MEng Individual Project

Imperial College London

Department of Electrical and Electronic Engineering

Hate Speech Detection and Hate Network Detection on Social Media

Author:
Jeetendra Joshi

CID:
01518903

Supervisor:
Prof. Jeremy Pitt

Second Marker:
Dr. Javier Barria

June 22, 2022

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

Abstract

The current problem with social media is the large reliance on the presence of non-malicious actors. However, the spreadability of social media ensures that malicious actors will be able to spread hateful views very easily. This project provides practical methodologies to detect hate speech through the development of research and the design of a hate speech detection model. These models were further developed to be utilised in a hateful network detector that has successfully identified hate networks on the Twitter platform. The hateful network detector can be successfully deployed to find chains and communities of hate speech users embedded within non-hate speech communities. The models investigated included K-NNs, transformers and GRUs have been used with various sets of tweets to attempt to produce a model that can easily distinguish hate and non-hate speech instances. The work in this paper also determined the effect of preprocessing on the effectiveness of a model with extensive testing performed to evaluate the effectiveness of the models and the type of preprocessing employed. This research further ponders the question as to why the methods implemented in this paper could not be utilised earlier by the Twitter platform as the methods in this paper showcase that self-regulation by the social media platforms should be possible.

Acknowledgements

I would like to thank Professor Jeremy Pitt and Dr David Burth Kurka for their continuous advice and supervision of the project in addition to their continuous input and for giving me the freedom of choice in how to construct and tailor the project to my interests.
I would also like to thank my friends and family for continuous help and support throughout my four years at Imperial and for allowing me to grow through my experience at university.

Contents

List of Figures

List of Tables

List of Abbreviations

ML	Machine Learning
DL	Deep Learning
K-NN	K-Nearest Neighbours
GNN	Graph Neural Network
LSTM	Long Short Term Memory
BiLSTM	Bidirectional Long Short-Term Memory
GRU	Gated Recurrent Unit
BiGRU	Bidirectional Gated Recurrent Unit
NBSVM	Naive Bayes Support Vector Machine
SVM	Support Vector Machine
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network

Chapter 1

Introduction

1.1 Motivation

In the last 25 years, since the development of Six Degrees, the first social media platform [10], there has been a massive increase in the use of social media, with growth from 2.86 billion people in 2017 to 3.6 billion in 2020 [11]. Social media has its many advantages allowing the public to connect and be able to spread vast amounts of information quickly. One of the current issues with the efficient spread of information is that malicious actors can use this key benefit to spread hate speech and extremist beliefs rapidly and at a mass scale. Companies such as Facebook are not actively trying to stop the spread of this information and are not putting enough effort into stopping the spread of hate and extremist content on their platforms [12]. The term "machine learning" was first coined in 1959 by Arthur Samuel [13] and ever since, there have been rapid developments in the field, going as far as branching out into sub-fields such as computer vision and natural language processing. This project will be focusing on the use of machine learning in the field of natural language processing to try and create an automated model that would be able to classify hate speech and detect hate networks on the Twitter platform.

1.2 Project Specification

The project aims to be able to develop a machine learning approach to detecting hate speech and developing a system that could identify malicious actors who are spreading this discrimination. Additionally, the system should also be able to determine the impact of individual actors within a social network to identify who the key players are in the outspread of hate speech. To do this, the project involves four stages. The first is to develop a classifier that can accurately distinguish between hate and non-hate speech instances. The second objective is to identify the account that a particular text is associated with, followed by checking for multiple instances of discrimination on a given account. The third part of this investigation involves developing a social network based on the connections of a malicious actor. The fourth and final part involves identifying key players within these social networks and determine how large a hate speech netowrk may have grown. The main goal of this project is to determine the difficulties in finding hate speech users and why social media platforms such as Twitter have not done more. In the case of success,

the report further raises the question as to why hate speech and hateful users on Twitter are still such a prominent problem.

1.3 Report Structure

Chapter 2 showcases a literature review to introduce current techniques used in hate speech detection in addition to a basic overview of how the overall system works concerning the relevant papers. The analysis and design chapter showcases the thought processes before implementation and is detailed in Chapter 3. It also details the requirements of the overall project. The implementation of the project is detailed in Chapter 4. Chapter 5 shows the overall evaluation of the system with overall conclusions and possible future works detailed in Chapter 6. The appendices contain various images, data and the original plan for how the project was going to be implemented.

1.4 Contributions

Throughout this project, various contributions were made. They are as follows:

1. Identification of the large prevalence of specific non-hate related tags within hate speech texts.
2. The effectiveness of preprocessing on the classification accuracy of K-NNs.
3. Investigation of 16 different preprocessing methods for 4 different deep learning methods over 50 epochs.
4. Development of an ensemble model that performs well at classifying hate speech and performs similarly to models developed in other papers.
5. A novel method to detect hate networks using the Tweepy API and a depth-based hate speech detection with the determination of optimal hate speech thresholding for the classification of hate users.

Chapter 2

Background

The following literature review/background chapter will focus on various machine learning methods and delve into current methods of hate speech detection. At this moment in time, though possible, manual hate speech detection would be completely impractical. There are 500 million tweets per day [14], making it impossible to monitor every tweet. This sheer volume makes it impossible, in addition to different human monitors having to distinguish between different tweets and classify each tweet with inherent natural bias. Furthermore, tweets are not constrained to a singular language, further complicating the problem and reinforcing that manual hate speech detection is unfeasible. Therefore, automated methods of hate speech detection are further studied. The literature review outlines various ML models that could be utilised in the field of hate speech detection. Additionally, it provides an explanations and links to a relevant paper that shows the implementation of the method in a hate speech detection system.

2.1 Literature Review

2.1.1 Convolutional Neural Networks

Convolutional neural networks have been used in the field of natural language processing and even further, in the detection of hate speech. The three main layer types are convolutional layers, pooling layers and fully connected layers are used in the definition of a convolutional neural network. Convolutional layers contain parameters which allow multiple values to be parsed into a singular value. Pooling layers are used to reduce the overall dimensions of a given input by essentially compressing the values. The two most common methods are compressing values into the maximum value or to the average of the surrounding input blocks. Fully connected layers take all inputs from the previous layer into each of its neurons and use them for a weight calculation output which is fed to the next layer as output.

CNNs have been used in the paper *"Investigating Deep Learning Approaches for Hate Speech Detection in Social Media"* [15] and have had varying results when compared to other models architectures. The loss function used for all CNNs was the categorical cross-entropy with an Adam optimiser to ensure that all of the models were consistent in optimiser choice. Additionally, Adam optimisers have the ability

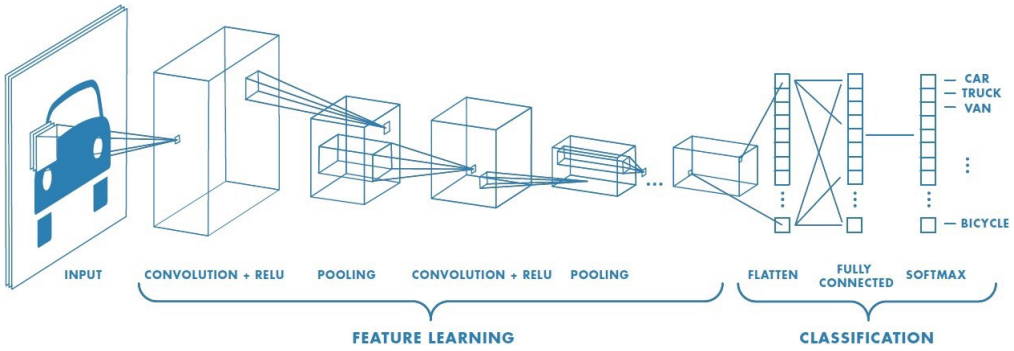


Figure 2.1: A visualisation of a CNN architecture being used on a computer vision problem [1]

to handle sparse gradients when the machine learning problem they are tasked with is quite noisy. Sparse gradients make it difficult to train networks correctly due to high noise in the input or high variance in the input space. All of the model architectures were deployed on 3 different datasets with the hate speech class having a presence of 31%, 5.8% and 7.1% respectively. The CNN performed on par with other models attaining accuracy of 90.47%, 90.35% and 90.17% respectively with the W2V, GloVe and FastText embeddings respectively on dataset 1. Additionally, it proved to be quite accurate on dataset 2 with slightly lower accuracy of 83.15%, 82.98% and 83.21% accuracy on the W2V, GloVe and FastText embeddings. When comparing F1-Scores, the model also performs to an average degree having F1-Scores of 89.81, 89.60 and 88.85 for the three of the previous embeddings on dataset 1.

2.1.2 RNNs, LSTMs and GRUs

Work has been done on trying to practically apply recurrent neural networks (RNNs) to the application of hate speech detection. RNNs are quite useful when looking at textual data, as they can look at previous inputs, which when looking at phrases and sentences, help to provide context to the meaning of each word. However, they primarily suffer from the vanishing/exploding gradient problem and do not have large reference windows from which they can infer meaning. The vanishing gradient problem occurs during training, at backpropagation when the derivative of the slope gets exponentially smaller. Similarly, the exploding gradient results in a massive increase in the slope of the gradient during the update of weights at backpropagation. Even though these drawbacks are present, RNNs have still proven to be useful in the field of hate speech detection and classification.

Long short-term memory networks (LSTMs) are effectively more sophisticated RNNs that can counteract the vanishing/exploding gradient problem. LSTMs are comprised of a hidden layer called a "gated cell". This is comprised of four layers that produce both an output and a cell state which is passed on to the next layer. Additionally, LSTMs consist of three sigmoid gates and one tanh layer to limit the information passed through the four layers. This helps determine whether a previous output actually will be used in the calculation of the output. These additions to the LSTM networks allow the network to be sensitive to the order of incoming values within the neural network.

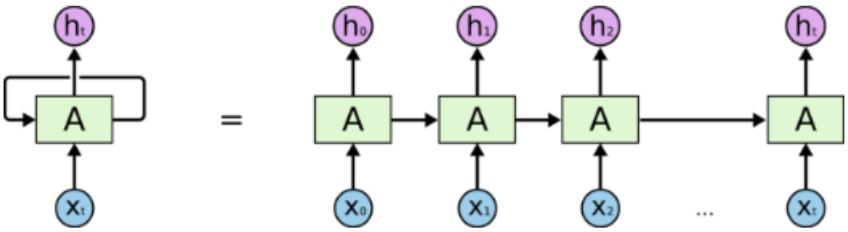


Figure 2.2: An image of an RNN [2]

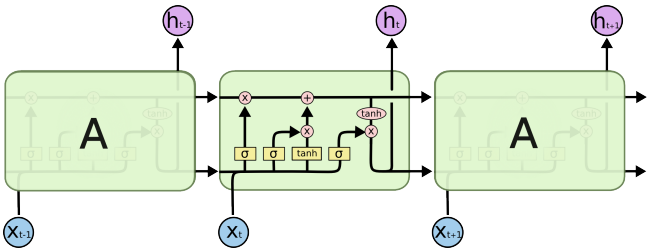


Figure 2.3: An image showing the internal structure of an LSTM [3]

Gated recurrent units(GRUs) are another variation of RNNs that aim to solve the vanishing gradient problem. The architecture is a variation through the model consisting of an update and reset gate. The update gate helps the model determine how much of the information from a previous time step is used in future calculations. In contrast, the reset gate is used to portion the amount of previous information used in future calculations. This helps solve the exploding gradient problem as the gradient is no longer allowed to inflate exponentially. These two gates allow the memory unit within GRUs to not be susceptible to the vanishing and exploding gradient problems that were a major issue when working with RNNs and had been dealt with through the use of a gated cell within LSTMs.

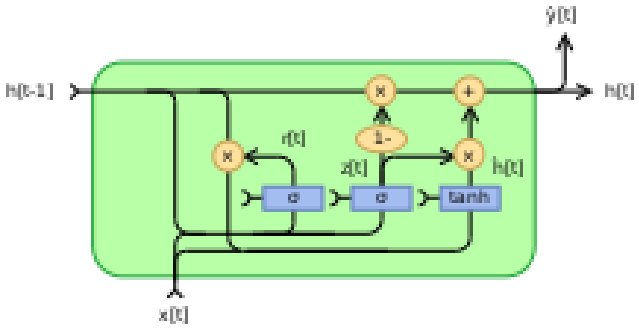


Figure 2.4: GRU Internal Architecture [4]

In the paper *"Effective hate-speech detection in Twitter data using recurrent neural networks"* [16], various LSTMs and LSTM ensembles have been used with great effect to classify hate speech into the classes of racism and sexism. The ensemble

of classifiers with all classes ("No additional features", "Neutral and Racist", "Neutral and Sexist", "Racist and Sexist" and "Neutral and Racist and Sexist") had the highest precision, recall and F1-Score (at 0.9300) when compared to other classifiers. This shows that the classifier was able to identify a high proportion of hate speech with a high proportion of hate speech classified compared to the overall number of tweets classified as hate speech. Ensemble learning provided greater results in all cases when compared to singular classifiers which lead to an interesting prospect which could be used in tandem with various machine learning techniques to identify hate speech at a higher accuracy.

GRUs have also been used in the paper *"Detecting Hate Speech on Twitter Using a Convolution-GRU Based Deep Neural Network"* [17]. This paper implemented the use of a convolutional network paired with a GRU as the main investigative model. This model (labelled $CNN+GRU$) and the model with removed dropout and global pooling layers (labelled $CNN + GRU_B$) performed the best when compared to the other models and additional state-of-the-art models. It was interesting to note that it performed better than all models on all datasets apart from one, proving it to be an efficient model for distinguishing the difference between hate speech and neutral tweets. Another fascinating finding is that most of the errors that occurred with the GRU models were on sexist hate speech comments followed by neutral tweets which could infer that ML models have difficulty when dealing with gender-oriented tweets. Overall, the GRU-based models performed with a high F1 score on all datasets peaking at 0.94 on the DT dataset.

2.1.3 KNNs and SVMs

K-Nearest Neighbours(KNNs) is a supervised learning algorithm which allows a classifier to classify a piece of data relative to its K-nearest data points depending on which are closest to it. The majority class assigned to the nearest data points is the same. KNNs are very easy to implement. Despite this, they become increasingly slower as the length of the training dataset increases. This is because the distance between every point has to be computed to be able to classify the point correctly. Another requirement is that the K has to be chosen such that it is the correct value. A K too small may result in a misclassification if the closest distance to a specific data point is an anomaly. However, a large K may result in too many points being used. This could result in an incorrect classification, as distant points are considered when they are irrelevant to the classification of a specific data point. Nevertheless, a benefit of KNNs is that they can be used for multi-class classification, which would be useful to classify hate speech into subclasses, as seen in the *"Effective hate-speech detection in Twitter data using recurrent neural networks"*[16].

Support Vector Machines(SVMs) are another supervised learning algorithm which can be used for binary class classification and multi-class classification when multiple SVMs are used together. SVMs aim to construct a hyperplane between points allowing them to be classified relative to their position relative to the hyperplane. The hyperplane is constructed to maximise the margin between itself and the closest point to it. A hard margin is used where data can be classified as linearly separable, whilst soft margins are used with hinge loss to try and optimise the hyperplane

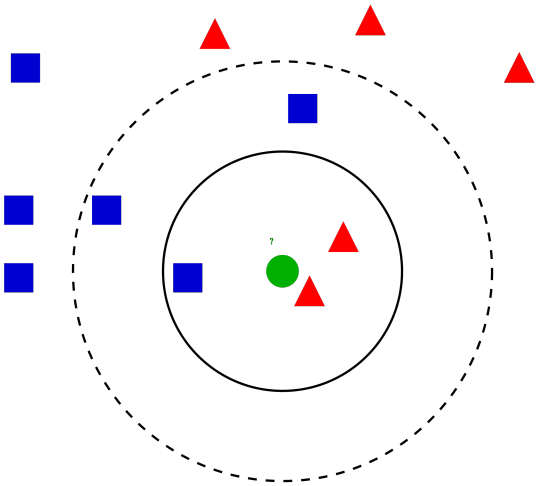


Figure 2.5: A visual representation of a K-NN Classifier [5]

definition. SVMs can be further developed to allow for non-linear classification whereby a kernel is applied by adding an extra dimension to the dataset. The benefit of SVMs is that they can classify points with high dimensionality and can be used with various kernels to help classify ranges of datasets. However, they suffer from being computationally slow and the efficiency decreases as the size of the dataset increases, resulting in difficulty when scaling.

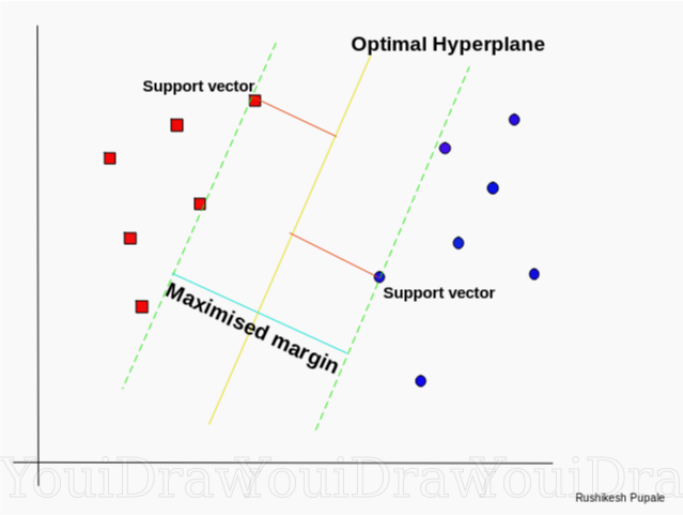


Figure 2.6: An image displaying hyperplane separation [6]

Nevertheless, both KNNs and SVMs have been used to classify hate speech on social media in the paper *Using KNN and SVM Based One-Class Classifier for Detecting Online Radicalization on Twitter* [18]. Both SVMs and KNNs had high accuracies at 90% and 97% respectively. The SVM model performed slightly better overall with higher precision, recall and F1-score meaning it could classify positive class tweets (instances of hate speech) to a better level than that of the SVM. The SVM also had higher TNR (true negative ratio) and NPV (negative predictive value) values showing it could distinguish neutral tweets at a higher efficacy when compared to KNNs. Furthermore, it can be seen that both classifiers' accuracies are

very dependent on the words, phrases and symbols used within each tweet with the presence of religious and war-related phrases having the highest impact compared to every other individual feature when determining the accuracy of the models. These specific features resulted in a decrease in classifier accuracy by 20-25% whilst removal of punctuation marks had very little effect possibly showing that pre-processing the punctuation does not have much impact on the accuracy of this classifier.

2.1.4 Transformers

Transformers are machine learning models specifically in the fields of natural language processing and computer vision [19]. In the field of natural language processing, they try to solve text-based tasks reliant on the sequence of the input to the model. This makes transformers especially useful for tasks such as language translation and language interpretation. Transformers were first introduced in the paper *"Attention Is All You Need"* [7]. Transformers consist of an encoder and decoder module as shown in Figure 2.7.

