MANCHESTER 1824
The University of Manchester

INVESTIGATING AUTOMATED HATE SPEECH DETECTION METHODS FOR SOCIAL MEDIA

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2024

Awais Ahmed

Supervised by Prof. Sophia Ananiadou

Department of Computer Science

Contents

Word Count: 13721

List of Tables

List of Figures

Abstract

Disclaimer: The project contains the use of words or phrases that many people will find offensive; however, this can't be avoided owing to the nature of the research being carried out.

This project explored various NLP approaches to automated hate speech detection. The aim of the research carried out was to provide a comprehensive analysis of the current state of hate speech detection when applied to data from social media. This included the investigation of various features, the difference between traditional NLP approaches (SVM, RF and LR) and deep learning (BiLSTM, BERT and RoBERTa), and how the classification task impacts model capabilities. The project produced results for each goal which successfully achieved what was intended. TF-IDF document vectors were found to be most successful when combined with traditional models. BERT and RoBERTa models were found to produce state-of-the-art performance, whilst traditional models were also shown to produce good performance. The dataset classification labels were found to improve hate speech detection performance in a binary setup, at the cost of a loss of categorical information.

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/`) and in The University's policy on presentation of Theses

Acknowledgements

I would like to express my gratitude to Prof. Sophia Ananiadou for her unwavering support and advice during this project, and to my family and friends for their support throughout the journey.

Abbreviations and Acronyms

- Machine Learning (ML)
- Neural Network (NN)
- Natural Language Processing (NLP)
- Out-of-vocab (OOV)
- Large Language Model (LLM)
- Bidirectional Long Short Term Memory (BiLSTM)
- Bidirectional Encoder Representations From Transformers (BERT)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Term Frequency-Inverse Document Frequency (TF-IDF)
- Bag of Words (BoW)
- Continuous Bag of Words (CBOW)
- Skip Gram (SG)
- Distributed Memory (DM)
- Distributed Bag of Words (DBOW)
- Embeddings From Language Model (ELMo)
- Bidirectional Language Model (BiLM)
- Support Vector Machine (SVM)
- Random Forest (RF)
- Logistic Regression (LR)

Chapter 1

Introduction

Hate Speech has been a prominent issue which has plagued humanity throughout history and has been a point of contention for whether the beliefs and opinions upheld by an individual can be considered freedom of speech whilst simultaneously violating human rights. To date, there is no universally agreed definition on what constitutes a piece of media as hate speech, with the United Nations defining it as "offensive discourse toward an individual or group of people based on inherent characteristics (gender, religion, race) that threatens social peace." [48] This makes the task of hate speech detection complex, as the occurrence of hateful words doesn't necessarily constitute a piece of text as being hateful. But with its rising prevalence through various social media platforms, it's become as important as ever to identify hate speech instances as well as provide sufficient punishment for repeat offenders.

The rise of hate speech on social media can be attributed to its rapid growth, which has increased connectivity between individuals whilst simultaneously making it easier for people to mask their identities. Victims of hate speech are vast, with the majority coming from minority groups. Individuals with a large social media following are also at high risk of receiving hate speech; an example being athletes. Common forms of hate speech tend to be rooted in racism, homophobia, sexism, ableism amongst others.

1.1 Motivations

With the clear issue that hate speech presents and no signs of it slowing down, it's crucial that social media companies are able to accurately identify instances of hate speech, not only to prevent its spread to large audiences but to then allow for appropriate punishment for offending individuals. Undetected hate speech is a dangerous influence that must be acknowledged since it not only encourages an increase in hate crimes against minority groups but also has the ability to incite rhetoric that is genocidal in nature. There

exist many examples in human history, past and present, where hate speech has played a major role in war crimes against minority groups. Detection is the first step in a larger chain in helping to combat hateful ideologies and agendas, which can ultimately lead to a peaceful society and a harmonious and successful use of social media. In this project, I want to explore the different methods available for detecting hate speech and investigate how adjustments to the processing pipeline, as well as the data available, can impact the resulting performance of models.

1.2 Aims

This project aims to provide an in-depth analysis and discussion surrounding of hate speech, specifically methods of hate speech detection. In line with the motivation, the project has the following aims:

- Explore and evaluate various feature extraction techniques and how they impact hate speech detection performance, using traditional, ML-based natural language processing (NLP) models.
- Analyse hate speech detection in its current state today, evaluating two NLP approaches, traditional and deep learning.
- Explore the impact of multi-class and binary classification setups on hate speech detection performance through the analysis and evaluation of multiple ML models.

The evaluation strategies will focus on assessing the effectiveness of the developed hate speech detection models, considering the performance within each NLP approach and comparative performance across approaches.

1.3 Deliverables

In line with the aims of this project, it has successfully produced the following:

- This dissertation, which serves as a thorough analysis into hate speech detection through presentation of various methods, evaluation of different strategies and key findings being detailed.
- Models created from ML-based NLP and language analysis of hate speech, which serve as an example into the overall effectiveness of each architecture, which can then be used as part of hate speech detection tools.

Chapter 2

Background

In order to offer the necessary background knowledge for subsequent chapters, this chapter will explain key concepts that are employed throughout the project. This includes the distinction between traditional and neural network-based approaches, feature extraction techniques that are employed, as well as the underlying architecture of respective models. In addition to this, a literature review of hate speech detection will be presented, which will consist of:

- A review of factors that have influenced the effectiveness of hate speech detection, including dataset composition and annotator reliability.
- A review of traditional and neural network-based NLP approaches to hate speech detection.

