = (0: 0.92, 1: 1.00) R = (0: 1.00, 1: 0.60) F1 = (0: 0.96, 1: 0.75)	A = 0.872500 P = (0: 0.94, 1: 0.66) R = (0: 0.90, 1: 0.77) F1 = (0: 0.92, 1: 0.71)
K Neighbours	A = 0.801167 P = (0: 0.83, 1: 0.59) R = (0: 0.94, 1: 0.25) F1 = (0: 0.88, 1: 0.32)	A = 0.772833 P = (0: 0.81, 1: 0.26) R = (0: 0.94, 1: 0.09) F1 = (0: 0.87, 1: 0.13)	A = 0.804000 P = (0: 0.82, 1: 0.79) R = (0: 0.99, 1: 0.11) F1 = (0: 0.90, 1: 0.17)	A = 0.939333 P = (0: 0.98, 1: 0.82) R = (0: 0.95, 1: 0.90) F1 = (0: 0.96, 1: 0.85)	A = 0.897333 P = (0: 0.93, 1: 0.76) R = (0: 0.94, 1: 0.73) F1 = (0: 0.94, 1: 0.74)
Multi-Layer Perceptron	A = 0.879833 P = (0: 0.87, 1: 0.98) R = (0: 1.00, 1: 0.39) F1 = (0: 0.93, 1: 0.55)	A = 0.797167 P = (0: 0.81, 1: 1.00) R = (0: 1.00, 1: 0.02) F1 = (0: 0.89, 1: 0.03)	A = 0.877667 P = (0: 0.88, 1: 0.89) R = (0: 0.98, 1: 0.46) F1 = (0: 0.93, 1: 0.60)	A = 0.956833 P = (0: 0.96, 1: 0.97) R = (0: 0.99, 1: 0.80) F1 = (0: 0.97, 1: 0.88)	A = 0.934500 P = (0: 0.95, 1: 0.87) R = (0: 0.97, 1: 0.79) F1 = (0: 0.96, 1: 0.83)
XGBoost	A = 0.912333 P = (0: 0.91, 1: 0.91) R = (0: 0.98, 1: 0.63) F1 = (0: 0.95, 1: 0.74)	A = 0.775000 P = (0: 0.81, 1: 0.30) R = (0: 0.94, 1: 0.10) F1 = (0: 0.87, 1: 0.15)	A = 0.894833 P = (0: 0.90, 1: 0.84) R = (0: 0.97, 1: 0.59) F1 = (0: 0.94, 1: 0.69)	A = 0.946167 P = (0: 0.94, 1: 0.96) R = (0: 0.99, 1: 0.77) F1 = (0: 0.97, 1: 0.85)	A = 0.925833 P = (0: 0.93, 1: 0.90) R = (0: 0.98, 1: 0.70) F1 = (0: 0.96, 1: 0.79)
Linear Discrimination	A = 0.581500 P = (0: 0.86, 1: 0.29) R = (0: 0.56, 1: 0.66) F1 = (0: 0.67, 1: 0.40)	A = 0.776500 P = (0: 0.82, 1: 0.38) R = (0: 0.93, 1: 0.16) F1 = (0: 0.87, 1: 0.23)	A = 0.778833 P = (0: 0.89, 1: 0.46) R = (0: 0.82, 1: 0.60) F1 = (0: 0.86, 1: 0.52)	A = 0.845333 P = (0: 0.94, 1: 0.59) R = (0: 0.87, 1: 0.77) F1 = (0: 0.90, 1: 0.67)	A = 0.728833 P = (0: 0.90, 1: 0.40) R = (0: 0.75, 1: 0.65) F1 = (0: 0.81, 1: 0.49)

Figure 9.3: The final table of results achieved by models tested using the 1000 comment 4Chan dataset.