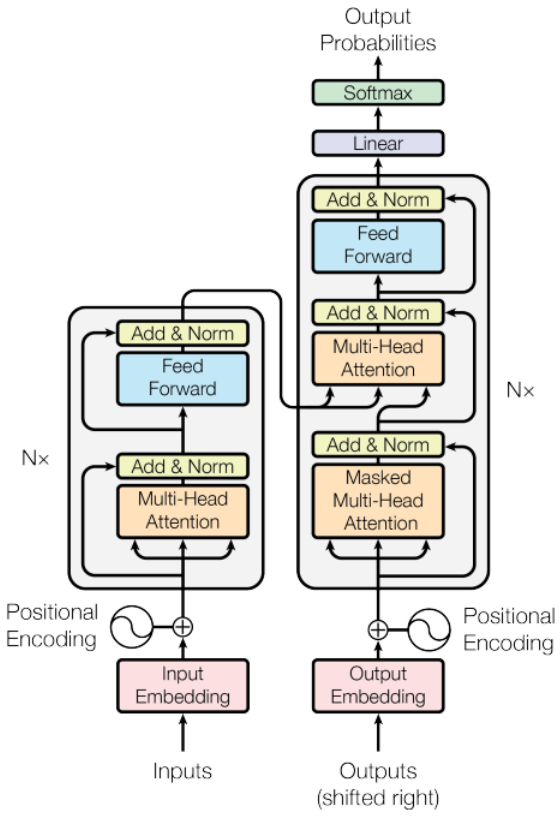


Figure 2.7: Transformer Model Architecture [7]

The transformer consists of an encoder and a decoder. Data is inputted into the transformer in parallel rather than sequentially. This allows the transformer to compute word embeddings simultaneously. Embeddings are used to allow a computer to determine meaning from a set of words by vectorising it and mapping it into an embedding space. Positional encoders produce a vector that allows position-dependent context to a set of words. This positional encoding is combined with a

word embedding to provide contextual information to the encoder. The combined vector is passed to a multi-head attention block which allows the transformer to focus on specific parts of the input sequence. The original problem with the attention vector produced is that each word places a high attention value on itself compared to other words in the vector. Instead, the vector is calculated via computing attention over several instances and outputting a weighted average over the several vectors. The output of the multi-head attention is passed through to a feed-forward network which prepares the input for use in the decoder module. As the vectors are independent of one another, the feed-forward network is parallelised to increase the overall efficiency of the transformer model.

The first input to the decoder block is the required output mapping to the input. This is embedded and combined with its positional encoding, similarly to the input used in the encoder module. This vector is passed to a masked multi-head attention block to produce an attention vector for the output. The difference between this block and the multi-head attention block is that the attention vector per word can only calculate values for words before this word. Words that appear later in the phrase relative to any given word are effectively masked by attaching a 0 value to the relevant attention vector value. The input from the encoder and this attention vector are then passed to another multi-head attention block to produce an encoder-decoder attention vector. The encoder-decoder attention block produces a relation vector for each word comparative to each other word in the phrase. The output of this block is passed to a feed-forward network which readies the output to a probability mapping. This is further throughput to a linear layer which expands the dimension of the output to map to every word possible in the required output. This is further passed to the output layer to produce probabilities for the word mappings. The highest probability element produced in the softmax output corresponds to the next word in the required output as the input is inputted recurrently in the network. This entire process is repeated for the length of any input till the end of input token is parsed by the transformer. At the end of all the attention layers and feed-forward layers, the input is normalised using layer values as opposed to mask values as this results in the stabilisation of values into the layers.

Transformers have been used in hate speech detection in the paper *"Hate Speech Detection in Twitter using Transformer Methods"* [20]. In the paper, transformers such as BERT and DistilBERT have been used to classify various instances of hate speech on Twitter. DistilBERT proved to have the highest values in both the accuracy and precision metrics, standing at 0.92 and 0.75 respectively. Comparatively to LSTM methods, the DistilBERT method proved to be much more effective, scoring higher in accuracy by 26%, precision by 9% and recall by 9%. All transformer models also resulted in lower evaluation loss results proving that LSTMs are not as good at classification tasks when compared to transformers. What was most surprising was that DistilBERT proved to have higher proficiency in all metrics compared to BERT with only half as many parameters.

2.1.5 Ensemble Learning

Ensemble learning is a technique which combines several machine learning architectures to come to a collective outcome. It becomes a very useful methodology when working with small or extremely large datasets in addition to being able to let specific classifiers specialise on specific values within the dataset to produce overall good performance. Ensemble learning is used to optimise various performance metrics such as accuracy and precision through utilising these values and can counteract overfitting through the use of different models adapting their respective weights differently when looking at a range of features. Various approaches are also taken when evaluating the overall consensus such as majority voting, median voting or weighted sum rules. Further information on these strategies can be read in an article on Ensemble Learning [21].

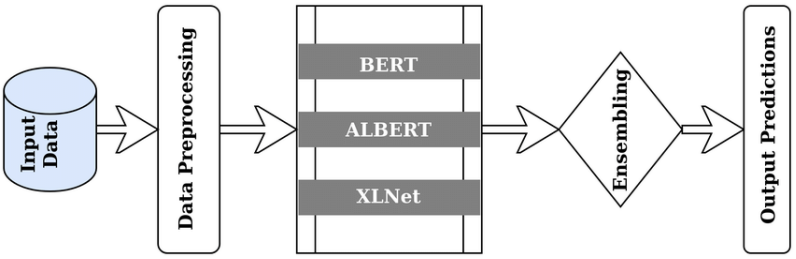


Figure 2.8: Ensemble model architecture consisting of only transformer models [8]

Ensemble Learning has been used in hate speech detection in tandem with transformers in the paper *"Hate Speech Detection using Transformer Ensembles on the HASOC dataset"* [22]. Each model was trained on a subset of the original training dataset and tested individually as well as on an ensemble. Every transformer performed well individually, all having relatively high F1-Scores. However, the ensemble of all the models proved to be more effective having a higher F1-Score as an ensemble rather than when computing the average of the individual F1-Scores. Additionally, all models proved to be working better on the dataset comprised of HASOC and OffensEval comparatively to the HASOC dataset individually. This seemed to be because the five transformers in the ensemble tended to have the same classification when averaging the overall classification as opposed to each of them individually trained transformers.

2.1.6 Graph Neural Networks

Graph neural networks represent a network in terms of nodes. Each node is given an initial value, represented by a vector. Every iteration, the graph is updated to better represent the problem that the graph is defined for. This is done through the connections between the nodes in the graph. The updates are done using (2.1)

$$h_i^{t+1} = f(h_i^t W + \sum_{j \in N(i)} \frac{1}{c_{ij}} h_j^t U) \tag{2.1}$$

The first term in the update allows the network to continuously propagate memory in the form of h_i from previous stages by multiplying it by a weight W. The weight

matrix that multiplies the memory is trained by the neural network to better update the GNN. The h_j^t term allows the node to be updated with the information from its neighbours. The U matrix is also trained by the neural network to allow the model to decide what features are most important when considering neighbours. The aggregation function (given by the term $\sum_{j \in N(i)} \frac{1}{c_{ij}}$) aggregates the sum of all nodes within the graph without any intrinsic order. This allows the model to be permutation-invariant as the values are not determined by the position in the vector. This ensures that there is no inherent order within a given graph. All of the previously mentioned parameters are used within a function f which is learned during the training process to allow for the best weight updates. Graph neural networks can be applied to several areas such as node classification, graph classification and link prediction. Figure 2.9 shows an example of a graph that could be classified through the use of a GNN.

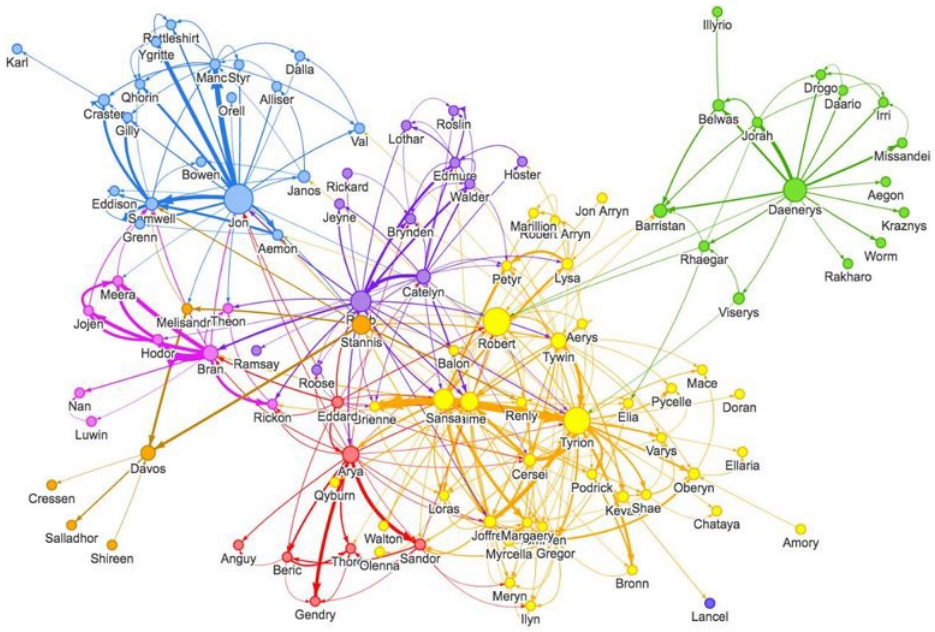


Figure 2.9: An image of a social network that could be classified using a GNN [9]

"Neighbours and Kinsmen: Hateful Users Detection with Graph Neural Network"[23] shows the use of graph neural networks in the classification of hate speech. The weighting of various nodes was further categorised into four categories which were defined as content-related properties, activity-related properties, sentiment-related properties and structure-related properties. These were identified as key features that the GNN could help to represent the overall social network and be able to identify certain users. The model performs quite well in the detection of hateful users, classifying at a 95% accuracy on the HateUser5k dataset and an 87% accuracy on the Tweet9k dataset. This performed better than the state-of-the-art GraphSage model which performed worse on metrics such as F1-Score and area under curve (AUC) when compared to the GNN.

Chapter 3

Analysis and Design

The analysis and design chapter breaks down what is required of the entire project. Initially, it contains a requirements section which captures the deliverables of the overall project. Following this, the analysis and design section takes a look at the choices made from initial development followed by a detailed description of the step-by-step process required to develop both the hate speech and hate network detection systems.

3.1 Requirements

The requirements section allows the overall deliverables to be captured in more detail and outline the format for the rest of the report. This allows the reader to focus on project deliverables and key points in the overall development of this system.

3.1.1 System Deliverables

The two system deliverables of the project are a hate speech detection model and a hate speech network detector. The first deliverable is a hate speech detection model which must be able to differentiate between hate and non-hate speech instances consistently. The hate speech network detector will then use this model to detect hateful users to see whether a network of hateful users can be detected through a series of tweets. This could allow for more robust hate detection on social media and detect multiple hateful users on social media as opposed to singular users with a singular instance of hate speech.

Hate Speech Detection Model

The hate speech detection model requires a series of steps before model development. Firstly a hate speech dataset must be collected to train eachh of the models on. Following this, a textual analysis must be conducted to determine what features are the most important when detecting hate speech. Additionally, the tweets should be preprocessed such that only the most important features are being used for hate speech detection. Additionally, this will ensure that the machine learning models are learning the correct patterns as opposed to incorrect trends in hate speech data.

Work should also be done to determine the optimal choice of preprocessing to allow for better hate speech detection for machine learning models.

Hate Speech Network Detector

The hate speech network detector needs to be able to detect hateful users. This requires the use of the hate speech detector to detect hateful users. This will be done through the development of a majority rule to determine whether a user is hateful. The system will then continue to recursively look at the connections of a hateful user to observe whether a hateful network is present. Further research can be done on hateful networks and their impact on social media thereafter.

3.2 Analysis and Design

This analysis and design section breaks down components of the hate speech detector and hate network detector further into individual subparts. Additionally, it showcases the thought processes on which tools were decided to be used to build an effective hate speech detection system and hate network detection model. It displays the flow of data from a tweet into the hate speech detection model followed by the use of the hate speech detection model in detecting a hate network.

3.2.1 Design Overview

The overall goal of this project is to design a system which can primarily classify hate speech followed by the generation of a social network through the connections of a hateful user. The classification of a user is done through the use of machine learning to handle the classification of hate speech over a given user's past ten tweets. Thereafter, the system will sift through the user's connections to find additional hateful users and generate a social network. It can also iterate through non-hate speech connections to try and delve into finding segments of a network that can be classified as part of a non-hate network.

3.2.2 Prerequisites

Choice of Programming Language

When developing the entire system, the first task is to determine which tools would be required for system development. Though most programming languages are suitable for the development of a hate speech detection system, the system requires quick machine learning model development and large instances of data processing. Both Python[24] and MATLAB[25] allow for complex and large data processing at fast speeds. Additionally, both programming languages have large user bases and detailed documentation which allow the developer of the hate speech detection system to quickly develop the system with a large amount of support.

Python - Python is a programming language that was initially conceived in 1989 with major version releases in 1991, 2000 and 2008. It has been widely popularised since its conception in addition to receiving constant developer updates and a large open-source community. Due to this large open-source community, many libraries have been developed which are cross-compatible and have allowed for rapid uptake of Python in the realm of machine learning through the development of libraries such as Keras[26] and Pytorch[27]. The main problem with Python is it is slow when compared to languages such as Java[28] and C++[29] as it is interpreted as opposed to a compiled language. However, the ML models produced by Python are relatively quick as the libraries are built-in highly optimized versions of C++. This made Python an optimal choice for building a hate speech detection system and it is much easier to deploy ML models in contrast to other languages.

MATLAB - MATLAB was developed in 1984 and is predominantly used "in industry and academia, including deep learning and machine learning, signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology"[30]. MATLAB allows for large-scale data processing with a large variety of toolkits to support machine learning methods. Similarly to Python, MATLAB can query Twitter through an API by establishing a connection. However, MATLAB suffers from dominantly dealing with numerical data in addition to not having the full range of machine learning methods such as BERT and DistilBERT, slowing down the development of the hate speech detection system as these models would have to be built from scratch.

Choice of Development Workspace

Due to the high processing power and memory demands, a local computer instance would be infeasible to train and test large machine learning models. Hence, a separate development workspace was required which access to either a GPU or TPU to allow for faster model development. Additionally, the use of a development workspace allows for use of cloud services such as Google Drive and Github to store datasets in addition to being accessible across multiple devices.

Google Colaboratory - Google Colaboratory is a hosted Jupyter notebook service, released in 2017, with allows scripts to be run on a cloud environment. The Jupyter notebooks are stored on Google Drive to allow seamless integration with other services. Benefits of Colaboratory include the ability to work with Colaboratory from any device due to its cloud infrastructure in addition to access to GPUs and TPUs to allow for faster development of machine learning models. These features make Colab an optimal workspace to work in when developing systems which require large data processing capabilities.

Amazon Sagemaker - Amazon Sagemaker was a platform released in 2017 to allow for machine learning model development. Similarly to Colab, the platform allows access to GPUs to speed up the machine learning model development process. The Sagemaker platform also allows for rapid labelling of large-scale data and is integrated with Python, more specifically Pytorch and Tensorflow. Additionally,

the models can be hosted on Amazon Web Services for continuous access for further development.

Choice of ML Framework

The next choice made was to determine which ML library to use to train various machine learning models rapidly to test and deploy in the hate network detector. The most popular and used libraries are Tensorflow, Pytorch and K-Train.

Tensorflow - Tensorflow is a DL framework built by Google in November 2015. Its API is quite simple, and with a large developer community, it is easy to obtain support. The training is quite visual meaning it becomes easier to observe what happens when a model is being developed. An issue with the Tensorflow framework is that a large proportion of models on the HuggingFace[31] library are built upon the Pytorch framework resulting in a smaller subset of models not being available whilst using the Tensorflow framework.

Pytorch - Pytorch[27] is a DL framework built in C++ by Facebook's AI research lab in 2015. Advantages of Pytorch over a DL framework like Tensorflow is that it is better for prompt code editing in addition to more detailed documentation being available. However, the main drawback of Pytorch is that it is hard to visualise both models and the development process. This made it a difficult choice when compared to Tensorflow.

K-Train - The K-Train [32] framework was developed in 2020 and acts as a DL library wrapper around Tensorflow Keras. It allows access to a range of readily available models such as GRUs and transformers with ease of access. Additionally, all of the preprocessing that is required in libraries such as Tensorflow and Pytorch is automatically offloaded onto the K-Train library when building a K-Train dataset. K-Train also integrates with HuggingFace to allow for a variety of models to be used when building ML models with the framework. The large downside of using the K-Train framework is that it does not have a large user base however, it does have detailed documentation to allow for fast development and model deployment.

3.2.3 Classification Model Development

The process behind the development of a hate speech classification model can be seen in Figure 3.1. The system displays the choices that a developer has to make through the choice of which data to input followed by which model choice. Additional work has to be done to determine the optimal preprocessing methods with an evaluation of the model versus industry-standard models to be done after training to ensure that the model can successfully detect hate speech correctly.

Dataset Collection and Analysis

Before any model development can take place, firstly a dataset comprised of instances of hate speech and non-hate speech must be collated. Due to the large amount of data required to train performant ML models, it would be illogical to

query Twitter continuously and label every tweet instance manually. Further problems arise from this method as the process would be time-consuming, and there would be inherent bias from the sole developer working on the project in addition to the Twitter API(Tweepy[33]) not having access to tweets that have already been taken down by the platform. Due to these restrictions, the decision was made to use existing datasets from previous research papers and works to develop a machine learning model that could correctly distinguish between hate and non-hate instances.

Text Analysis

After the dataset has been decided, further work related to the actual contents of the dataset should be performed. This included frequency analysis to help perform thorough text preprocessing, for example, filtering out words that provide no meaning. Additionally, the results from the frequency analysis help provide a set of terms to allow the hateful network detector to query Twitter with words commonly found in hate speech phrases.

Text Preprocessing

The text analysis impacts the methods used in text preprocessing. Text preprocessing should be conducted thoroughly to ensure that only the most important features are inputted into the ML models whilst avoiding overprocessing the tweets. Different methods of preprocessing will be applied to the datasets to evaluate the effect of preprocessing on hate speech detection. Text preprocessing undertakes processes such as punctuation removal and user anonymisation such that the tweets inputted to the model are generalised and contain only the key parts of any given tweet as opposed to large portions of text that add no value when considering the classification of hate speech.

Model Choice

After the tweets have been preprocessed using several methods, various models will be trained on the dataset and used to classify hate and non-hate speech instances. Models such as K-NNs, GRUs and transformers will be trained and tested to determine which model performs the best at the task. To account for the various data inputs required for each of these models, further preprocessing must take place to ensure that the data is in the correct format. For example, with a K-NN, the text must be converted to a numeric data point to determine the distance between a test point and its nearest neighbours.

3.2.4 Hateful Network Detector

A system diagram of the hateful network detector is displayed on Figure 3.2. The boxes highlighted in red with bolded text are points within the system which will utilise the hate speech detector to allow further actions to pass.