2.1 Traditional NLP Approaches

As part of the traditional approaches, three ML models and one feature extraction technique were selected for use in this project. The models are: Support Vector Machine (SVM), Logistic Regression (LR) and Random Forest (RF), whilst TF-IDF over bag of words is used to extract features. This section provides a brief overview of the mathematical foundations of each model, as well as the intuition behind the feature extraction technique of choice.

2.1.1 Bag of Words

The simplest and most well-known approach to feature extraction is the bag of words (BoW) model [21]. The intuition is to represent a document of text over the entire vocabulary of the corpus. The presence or absence of a word within a document is then encoded using the total count for that word. This produces a document-term matrix,

where each document's dimensionality is equal to the vocabulary size. These extracted features can then serve as an input into machine learning models.

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Figure 2.1: Bag of words example [41]

This can effectively capture the essence of the document's meaning through its set of constituent words, which is capable of providing relevant features needed for sequence classification tasks such as hate speech. In addition, all words within a document contribute to its vector representation, making it robust to noise and mitigating issues such as out-of-vocabulary (OOV) terminology.

A drawback with this approach however is its lack of ability to provide information on the relative importance of each word to the document in question. This is most commonly seen with stop words where they have a high incidence in documents but provide comparatively low information related to its meaning.

2.1.2 TF-IDF

In order to allow for a more robust representation of documents where word importance is considered, a statistical measure can be added on top of the typical BoW approach. This comprises two components: Term Frequency (TF) as seen in Equation 2.1, and the Inverse Document Frequency (IDF) as seen in Equation 2.2 This is just one of the feature extraction techniques to be used in subsequent chapters.

$$TF(t,d) = \frac{\text{Number of times } t \text{ appears in } d}{\text{Total number of terms in } d} \tag{2.1}$$

The TF is the relative frequency of a word within a single document, normalising this value against the document length to allow for comparison of terms across documents of varying length.

$$IDF(t) = \log \frac{\text{Total number of documents}}{\text{Number of documents containing } t} \tag{2.2}$$

The IDF measures the importance of a term relative to the entire corpus, with higher

weights being given to terms which occur less frequently in the document. The intuition behind this is that rarer terms are likely to provide greater semantic importance to documents.

$$TF\text{-}IDF(t,d) = TF(t,d) \cdot IDF(t) \tag{2.3}$$

When combined, both the relative frequency of a term in a given document and its calculated importance across a corpus are combined to provide a weight, indicating word importance within a single document. This approach improves over the simple BoW approach but still leaves room for improvement in a few areas, with word order being neglected and word semantics unable to be captured. Furthermore, the vectors produced are sparse, with many zero values. The succeeding techniques look to address these issues by utilising word embedding based approaches.

2.1.3 Logistic Regression

Logistic Regression (LR) is one of the most common and well-known ML models. It models the probability that a given input belongs to a certain class. In its original form, it can handle binary classification problems, but can be extended to multi-class classification problems by adjusting the logistic function and the loss function. In the case of hate speech detection across multiple, non-ordinal categories, Multi-nominal Logistic Regression is the desired approach [26].

Logits are calculated, which represent a score of an input x belonging to a class k . These represent linear class predictions.

$$z_k = w_k^T x + b_k \tag{2.4}$$

Here, z_k represents the logit, w_k represents the weights and b_k is the bias term for class k.

The prediction function calculates the probability of a given input belonging to a class k out of K classes using the derived logits. This is known as the Soft max function.

$$\hat{y}_k = \text{Softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.5}$$

Here, $\hat{y}_k$ is the predicted probability that input x belongs to class k. The complete prediction models the probability distribution over K classes, such that $\hat{y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_K]$.

The loss function is used to calculate the error between the predicted probabilities of the

prediction function and the actual class labels.

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik}) \tag{2.6}$$

Here, L is the total loss over all samples, N is the number of samples, K is the number of classes, y_{ik} is a binary indication of whether class k is the correct label, and $\hat{y}_{ik}$ is the predicted probability.

The goal is to minimise the total loss over all samples through adjustment of the weight and bias values. This is normally done through an optimisation process. A common approach to optimisation for logistic regression is gradient descent, which iteratively updates weight and bias values in a manner which reduces the loss, based on the gradient on the loss function.

2.1.4 Support Vector Machine

Support Vector Machines (SVM) are another popular ML model used for classification and regression problems. It utilises decision boundaries to determine the class for a given input. The aim is to find a hyperplane that separates the classes in the feature space. The best separation is achieved by maximizing the margin between the closest points of the classes and the hyperplane. As part of a multi-class classification problem, a one vs one approach can be employed, where multiple binary classifiers are trained, through various combinations of all classes [22]. Predictions are calculated by combining the predictions of each classifier through a voting procedure to determine the final class.

The prediction function outputs a discrete value corresponding to the class the input belongs to.

$$f_{ij}(x) = \text{sign}(w_{ij}^T x + b_{ij}) \tag{2.7}$$

In the one vs one approach, there isn't a single prediction function; instead, there are multiple decision functions, one for each pair of classes. This is represented by the classes i and j . Here, $f_{ij}(x)$ represents the predicted class for an input feature x, w represents the weights and b is the bias.

The loss function, in the context of SVM's is the hinge loss.

$$L_{ij} = \max(0, 1 - y_{ij}(w_{ij}^T x + b_{ij})) \tag{2.8}$$

Here, L_{ij} represents the loss for a classifier differentiating between classes i and j and y_{ij} is the ground-truth label of input x.