3000 Comment Multi-Platform Dataset Results
(20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.949444 P = (0: 0.95, 1: 0.95) R = (0: 0.99, 1: 0.79) F1 = (0: 0.97, 1: 0.86)	A = 0.891333 P = (0: 0.90, 1: 0.86) R = (0: 0.98, 1: 0.55) F1 = (0: 0.93, 1: 0.67)	A = 0.901500 P = (0: 0.92, 1: 0.81) R = (0: 0.96, 1: 0.66) F1 = (0: 0.94, 1: 0.73)	A = 0.914889 P = (0: 0.91, 1: 0.98) R = (0: 1.00, 1: 0.59) F1 = (0: 0.95, 1: 0.74)	A = 0.901389 P = (0: 0.90, 1: 0.91) R = (0: 0.99, 1: 0.55) F1 = (0: 0.94, 1: 0.68)
Decision Tree	A = 0.917667 P = (0: 0.91, 1: 0.94) R = (0: 0.99, 1: 0.64) F1 = (0: 0.95, 1: 0.76)	A = 0.848722 P = (0: 0.89, 1: 0.65) R = (0: 0.92, 1: 0.56) F1 = (0: 0.91, 1: 0.60)	A = 0.890889 P = (0: 0.90, 1: 0.82) R = (0: 0.97, 1: 0.58) F1 = (0: 0.93, 1: 0.68)	A = 0.873222 P = (0: 0.92, 1: 0.70) R = (0: 0.93, 1: 0.67) F1 = (0: 0.92, 1: 0.68)	A = 0.843889 P = (0: 0.89, 1: 0.63) R = (0: 0.91, 1: 0.57) F1 = (0: 0.90, 1: 0.60)
Naive Bayes	A = 0.701944 P = (0: 0.88, 1: 0.35) R = (0: 0.73, 1: 0.59) F1 = (0: 0.80, 1: 0.44)	A = 0.830222 P = (0: 0.87, 1: 0.61) R = (0: 0.92, 1: 0.49) F1 = (0: 0.90, 1: 0.54)	A = 0.437833 P = (0: 0.88, 1: 0.24) R = (0: 0.34, 1: 0.81) F1 = (0: 0.49, 1: 0.37)	A = 0.936056 P = (0: 0.96, 1: 0.85) R = (0: 0.97, 1: 0.82) F1 = (0: 0.96, 1: 0.83)	A = 0.810000 P = (0: 0.95, 1: 0.51) R = (0: 0.81, 1: 0.83) F1 = (0: 0.87, 1: 0.63)
SVC	A = 0.912889 P = (0: 0.90, 1: 0.99) R = (0: 1.00, 1: 0.57) F1 = (0: 0.95, 1: 0.72)	A = 0.872389 P = (0: 0.86, 1: 0.96) R = (0: 1.00, 1: 0.39) F1 = (0: 0.93, 1: 0.55)	A = 0.892611 P = (0: 0.89, 1: 0.92) R = (0: 0.99, 1: 0.50) F1 = (0: 0.94, 1: 0.65)	A = 0.956500 P = (0: 0.96, 1: 0.96) R = (0: 0.99, 1: 0.82) F1 = (0: 0.97, 1: 0.88)	A = 0.933944 P = (0: 0.94, 1: 0.91) R = (0: 0.98, 1: 0.74) F1 = (0: 0.96, 1: 0.81)
AdaBoost	A = 0.940111 P = (0: 0.94, 1: 0.93) R = (0: 0.99, 1: 0.75) F1 = (0: 0.96, 1: 0.83)	A = 0.878556 P = (0: 0.90, 1: 0.76) R = (0: 0.95, 1: 0.58) F1 = (0: 0.93, 1: 0.66)	A = 0.896889 P = (0: 0.91, 1: 0.81) R = (0: 0.96, 1: 0.63) F1 = (0: 0.94, 1: 0.71)	A = 0.930667 P = (0: 0.95, 1: 0.86) R = (0: 0.97, 1: 0.78) F1 = (0: 0.96, 1: 0.82)	A = 0.895500 P = (0: 0.92, 1: 0.76) R = (0: 0.95, 1: 0.68) F1 = (0: 0.94, 1: 0.72)