Choice of Tweet Extraction

Once the model/models have been created, a hateful network can be detected. Firstly, tweets must be extracted from the Twitter platform using phrases and words

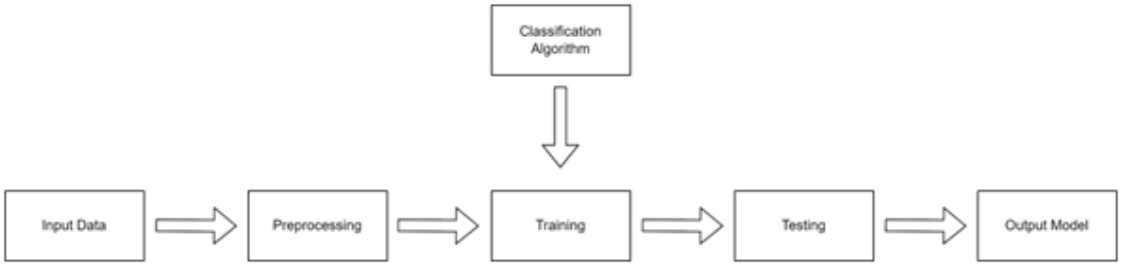


Figure 3.1: Overview of the Hate Speech Detection System Development Process

commonly contained in hate speech. These phrases can be found using methods seen in Subsection 3.2.3. This can be done in one of two ways:

BeautifulSoup - BeautifulSoup[34] is a Python library that can be used to extract HTML and XML data. This makes it useful to use on the Twitter platform to extract text data from Twitter pages. Additionally, BeautifulSoup has a big open source community that has used it so reaching support would be easy to do. The main problem with BeautifulSoup is the ability to query specific phrases. This would need to be manually done to specifically find pages with hate speech contained on them. This would be difficult to do as the proportion of hate users compared to non-hate users is massive.

Tweepy - The next library which would be used is the Tweepy[33] library. This library directly integrates with Twitter allowing queries for specific terms, making it much easier to find instances of hate speech. The main problem with Tweepy is the need for a developer account but this was done near the beginning of the project in the case the Tweepy API was utilised. Additionally, the API is easy to use with clear documentation to allow for tweet extraction to be completed rapidly. Additionally, Tweepy allows user information to be extracted which makes it much easier to find networks of users.

Tweet and Hate User Classification

Each tweet can then be classified after being preprocessed to ensure that the new tweets are as similar as possible to the training data. Each tweet that is flagged as hate speech has its user stored as a preliminary set of hate users. Each of the users in this set has their Twitter account queried for the last ten tweets. Each of these tweets is then classified. A majority rule will then be used to determine whether a user is classified as a hate user. Each of these hate users will then have their friends and followers queried to determine whether there is any sort of hate network present. A "friend" in this context is classified as someone this user follows as opposed to a "follower" who is classified as someone that follows this user.

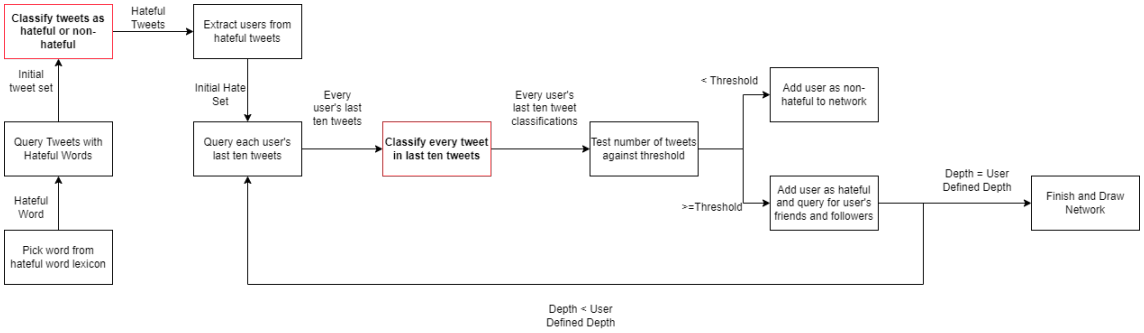


Figure 3.2: Overview of the Hateful Network Detector

Chapter 4

Implementation

The implementation section details how each specific component was built with examples detailed throughout. It outlines each section of the the system, following from Chapter 3, on how every element of the system has been chosen with reasoning behind the various decisions. It details the implementation of each of the system deliverables, the hate speech detection and hate network detection systems, at a more granular level, showcasing how the analysis and design section influenced the end products delivered and how each task was carried out in the same order that Chapter 3 outlined. It showcases how activities such as the text analysis and preprocessing were conducted to built the hate speech and hate network detection systems.

4.1 Classification Model Development

4.1.1 Dataset Collection and Analysis

Firstly, an appropriate dataset/datasets for hate speech classification must be found. There are many publicly available datasets for hate speech detection in addition to the work done prior on detecting hate speech. Table 4.1 displays the various datasets in summary, with the tasks that they were used to perform.

Thomas Davidson's dataset [35] is comprised of 24802 entries of data. It contains tweets that can be used for classification and have been separated into three classes(Hate, Offensive, Neither). This distinction in the dataset is quite useful as profanity is very common on social media without its exclusive use in hate speech. Approximately, 6% of the entire dataset is comprised of hate speech which is quite useful as though hate speech is a large issue, the proportion of hate speech online compared to all general use social media is quite small.

The dataset used in *"Hate Speech Dataset from a White Supremacy Forum"* [36] is quite expansive, comprised of 9916 features with an 11% split of hate-related comments. This paper makes an important distinction between phrases, using three classes to categorise their data (Hate, Relation, Not). The relation class is quite important as this highlights phrases which are not hateful when used by themselves, however, when put into context, the phrase is considered hate speech. This distinction allows these phrases to be categorised in a binary hate speech classifier within

Dataset Name	**Size of Dataset**	**Hate Speech Percentage(%)**	**Dataset Use**	**Research Paper Name**
Thomas Davidson Dataset	24802	6	Multiclass Classification	Automated hate speech detection and the problem of offensive language[35]
White Supremacy Forum Dataset	9916	11	Binary Classification	Hate Speech Dataset from a White Supremacy Forum [36]
Predictive Features for Hate Speech Detection on Twitter Dataset	16914	32	Multiclass Classification	Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter [37]
OLID Dataset	14100	32.9	Binary and Multiclass Classification	Predicting the Type and Target of Offensive Posts in Social Media [38]
Woolwich Murder Dataset	450000	11	Binary Classification	Cyber Hate Speech on Twitter: An Application of Machine Classification and Statistical Modeling for Policy and Decision Making [39]
Human Labeled Corpus Dataset	35000	15.7	Binary Classification	A Large Human-Labeled Corpus for Online Harassment Research [40]
Sharma Roshan Dataset	31962	7	Binary Classification	Twitter-Sentiment-Analysis[41]

Table 4.1: Summary of the various datasets used in hate speech classification

the "Hate" class as overall, they have negative connotations.

The paper *"Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter"* [37] also discusses very important details when looking at dataset features. The dataset used was comprised of 16914 tweets of which 32% of the tweets were classified as hate. This paper looked to classify tweets into three classes, two of which are widely considered to be hate speech(Sexist, Racist, Neither). Important distinctions were made in this paper as the researchers looked for additional information such as geographical information and demographic distribution within the dataset. It was interesting to note that the F1-Score of the model with only gender as an additional feature proved to be the most effective whilst the model trained on the dataset containing gender, location and tweet length resulted in the worst-performing model showing that the model was not learning well from the additional features.

"Predicting the Type and Target of Offensive Posts in Social Media" [38] used a dataset comprised of 14100 tweets to develop various machine learning models which were first able to categorise whether a tweet was offensive, followed by whether a tweet was then targeted. Additionally, the model was also able to distinguish whether an offensive tweet was targeted at an individual or a group. The models developed were of types SVM, BiLSTM, and CNN. What makes a BiLSTM unique in this circumstance is that it can run the input through the LSTM layer in a forward and backward manner which allows the LSTM to characterise tweets using more context than with a normal LSTM. Even with this additional context, the CNN outperformed the other models in offensive language detection and offensive language categorisation but the BiLSTM and CNN performed similarly when identifying whether an offensive tweet was targeted at an individual or a group.

The dataset used in Twitter-Sentiment-Analysis[41] was a composite of 31962 tweets of which 7% were considered hate speech. The model was split into a 75:25 ratio and tokenized after which the encoded dataset was standardised with mean and standard deviation equivalent to the dataset. The training dataset was then fitted with a random forest, a logistic regression model and a decision tree classifier. All 3 models performed with accuracies greater than 90% on the validation datasets but suffered when observing the F1-score with the greatest F1-Score having a performance of 60%.

The datasets that this paper will be utilising are from the Twitter-Sentiment-Analysis [41] Github repository and the dataset utilised in *Automated hate speech detection and the problem of offensive language [35].* These datasets, though imbalanced, seemed to provide a vast range of tweets with a large set of tweets, at a size of 55000 tweets. Additionally, more diverse methods such as the application of transformers and BiGRUs have not been tested on these datasets. The use of these deep learning methods could allow for hidden relationships to be modelled where they could not be formed before.

4.1.2 Text Analysis

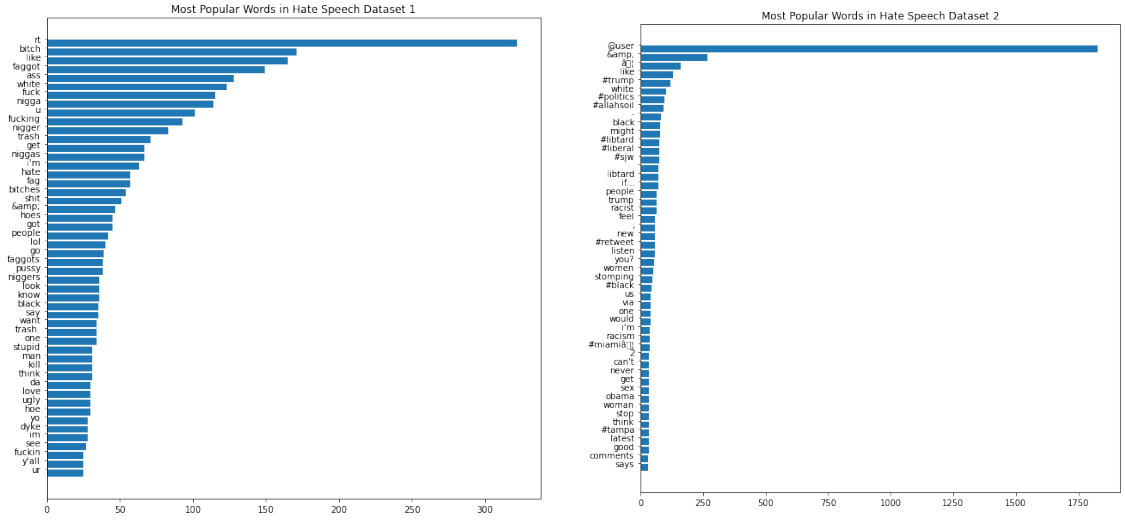
Analysis was conducted on the test and training datasets to observe key insights into specific words that were common in hate speech. This would help provide support for text preprocessing and ensure that no key information is removed when conducting tweet preprocessing. Before conducting the frequency analysis on the data, common words (also known as stop words) are removed using NLTK[42]. This was to ensure that only insightful data remained within the tweets as opposed to words that may clutter the analysis. The frequency of words within the hate speech class with removed stop words can be seen in Figure 4.1. As can be seen in these figures, profanity is common in hate speech in addition to lots of hate speech appearing as a result of a retweet. However, the presence of a retweet, @USER and URL tags within hate speech should not be classified as hate speech as these specific words and phrases do not directly relate to hate speech and are results of the platform. Hence, during preprocessing, two methods will be used to determine the impact of URLs and @USER tags on the classification of hate speech. Furthermore, the models developed will try to be trained on both imbalanced and balanced datasets to determine their impact. The methods are detailed in Table 4.2.

Dataset Type	Methods Applied
1	The two datasets are separately preprocessed and subsequently combined. The entire dataset is split into a train - test split of ratio 90:10.
2	The two datasets are kept separate and preprocessed. The Sharma Roshan dataset is first sampled for hate speech samples. An equivalent amount of non-hate speech samples are extracted randomly and together with the hate speech samples, form the training dataset. The Thomas Davidson dataset is then treated as the test dataset.

Table 4.2: Different Dataset Type Explanations

4.1.3 Text Preprocessing

The text from the dataset must be preprocessed such that it can provide meaningful context to the transformer model. Most punctuation in the dataset is removed including speech marks, apostrophes and colons. Certain punctuation is still kept in the text, with periods, commas, question marks and exclamation marks kept within the text. This set of punctuation helps keep context and the grammatical structure within literary pieces. The question mark and exclamation mark are relevant due to how they are used in addition to helping the transformer gain context of a specific phrase. In the paper, *"How does Punctuation Affect Neural Models in Natural Language Inference"* [43], experiment 2 demonstrated how the removal of non-meaningful punctuation does not affect the classification accuracy of BERT models. In the case where non-transformer models were being trained, the punctuation would have to be kept within the pieces of text to ensure that it does not affect



(a) Word Frequency for words in dataset 1

(b) Word Frequency for words in dataset 2

Figure 4.1: Frequency analysis of the two datasets

the classification accuracy.

Multiple instances of white space were also removed and replaced by single spaces as they convey no additional meaning through their repeated occurrence. Hyperlinks were wither removed as the transformer model would not be able to express any meaning from a hyperlink as it cannot access the data source and would not be able to derive any meaning from it or tokenized to a URL tag to observe the impact of the tag contained in the sentence. Capitalisation also shows no effect on the transformer models as it does not change the meaning of any given sentence. Hence, the texts provided to the models were preprocessed such that every letter is set to lower case. Additionally, new lines from the text were also removed as they do not provide any meaningful content that can be learned by the transformer. Emojis are kept within the text following the paper *"Twitter Sentiment Analysis with Emojis"*[44] as they can provide greater performance benefits in terms of precision and recall as denoted by the F1-Score metric denoted in the paper.

Additionally, as the text that the model is trained on is a set of tweets, the data naturally contains hashtags and retweets. The hashtags generally contain subjects that are relevant to the content of the tweet. This may allow the model to gain a more complete view of the topic that is being portrayed hence they are kept in the tweet. The retweet function of a tweet allows a user to essentially target the nature of their content to a person or group of people. In this case, the specific user was replaced by a token. This would allow the transform to maintain some context in the case where hate speech is directed at a specific user, however, it would also allow for the model to maintain the anonymity of the users within the dataset. Figure 4.2 gives an example of the total preprocessing from a tweet to data that is input into the transformer model.

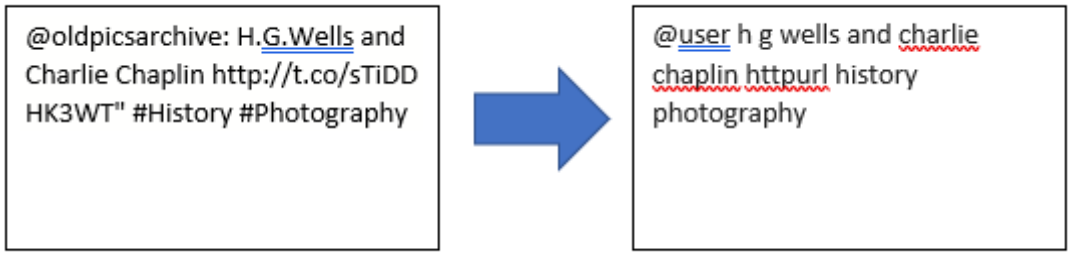


Figure 4.2: An example of the preprocessing of text before it is inputted in to an ML model

4.1.4 Model Choice

Once the dataset has been preprocessed, some data still has to be altered in terms of the classes that the dataset evaluates. The Thomas Davidson dataset involves three classes (Hate Speech, Offensive Language and Neither). This is further subdivided to allow for binary classification whereby all elements in the "offensive language" and "neither" classes are changed to the negative class whilst only samples in the "hate speech" class are kept as the positive class. This decision was made to allow the ML models to learn the context of offensive language in a non-hate speech setting as it becomes increasingly popular on social media. Additionally, the offensive language class is not necessarily an instance of hate speech, further contributing to the decision to merge the "offensive language" class with the "neither" class.

The dataset is now prepared to be used in training various models so different classifier architectures are experimented with to observe which model is most suited and able to discriminate between instances of hate speech and non-hate speech. The first model to be implemented was the K-Nearest Neighbours algorithm to see whether encoding phrases via frequency could distinguish hate speech to a great degree.

K-Nearest Neighbours

The K-Nearest Neighbours algorithm was the first approach taken to classify hate speech. Every tweet in both the test and training datasets was encoded using Scikit-Learn's[45] `LabelEncoder` encoder to provide unique IDs for each tweet. This allowed each sentence to be represented as a singular number.

The K-NN algorithm was also part of Sci-Kit-Learn's native library. This allowed different distancing metrics to be used, particularly the "uniform" and "distance" based metrics. The uniform metric means each neighbour within the dataset is weighted equally whilst the distance metric ensures that each data point is weighted dependent on its distance from the test point. Additionally, the text was preprocessed separately to determine the effect on text classification. The preprocessing was varied to see the effect of the inclusion of URL and @USER tags within the dataset as these tags have high frequency but should bear no significance on the classification of hate speech. The K-NN was tested on 4 different data configurations as seen in Table 4.3.

Configuration	Distancing	Preprocessing Type
1	Uniform	Included @USER and URL tags
2	Uniform	Removed @USER and URL tags
3	Distance	Included @USER and URL tags
4	Distance	Removed @USER and URL tags

Table 4.3: Different Data Preprocessing methods used in the development of the K-NN models

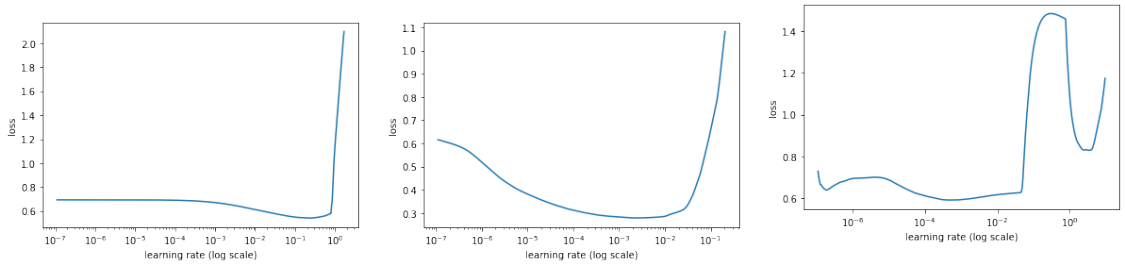
Deep Learning Methods

The following models were trained on various configurations as can be seen in Table D.1. This was to determine the effect of various preprocessing methods and changes in the training conditions that allowed for better results when distinguishing hate and non-hate speech instances. Due to the high frequency of user tags and URLs, whether or not these tags were included formed part of the configuration. Additionally, having non-equal class weights in both equal and non-equal datasets allows the observation of whether hate speech can be detected at a higher accuracy to be made at the cost of misclassifying non-hate speech. The datasets were also changed in the same manner as when using the K-Nearest Neighbours algorithm to observe the impact of equivalent sets of hate speech and non-hate speech on a given model's performance. The impact of lemmatization was also analysed to see whether applying a morphological analysis to a tweet allows a model to extract more information from a tweet. Lemmatization was used over stemming as stemming only considers the root of the word whilst lemmatization allows the context of a word in a given phrase to be used. Additionally, the NBSVM and BiGRU models were trained up to 50 epochs whilst the transformer models(BERT and DistilBERT) were trained to 2 epochs. As the transformer models were pre-trained on the English vocabulary for a large number of epochs, increasing the number of epochs that transformers were trained on does not result in better classification results. This was further justified in the initial investigation with the use of Tensorflow where there was no change in evaluation metrics once the transformer models trained past 20 epochs.