The minimisation of the loss function through optimisation algorithms like gradient descent allow for the selection of an optimal decision boundary.

2.1.5 Random Forest

Random Forest (RF) is an ensemble ML method which utilises a multitude of decision trees. Each tree predicts a class for a given input. The final class prediction is calculated using majority voting across all trees. Decision trees can naturally handle multi-class classification problems. The intuition behind such an ensemble method is to improve overall classification accuracy [38] To understand Random Forest, a brief description of decision trees follows.

The root of the decision tree corresponds to the entirety of the dataset. Features are then derived from the dataset. The dataset is then split into subsets based on its features. The split creates branches for each possible value of the selected feature. The algorithm then recursively repeats the process of feature selection and splitting until a stopping criterion is met. Each leaf node represents the majority class among the data whose split lead to the leaf. When classifying a new input, the decision tree routes it through according to the same splits, until it reaches a leaf node, and the input is assigned the label of that leaf. [25]

There exist different criterion to determine how decision trees create splits in data, such as Gini Impurity, Entropy and Information Gain.

$$Gini = 1 - \sum_{i=1}^{n} p_i^2 \tag{2.9}$$

Gini Impurity is a measure of how often a randomly chosen element of a class would be incorrectly labelled if it were labelled randomly and independently according to the distribution of labels that exist within the dataset. The aim is to select a split based on the feature which provides the lowest score. The lower the score, the more homogeneous the labels are in groups associated with each split.

2.2 Neural Network NLP Approaches

Advanced approaches to creating hate speech detection models exist over the traditional. These include deep learning approaches such as: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Large Language Models (LLMs). Many feature extraction techniques also utilise the neural network (NN) architecture.

It's important to make the distinction between NNs and deep learning. Whilst a simple NN consists of a singular hidden layer between input and output, deep learning architectures specifically contain multiple hidden layers between input and output. Thus, deep learning models are better equipped for complex tasks compared to their shallow counterparts. We can establish that deep learning is a subset of NNs. This section provides an

overview of the foundations of the chosen deep learning models and feature extraction techniques.

2.2.1 Word2Vec

Word2Vec, is an architecture which serves as a feature extraction technique, where each word is represented as a dense, continuous, vector [34]. Building upon initial Feed-forward Neural Net Language Models (NNLM), these vectors are more commonly known as word embeddings [7], which aim to capture the semantic relationships of a word based on the context in which it occurs. The context refers to the group of words which surround a given target word. This allows for semantically similar words to be represented closer together in the vector space and share numerically similar embeddings. Word2Vec models are usually trained on extremely large corpus in order to achieve meaningful embeddings. There are two training approaches: Continuous Bag of Words (CBOW) and Skip Gram (SG).

The CBOW model aims to predict the target word from its surrounding context. It does so by aggregating the vectors of the context into a projection layer and using this to predict the target word (e.g. the vector for "cat" from the passage "The cat sat on the mat" would utilise said context).

The SG model aims to predict the surrounding context words using the target word. It does so by taking the target word vector as input into the projection layer and predicting words within a specified context window. This model is more computationally expensive, as it predicts vectors for multiple context words. However, it tends to achieve greater semantic accuracy in word embeddings compared to CBOW, with syntactic accuracy being similar when trained on the same data. [34].

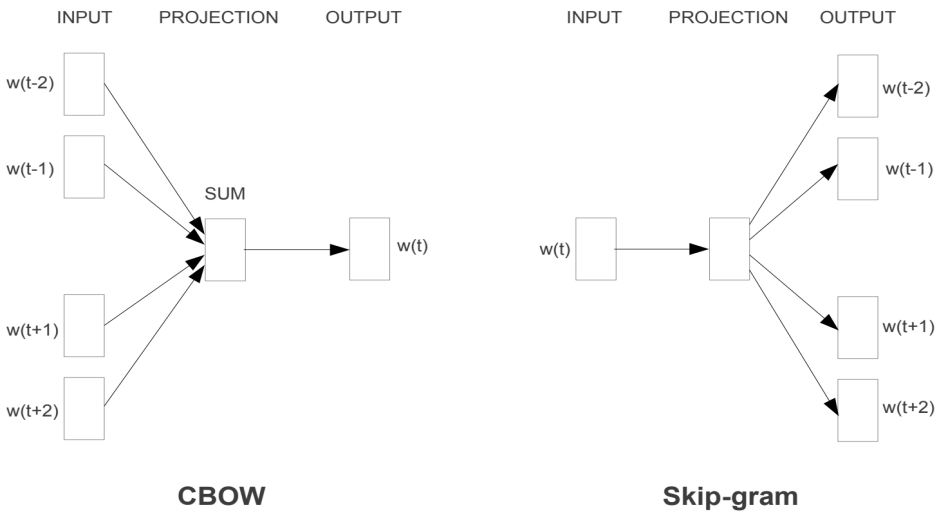


Figure 2.2: CBOW and SG architectures [34]

Each word in the corpus is initially represented as a high-dimensional vector and assigned random values. During training, the word vectors are updated based on the underlying objective function, which aims to position semantically similar words closer together in the vector space.

The benefit of embeddings lies in their ability to capture semantic relationships, which allows for better understanding of document content at the word level, beyond the simple presence of a word. The ability for embeddings to capture deeper linguistic patterns compared to a typical TF-IDF model may make them more robust at classification tasks. However, an issue with this approach is its inability to capture embeddings for OOV words which weren't encountered during the unsupervised, training process. For the task of hate speech detection, it's important to note that classification occurs at the document level, and so embeddings derived for each word within a document must be aggregated to produce a document vector representation. These methods will be explored in subsequent chapters examining methodology.