Gaussian Process	A = 0.837833 P = (0: 0.83, 1: 0.97) R = (0: 1.00, 1: 0.20) F1 = (0: 0.91, 1: 0.33)	A = 0.828000 P = (0: 0.82, 1: 0.94) R = (0: 1.00, 1: 0.17) F1 = (0: 0.90, 1: 0.28)	A = 0.851056 P = (0: 0.85, 1: 0.94) R = (0: 1.00, 1: 0.26) F1 = (0: 0.91, 1: 0.41)	A = 0.939500 P = (0: 0.93, 1: 0.98) R = (0: 1.00, 1: 0.71) F1 = (0: 0.96, 1: 0.82)	A = 0.863722 P = (0: 0.93, 1: 0.63) R = (0: 0.90, 1: 0.74) F1 = (0: 0.91, 1: 0.68)
K Neighbours	A = 0.810833 P = (0: 0.81, 1: 0.96) R = (0: 1.00, 1: 0.05) F1 = (0: 0.89, 1: 0.10)	A = 0.885500 P = (0: 0.89, 1: 0.88) R = (0: 0.98, 1: 0.49) F1 = (0: 0.93, 1: 0.63)	A = 0.811556 P = (0: 0.81, 1: 0.85) R = (0: 1.00, 1: 0.09) F1 = (0: 0.89, 1: 0.15)	A = 0.937333 P = (0: 0.98, 1: 0.81) R = (0: 0.95, 1: 0.91) F1 = (0: 0.96, 1: 0.85)	A = 0.904556 P = (0: 0.94, 1: 0.77) R = (0: 0.94, 1: 0.75) F1 = (0: 0.94, 1: 0.76)
Multi-Layer Perceptron	A = 0.912000 P = (0: 0.90, 1: 0.99) R = (0: 1.00, 1: 0.58) F1 = (0: 0.95, 1: 0.73)	A = 0.8437222 P = (0: 0.84, 1: 0.95) R = (0: 1.00, 1: 0.24) F1 = (0: 0.91, 1: 0.38)	A = 0.894611 P = (0: 0.89, 1: 0.90) R = (0: 0.99, 1: 0.52) F1 = (0: 0.94, 1: 0.66)	A = 0.952333 P = (0: 0.95, 1: 0.95) R = (0: 0.99, 1: 0.80) F1 = (0: 0.97, 1: 0.87)	A = 0.930000 P = (0: 0.95, 1: 0.86) R = (0: 0.97, 1: 0.78) F1 = (0: 0.96, 1: 0.82)
XGBoost	A = 0.937667 P = (0: 0.94, 1: 0.94) R = (0: 0.99, 1: 0.74) F1 = (0: 0.96, 1: 0.83)	A = 0.899500 P = (0: 0.91, 1: 0.82) R = (0: 0.97, 1: 0.62) F1 = (0: 0.94, 1: 0.71)	A = 0.903278 P = (0: 0.92, 1: 0.84) R = (0: 0.97, 1: 0.66) F1 = (0: 0.94, 1: 0.73)	A = 0.943167 P = (0: 0.95, 1: 0.93) R = (0: 0.98, 1: 0.78) F1 = (0: 0.97, 1: 0.85)	A = 0.918111 P = (0: 0.93, 1: 0.87) R = (0: 0.97, 1: 0.69) F1 = (0: 0.95, 1: 0.77)
Linear Discrimination	A = 0.718222 P = (0: 0.88, 1: 0.38) R = (0: 0.75, 1: 0.60) F1 = (0: 0.81, 1: 0.46)	A = 0.900778 P = (0: 0.90, 1: 0.90) R = (0: 0.99, 1: 0.56) F1 = (0: 0.94, 1: 0.69)	A = 0.887833 P = (0: 0.92, 1: 0.74) R = (0: 0.95, 1: 0.65) F1 = (0: 0.93, 1: 0.69)	A = 0.946111 P = (0: 0.96, 1: 0.89) R = (0: 0.97, 1: 0.83) F1 = (0: 0.97, 1: 0.86)	A = 0.917333 P = (0: 0.94, 1: 0.81) R = (0: 0.95, 1: 0.77) F1 = (0: 0.95, 1: 0.79)