The initial models that were developed were the transformer models BERT and DistilBERT. They were developed using the Tensorflow framework. This meant that most of the preprocessing for the model required separate tokenizers to be created for transformer models. Additionally, it became much more difficult to make training models more streamlined as the models had various needs for preprocessing, for example, the BERT and DistilBERT models required the use of the `AutoTokenizer` function to tokenize input tweets into sequences of numbers. This same tokenizer could not be used for BiGRU and NBSVM implementations. The models also performed exceptionally poorly. A similar implementation also was created in Pytorch with similar performance. Hence, further development was done on the K-Train wrapper. This meant that the preprocessing was offloaded to the K-Train library when preparing the dataset as opposed to having separate objects dealing with the preprocessing. Additionally, K-Train allowed the NBSVM and BiGRU models to

be built using the existing code as opposed to having to develop new models separately. This made continuing development with K-Train much easier as opposed to developing with Tensorflow and Pytorch.

Overall, a total of 64 models were developed with varying changes made as can be seen in Table D.1. Each model was trained with a specific learning rate. This was determined using K-Train's `lr_find` function to determine the optimal learning rate. This produces a graph where the learning rate is determined on the steepest negative gradient that can be found in the graph within the graph. Examples of these graphs can be seen in Figure 4.3. Every model was trained with an Adam optimizer and using a triangular learning rate policy as detailed in *Cyclical Learning Rates for Training Neural Networks* [46]. The Adam optimizer was chosen as it is can handle sparse gradients on very noisy datasets in addition to its effectiveness with various hyperparameters on different ML models. A triangular learning rate policy allows models to train much faster and achieve high accuracy values with no additional cost.



(a) Optimal Learning Rate Finder for BiGRU

(b) Optimal Learning Rate Finder for BERT

(c) Optimal Learning Rate Finder for DistilBERT with Class Weights

Figure 4.3: Graphs used to determine learning rates for different models

Each model was then evaluated against its respective test set to determine its performance metrics (Accuracy, Precision, Hate Speech Accuracy(Recall), Non-Hate Speech Accuracy, F1-Score, Performance). The performance metric was calculated as the average of the hate speech accuracy and non-hate speech accuracy. This metric was then used to determine the best model. This metric was developed as the desired model should perform well on both hate and non-hate speech instances. This cannot be captured by standardised metrics when an imbalanced test set is utilised. The results of every model are detailed in Appendix D. These results will be evaluated in Chapter 5.

4.1.5 Hate Speech Detection System Diagram

The system diagram for the overall development of the hate speech detection system can be seen in Figure 4.4. The diagram showcases the overall flow of how each of the ML models will be trained. Every rectangle within the diagram showcases a data structure whilst the rounded rectangles display a module or action taking place. Each part of the system is also segmented into the goal of the system through the coloured key denoted on the right side of the diagram. Variables being split up ,such as the test and training data partitions, are shown using smaller arrows to

denote the various locations that act as their endpoint. The diagram showcases the original data being transformed into preprocessed data. This is further followed by the data being broken into a training and testing split. A validation set was not required as the models to be trained had already been decided. Additionally, the lack of hate speech samples meant that the models developed should be exposed to as many samples of hate speech and non-hate speech samples as possible. The model is then trained using the triangular learning rate policy and evaluated against the test variables.

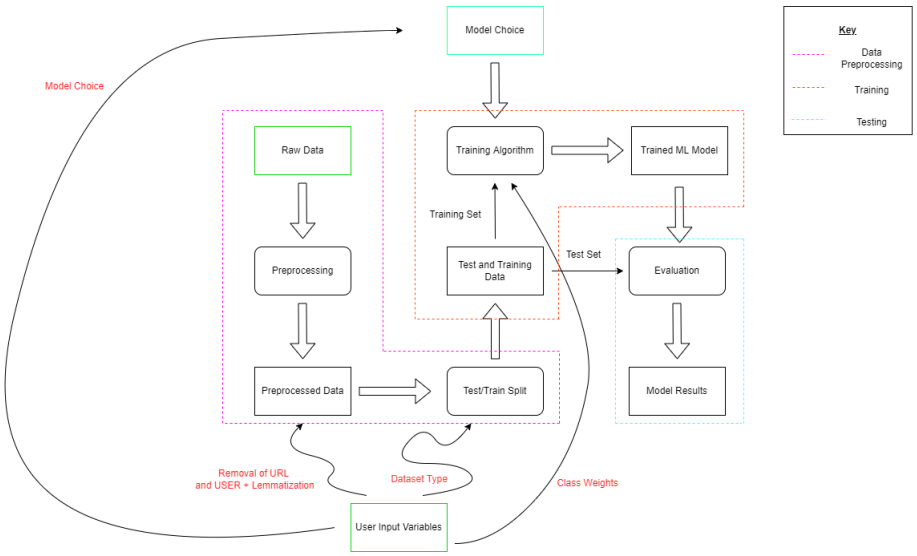


Figure 4.4: ML Development System Diagram

The main choices that a user can make are with the dataset choices and the user state variables. The diagram begins with 2 initial variables, the raw data and the user input data. The raw data consists of a dataset consisting of a set of tweets with each tweet's respective label as to whether it is hateful or not. The user input variables dictate 5 different states within the hate speech detection system diagram. The first choice is the choice of which model is being developed. The choices include BiGRUs, NBSVMs and transformer models. The next 2 state variables are the "removal of URL and @USER" variable and the "Lemmatization" variable. These variables dictate partial segments of the preprocessing, specifically whether the URL and @USER tags will be removed and whether each word in the tweet would be lemmatized. The following input variable determines the dataset type, whether the dataset is smaller and balanced or whether the dataset is larger but unbalanced. The final variable impacts the training process by determining the learning rate in addition to whether the model is trained using uneven class weights due to the imbalanced class distribution within the original composite datasets.

4.2 Hate Network Detection

4.2.1 Tweet Extraction

Tweets were extracted from the Twitter platform using the Tweepy API. Using the results of Section 3.2.3, the most common words within hate speech seem to be profanity and words associated with racist remarks. These words and phrases were put into a list. From this list, a random word would be chosen and queried on Twitter. This would return tweets containing the chosen word from the past seven days and ensured that the system would not query inactive users.

4.2.2 Tweet and Hate User Classification

The best models used in Section 4.1 were then used to develop the hate speech network detector. Each of the tweets from Subsection 4.2.1 is first preprocessed to ensure that the data is as close to the training data as possible in terms of format. The models are then used to classify the initial set of users. These models used a majority rule such that a consensus is made as to whether a tweet is considered hate speech. This ensured that the overall accuracy of the system would increase as the different models may have picked up different characteristics to determine whether a tweet is classified as hate. This helps form a primary set of users that could be possible sets of hate users. Each user that has an initial tweet classified as hate is separated into a new list.

Each member of this new list has their last ten tweets queried. Each of these tweets is further preprocessed and classified. Further investigations were done to determine a threshold as to whether a user would be considered a hate user. The threshold was defined as the minimum number of tweets that a user has produced in the last ten tweets that are hateful. If a user has a number greater than this given threshold, their friends and followers are looked up using the Tweepy API. The friends and followers have their last ten tweets preprocessed, classified and tested against the threshold. The process then continues up to a defined depth. The overall algorithm is described in Figure 4.5 as Python pseudo-code. The algorithm denotes the two initial variables, `depth` and `threshold` which denote the number of friend and follower queries made in addition to the threshold at which a user is determined to be a hate user respectively. Each tweet is preprocessed before being input to the classification function which utilises the machine learning models built for the hate speech detector. The algorithm continues from 1 to `depth` denoting the size of the network.

4.2.3 Hate Network Detection System Diagram

Figure B.1 showcases the overall system diagram. However, the image can be broken down into two main segments which are shown in Figures 4.6 and 4.7. These showcase the implementation of the system in two different parts. The first part showcases the use of the word lexicon. A word would be queried using Tweepy which provides a list of tweets containing the query word. Each of these tweets is prepro-

```
depth = d
threshold = t
for z in range(1 , depth):
    for i in hate_user_stats:
      hate_user_friends = i[1]
      hate_user_followers = i[2]
      for j in hate_user_friends:
          lastTenTweets = queryLastTenTweets(user_id = j)
          y_pred = 0
          for k in lastTenTweets:
            k = preprocess(k)
            y_pred += classify(k)
          if y_pred >= threshold:
            new_hate_users.append(j)

      for j in hate_user_followers:
          lastTenTweets = queryLastTenTweets(user_id = j)
          y_pred = 0
          for k in lastTenTweets:
            k = preprocess(k)
            y_pred+= classify(k)
          if y_pred >= threshold:
            new_hate_users.append(j)
    hate_user_stats = get_friends_and_followers(new_hate_users)
```

Figure 4.5: Hate Network Detector Algorithm

cessed and classified which provides a list of classifications as the list of tweets is passed through the ML model/models developed. The list of classifications is cross-correlated with the tweet list to extract the set of users.

The next part of the flow diagram runs until a user-defined depth, `d` , to generate a hate network from the set of users. Each of the users from the hate set has their last ten tweets queried. Each of these tweets is further preprocessed and classified, in the same fashion the initial set of tweets was but at a larger scale as ten tweets are utilised per user as opposed to the original one. This difference was made as the initial set of users works as an initial survey of users whereas the current classification needs more in-depth classification to determine whether a user is determined as a "hateful user". This provides a list of ten classifications per user which is then compared against a threshold, `t`. If there are $\geq$ `t` classifications that count as hateful tweets, the user who created these ten tweets is categorised as hateful. This hateful user has their friends and followers queried to determine further branches of the network. As the diagram shows, the friends and followers have their last ten tweets preprocessed, classified and evaluated against `t`. This continues up till `d` where the network stops growing.

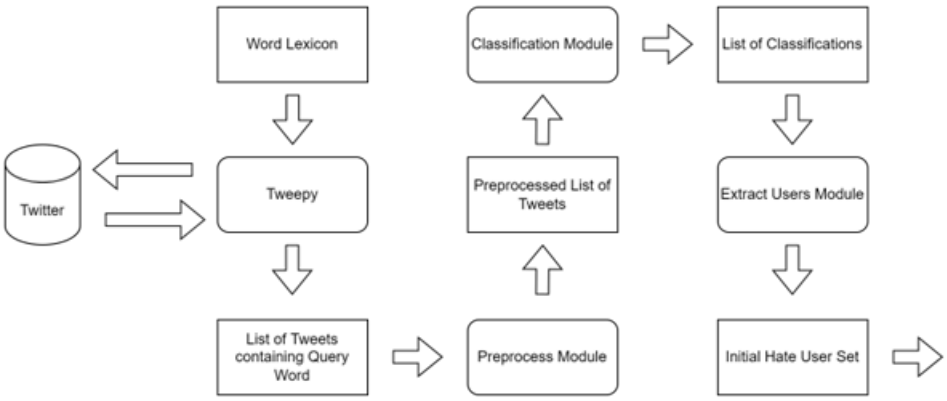


Figure 4.6: First Part of Hate Network System Diagram

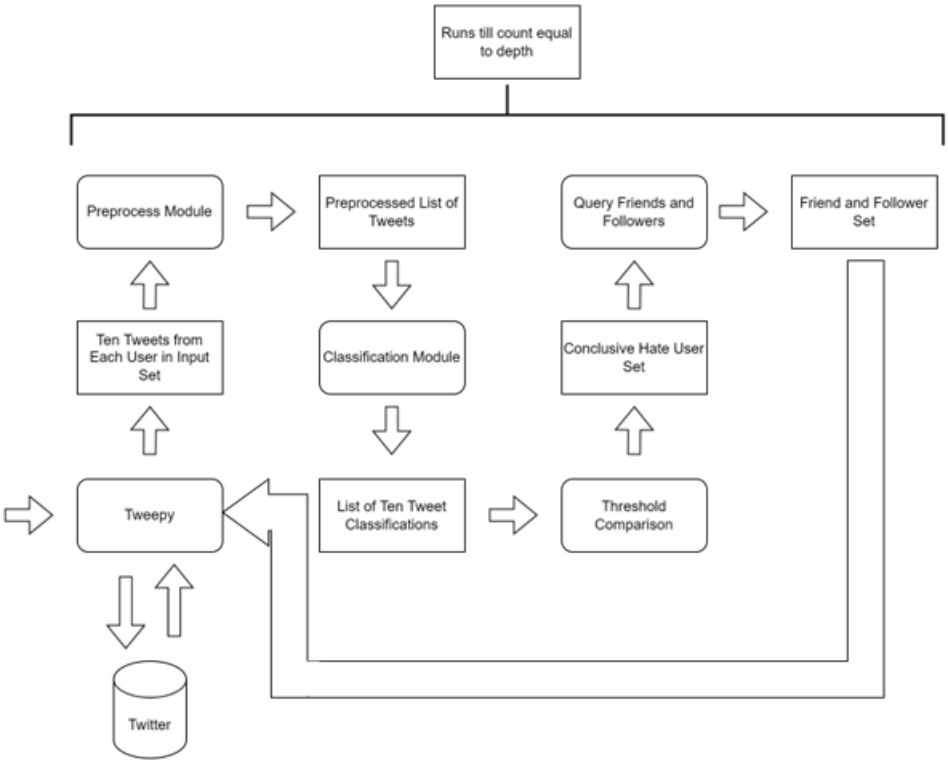


Figure 4.7: Second Part of Hate Network System Diagram

Chapter 5

Evaluation

5.1 Hate Speech Detection

The evaluation section details how each sub-part of the project, the hate speech detection system and hate network detection system, were tested and evaluated. It also looks into the various configurations of data preprocessing that were implemented in addition to further methodologies that were implemented but were not successful. Evaluation of the hate speech detection system were conducted at a model and final hate speech detection system level, further comparing the utilised system to state-of-the-art models whilst the hate network detection system has its thresholding mechanism and functionality tested individually.

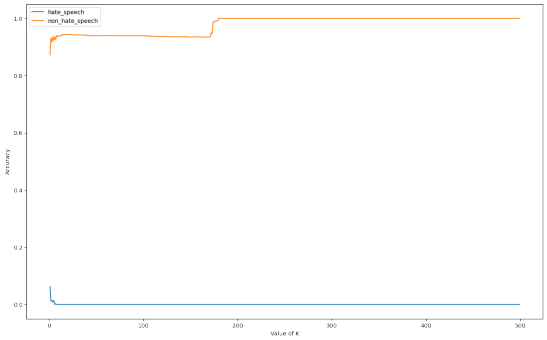
5.1.1 K-Nearest Neighbours

The K-NNs were developed with two different preprocessing methods. The first method involved the removal of the @USER and URL tokens as they had a high frequency within both the test and training datasets as can be seen in Section 3.2.3. The K-NN's used various distance metrics to determine the class of a specific test sample. The two distancing methods used were uniform and distance-based weighting. Overall, 8 different combinations of methods were tested for the K-NN algorithms. These configurations can be seen in Table 5.1.

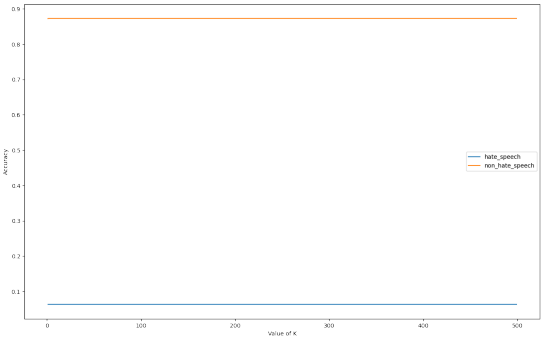
Configuration	Removal of @USER and URLs	Dataset Type	Distancing
1	Yes	1	Uniform
2	Yes	1	Distance
3	Yes	2	Uniform
4	Yes	2	Distance
5	No	1	Uniform
6	No	1	Distance
7	No	2	Uniform
8	No	2	Distance

Table 5.1: Configurations for Different K-NN experiments

Figure 5.1 shows the results of configurations 1 and 2. The mean accuracy values for the non-hate speech class and hate speech class were 87% and 6% respectively when using distance-based weighting whilst the accuracy values were 98% and 0.02% respectively for uniform weighting. These results are due to the lack of samples of hate speech within the total dataset. The total dataset is comprised of 6.1% of hate speech, therefore a major class imbalance results in less varied results.



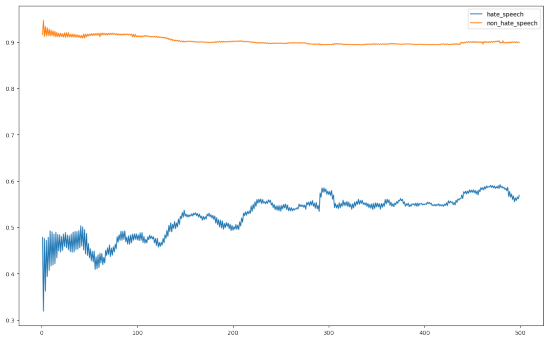
(a) Graph of Accuracy vs K using a K-NN with Uniform Neighbour Weighting with dataset type 1



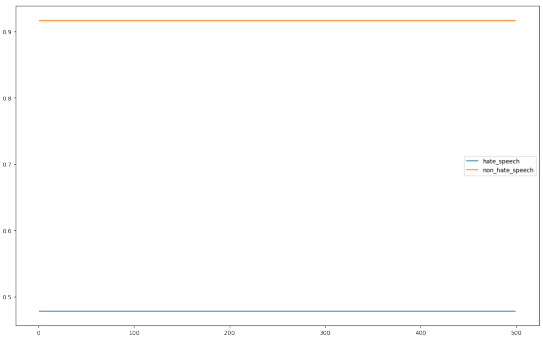
(b) Graph of Accuracy vs K using a K-NN with Distance Neighbour Weighting with the dataset type 1

Figure 5.1: Results of Accuracy vs K with two different weightings for dataset type 1 with removal of URL and @USER tokens

The next set of results from configurations 3 and 4 can be seen in Figure 5.2. The K-NN implementation with uniform distancing was quite effective with mean accuracy values of 52% and 90% for the hate speech and non-hate speech classes respectively. Using distance-based weighting, the model performed with mean accuracy values of 48% and 92% respectively for the hate speech and non-hate speech classes.



(a) Graph of Accuracy vs K using a K-NN with Uniform Neighbour Weighting with dataset type 2



(b) Graph of Accuracy vs K using a K-NN with Distance Neighbour Weighting with dataset type 2

Figure 5.2: Results of Accuracy vs K with two different weightings for dataset type 2 with removal of URL and @USER tokens

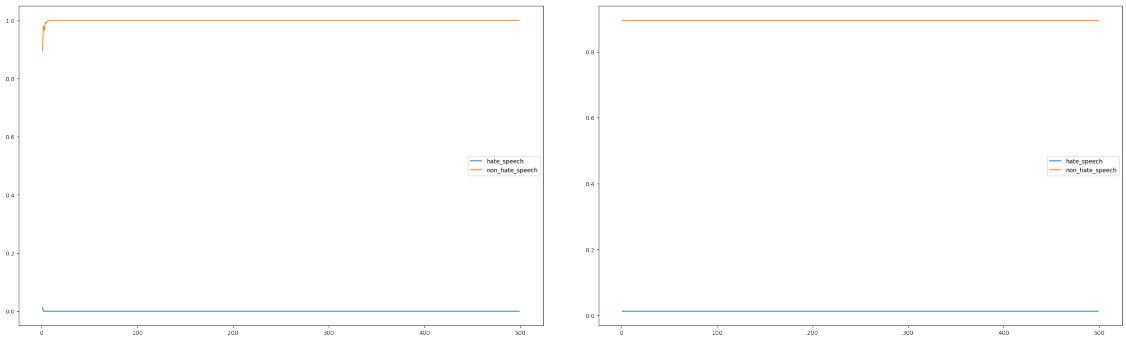
The following set of configurations all involved the inclusion of the URL and @USER tags to observe the effect of the inclusion of these tags. These results can

Configuration	Average Hate Speech Accuracy(%)	Average Non-Hate Speech Accuracy(%)
5	0.02	100
6	1.3	90
7	96	92
8	62	91

Table 5.2: Average Hate and Non-Hate Classification Accuracies configurations using URL and USER tokens

be observed in Figures 5.3 and 5.4. The second preprocessing method seemed to be much more effective at classifying hate speech but seems to be extracting the URL and @USER tokens as instances of hate speech. The K-NN that performed the best at classifying non-hate speech was trained using configuration 5. However, this model was completely inaccurate, at classifying hate speech, standing at a 0.02% classification rate. This was due to the large class imbalance within the dataset as only 6% contains hate speech meaning there is not much lexical diversity within the hate speech when compared to instances of the non-hate speech class. Table 5.2 shows the results of the remaining KNNs when using the second preprocessing method. The models all performed well at the classification of non-hate speech but only two models had a classification rate of above 50%.