2.2.2 Doc2Vec

Whilst word embeddings generate vectors for individual words, this approach does not directly produce fixed-size, document vectors, which are needed as an accepted form of input for ML models. Whilst methods of aggregation can be used to achieve document vectors, this loses word order in the same way the BoW approach does. Doc2Vec, also known as Paragraph Vectors, is a feature extraction technique that produces document embeddings. [28]. Created as an extension of Word2Vec, their underlying architecture is similar, utilising a two-layer, neural network (NN) with the additional component of a document vector being used during the training process. There are two training approaches to implementing the Doc2Vec model: Distributed Memory (DM) and Distributed Bag of Words (DBOW).

The DM model works by associating each document and its corresponding words with unique vectors. The model then aims to predict the next word using the context, which is represented by an aggregation of the document and word vectors into a projection layer. This is analogous to the CBOW model, where the difference lies in the addition of a document vector, which serves as additional context.

The DBOW model works by associating each document with its own vector and uses this as an input into the projection layer. The model then aims to predict randomly sampled words from the document using the vector. This is analogous to the SG model employed by Word2Vec, where instead, the "target" word is now the document vector and there's no context words.

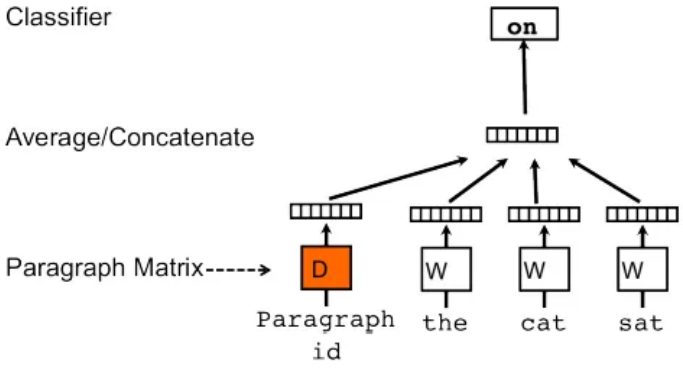


Figure 2.3: Distributed Memory (DM) model [28]

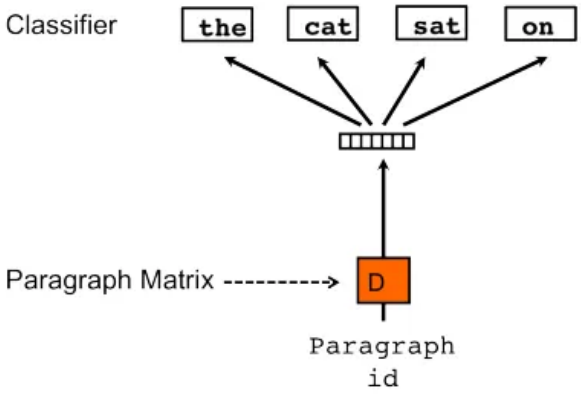


Figure 2.4: Distributed Bag of Words (DBOW) model [28]

Document Embeddings provide the correct dimensionality of input needed for many ML models, and so Doc2Vec provides an effective solution whilst also capturing semantic properties of words. Doc2Vec models based on the DM implementation can also take into account limited word order, providing an advantage over both the BoW approach and averaged word embeddings. This makes them an ideal feature extraction technique for a range of sequence classification tasks, including hate speech detection.

2.2.3 FastText

FastText is a word embedding technique used to represent words by its whole as well as constituent n-grams [9], [24]. It can be seen as an extension from Word2Vec, which produces unique vectors for each word but ignores morphology. Approaches to training word embeddings are similar to Word2Vec with CBOW and SG models being employed, albeit with modifications to allow for the derivation of n-gram vector representations. Thus, the representation of a word can be derived by summing the n-gram embeddings and the embedding for the word itself.

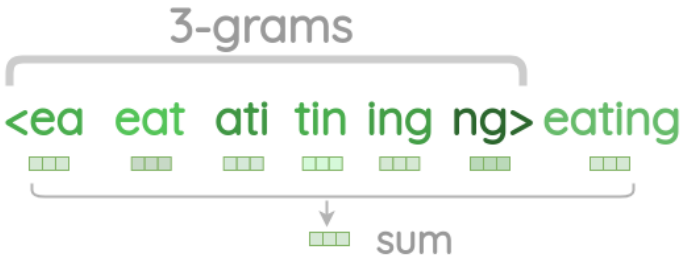


Figure 2.5: FastText derivation of word embedding from its constituent n-grams [12]

The incorporation of advanced techniques provides FastText with advantages over fixed word embeddings such as Word2Vec. It allows for the representation of OOV words, by representing them as a bag of character n-grams. In this way, any unseen words during the training process can be represented by the summed vector representations of its n-grams. It can also capture morphological properties of words by considering their internal structure, allowing for parameter sharing which can create more comprehensive vector representations for semantically similar words (e.g. eat, eating and eaten). In this way, nuances in the structure of a word can contribute to its position within the vector space. A potential disadvantage, however, is that the generation of n-gram vectors for each word encountered in the training process requires more computational space and time. This may be an issue depending on the size of the dataset the model is trained on.