Figure 9.4: The final table of results achieved by models tested using the 3000 comment multi-platform dataset.

Acknowledgements

I would like to give a massive thank you to my project supervisor Soumyabrata Dev for voluntarily taking my project on at a late stage, and for his invaluable support and guidance throughout the duration of this project.

Bibliography

1. O'Dea, B. Facebook reports increased removal of hate speech across its platforms. *SiliconRepublic.* https://www.siliconrepublic.com/business/facebook-increase-removal-hate-speech-instagram (2021).
2. Wagner, K. Twitter Penalizes Record Number of Accounts for Posting Hate Speech. *TIME.* https://time.com/6080324/twitter-hate-speech-penalties/ (2021).
3. Ray, S. Reddit Says It Has Removed 7,000 'Hateful' Subreddits Since Changing Hate Speech Policy In June. *Forbes.* https://www.forbes.com/sites/siladityaray/2020/08/20/reddit-says-it-has-removed-7000-hateful-subreddits-since-changing-hate-speech-policy-in-june/?sh=2b54cd71233e (2020).
4. trackmyhashtag. *trackmyhashtag* 2021. https://www.trackmyhashtag.com/.
5. Rudnicki, K. & Steiger, S. *Online hate speech* (DeTACT, 2020). https://www.media-diversity.org/wp-content/uploads/2020/09/DeTact_Online-Hate-Speech.pdf.
6. Jourová, V. *Code of Conduct - Illegal online hate speech Questions and answers* (European Commission, 2016). https://ec.europa.eu/info/sites/default/files/code_of_conduct_hate_speech_en.pdf.
7. Hawdon, J., Oksanen, A. & Räsänen, P. *Online Extremism and Online Hate Exposure among Adolescents and Young Adults in Four Nations* (Nordicom, 2015). https://www.nordicom.gu.se/sites/default/files/kapitel-pdf/nordicom-information_37_2015_3-4_29-37.pdf.
8. Inc., T. Introducing the new Twitter Transperency Center. *Twitter.* https://blog.twitter.com/en_us/topics/company/2020/new-transparency-center (2020).
9. L1GHT. *Rising Levels of Hate Speech Online Toxicity During This Time of Crisis* (L1GHT, 2020). https://l1ght.com/Toxicity_during_coronavirus_Report-L1ght.pdf.
10. Han, E. *Countering hate on TikTok* (TikTok, 2020). https://newsroom.tiktok.com/en-us/countering-hate-on-tiktok.
11. Facebook. *Hate Speech* (Facebook, 2021). https://transparency.fb.com/data/community-standards-enforcement/hate-speech/facebook/.
12. University, C. Increase in online hate speech leads to more crimes against minorities. *PHYSORG.* https://phys.org/news/2019-10-online-speech-crimes-minorities.html (2019).
13. Williams, M., Burnap, P., Javed, A., Liu, H. & Ozalp, S. CORRIGENDUM TO: HATE IN THE MACHINE: ANTI-BLACK AND ANTI-MUSLIM SOCIAL MEDIA POSTS AS PREDICTORS OF OFFLINE RACIALLY AND RELIGIOUSLY AGGRAVATED CRIME. *Brit J Criminol* (2020).
14. Kovács, G., Alonso, P. & Saini, R. Challenges of Hate Speech Detection in Social Media. *Springer Nature.* https://link.springer.com/content/pdf/10.1007/s42979-021-00457-3.pdf (2020).
15. Davidson, T., Warmsley, D., Macy, M. & Weber, I. *Automated Hate Speech Detection and the Problem of Offensive Language* (Cornell University, 2017). https://arxiv.org/pdf/1703.04009.pdf.
16. Bhattacharya, D. & Weber, I. Racial Bias in Hate Speech and Abusive Language Detection Datasets. *ACL Anthology.* https://aclanthology.org/W19-3504/ (2019).
17. Röttger, P. *et al. HATECHECK: Functional Tests for Hate Speech Detection Models* (The Alan Turing Institute, 2021). https://arxiv.org/pdf/2012.15606.pdf.

18. Sap, M., Card, D., Gabriel, S., Choi, Y. & Smith, N. *The Risk of Racial Bias in Hate Speech Detection* (Aclanthology, 2019). https://aclanthology.org/P19-1163.pdf.

19. Founta, A.-M., Djouvas, C., Chatzakou, D., Leontiadis, I. & Blackburn, J. *Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior* (Cornell University, 2018). https://arxiv.org/abs/1802.00393.

20. Salminen, J. *et al.* Developing an online hate classifier for multiple social media platforms. *Human-centric Computing and Information Sciences.* https://hcis-journal.springeropen.com/track/pdf/10.1186/s13673-019-0205-6.pdf (2020).

21. Kansara, K. & Shekokar, N. A Framework for Cyberbullying Detection in Social Network. *Semantic Scholar.* https://inpressco.com/a-framework-for-cyberbullying-detection-in-social-network/ (2015).

22. Ramampiaro, H. Detecting Offensive Language in Tweets Using Deep Learning. *Cornell University.* https://arxiv.org/abs/1801.04433 (2018).

23. Lee, H. S. & Lee, H. R. An abusive text detection system based on enhanced abusive and non-abusive word lists. *Yonsei University.* https://yonsei.pure.elsevier.com/en/publications/an-abusive-text-detection-system-based-on-enhanced-abusive-and-no (2018).

24. HASOC. *HASOC (2019)* 2019. https://hasocfire.github.io/hasoc/2019/.

25. Mozafari, M. Hate speech detection and racial bias mitigation in social media based on BERT model. *PLOS ONE.* https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0237861 (2020).

26. Alshalan, R. & Al-Khalifa, H. A Deep Learning Approach for Automatic Hate Speech Detection in the Saudi Twittersphere. *MDPI.* https://www.mdpi.com/2076-3417/10/23/8614 (2020).

27. Vigna, F. D., Cimino, A. & Petrocchi, M. Hate me, hate me not: Hate speech detection on Facebook. *First Italian Conference on Cybersecurity.* http://ceur-ws.org/Vol-1816/paper-09.pdf (2017).

28. Perspective. *Using Machine Learning to Reduce Toxicity Online* 2021. https://www.perspectiveapi.com.

29. twohat. *Introducing... Sift Ninja!* 2021. https://www.twohat.com/blog/introducing-sift-ninja/.

30. Reddit. *API Terms of Use* 2016. https://www.reddit.com/wiki/api-terms.

31. Twitter. *Developer Agreement and Policy* 2020. https://developer.twitter.com/en/developer-terms/agreement-and-policy.

32. 4Chan. *4Chan API* 2021. https://github.com/4chan/4chan-API.