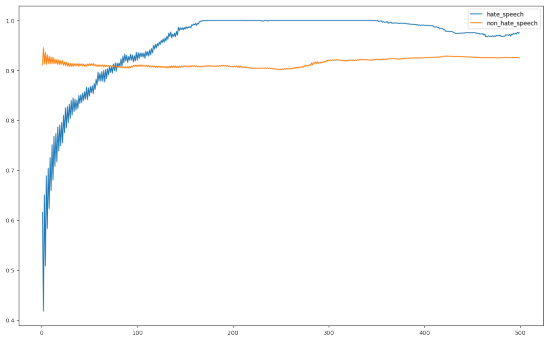
Through these investigations, the impact of K on the classification of hate speech was also investigated. The figures demonstrate that the accuracy of the hate speech class either stabilises or increases as the value of K increases. K-NNs that used a distance-based neighbour classification had no increase in the accuracy when larger values of K were used. The value of K seemed to have the biggest impact on configurations 3 and 7 with the accuracy values for both classes displaying fluctuation as K increased from 1 to 500. It seemed to have the largest impact on configuration 7 which is the only configuration to display a higher hate accuracy than non-hate accuracy at the intersection point of K $\approx$ 80.



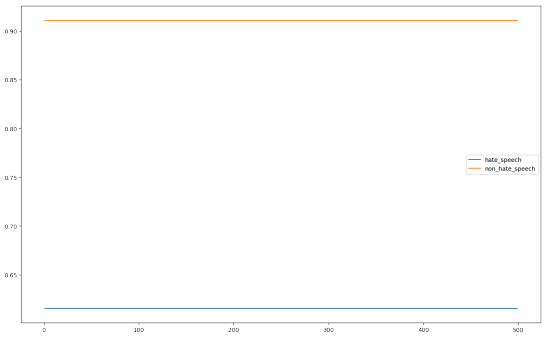
(a) Graph of Accuracy vs K using a K-NN with Uniform Neighbour Weighting with the dataset type 1

(b) Graph of Accuracy vs K using a K-NN with Distance Neighbour Weighting with dataset type 1

Figure 5.3: Results of Accuracy vs K with two different weightings for dataset type 1 without removal of URL and @USER tokens



(a) Graph of Accuracy vs K using a K-NN with Uniform Neighbour Weighting with dataset type 2



(b) Graph of Accuracy vs K using a K-NN with Distance Neighbour Weighting with dataset type 2

Figure 5.4: Results of Accuracy vs K with two different weightings with dataset type 2 without removal of URL and @USER tokens

5.1.2 Deep Learning Methods

This section denotes key findings within the classification of hate speech from Tables D.2 to D.21. Each of these tables was produced from the use of BERT, NBSVM, BiGRU and DistilBERT models trained at various epochs using the configurations detailed in Table D.1. The tables provide the accuracy, precision, recall, non-hate speech accuracy, F1-Score and performance of any given model at a specific epoch. The metrics that were used to determine the efficacy of a model were the recall, non-hate speech accuracy and performance. Accuracy and precision are not used as primary factors when evaluating the models as they are heavily reliant on the test set being balanced however, both test sets used consisted of imbalanced test sets. Hence, the individual class-based accuracies would serve as a better metric with performance being calculated as the average of the class-based accuracies.

BERT

The BERT model was trained over 2 epochs for each of the various configurations. All of the BERT models performed poorly when observing the precision metric. This is due to the class imbalance within the testing datasets. The models performed similarly no matter how many epochs they were trained for. Configurations 14 and 15 performed particularly poorly when classifying non-hate speech as can be seen in Tables D.2 and D.3. Models that performed well at the accuracy metric were generally overfitting for the non-hate speech class which meant that they were unable to be deployed when classifying tweets on the Twitter platform. The BERT models that performed the best at classifying both hate and non-hate speech were configurations 4, 5, 12 and 13. These four models classified both classes at an accuracy of $>80\%$. As a result, these models performed well in the performance metric when compared to other models. Since these 4 models performed the greatest, it suggests that the models were able to classify hate speech to the same degree regardless of the inclusion of the URL and USER tags.

BiGRU

The BiGRU performed similarly to the BERT models across all configurations however, it was trained for a larger set of epochs. The configurations that were the most performant with the BERT model were also the most performant with the BiGRU instances. However, in these configurations, the models became much worse at classifying hate speech instances after being trained for more epochs. This seems to be due to the number of non-hate speech samples in these configurations as these models were trained on imbalanced datasets with imbalanced class weightings. Even though the class weight was proportional to the imbalance within the dataset, the models tended to classify non-hate speech samples at a higher accuracy even if it meant the model misclassified a larger proportion of hate speech samples. After training the BiGRU model for larger sets of epochs, configurations 14 and 15, which overfit towards the hate speech class, tended to surrender a high hate speech accuracy to begin classifying larger proportions of non-hate speech class samples.

DistilBERT

Similarly to the BERT model, the DistilBERT models were only trained for 2 epochs with the same learning rates. Over the 16 configurations, this model performed near the same across all metrics when being evaluated. This is impressive considering how much smaller the network is compared to the BERT network, containing 40% fewer parameters. This makes DistilBERT a much more ideal choice to train, especially as it is 60% than its predecessor. It also took much less time to train DistilBERT models compared to BERT models with each epoch taking 17 minutes as opposed to a BERT model take approximnately 30 minutes to train each epoch. In the same fashion as both the BiGRU and BERT models, the best configurations to classify both hate speech and non-hate speech instances were configurations 4, 5, 12 and 13. This implies that the best method to train an ML model to classify both hate and non-hate speech instances is the use of an imbalanced dataset with class weightings, regardless of the inclusion of the URL tags and @USER tags within a given set.

NBSVM

The NBSVM models performed the same across all configurations as well. This leads to the conclusion that the configuration has a larger impact as opposed to the models. Similarly to the BiGRU, the NBSVM models that are trained on the optimal configurations tend to bias toward classifying the non-hate speech class at a higher accuracy even when the trade-off results in lowering the accuracy of the classifying the hate speech class. For this reason, the final models used were generally trained on fewer epochs to ensure high accuracy in both classes.

Summary

Overall, all the models with optimal configurations performed well when training for fewer epochs. Across all the configurations, the optimal configurations were 4, 5, 12 and 13. Overall the models performed worse as the number of epochs increased. However, models that were overfitted towards one specific class, for example, configuration 15 across all models, began classify the non-overfitted class more correctly as the epochs increased. This indicates that overfitted models can begin learning

features within non-overfitted classes if given enough epochs at the cost of the over-fitted class. As a result of multiple models performing so well, an ensemble of the models was used. This both ensured a high accuracy and stability in the case of classifying hate speech. Five different models were used in the ensemble to allow a majority ruling. Each model was given equivalent weighting within the ensemble as they all performed similarly. The models that were picked are shown in Table 5.3.

Model Number	Configuration	Model Type	No of Epochs
1	12	BERT	1
2	13	BERT	1
3	13	NBSVM	1
4	13	BiGRU	1
5	13	DistilBERT	1

Table 5.3: Models used in Ensemble Architecture

K-NNs were not used in the ensemble architecture due to their over-reliance on the frequency of URL and @USER tags to attain higher accuracy values. Additionally, these tags have no meaning in the context of classifying hate speech as can be seen from previous results with the deep learning architectures. To ensure the ensemble performs well, each model and the ensemble were tested on another sampled test set. The results from this test can be seen in Table 5.4.

Model Number	Hate Speech Accuracy(%)	Non Hate Speech Accuracy(%)	Performance
1	82.0	88.4	85.2
2	81.1	89.2	85.2
3	96.4	95.6	96.0
4	95.4	95.9	95.6
5	84.7	84.9	84.8
Ensemble	87.4	90.9	89.6

Table 5.4: Accuracy and Performance values for individual models and ensemble models

The ensemble is further tested against the dataset found in the paper *"HateCheck: Functional Tests for Hate Speech Detection Models"*[47]. The ensemble was not trained on the data provided in the HateCheck paper due to time constraints though parts of the training sets with some of the models overlapped. The dataset was comprised of 3728 samples with 2563 hate speech samples and 1165 non-hate speech samples. This paper also compared several models against different types of hate speech, even classifying them by category. However, the ensemble will only be compared to the models in the paper by the ability to differentiate between hate and non-hate speech instances. The results can be seen in Table 5.5. Overall, the ensemble performs similarly to the BERT models developed in the paper, with the ensemble performing slightly better with the performance metric when compared to the B-D and SN models overall. It is, however, good to note that the SN model

performed much better at the classification of non-hate speech samples than all the other models. In the context of the purpose of this project, the P model performed the best when compared to all other models, attaining a 76.6% accuracy and a performance of 68.9. Overall, the ensemble developed in this paper does very well considering the data in the HateCheck dataset is not in the format of a tweet.

Model Name	Hate Speech Accuracy(%)	Non Hate Speech Accuracy(%)	Accuracy(%)	Performance
Ensemble	87.4	25.9	68.2	56.7
B-D	75.5	36.0	63.2	55.8
B-F	65.5	48.5	60.2	57.0
P	89.5	48.2	76.6	68.9
SN	9.0	86.6	33.2	47.8

Table 5.5: Models Comparison on HateCheck Dataset

5.1.3 Failed Attempts

Data Augmentation

To combat the effect of the class imbalance within the test and training datasets, data augmentation was used. The Python library that was used for the text augmentation was Augly[48], a data augmentation library. This was to increase the size of the training data and allow the models to be able to collate more information about the difference between hate speech and non-hate speech instances. The first model that was tested on this data was an NBSVM. The NBSVM was trained over 50 epochs with a learning rate of 0.01. This learning rate was determined in the same fashion as the learning rates for deep learning models in Subsection 4.1.4. These results can be seen in Table 5.6.

Epochs	Hate Speech Accuracy(%)	Non Hate Speech Accuracy(%)	Performance
1	41.0	80.0	60.5
2	38.0	81.0	59.5
5	38.0	82.0	60.0
10	38.0	83.0	60.5
20	39.0	82.0	60.5
30	39.0	82.0	60.5
40	40.0	82.0	61.0
50	40.0	81.0	60.5

Table 5.6: NBSVM results for augmented data

To ensure that this was not only the case for the NBSVM, a BiGRU model was also developed and trained over 50 epochs. The learning rate used was 0.0006. As can be seen in both Table 5.6 and 5.7, augmenting the data did not assist in classifying hate speech. This is probably because the data is unable to learn any diversity from the data as the augmentation is only able to simulate typos. Current data augmentation techniques would not be able to assist in the classification of hate speech. With the ability to just simulate typos, smarter data augmentation

would be required to replace particular words with synonyms and in the given input context. As the performance of both the NBSVM and BiGRU was poor, further testing was not conducted on the BERT and DistilBERT models as they seemed to show similar classification results when using other methods.

Epochs	Hate Speech Accuracy(%)	Non Hate Speech Accuracy(%)	Performance
1	58.0	63.0	60.5
2	56.0	64.0	60.0
5	51.0	69.0	60.0
10	56.0	63.0	59.5
20	52.0	66.0	59.0
30	50.0	64.0	57.0
40	55.0	63.0	59.0
50	56.0	61.0	58.5

Table 5.7: BiGRU results for augmented data

Adaptive Class Weighting

To improve the accuracy of each class, class weights were adapted per epoch dependent on the accuracy of both the non-hate speech and hate speech classes. Initially, due to the imbalance of class weights, the hate speech class was weighted at approximately 16 times the weight of the non-hate speech class as the ratio of hate to non-hate speech instances was 1:16. For the first epoch, the model was trained on the initial class weights. From then onwards, the class weights would be adapted such that the class weight of a given class was proportional to the class accuracy of the other class. As this was binary classification, this would hopefully ensure that both classes would increase in accuracy as the weights were adapted. The model that was trained was a BiGRU model with a learning rate of 0.002. The results of the investigation can be observed in 5.8.

Epochs	Hate Speech Accuracy(%)	Non Hate Speech Accuracy(%)
1	78.0	91.0
2	49.0	99.0
3	51.0	98.0
4	52.0	98.0
5	49.0	98.0

Table 5.8: BiGRU results for adaptive class weighting

The results show that the hypothesis did not work as intended whereby the accuracy of each class would steadily increase over a given number of epochs. After the first epoch, the accuracy of the hate speech class diminished significantly, from 78% to 49%, with a raise in the accuracy of the non-hate speech class instances. Thereafter, there were only minor fluctuations in the accuracy of both class instances. This meant that this model could not be utilised in the final ensemble as the model

needs to be able to distinguish both classes with high accuracy as opposed to models that were only able to classify one class extremely well. The model tended to overfit towards the non-hate speech class instance as the proportional class weights did not seem to make the overall system perform well.

5.2 Hate Network Detection

5.2.1 Thresholding

After the model had been developed, further work had been done to find hate networks on Twitter. An initial optimal threshold had to be found to determine whether a user should be classified as a hate user. Every word from the query list was queried on Twitter. Each of the tweets was then classified to reduce the set of users. This led to a set of 63 users that had all posted at least one tweet classified as hate speech. Each user had their last ten tweets run through the ensemble of ML models and classified. Testing was done to find an optimal threshold to determine whether a user would be classified as hateful by varying the thresholding amount. The threshold varied from 1 to 10 tweets to determine whether a user was hateful. Figure 5.5 displays the number of users classified as "hate users" as the threshold varies from 1 to 10.

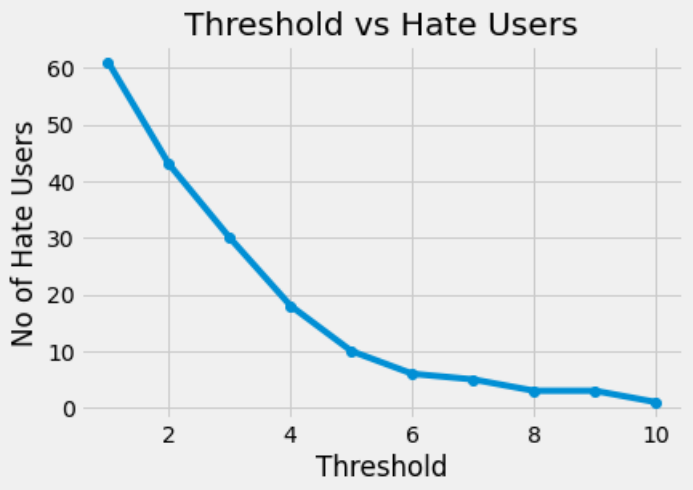


Figure 5.5: Thresholding Value vs Number of Hate Users

The graph displays an exponential decay as the threshold value increases similar to an elbow plot. The decreasing behaviour was expected as the requirement for a user to be classified becomes more restrictive as the number of tweets required to be classified as a hate user increased. The magnitude of the graph's gradient decreases until the threshold value equals 5. It tends to stabilise and finds a minimum thereafter. The value of 7 was determined to be an optimal threshold value for the investigation in this paper. This would give an inherent advantage when detecting hate networks as the most impactful hate users would be determined to have >7 hateful tweets. Further work can be done to implement a decaying threshold function that decreases the threshold as the depth of the network being evaluated

increases.

5.2.2 Detecting Hate Networks

Initially, two users were detected to be hateful using the threshold value of 7. Each of the users had their entire set of followers and friends added to a user list. Each of the members within the user list had their most recent ten tweets queried and classified by the ensemble with the same thresholding value being used. This is known as a depth 1 network as only the initial two users that were found to be hateful had their friends and followers queried. This resulted in Figure 5.6. This network is quite expansive as a result of the network containing >2000 users. The lines on the image show the connections between two users. Red lines denote connections between two hateful users in this network. It is important to note that a grey connection does not ensure that a user has posted only non-hateful content. It only ensures that a user may post hateful content but in the last ten tweets, they produced fewer than 7 hateful tweets. As previously mentioned, having a decaying thresholding function for a larger depth network would result in clustered red nodes and connections towards the centroid positions within the network as the number of red nodes and connections grows more sparsely as the network grows. Further work could be done to observe whether two hate networks begin to connect through a mutual connection and at what depth this occurs.

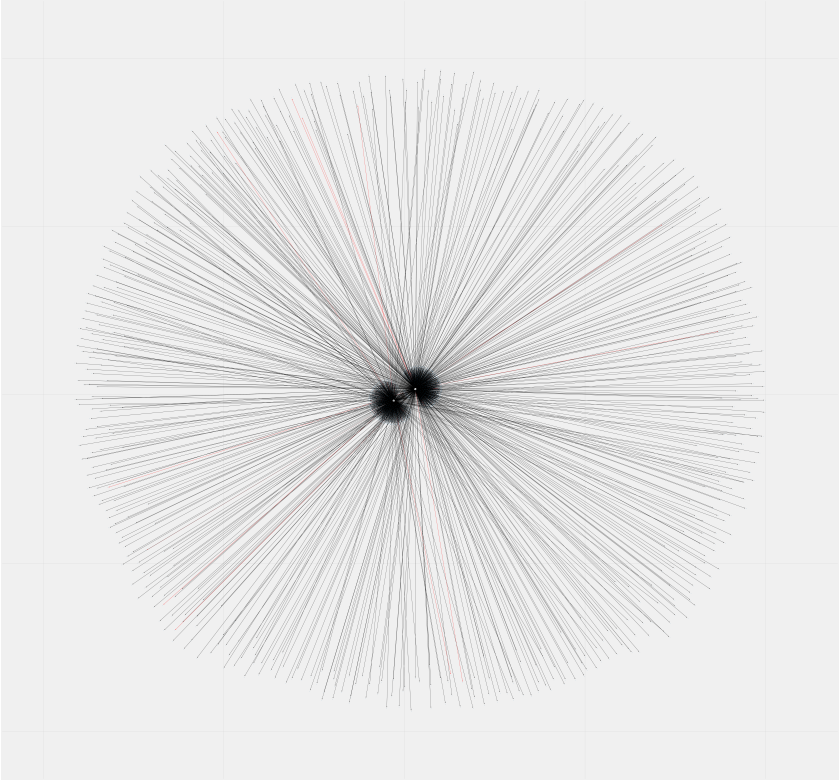
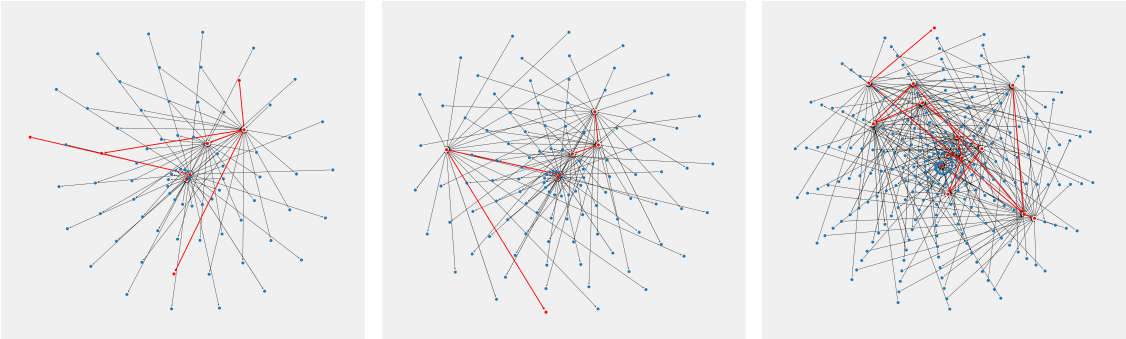


Figure 5.6: Depth 1 Network Querying all Friends and Followers

Figure 5.6 required a large amount of computational resource power and large amounts of time to compute even a network with only two primary hate users. This

was largely in part to the rate limit applied by the Tweepy API to only be able to request 900 tweets every 15 minutes. This meant that generating large networks composite of every friend and follower would result in large network generation times. Hence, further experiments were done limiting the number of friends and followers to 10. The impact of depth of a network was also impacted to see the variability of networks at different depths.