2.2.4 Embeddings from Language Models (ELMo)

The suitability and vastness of word embeddings in tasks such as sequence classification is clear, but a drawback with approaches covered up until now is the inability to produce multiple vector representations for a given word. These embeddings can be seen as static, [56] lacking contextual awareness of the sentence and not being able to model the polysemous nature of language. The introduction of contextualised word embeddings aims to capture both the semantics and polysemous aspects of words

ELMo [42] is a method used to create contextualised word embeddings, whereby the vectors produced for each token is a function of the entire input sentence. Its name is derived from its underlying architecture, made up of a Bidirectional Long Short Term Memory (BiLSTM), a particular type of Recurrent Neural Network (RNN) which, for the purpose of ELMo is known more formally as a Bidirectional Language Model (BiLM) due to its application in language modelling [42].

ELMo is made up of three layers. The first layer is known as the embedding layer, a Convolutional Neural Network (CNN) [29] where input words are broken down at the character-level, from which pre-trained word embeddings are derived. The use of character embeddings allows ELMo to handle OOV words not encountered during training. This provides an initial vector representation of each word in a sentence before being

passed through the BiLM. The BiLM is made up of two layers, with each consisting of a forward and backward pass. This allows ELMo to effectively leverage both the preceding and following context for a given word. Each layer produces intermediate vectors for each word and extract varying linguistic features (syntax and semantics). The initial vector along with the 2 intermediate vectors are then combined through a weighted sum, producing the contextualised word embeddings [42].

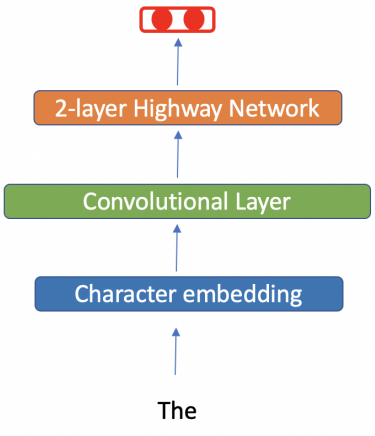


Figure 2.6: Example illustration of how the word vector representation of "The" is generated from its morpheme composition [13]

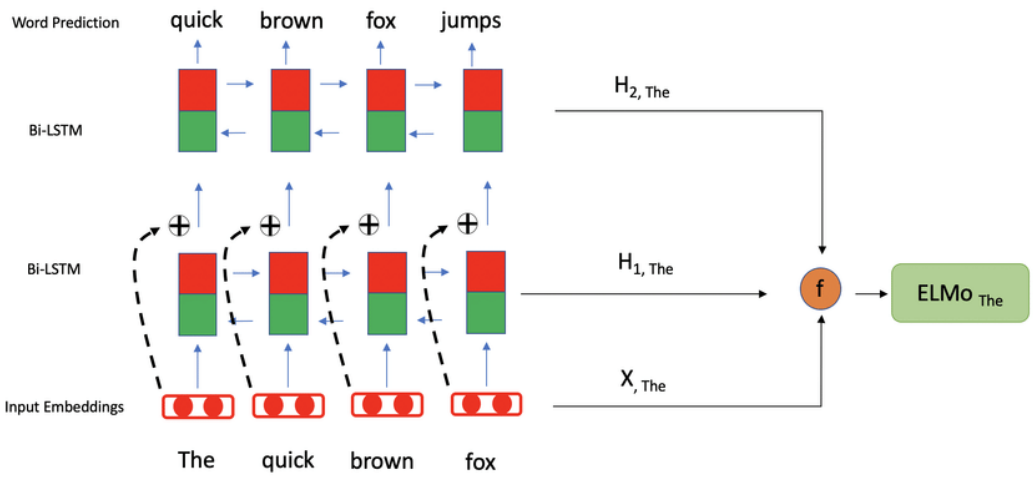


Figure 2.7: Example illustration of the ELMo Architecture predicting the word vector for the word "The", given the context "quick brown fox jumps" [13]

Contextual embeddings provide an improvement over static embeddings in semantic and polysemous word representation. Regarding hate speech detection, this could prove to be useful, as hate speech is context-sensitive. (E.g. the word "jihad" is inoffensive, but could be used to incite hate speech when used against an ethnic minority, thus having multiple vector representations can lead to greater clarity in detecting offence.) However, as ELMo only embeddings at the word level, averaging techniques need to be used in order to obtain document-level vectors, a common drawback of all word embedding

approaches. Consequently, there's an inability to preserve word-order when used to represent documents.

2.2.5 Bidirectional Long Short-Term Memory (BiLSTM)

BiLSTMs are a type of RNN and a class of deep neural networks that take word order into account. In RNNs, the current state is predicted based on the current input as well as previous hidden state. However, an issue with RNNs is that they suffer from the vanishing gradient problem. This occurs during model training when backpropagation is performed to update model weights. As the gradient is propagated through multiple layers, it can get small enough to be negligible, preventing weights associated with early inputs being updated. They lack the ability to effectively learn long term dependencies for inputted sequences. RNNs are unable to capture information in longer sentences, impacting their ability to detect hate speech effectively.