Figure 5.7 showcases a hate network at depths 3, 4 and 10. As expected with an increased depth, the network increases in size as the depth increases as a greater number of hate users are detected. Additionally, the number of hate users increases as a byproduct of the increased depth. As the depth increases from depth 4 to depth 10, many more interconnections are made between any given hate user to many more hate users in contrast to lower depth networks. More connections between hate users seem to denote an underlying hate network between users that allows hate speech to be much more widespread as users may fall into a hate network unknowingly, as can be seen through the ratio of non-hate users to hate users in all 3 figures.



(a) Depth 3 Hate Network (b) Depth 4 Hate Network (c) Depth 10 Hate Network

Figure 5.7: Social Networks with Different Depths denoting hate users and hate connections in red

5.3 Evaluation Findings

The evaluation section has detailed how each method has worked and performed to detect hate speech. With many methods performing quite well, it is still very unclear as to why hate speech is still such a large problem on social media platforms. Furthermore, it can be used to detect hate networks using a relatively simple algorithm. The work in this paper shows that social media platforms are not doing enough to justify the large presence of hate users still on the platform. Even simple methods such as K-NNs prove to be quite effective with certain preprocessing methods whilst slightly more complex methods such as NBSVMs and BiGRUs also perform very well on a small dataset. Testing against other state-of-the-art models also further proves that there are already more effective methods and with the social media platforms having the largest data access to their platforms, hate speech and hate network detection should not be as large a problem as it is currently.

Chapter 6

Conclusion

The conclusion chapter consists of a concluding remarks section outlining briefly, the entire experimental process throughout the investigation. It details the project process with a brief outline of the final results of the project. The next section details reflections on the project as a whole consisting of the decisions made from the inception through to the conclusion of the entire project. It follows through on the various decisions made and showcases what other decisions could be made, further expanding the entire design space in this project. This is further followed by a future work section detailing works that could further expand on what has been conducted on this paper.

6.1 Concluding Remarks

Overall, this project outlines different methods that have been used to detect hate speech. These different implementations included the use of KNNs, BiGRU, NBSVM and transformer models to detect hate speech. Additional works were done to determine the effects of preprocessing on the classification of hate and non-hate speech instances, with 16 various configurations used and certain models being trained up to about 50 epochs to determine whether training models for larger numbers of epochs results in better classification results or whether a model begins to overfit its training set. Further investigations were done to determine whether an ensemble of models performed better than the individual models themselves and though some of the models were more performant individually, the ensemble provided a more stable classification when compared to the models individually.

The ensemble was then used to find different hate networks with work on Twitter by randomly querying a lexicon built from the frequency analysis of hate speech within the test and training query sets. A set of users was identified by using the ensemble with a subset of users found as hate users by looking at a user's past ten tweets. Further work was done to identify the correct threshold to test users. After finding an initial user, hate networks were identified by querying the initial user's friends and followers to see whether a hate network could be identified through mutual connections. A hate network would then grow up to a defined depth to see how expansive a hate network could be and how hate users could be identified in a vast network filled with non-hate users.

Overall, multiple models were developed that classified at an accuracy that was >80% across both classes. Additionally, multiple configurations were tried and tested to create optimal data preprocessing methods for hate speech classification that can be used for further investigation. Furthermore, hate networks were detected through the use of the hate network detection system which displays its effectiveness. Further work was done to find optimal thresholding metrics to determine whether a user can be categorised as a hateful user.

6.2 Reflections

The entire project outlines various routes to the development of hate speech and hate network detection systems. The overall project only undertook one path in a very large design space. However, this paper showcases that there are further design paths that can be explored to ensure that hate speech is being detected optimally. A general overview of the design space that was worked in can be seen in Figure 6.1. It shows how large the design space is with only what was considered within the scope of this project. The path in the red dashed box indicates the path that was taken in the development of the project with the arrows branching outside denoting further routes that were possible throughout the project. Further developments can be seen in Section 6.3.

The first developments that were made were the K-NNs however, they could be further improved through a class weight implementation where the hate speech class was weighted at a larger value similarly to the class weights in the DL development process. Additionally, initial development of the DL methods was undertaken in Tensorflow with later developments taking place with the use of K-Train due to its ease of use and speed of deployment. However, further development in either PyTorch or Tensorflow would have allowed for more accurate control over the development of the DL models. Additionally, only 2 transformer models were used. Due to the large processing power and memory utilisation, a model like BERTweet[49], a transformer trained primarily on tweets, could not be used to train a hate speech detector. Additionally, only 5 models were used in the ensemble with all configurations being configurations 12 and 13 whilst configurations 4 and 5 also performed quite well. More models could be utilised to improve the performance of the ensemble whilst further developments could also look at which assortment of models performs the best at classifying hate speech when used as an ensemble.

Additional decisions were also made when the development of the hate network detector took place. The hate network only looks at the last ten tweets of a user and classifies them as hateful if seven of the ten tweets were classified as hateful. More tweets could be queried with different threshold values. Furthermore, observing hate users to look at their impact could also be done to determine the effect of an individual user on their local network and how hate communities form within a social network could also be researched through the work done in this paper. The conclusion to be made is why Twitter and other social media platforms have not done more to deal with hate speech. This project was undertaken by a sole developer at a university with a limited time frame of 6 months. With the resources that these mass media corporations have, more work can be undertaken with hate

speech no longer being a problem.

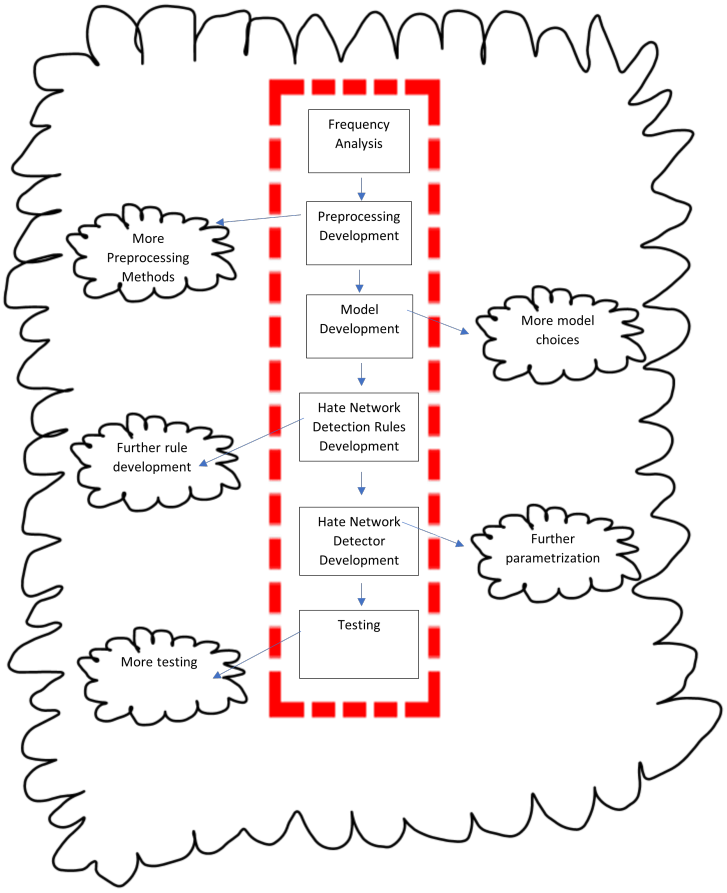


Figure 6.1: An illustration of the large design space that of which only the surface has been scraped

6.3 Future Work

Though this project provided many results to determine how to classify hate speech effectively and how to detect hate networks from hate users, more extensive research can be done to further improve on the current results by increasing the classification accuracy and looking at other methods to vary whilst training a model.

The first two major changes would be to observe hate speech on other social media platforms such as Facebook and Instagram. Hate speech can be spreading on more than one platform and through this project, work was done to detect hate speech on Twitter. However, similarly to how certain nuances were found when dealing with tweets, such as changing specific users to a user token and ensuring that the maximal length was 280 characters, the model that was developed and used in this project cannot be directly exported to directly work on another platform. Further work can be done to build models that are platform-specific in addition to a need for other social media platforms to release APIs such as Tweepy to ensure that projects can be done to deal with problems on social media. Dealing with other

forms of media can also be undertaken to address this problem with Instagram, as an example, for the need to deal with hate speech on images. Though work has been done to identify hate speech in images through the paper *"Multimodal Meme Dataset (MultiOFF) for Identifying Offensive Content in Image and Text"*[50], work can be done to apply this paper on a social media platform to prevent the spread of hate speech online.

The second major change would be able to detect hate speech in multiple languages as this project primarily focused on the detection of hate speech in English. Though a translation-based platform could be used, current translators struggle to directly translate between different languages to the English language. Hence, other hate speech detection models can be trained and tested on either a dataset comprised of a sole language or a multilingual dataset with the effectiveness of each of the models evaluated against one another. This would prove to be quite effective as English only accounted for 25.9% of media on the internet[51]. Hence, the hate speech detection method produced in this paper would only cover approximately 1/4 of current online media which means a large proportion of hate speech could be undetected with large hate networks being formed using languages other than English.

The next major change would be to be able to work with more datasets. Due to time constraints and computational resources, only two datasets were used as composites to form an entire dataset. This meant that the models trained were only exposed to a small subset of hate speech due to the lack of hate speech already within utilised datasets. Exposing the model to more varied data would allow the model to perform better. A possible solution to this is to create a dataset through querying Twitter continuously to label data with Twitter moderators labelling the tweets to ensure no bias and that the tweets were permissible by moderators of the platform..

There are many more improvements that could be made such as:

1. Using a reinforcement learning technique to ensure that the model is adaptive and can adapt to new phrases and colloquialism that is used online.

2. Further work to be done on an optimal threshold as only an optimal band was used to determine whether a user can be classified as hateful.

3. Querying more tweets to determine whether a user is hateful as the system only uses the 10 most recent tweets.

4. Similar to how different class weights were used in configurations for DL networks, K-NNs may benefit from having weighted neighbours.

5. Using a GNN to perform node classification on a given network after being trained on the characteristics of a hate speech network.

6. As the models developed are pretrained on the English vocabulary, they may struggle to pick up on certain terms that are misspelled, or include leetspeak, where letters are replaced by numbers and symbols.

Bibliography

[1] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way, Dec 15, 2018. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a5

[2] Aditi Mittal. Understanding rnn and lstm, Oct 12, 2019. URL https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[3] Christopher Olah. Understanding lstm networks, Aug 27, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[4] Wikipedia. Gated recurrent unit, -11-24 2021. URL https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[5] Tavish Srivastava. Introduction to k-nearest neighbors: A powerful machine learning algorithm (with implementation in python & r), Mar 26, 2018. URL https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/.

[6] Rushikesh Pupale. Support vector machines(svm) — an overview, Jan 16, 2018. URL https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 1–11, 2017.

[8] Sunil Gundapu and Radhika Mamidi. Transformer based automatic covid-19 fake news detection system, Jan 2021.

[9] Rishabh Anand. An illustrated guide to graph neural networks, Mar 30, 2020. URL https://medium.com/dair-ai/an-illustrated-guide-to-graph-neural-networks-d5564a551783.

[10] Chenda Ngak. Then and now: a history of social networking sites, Jul 6 , 2011. URL https://www.cbsnews.com/pictures/then-and-now-a-history-of-social-networking-sites/.

[11] Statista Research Department. Number of social media users 2025, Sep 10, 2021. URL https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.

[12] BBC News. Frances haugen says facebook is 'making hate worse', Oct 25, 2021. URL https://www.bbc.co.uk/news/technology-59038506.

[13] Wikipedia. Machine learning, Dec 19 2021. URL https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1061072611.

[14] Salman Aslam. Twitter by the numbers (2021): Stats, demographics & fun facts, Jan 3, 2021. URL https://www.omnicoreagency.com/twitter-statistics/.

[15] Prashant Kapil, Asif Ekbal, and Dipankar Das. Investigating deep learning approaches for hate speech detection in social media. May 29, 2020. URL https://arxiv.org/abs/2005.14690.

[16] Georgios K. Pitsilis, Heri Ramampiaro, and Helge Langseth. Effective hate-speech detection in twitter data using recurrent neural networks, Jul 26, 2018. URL https://link.springer.com/article/10.1007%2Fs10489-018-1242-y.

[17] Ziqi "Zhang, David Robinson, and Jonathan" Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. In Aldo "Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish" Alam, editors, *The Semantic Web*, pages 745–760, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93417-4.

[18] Swati Agarwal and Ashish Sureka. Using knn and svm based one-class classifier for detecting online radicalization on twitter. In Raja Natarajan, Gautam Barua, and Manas Ranjan Patra, editors, *Distributed Computing and Internet*, pages 431–442, Cham, Technology 2015. Springer International Publishing. ISBN 9783-319149776. ID: 10.1007978-3319149776_47.

[19] Wikipedia. Transformer (machine learning model), -12-18 2021. URL https://en.wikipedia.org/w/index.php?title=Transformer_(machine_learning_model)&oldid=1060927675.

[20] Raymond T. Mutanga, Nalindren Naicker, and Oludayo O. Olugbara. Hate speech detection in twitter using transformer methods. *(IJACSA) International Journal of Advanced Computer Science and Applications*, 11:1–7, 2020.

[21] Robi Polikar. Ensemble learning. *Scholarpedia*, 4(1):2776, /1/11 2009. doi: 10.4249/scholarpedia.2776. URL http://www.scholarpedia.org/article/Ensemble_learning.

[22] Pedro Alonso, Rajkumar Saini, and György Kovács. Hate speech detection using transformer ensembles on the hasoc dataset. In Alexey Karpov and Rodmonga Potapova, editors, *Speech and Computer*, pages 13–21, Cham, 2020. Springer International Publishing. ISBN 978-3-030-60276-5.

[23] Shu Li, Nayyar A. Zaidi, Qingyun Liu, and Gang Li. Neighbours and kinsmen: Hateful users detection with graph neural network. In Kamal Karlapalem, Hong

Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty, editors, *Advances in Knowledge Discovery and Data Mining*, pages 434–446, Cham, 2021. Springer International Publishing. ISBN 978-3-030-75762-5.

[24] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[25] *MATLAB version 9.12.0.1952421 (R2022a) Update 1*. The Mathworks, Inc., Natick, Massachusetts, 2022.

[26] Francois Chollet et al. Keras, 2015. URL https://uk.mathworks.com/discovery/what-is-matlab.html.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[28] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[29] ISO. *ISO/IEC 14882:1998: Programming languages — C++*. pub-ISO, pub-ISO:adr, September 1998. ISBN ???? URL http://webstore.ansi.org/ansidocstore/product.asp?sku=ISO%2FIEC+14882%2D1998;http://webstore.ansi.org/ansidocstore/product.asp?sku=ISO%2FIEC+14882%3A1998;http://www.iso.ch/cate/d25845.html;https://webstore.ansi.org/. Available in electronic form for online purchase at `http://webstore.ansi.org/` and `http://www.cssinfo.com/`.

[30] Inc The MathWorks. What is matlab?, 1994-2022. URL https://github.com/fchollet/keras.

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2019. URL https://arxiv.org/abs/1910.03771.

[32] Arun S. Maiya. ktrain: A low-code library for augmented machine learning. *arXiv preprint arXiv:2004.10703*, 2020.

[33] Joshua Roesslein. Tweepy: Twitter for python! 2020. URL https://github.com/tweepy/tweepy.

[34] Leonard Richardson. Beautiful soup documentation. *April*, 2007.

[35] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515, 2017.

[36] Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. Hate Speech Dataset from a White Supremacy Forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5102. URL https://www.aclweb.org/anthology/W18-5102.

[37] Zeerak Waseem and Dirk Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop*, pages 88–93, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-2013.

[38] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval), 2019.

[39] Peter Burnap and Matthew Leighton Williams. Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy & Internet*, 7:223–242, 2015.

[40] Jennifer Golbeck, Zahra Ashktorab, Rashad O. Banjo, Alexandra Berlinger, Siddharth Bhagwan, Cody Buntain, Paul Cheakalos, Alicia A. Geller, Quint Gergory, Rajesh Kumar Gnanasekaran, Raja Rajan Gunasekaran, Kelly M. Hoffman, Jenny Hottle, Vichita Jienjitlert, Shivika Khare, Ryan Lau, Marianna J. Martindale, Shalmali Naik, Heather L. Nixon, Piyush Ramachandran, Kristine M. Rogers, Lisa Rogers, Meghna Sardana Sarin, Gaurav Shahane, Jayanee Thanki, Priyanka Vengataraman, Zijian Wan, and Derek Michael Wu. A large labeled corpus for online harassment research. In *Proceedings of the 2017 ACM on Web Science Conference*, WebSci '17, page 229–233, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348966. doi: 10.1145/3091478.3091509. URL https://doi.org/10.1145/3091478.3091509.

[41] Roshan. Sharma. Twitter-sentiment-analysis. https://github.com/sharmaroshan/Twitter-Sentiment-Analysis, 2019.

[42] Michael Elhadad. Natural language processing with python steven bird, ewan klein, and edward loper (university of melbourne, university of edinburgh, and bbn technologies) sebastopol, ca: O'reilly media, 2009, xx+482 pp; paperbound, isbn 978-0-596-51649-9, $44.99; on-line free of charge at nltk.org/book. *Computational Linguistics*, 36:767–771, 12 2010. doi: 10.1162/coli_r_00022.

[43] Adam EK, Jean-Philippe Bernardy, and Stergios Chatzikyriakidis. How does punctuation affect neural models in natural language inference anonymous acl submission. 05 2020.

[44] Dane Hankamer and David Liedtka. Twitter sentiment analysis with emojis. 2019.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[46] Leslie N. Smith. Cyclical learning rates for training neural networks, 2015. URL https://arxiv.org/abs/1506.01186.

[47] Paul Röttger, Bertie Vidgen, Dong Nguyen, Zeerak Waseem, Helen Margetts, and Janet Pierrehumbert. HateCheck: Functional tests for hate speech detection models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 41–58, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.4. URL https://aclanthology.org/2021.acl-long.4.

[48] Zoe Papakipos and Joanna Bitton. Augly: Data augmentations for robustness, 2022.

[49] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. Bertweet: A pre-trained language model for english tweets, 2020. URL https://arxiv.org/abs/2005.10200.

[50] Shardul Suryawanshi, Bharathi Raja Chakravarthi, Mihael Arcan, and Paul Buitelaar. Multimodal meme dataset (MultiOFF) for identifying offensive content in image and text. In *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*, pages 32–41, Marseille, France, May 2020. European Language Resources Association (ELRA). ISBN 979-10-95546-56-6. URL https://aclanthology.org/2020.trac-1.6.

[51] Statista Research Department. Most common languages used on the internet as of january 2020, by share of internet users, Jan 10, 2020. URL https://www.statista.com/statistics/262946/share-of-the-most-common-languages-on-the-internet/.

[52] Ethics approval overview. URL https://www.imperial.ac.uk/research-ethics-committee/ethics-approval-overview/.

Appendix A

Project Files

The project files can be located at https://github.com/JeetendraJoshi/final_year_project. It contains all the source code for machine learning model development in addition to the training and test data and gives access to the original results used to calculate the results in Appendix D. It does not contain the Twitter developer keys utilised in the project however, as long as someone has access to a Twitter developer account, they will be able to utilise the page.

Appendix B

Appendix for Images

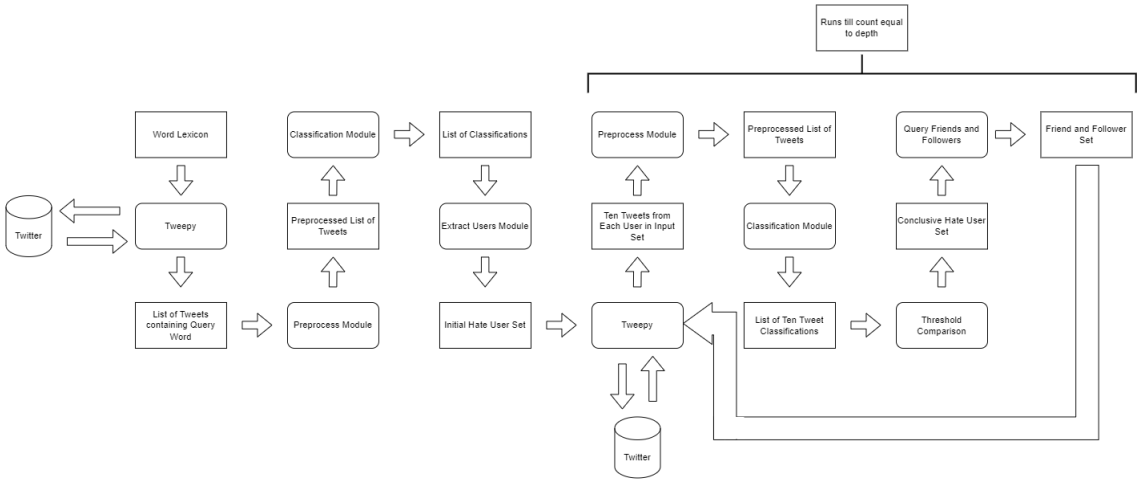


Figure B.1: Hate Network Detector System Diagram

Appendix C

Appendix for Original Planning

C.1 Implementation Plan

Therefore, the overall design process is as follows:

1. Develop a classification model for hate speech detection
2. Develop a system that can find hate speech online and classify the last 10 tweets of the user
3. Generate a social network based on the connections of various users
4. Classify the network with a graph neural network to classify various parts of the network as hateful.

The implementation plan outlines the key steps in the implementation of a system that is able to classify hate speech and generate social networks based on the classification of hateful users. The classification model must first delve into using an appropriate hate speech dataset and training a classifier on the given hate speech. The text must also be pre-processed to ensure that no irrelevant information is fed to the transformer and that all the input data is in a suitable format. The model must then be trained and learn how to identify a hateful user based on a range of tweets. Thereafter, the entire system must be able to create a graph that can be used to scout out various hateful users within a given social network. The GNN employed must then be able to identify specific users based on various features, such as number of hateful tweets and number of connections, to determine users that spread the most discrimination and hate within a given social network.

The initial model for classification should be developed before the first week of February. This can then be used to develop the rest of the system by being deployed online to find hateful users. This system should be developed by the third week of February. Thereafter, the generation of a social network should be completed in two weeks after this development. Finally the overall product should be delivered in the first week of April. This timeline allows for a great overall development plan with the ability to change various timings due to the length of the project available. Figure C.1 gives an outline of the project details and as shown, details that there is time towards the end of the project so in the case of tasks taking too long, there is sufficient time to still complete the overall project. Each number on the Gantt chart details a week.

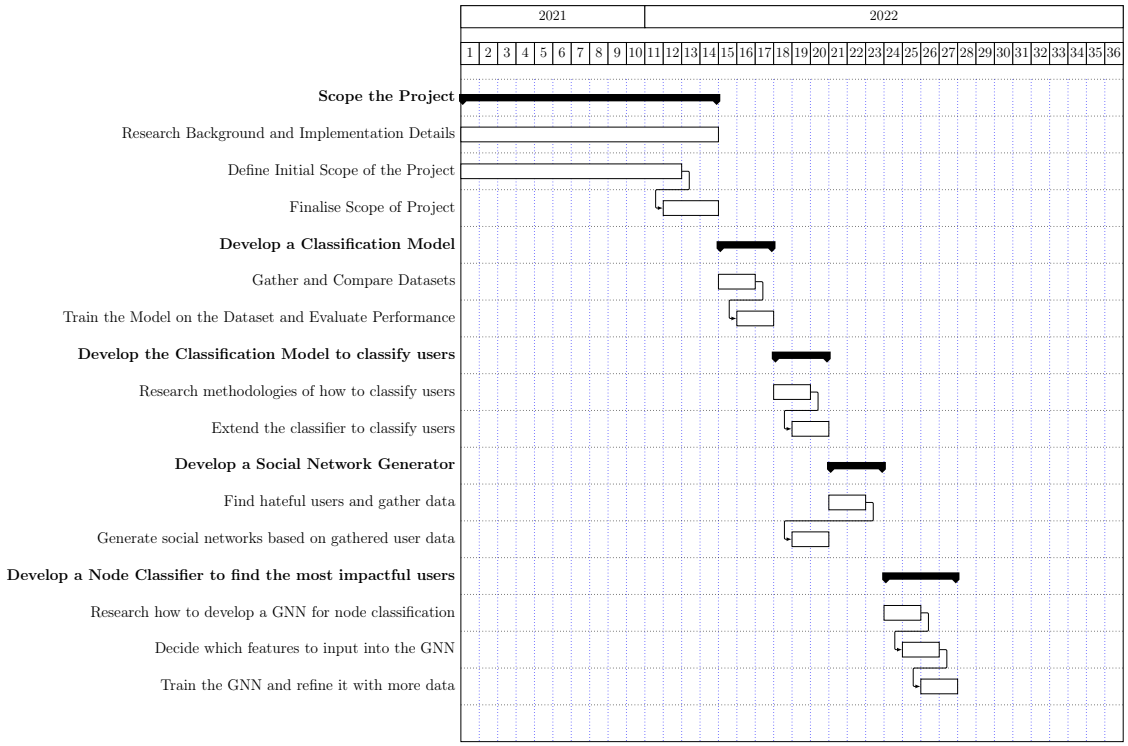


Figure C.1: Gantt chart detailing project timeline

C.2 Fallback Plan

The fallback of this project would involve the changing of the topic to create a bibliography of the classification of hate speech using pre-existing models covered in Chapter 2. This would ensure that no redundant work would be complete in the case that the GNN is unable to be completed. Additionally, this would be further complemented by investigating a new method of hate speech classification and applying it to a given dataset. It would further involve the comparison of this new method with current state of the art models to observe whether this new method would be able to attain higher metrics in terms of accuracy and precision when classifying various pieces of hate speech data.

C.3 Ethical, Legal and Safety Plan

C.3.1 Imperial College Ethical Approval Conditions

Below are the Imperial College London ethic review considerations [52]:

Being 'ethical' means acting in accordance with a set of core values and principles, in particular, integrity, compliance with the law, respect for human rights and avoiding unnecessary risk to people's safety and well-being. Imperial College London seeks to ensure that any potential ethical risks arising from research are limited strictly in proportion to the importance of the intended benefits.

A researcher must, therefore, consider the ethical implications of any work that:

- has the potential to damage the mental or physical health of human participants, (e.g. volunteers, College staff and students, or patients,) or others who

may be affected

- the potential to jeopardise the safety and liberty of people affected by the research (e.g. volunteers working in sensitive situations or abroad)
- has the potential to compromise the privacy of individuals whose data is involved in the work
- involves methods (e.g. genetic research, interviews, questionnaires, randomised control trial) or subject matter (e.g. recreational and controlled drugs, human impact on the environment) that are sensitive and therefore need to be managed consistently with the College's high public reputation
- carries a risk of an actual or perceived conflict of interest on the part of researchers and/or the College

C.3.2 Ethical Concerns

The main ethical concerns are the nature of the data that is involved in addition to the availability of data online. All data online is posted via an account especially on the platform ,Twitter, where media is freely available to any user who requests this data. The data used is both for training and testing of the machine learning models and could possibly be tracked back to specific user accounts in addition to the values of hate speech that are tracked within the investigation. Data used in training will specifically be anonymised but with the possibility of the machine learning model being deployed online, certain users may be tracked through the posting of hate speech.

C.3.3 Legal Concerns

The project does not entail any legal considerations. The machine learning algorithm will only be used for hate speech classification on data available online. Additionally, the machine learning algorithm will be generating social networks. The only legal concerns are data anonymisation but since the data trained by the model is anonymised, there should be no legal worry as the data can no longer be traced back to a specific user hence, there is no infringement of GDPR policies.

C.3.4 Safety Assessment

There are no physical safety concerns overall with this project as the project is not hardware related and all work on the project is software-related.

Appendix D

Appendix for Test Results

Configuration	Inclusion of URLs and User Tags	Non-Equal Class Weights	Dataset Type	Lemmatization
0	No	No	1	No
1	No	No	1	Yes
2	No	No	2	No
3	No	No	2	Yes
4	No	Yes	1	No
5	No	Yes	1	Yes
6	No	Yes	2	No
7	No	Yes	2	Yes
8	Yes	No	1	No
9	Yes	No	1	Yes
10	Yes	No	2	No
11	Yes	No	2	Yes
12	Yes	Yes	1	No
13	Yes	Yes	1	Yes
14	Yes	Yes	2	No
15	Yes	Yes	2	Yes

Table D.1: Table displaying all data configurations when training various models

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.951542	0.660777	0.510929	0.981917	0.576271	0.746423	0.00001
1	0.951718	0.663121	0.510929	0.982106	0.577160	0.746517	0.00001
2	0.612194	0.097748	0.695105	0.607117	0.171394	0.651111	0.00001
3	0.633418	0.101094	0.678322	0.630668	0.175964	0.654495	0.00001
4	0.859031	0.288911	0.811475	0.862309	0.426112	0.836892	0.00001
5	0.841586	0.268462	0.844262	0.841401	0.407383	0.842832	0.00001
6	0.058992	0.057740	0.999301	0.001413	0.109171	0.500357	0.00001
7	0.098212	0.059433	0.986713	0.043806	0.112113	0.515260	0.00001
8	0.952775	0.673759	0.519126	0.982671	0.586420	0.750898	0.00001
9	0.952070	0.683594	0.478142	0.984743	0.562701	0.731442	0.00001
10	0.475608	0.079846	0.768531	0.457671	0.144662	0.613101	0.00001
11	0.468513	0.078503	0.761538	0.450501	0.142334	0.606020	0.00001
12	0.869075	0.306269	0.814208	0.872857	0.445108	0.843533	0.00001
13	0.865727	0.299189	0.806011	0.869844	0.436391	0.837927	0.00001
14	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.00001
15	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.00001

Table D.2: Table displaying testing results for BERT model for 1 Epoch

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.953480	0.688889	0.508197	0.984178	0.584906	0.746187	0.00001
1	0.954185	0.687943	0.530055	0.983424	0.598765	0.756740	0.00001
2	0.389582	0.075703	0.854545	0.361110	0.139085	0.607828	0.00001
3	0.561368	0.090878	0.733566	0.550828	0.161720	0.642197	0.00001
4	0.865022	0.301193	0.827869	0.867583	0.441691	0.847726	0.00001
5	0.849163	0.282416	0.868852	0.847806	0.426273	0.858329	0.00001
6	0.087237	0.059017	0.991608	0.031859	0.111404	0.511734	0.00001
7	0.147520	0.061529	0.966434	0.097375	0.115692	0.531904	0.00001
8	0.955066	0.714286	0.505464	0.986061	0.592000	0.745763	0.00001
9	0.956828	0.730038	0.524590	0.986626	0.610493	0.755608	0.00001
10	0.577815	0.090637	0.699301	0.570376	0.160475	0.634839	0.00001
11	0.502239	0.081697	0.744755	0.487389	0.147242	0.616072	0.00001
12	0.869780	0.312563	0.849727	0.871162	0.457017	0.860444	0.00001
13	0.862907	0.301923	0.857923	0.863251	0.446657	0.860587	0.00001
14	0.074648	0.058369	0.993706	0.018370	0.110262	0.506038	0.00001
15	0.095267	0.059251	0.986713	0.040680	0.111789	0.513697	0.00001

Table D.3: Table displaying testing results for BERT model for 2 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.949780	0.742081	0.418367	0.989211	0.535073	0.703789	0.0010
1	0.935154	0.000000	0.000000	1.000000	0.000000	0.500000	0.0001
2	0.735302	0.097205	0.432867	0.753822	0.158759	0.593344	0.0060
3	0.535327	0.074143	0.613986	0.530510	0.132309	0.572248	0.0007
4	0.889515	0.349112	0.793011	0.896285	0.484799	0.844648	0.0080
5	0.899207	0.367478	0.786301	0.906968	0.500873	0.846635	0.0090
6	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
7	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
8	0.928811	0.000000	0.000000	1.000000	0.000000	0.500000	0.0001
9	0.953128	0.767442	0.433071	0.990555	0.553691	0.711813	0.0010
10	0.627043	0.085693	0.565035	0.630840	0.148817	0.597937	0.0010
11	0.483194	0.070512	0.653147	0.472787	0.127283	0.562967	0.0004
12	0.873833	0.320000	0.812834	0.878136	0.459215	0.845485	0.0010
13	0.889515	0.349942	0.820652	0.894291	0.490658	0.857471	0.0020
14	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
15	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001

Table D.4: Table displaying testing results for BiGRU model for 1 Epoch

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.952599	0.783410	0.433673	0.991104	0.558292	0.712389	0.0010
1	0.943260	0.671642	0.244565	0.991709	0.358566	0.618137	0.0001
2	0.568051	0.070005	0.527972	0.570505	0.123619	0.549238	0.0060
3	0.640036	0.086544	0.548252	0.645656	0.149490	0.596954	0.0007
4	0.875066	0.316249	0.779570	0.881765	0.449961	0.830667	0.0080
5	0.893744	0.350126	0.761644	0.902825	0.479724	0.832234	0.0090
6	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
7	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
8	0.936035	0.720430	0.165842	0.995067	0.269618	0.580454	0.0001
9	0.955242	0.782222	0.461942	0.990744	0.580858	0.726343	0.0010
10	0.623774	0.091069	0.614685	0.624331	0.158636	0.619508	0.0010
11	0.478150	0.068950	0.643357	0.468034	0.124552	0.555695	0.0004
12	0.906432	0.393197	0.772727	0.915865	0.521190	0.844296	0.0010
13	0.909956	0.398582	0.763587	0.920106	0.523765	0.841846	0.0020
14	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
15	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001

Table D.5: Table displaying testing results for BiGRU model for 2 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.945198	0.651685	0.443878	0.982396	0.528073	0.713137	0.0010
1	0.947489	0.643443	0.426630	0.983607	0.513072	0.705118	0.0001
2	0.547391	0.070029	0.557343	0.546782	0.124424	0.552062	0.0060
3	0.601622	0.085924	0.612587	0.600951	0.150710	0.606769	0.0007
4	0.896916	0.349788	0.666667	0.913068	0.458834	0.789867	0.0080
5	0.913304	0.395387	0.657534	0.930885	0.493827	0.794210	0.0090
6	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
7	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
8	0.945198	0.686747	0.423267	0.985202	0.523737	0.704235	0.0001
9	0.951542	0.703846	0.480315	0.985455	0.570983	0.732885	0.0010
10	0.636767	0.087582	0.562238	0.641331	0.151555	0.601784	0.0010
11	0.622725	0.086984	0.583217	0.625145	0.151389	0.604181	0.0004
12	0.921057	0.436426	0.679144	0.938125	0.531381	0.808635	0.0010
13	0.920352	0.423077	0.627717	0.940644	0.505470	0.784181	0.0020
14	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001
15	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.0001

Table D.6: Table displaying testing results for BiGRU model for 5 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.938502	0.571906	0.436224	0.975771	0.494935	0.705998	0.0010
1	0.950661	0.666667	0.478261	0.983418	0.556962	0.730839	0.0001
2	0.572893	0.070792	0.527972	0.575643	0.124845	0.551808	0.0060
3	0.567365	0.080004	0.618881	0.564210	0.141691	0.591546	0.0007
4	0.906960	0.376582	0.639785	0.925702	0.474104	0.782744	0.0080
5	0.909251	0.377451	0.632877	0.928249	0.472876	0.780563	0.0090
6	0.073599	0.058271	0.993007	0.017300	0.110082	0.505153	0.0001
7	0.070290	0.058256	0.996503	0.013574	0.110077	0.505039	0.0001
8	0.946256	0.682657	0.457921	0.983684	0.548148	0.720803	0.0001
9	0.948194	0.657040	0.477690	0.982055	0.553191	0.729873	0.0010
10	0.550297	0.081575	0.662238	0.543442	0.145257	0.602840	0.0010
11	0.695961	0.099146	0.527972	0.706248	0.166943	0.617110	0.0004
12	0.930573	0.478632	0.598930	0.953971	0.532067	0.776451	0.0010
13	0.928282	0.458244	0.581522	0.952327	0.512575	0.766924	0.0020
14	0.146552	0.060329	0.946154	0.097589	0.113426	0.521872	0.0001
15	0.083121	0.058585	0.988112	0.027705	0.110611	0.507909	0.0001

Table D.7: Table displaying testing results for BiGRU model for 10 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.940441	0.595070	0.431122	0.978232	0.500000	0.704677	0.0010
1	0.947489	0.633588	0.451087	0.981911	0.526984	0.716499	0.0001
2	0.568979	0.069274	0.520280	0.571961	0.122268	0.546120	0.0060
3	0.559900	0.076428	0.597902	0.557573	0.135531	0.577738	0.0007
4	0.900617	0.352761	0.618280	0.920422	0.449219	0.769351	0.0080
5	0.885991	0.315927	0.663014	0.901318	0.427940	0.782166	0.0090
6	0.364524	0.070181	0.817483	0.336788	0.129264	0.577135	0.0001
7	0.221079	0.063239	0.904895	0.179206	0.118217	0.542051	0.0001
8	0.945727	0.665517	0.477723	0.981597	0.556196	0.729660	0.0001
9	0.948546	0.660650	0.480315	0.982244	0.556231	0.731280	0.0010
10	0.656781	0.087357	0.523776	0.664925	0.149740	0.594351	0.0010
11	0.713029	0.104097	0.522378	0.724703	0.173600	0.623541	0.0004
12	0.912423	0.392294	0.598930	0.934541	0.474074	0.766736	0.0010
13	0.915419	0.398182	0.595109	0.937630	0.477124	0.766369	0.0020
14	0.259936	0.064307	0.872727	0.222413	0.119787	0.547570	0.0001
15	0.321430	0.066783	0.829371	0.290327	0.123612	0.559849	0.0001