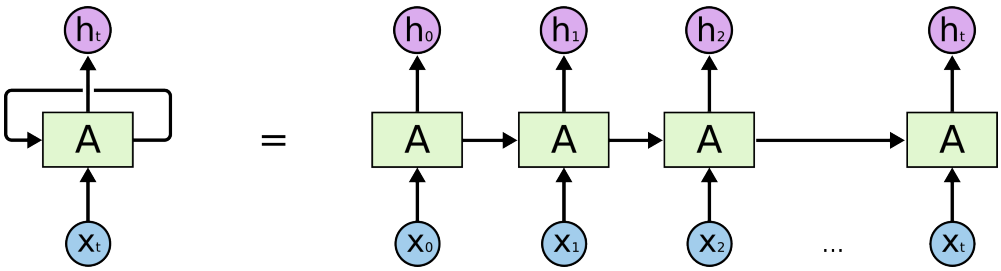


Figure 2.8: RNN/LSTM high-level architecture diagram [55]

LSTMs were designed to overcome this issue by employing the use of memory cells and gates, performing better with regard to long term dependencies. BiLSTM utilise the LSTM architecture, but inputs into models are provided as normal as well as in reverse. Thus, the network contains information not only on past states but future states as well, allowing for better predictions to be made. In LSTM units, there exist three gates. The forget gate allows the model to forget irrelevant information from the previous state. The input gate filters incoming data and updates the cell state with new information that is deemed important. The output gate influences the next hidden state and the network's output at the current time-step.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.10}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.11}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.12}$$

Here, f_t, i_t and o_t are the formulas for the forget, input and output gates. σ is the sigmoid activation function, W_f, W_i and W_o are weight matrices for the forget, input and output gates. b_f, b_i and b_o are bias terms for the forget, input and output gates. h_{t-1} is the previous hidden state and x_t is the current input.

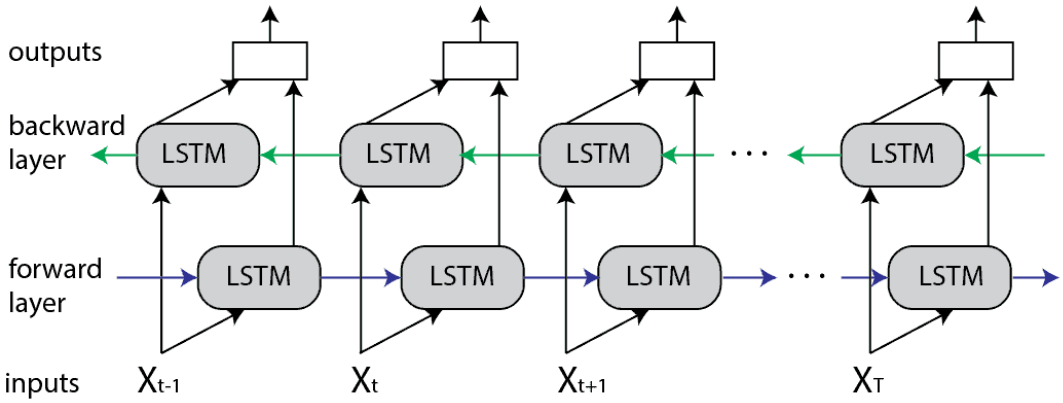


Figure 2.9: BiLSTM high-level architecture diagram [55]

2.2.6 Transformers (BERT)

BERT [17] is a type of LLM which utilises the encoder-based transformer architecture. A key component of this is the attention mechanism [49], which allows models to focus on the most relevant parts of an input sequence, enhancing the ability of the model to capture relationships between words. Each transformer block consists of two main components. First is the self-attention layer, which allows the model to weigh the importance of each word in a sequence and allow it to focus on the most important parts. Second is a feedforward network which takes as input the outputs from the self-attention layer and applies additional transformations to the inputted data. BERT, as with many LLMs, consists of multiple layers of Transformer blocks to allow for simultaneous processing of inputs. BERT can be applied to different downstream tasks when fine-tuned with additional layers on top. In the case of text classification, a dense layer is often combined with the softmax activation function. This can be seen in Figure 2.10

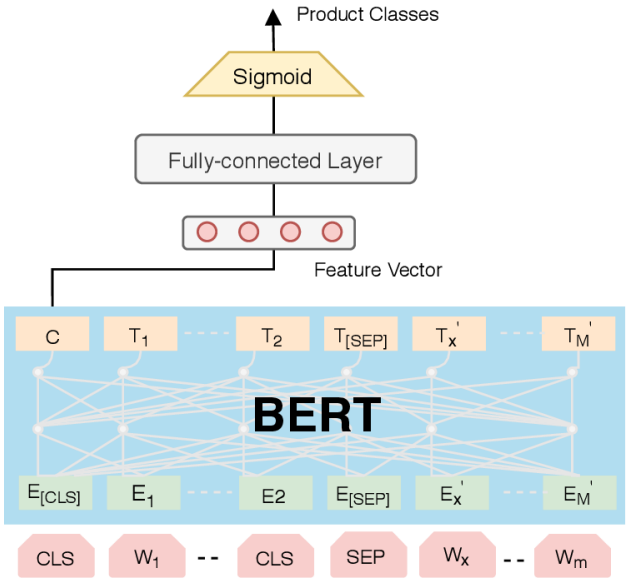


Figure 2.10: BERT high-level architecture for sequence classification [54]

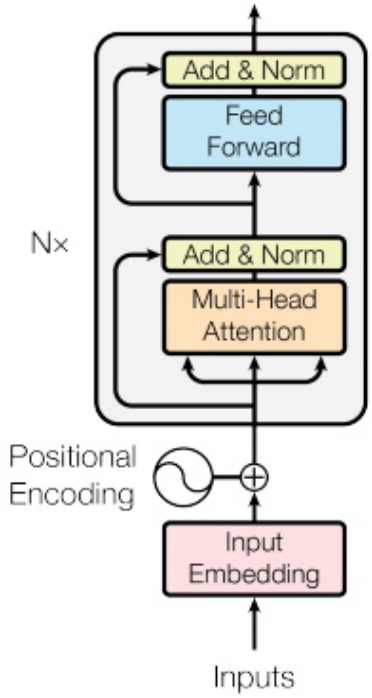


Figure 2.11: Encoder-based Transformer architecture diagram [49]

2.3 Literature Review

Over the years, there have been many published journals which have focused on hate speech detection. Being such a prevalent issue, it's important that hate speech detection continues to be investigated in order to find ways to mitigate its impact on society. In this section, relevant literature will be discussed in order to provide context for the research being conducted in this paper. I will focus on literature which has investigated the impact of various factors on hate speech detection performance.