Table D.8: Table displaying testing results for BiGRU model for 20 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.940088	0.595588	0.413265	0.979178	0.487952	0.696222	0.0010
1	0.943260	0.580420	0.451087	0.977388	0.507645	0.714238	0.0001
2	0.568737	0.069075	0.518881	0.571789	0.121919	0.545335	0.0060
3	0.578784	0.077795	0.580420	0.578684	0.137201	0.579552	0.0007
4	0.894273	0.335735	0.626344	0.913068	0.437148	0.769706	0.0080
5	0.900441	0.348024	0.627397	0.919209	0.447703	0.773303	0.0090
6	0.369406	0.069549	0.802098	0.342911	0.127999	0.572504	0.0001
7	0.326070	0.066924	0.825175	0.295508	0.123807	0.560341	0.0001
8	0.940617	0.603715	0.482673	0.975716	0.536451	0.729195	0.0001
9	0.946608	0.646617	0.451444	0.982244	0.531685	0.716844	0.0010
10	0.598717	0.082065	0.584615	0.599580	0.143927	0.592098	0.0010
11	0.700278	0.101726	0.535664	0.710358	0.170982	0.623011	0.0004
12	0.906256	0.368333	0.590909	0.928504	0.453799	0.759707	0.0010
13	0.919119	0.409901	0.562500	0.943848	0.474227	0.753174	0.0020
14	0.343784	0.067425	0.808392	0.315334	0.124468	0.561863	0.0001
15	0.338296	0.067445	0.816084	0.309040	0.124593	0.562562	0.0001

Table D.9: Table displaying testing results for BiGRU model for 30 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.942026	0.619772	0.415816	0.981071	0.497710	0.698444	0.0010
1	0.940264	0.547855	0.451087	0.974185	0.494784	0.712636	0.0001
2	0.565872	0.068860	0.520979	0.568621	0.121643	0.544800	0.0060
3	0.582819	0.077973	0.575524	0.583266	0.137338	0.579395	0.0007
4	0.886696	0.315646	0.623656	0.905148	0.419151	0.764402	0.0080
5	0.900617	0.340289	0.580822	0.922599	0.429150	0.751710	0.0090
6	0.458863	0.075653	0.746853	0.441228	0.137390	0.594041	0.0001
7	0.360247	0.068346	0.798601	0.333405	0.125917	0.566003	0.0001
8	0.936388	0.563798	0.470297	0.972112	0.512821	0.721204	0.0001
9	0.941850	0.581470	0.477690	0.975255	0.524496	0.726473	0.0010
10	0.632329	0.085563	0.554545	0.637092	0.148252	0.595819	0.0010
11	0.681556	0.098234	0.552448	0.689462	0.166807	0.620955	0.0004
12	0.907841	0.371773	0.577540	0.931145	0.452356	0.754343	0.0010
13	0.908546	0.373534	0.605978	0.929527	0.462176	0.767753	0.0020
14	0.380987	0.068972	0.778322	0.356657	0.126715	0.567489	0.0001
15	0.439616	0.072125	0.734266	0.421573	0.131349	0.577919	0.0001

Table D.10: Table displaying testing results for BiGRU model for 40 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.942026	0.619772	0.415816	0.981071	0.497710	0.698444	0.0010
1	0.937797	0.523511	0.453804	0.971359	0.486172	0.712581	0.0001
2	0.565105	0.068341	0.517483	0.568021	0.120737	0.542752	0.0060
3	0.558165	0.076286	0.599301	0.555646	0.135344	0.577473	0.0007
4	0.879824	0.306733	0.661290	0.895154	0.419080	0.778222	0.0080
5	0.869956	0.280846	0.654795	0.884746	0.393092	0.769770	0.0090
6	0.532462	0.081982	0.696503	0.522417	0.146697	0.609460	0.0001
7	0.340516	0.067610	0.815385	0.311438	0.124866	0.563411	0.0001
8	0.934449	0.546784	0.462871	0.970594	0.501340	0.716733	0.0001
9	0.944317	0.612457	0.464567	0.978844	0.528358	0.721705	0.0010
10	0.533228	0.075958	0.634965	0.526999	0.135684	0.580982	0.0010
11	0.683977	0.098671	0.550350	0.692159	0.167340	0.621255	0.0004
12	0.904670	0.359191	0.569519	0.928315	0.440538	0.748917	0.0010
13	0.896916	0.336842	0.608696	0.916902	0.433688	0.762799	0.0020
14	0.408425	0.070338	0.757343	0.387059	0.128722	0.572201	0.0001
15	0.442077	0.072488	0.734965	0.424143	0.131961	0.579554	0.0001

Table D.11: Table displaying testing results for BiGRU model for 50 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.954537	0.736842	0.459016	0.988698	0.565657	0.723857	0.00001
1	0.952247	0.672727	0.505464	0.983048	0.577223	0.744256	0.00001
2	0.390187	0.067459	0.746154	0.368390	0.123732	0.557272	0.00001
3	0.487229	0.075824	0.704895	0.473901	0.136919	0.589398	0.00001
4	0.860793	0.292157	0.814208	0.864005	0.430014	0.839106	0.00001
5	0.842643	0.270670	0.849727	0.842155	0.410561	0.845941	0.00001
6	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.00001
7	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.00001
8	0.953833	0.708000	0.483607	0.986250	0.574675	0.734928	0.00001
9	0.953480	0.717949	0.459016	0.987568	0.560000	0.723292	0.00001
10	0.437114	0.072696	0.744755	0.418276	0.132463	0.581516	0.00001
11	0.416899	0.070462	0.746853	0.396694	0.128776	0.571774	0.00001
12	0.842467	0.270435	0.849727	0.841966	0.410290	0.845847	0.00001
13	0.849515	0.280576	0.852459	0.849312	0.422192	0.850886	0.00001
14	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.00001
15	0.057701	0.057701	1.000000	0.000000	0.109106	0.500000	0.00001

Table D.12: Table displaying testing results for DistilBERT model for 1 Epoch

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.957357	0.758333	0.497268	0.989075	0.600660	0.743171	0.00001
1	0.956123	0.760000	0.467213	0.989829	0.578680	0.728521	0.00001
2	0.353710	0.068304	0.806993	0.325954	0.125948	0.566473	0.00001
3	0.428439	0.071472	0.742657	0.409198	0.130395	0.575928	0.00001
4	0.881586	0.329241	0.806011	0.886796	0.467512	0.846403	0.00001
5	0.876476	0.322375	0.830601	0.879638	0.464477	0.855120	0.00001
6	0.062503	0.057872	0.997902	0.005224	0.109399	0.501563	0.00001
7	0.057782	0.057706	1.000000	0.000086	0.109114	0.500043	0.00001
8	0.957181	0.735632	0.524590	0.987003	0.612440	0.755797	0.00001
9	0.955947	0.743697	0.483607	0.988510	0.586093	0.736058	0.00001
10	0.422225	0.070624	0.741259	0.402689	0.128962	0.571974	0.00001
11	0.459186	0.075395	0.743357	0.441785	0.136905	0.592571	0.00001
12	0.875771	0.320255	0.825137	0.879262	0.461421	0.852199	0.00001
13	0.856035	0.290622	0.855191	0.856093	0.433818	0.855642	0.00001
14	0.061050	0.057823	0.998601	0.003640	0.109316	0.501121	0.00001
15	0.063269	0.057952	0.998601	0.005995	0.109547	0.502298	0.00001

Table D.13: Table displaying testing results for DistilBERT model for 2 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.951894	0.626230	0.545714	0.978592	0.583206	0.762153	0.010
1	0.953480	0.651316	0.556180	0.980071	0.600000	0.768126	0.010
2	0.594319	0.081440	0.586713	0.594784	0.143028	0.590749	0.002
3	0.594036	0.081141	0.584615	0.594613	0.142504	0.589614	0.002
4	0.853921	0.291985	0.778626	0.859523	0.424705	0.819074	0.010
5	0.862731	0.300201	0.784777	0.868342	0.434277	0.826559	0.010
6	0.099867	0.057894	0.955944	0.047446	0.109177	0.501695	0.002
7	0.096074	0.057777	0.958042	0.043292	0.108981	0.500667	0.010
8	0.953480	0.652174	0.517241	0.981979	0.576923	0.749610	0.010
9	0.949075	0.636923	0.547619	0.977723	0.588905	0.762671	0.010
10	0.551790	0.081763	0.661538	0.545069	0.145538	0.603304	0.005
11	0.542889	0.080669	0.674126	0.534945	0.144096	0.604535	0.005
12	0.853216	0.282964	0.769634	0.859248	0.413793	0.814441	0.010
13	0.847401	0.263352	0.812500	0.849709	0.397775	0.831104	0.010
14	0.080902	0.058159	0.982517	0.025693	0.109817	0.504105	0.002
15	0.078481	0.058161	0.985315	0.022952	0.109838	0.504133	0.002

Table D.14: Table displaying testing results for NBSVM model for 1 Epoch

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.955066	0.702128	0.471429	0.986854	0.564103	0.729142	0.010
1	0.954714	0.673684	0.539326	0.982516	0.599064	0.760921	0.010
2	0.620708	0.084896	0.569930	0.623817	0.147779	0.596874	0.002
3	0.615139	0.083949	0.572028	0.617779	0.146411	0.594904	0.002
4	0.910485	0.411674	0.681934	0.927490	0.513410	0.804712	0.010
5	0.914185	0.418960	0.719160	0.928221	0.529469	0.823690	0.010
6	0.108946	0.057998	0.947552	0.057594	0.109305	0.502573	0.002
7	0.103256	0.057951	0.953147	0.051214	0.109259	0.502180	0.010
8	0.954890	0.709091	0.448276	0.987986	0.549296	0.718131	0.010
9	0.950837	0.686792	0.481481	0.984331	0.566096	0.732906	0.010
10	0.641892	0.090619	0.576224	0.645913	0.156609	0.611068	0.005
11	0.587943	0.085442	0.632867	0.585192	0.150557	0.609030	0.005
12	0.914009	0.413115	0.659686	0.932363	0.508065	0.796025	0.010
13	0.906784	0.372294	0.732955	0.918279	0.493780	0.825617	0.010
14	0.089053	0.058280	0.975524	0.034771	0.109990	0.505148	0.002
15	0.086955	0.058632	0.984615	0.031987	0.110674	0.508301	0.002

Table D.15: Table displaying testing results for NBSVM model for 2 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.957004	0.784946	0.417143	0.992488	0.544776	0.704816	0.010
1	0.954890	0.731481	0.443820	0.989096	0.552448	0.716458	0.010
2	0.636202	0.088344	0.569231	0.640303	0.152950	0.604767	0.002
3	0.626034	0.085028	0.561538	0.629983	0.147692	0.595761	0.002
4	0.928987	0.490530	0.659033	0.949072	0.562432	0.804053	0.010
5	0.930749	0.488506	0.669291	0.949566	0.564784	0.809428	0.010
6	0.135617	0.058831	0.932168	0.086841	0.110678	0.509504	0.002
7	0.124158	0.058526	0.939860	0.074209	0.110191	0.507035	0.010
8	0.955947	0.769231	0.402299	0.992116	0.528302	0.697207	0.010
9	0.952070	0.750000	0.420635	0.989994	0.538983	0.705315	0.010
10	0.632208	0.090657	0.595105	0.634480	0.157345	0.614792	0.005
11	0.617157	0.088037	0.602098	0.618079	0.153613	0.610088	0.005
12	0.928811	0.478516	0.641361	0.949556	0.548098	0.795459	0.010
13	0.924934	0.434164	0.693182	0.940259	0.533917	0.816721	0.010
14	0.115160	0.058953	0.958042	0.063546	0.111071	0.510794	0.002
15	0.111407	0.059322	0.969231	0.058879	0.111801	0.514055	0.002

Table D.16: Table displaying testing results for NBSVM model for 5 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.955771	0.792899	0.382857	0.993427	0.516378	0.688142	0.010
1	0.954361	0.736585	0.424157	0.989848	0.538324	0.707003	0.010
2	0.641246	0.089739	0.570629	0.645570	0.155089	0.608100	0.002
3	0.640762	0.086167	0.544056	0.646684	0.148771	0.595370	0.002
4	0.937621	0.546988	0.577608	0.964407	0.561881	0.771008	0.010
5	0.937974	0.533487	0.606299	0.961844	0.567568	0.784071	0.010
6	0.158133	0.059559	0.918881	0.111549	0.111868	0.515215	0.002
7	0.148610	0.059206	0.923776	0.101143	0.111280	0.512460	0.010
8	0.955947	0.798780	0.376437	0.993805	0.511719	0.685121	0.010
9	0.951718	0.762626	0.399471	0.991127	0.524306	0.695299	0.010
10	0.641892	0.091248	0.581119	0.645613	0.157730	0.613366	0.005
11	0.628455	0.089595	0.593706	0.630583	0.155694	0.612145	0.005
12	0.940088	0.554688	0.557592	0.967693	0.556136	0.762642	0.010
13	0.933921	0.476190	0.653409	0.952470	0.550898	0.802940	0.010
14	0.139612	0.059987	0.948252	0.090095	0.112835	0.519174	0.002
15	0.138159	0.060280	0.955245	0.088126	0.113403	0.521685	0.002

Table D.17: Table displaying testing results for NBSVM model for 10 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.954185	0.767857	0.368571	0.992676	0.498069	0.680624	0.010
1	0.952247	0.722513	0.387640	0.990036	0.504570	0.688838	0.010
2	0.642013	0.089565	0.567832	0.646555	0.154726	0.607194	0.002
3	0.645644	0.086316	0.536364	0.652336	0.148701	0.594350	0.002
4	0.940969	0.584795	0.508906	0.973116	0.544218	0.741011	0.010
5	0.948194	0.633028	0.543307	0.977333	0.584746	0.760320	0.010
6	0.176492	0.059836	0.902098	0.132060	0.112228	0.517079	0.002
7	0.164185	0.059646	0.913287	0.118315	0.111978	0.515801	0.010
8	0.955947	0.806250	0.370690	0.994181	0.507874	0.682435	0.010
9	0.950837	0.761905	0.380952	0.991505	0.507937	0.686229	0.010
10	0.730065	0.129784	0.545455	0.743035	0.209677	0.644245	0.005
11	0.630190	0.089394	0.588811	0.632724	0.155222	0.610768	0.005
12	0.939031	0.551136	0.507853	0.970149	0.528610	0.739001	0.010
13	0.941850	0.528646	0.576705	0.965997	0.551630	0.771351	0.010
14	0.164831	0.060813	0.932867	0.117801	0.114183	0.525334	0.002
15	0.163136	0.061096	0.939860	0.115574	0.114735	0.527717	0.002

Table D.18: Table displaying testing results for NBSVM model for 20 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.954890	0.767045	0.385714	0.992300	0.513308	0.689007	0.010
1	0.952423	0.723958	0.390449	0.990036	0.507299	0.690243	0.010
2	0.643021	0.089632	0.566434	0.647711	0.154772	0.607072	0.002
3	0.641528	0.084504	0.530070	0.648354	0.145769	0.589212	0.002
4	0.939207	0.571429	0.488550	0.972738	0.526749	0.730644	0.010
5	0.948018	0.641447	0.511811	0.979411	0.569343	0.745611	0.010
6	0.188113	0.060150	0.893706	0.144906	0.112713	0.519306	0.002
7	0.175080	0.060187	0.909790	0.130090	0.112905	0.519940	0.010
8	0.955419	0.791411	0.370690	0.993617	0.504892	0.682154	0.010
9	0.950837	0.751269	0.391534	0.990749	0.514783	0.691142	0.010
10	0.642658	0.086710	0.544755	0.648653	0.149606	0.596704	0.005
11	0.626195	0.088204	0.586713	0.628613	0.153354	0.607663	0.005
12	0.932511	0.498638	0.479058	0.965237	0.488652	0.722147	0.010
13	0.937269	0.494845	0.545455	0.963179	0.518919	0.754317	0.010
14	0.097163	0.058700	0.974126	0.043463	0.110727	0.508795	0.002
15	0.174595	0.061330	0.930070	0.128335	0.115072	0.529202	0.002

Table D.19: Table displaying testing results for NBSVM model for 30 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.954185	0.752809	0.382857	0.991737	0.507576	0.687297	0.010
1	0.951894	0.708543	0.396067	0.989096	0.508108	0.692582	0.010
2	0.645241	0.088439	0.553147	0.650880	0.152497	0.602013	0.002
3	0.640600	0.084473	0.531469	0.647283	0.145775	0.589376	0.002
4	0.940264	0.581818	0.488550	0.973874	0.531120	0.731212	0.010
5	0.946079	0.619048	0.511811	0.977333	0.560345	0.744572	0.010
6	0.193681	0.060210	0.888112	0.151158	0.112774	0.519635	0.002
7	0.185087	0.060556	0.904196	0.141053	0.113511	0.522624	0.010
8	0.955771	0.801242	0.370690	0.993993	0.506876	0.682341	0.010
9	0.950837	0.756477	0.386243	0.991127	0.511384	0.688685	0.010
10	0.635476	0.088261	0.569930	0.639490	0.152851	0.604710	0.005
11	0.624864	0.087640	0.584615	0.627328	0.152430	0.605972	0.005
12	0.922643	0.433566	0.486911	0.954090	0.458693	0.720501	0.010
13	0.930044	0.446809	0.536932	0.956040	0.487742	0.746486	0.010
14	0.180002	0.061467	0.925874	0.134330	0.115281	0.530102	0.002
15	0.185006	0.061700	0.923776	0.139768	0.115674	0.531772	0.002

Table D.20: Table displaying testing results for NBSVM model for 40 Epochs

Configuration	Accuracy	Precision	Hate Speech Accuracy (Recall)	Non-Hate Speech Accuracy	F1-Score	Performance	Learning Rate
0	0.954361	0.751381	0.388571	0.991549	0.512241	0.690060	0.010
1	0.952247	0.717949	0.393258	0.989660	0.508167	0.691459	0.010
2	0.645483	0.088591	0.553846	0.651094	0.152748	0.602470	0.002
3	0.639713	0.083620	0.526573	0.646641	0.144322	0.586607	0.002
4	0.939736	0.577039	0.486005	0.973495	0.527624	0.729750	0.010
5	0.942731	0.584337	0.509186	0.973933	0.544180	0.741560	0.010
6	0.200097	0.060541	0.886014	0.158095	0.113338	0.522055	0.002
7	0.194125	0.060949	0.900000	0.150901	0.114167	0.525451	0.010
8	0.955419	0.802548	0.362069	0.994181	0.499010	0.678125	0.010
9	0.951189	0.761658	0.388889	0.991316	0.514886	0.690102	0.010
10	0.635355	0.087428	0.563636	0.639746	0.151376	0.601691	0.005
11	0.628576	0.088058	0.581119	0.631482	0.152940	0.606300	0.005
12	0.916476	0.400000	0.481675	0.947856	0.437055	0.714766	0.010
13	0.925639	0.421525	0.534091	0.951531	0.471178	0.742811	0.010
14	0.188194	0.061723	0.920280	0.143365	0.115687	0.531822	0.002
15	0.192067	0.061796	0.916783	0.147690	0.115787	0.532237	0.002

Table D.21: Table displaying testing results for NBSVM model for 50 Epochs