2.3.1 Data Annotation

NLP methods related to hate speech detection are typically supervised, requiring labelled data. Annotations are mostly performed manually and are logistically planned out by authors of research. Annotations can either be performed by experts within the field of research or be crowdsourced. The difference in domain knowledge can result in differing levels of quality, which can impact the effectiveness of hate speech detection models.

Nobata et al., as part of wider research into hate speech detection models, investigated whether untrained annotators were able to effectively label instances of hate speech [37]. Annotators were crowdsourced from Amazon Mechanical Turk (AMT) and compared to in-house, trained annotators. Instances of hate speech were categorised as a binary problem. It was found that the untrained annotators had an agreement rate of 0.867 and a Kappa score of 0.401. Trained annotators were able to produce an agreement rate of 0.922 and a Kappa score of 0.843. For fine-grained classification of subtypes of abuse, kappa scores dropped for both groups, but were comparatively much poorer for untrained annotators. Crowdsourced annotations from untrained individuals may not be effective for the task of hate speech labelling, which could ultimately impact resulting detection models. This is likely to stem from misunderstanding what constitutes a piece of text as hate speech.

Ross et al. provided a more fine-grained study investigating annotator reliability and its consequences on hate speech detection [44], arguing that accuracy of annotations is essential for establishing an upper bound on performance as well as allowing models to learn the characteristics of hate speech. They presented two different groups with potentially hateful messages and showed one group a comprehensive definition of hate speech beforehand. They found that in both cases, annotator reliability was low and providing a definition of hate speech did not improve results. In my opinion, this highlights the need for more comprehensive and understandable guidelines for hate speech annotation. In addition, poor performance can be attributed to a lack of understanding of the area.

2.3.2 Hate Speech Detection Models

It has also been the case that ML-based NLP models have been employed throughout the literature in order to: establish baseline performance, examine the effectiveness of each approach to detecting hate speech, and determine the state-of-the-art.

Davidson et al. researched the ability of hate speech detection models to distinguish hateful posts from other forms of offensive language [15]. This distinction is important, as it's common for offensive language to be found in many social media posts, but not all offensive posts are considered hate speech. They employed a logistic regression model

capable of performing multi-class classification between hate, offensive and neutral categories, using L1 regularization to reduce data dimensionality. They were able to achieve an overall precision of 0.91, recall of 0.90 and F1 score of 0.9. Upon closer inspection of results, the model confused instances of hate speech and offensive language, misclassifying 40% of hate speech instances.

Asogwa et al. did the same for traditional NLP approaches to hate speech detection in order to create interpretable models to automatically identify instances of hate speech on social media platforms [6]. By employing SVM and Naive Bayes (NB) techniques in a binary manner, they were able to achieve accuracies of 0.98 and 0.80 respectively.

Gibert et al. took the task of comparing NLP approaches to hate speech detection in order to derive baseline performances for a created dataset [16]. They produced three different models: SVM, CNN and LSTM. Classification was tackled in a binary manner. It was found that overall, the LSTM classifier achieved best results, but SVMs were also able to achieve similar performance employing a simple BoW approach, highlighting how the minimal gap in performance gains between traditional and deep learning models.

Similarly, Lu also carried out a comparative analysis of various NLP approaches, in order to find the most effective methods of mitigating the presence of hate speech on social media [31]. She investigated the following models: Naive Bayes, Random Forest, BiLSTM and BERT and classified hate speech in a binary manner. She found that BERT was most effective, resulting in an accuracy of 0.84 and F1 score of 0.77. In addition, Naive Bayes was also able to demonstrate good performance with an accuracy of 0.69 and F1 score of 0.67, which were better results than produced by BiLSTM.

A stronger focus on deep learning approaches to hate speech detection was taken by Saleh et al., who employed BERT based models as well as BiLSTM [45]. They leveraged domain-specific and agnostic word embeddings, which served as the embedding layer to the LSTM. Each model was individually trained on three open-source datasets for binary classification. They found that BERT was able to outperform all combinations of BiLSTM with domain-specific and agnostic word embeddings, with an F1 score of 0.96 on a combined dataset from all three sources. However, it's also noted that domain-specific word embeddings are better able to detect hate terms and abbreviations commonly found on social media compared to BERT. A potential issue with this research is the conflation of various datasets into one may produce uninterpretable results, as each individual dataset possesses different sub-labels for categorising hate-speech.

Mathew et al. also focused on examining the performance of state-of-the-art models, as well as how aspects of bias and interpretability in hate speech are accounted for by each approach [32]. The issue of bias is common in hate speech detection, where misclassification of comments can occur due to the inclusion or referral to commonly attacked

identities (Muslims, Blacks and Homosexual). The following models were used: CNN-GRU, BiLSTM, BiLSTM with attention, BERT and variations of the same models which utilise human annotator rationale as a form of supervised attention. BERT with supervised attention provided the best performance for hate speech detection in both accuracy and F1 score, followed by BERT without rationale. BiLSTM with attention and CNN-GRU were able to produce similar performance. What's most interesting is that the use of supervised attention was able to reduce unintended bias for all approaches, making it a feature researchers would look into gathering as part of future work.

2.3.3 Feature Extraction Techniques

When dealing with supervised NLP approaches to hate speech detection, a key step in the pre-processing pipeline is to decide how to extract relevant features from data to convert it into a format understandable to ML models. Feature extraction turns unprocessed inputs (most commonly in the form of text) into a set of useful features (numbers) that machine learning models can be trained on to identify patterns within the data and thus make useful predictions. There are many methods of extracting numerical features with differing levels of complexity. Existing literature has also focused on the impact of feature extraction techniques on hate speech detection performance. The extraction of relevant features plays a crucial role in NLP, and the existence of multiple techniques leads to uncertainty over which is best for a given application. Whilst this has been touched upon in the previous subchapter, we look into more detailed results across the literature here.

Gupta and Hovy carried out a comparative analysis of feature extraction for hate speech detection on tweets [20]. They used both word and document level embeddings such as Word2Vec, Glove, Doc2Vec and Skip-Thought. They leveraged both domain-agnostic and domain-specific (pertaining to social media and hate speech) embeddings and compared performance across three different datasets. It was found that domain-specific embeddings outperformed domain agnostic embeddings. Doc2Vec performed poorly across all embeddings, with the lowest accuracy and F1 measures across the board. Word2Vec was the best performing word embedding across all three datasets, and Skip-Thought was the best performing document embedding. This was only performed on a Logistic Regression model, and so leaves room for comparative analysis using a range of NLP approaches. Furthermore, the conflation of categories into a binary classification problem fails to distinguish between offensive and hateful examples, confusing what's considered hate speech.

Languages other than English have also been looked into. Mahamat et al. aimed to detect instances of hate speech in the Chadian and French languages [4] and compared Word2Vec, Doc2Vec and FastText embeddings on hate speech detection for four different NLP approaches: Logistic Regression, Support Vector machine, Random Forest and

K-Nearest Neighbour. They found that out of all combinations, FastText paired with an SVM model produced the best performance for detecting instances of hateful posts. SVM's also outperformed all other approaches, with the highest average accuracy across all embeddings. FastText embeddings were found to outperform Doc2Vec and Word2Vec across all NLP approaches for detecting hateful, offensive and insulting instances of posts.

Jain et al. carried out a comparative analysis of word embeddings (Word2Vec, Glove, FastText and ELMo) in order to determine their suitability to hate speech detection [23]. As word embeddings have evolved over time, they've been able to better capture the complexities of language, and it is thought that this can lead to improvements in detecting hateful instances on social media. They trained a neural network model using a hate dataset consisting of 2000 tweets and varying the embeddings. It was found that ELMo outperformed all other embedding methods, with the highest accuracy and F1 score. This was followed by FastText and Glove, with Word2Vec being last. The ability for ELMo to differentiate word context allowed it to perform better than all other methods. This research could be further expanded upon by using a larger dataset which differentiates hate speech from racist/sexist posts.

Furthermore, the impact of different feature extraction techniques was investigated by Alsagheer et al. as part of wider research into detecting hate speech against athletes on social media [5]. In this study, BoW, Word2Vec, Perspective Scores and BERT embeddings were used to each train a Logistic regression model. Perspective Scores are features derived from trained classifiers which categorise documents into different labels. Data was collected from social media and consisted of comments and posts directed against three Olympic athletes. When trained to detect hate speech against each individual athlete, Word2Vec obtained the best results in terms of accuracy and F1 score (0.86 and 0.91). However, when combining all the data, Perspective Scores obtained the best results. This research shows how the composition of a dataset is important in determining the best feature extraction technique. Again, the use of a single NLP approach to hate speech detection could be expanded upon by investigating multiple approaches.

Reference	Model	Results
Davidson et al. [15]	LR	Precision: 0.91 Recall: 0.90 F1 Score: 0.90
Asogwa et al. [6]	SVM and NB	SVM Accuracy: 0.98 NB Accuracy: 0.80
Gibert et al. [16]	SVM, CNN and LSTM	SVM Accuracy: 0.74 CNN Accuracy: 0.70 LSTM Accuracy: 0.78
Lu [31]	NB, RF, BiLSTM and BERT	NB F1 Score: 0.62 RF F1 Score: 0.54 BiLSTM F1 Score: 0.60 BERT F1 Score: 0.77
Saleh et al. [45]	BERT and BiLSTM (with domain-specific and agnostic embeddings)	BERT F1 Score: 0.96 BiLSTM (domain-specific): 0.94 BiLSTM (domain-agnostic): 0.94
Mathew et al. [32]	CNN-GRU, BiLSTM and BERT	CNN-GRU: 0.61 BiLSTM: 0.58 BERT: 0.67

Table 2.1: Summary of related work on hate speech detection

2.3.4 Datasets

With this research also comes the creation of curated datasets, which has provided others with a means to carry out further research. The composition of these datasets, including size, balance, annotation quality and cardinality has a large effect on hate speech detection performance.

The table comprises some of the most popular and widely used hate speech datasets, some of which have also been used throughout research in previous sections. The composition of each dataset vastly differs from one another, which impacts the effectiveness of the NLP approaches used to detect hate speech.

Dataset	Cardinality	Data Split	Total	Source
Davidson [15]	Multi-Class	Hate - 1430 Offensive - 4163 Neither - 19190	24802	Twitter
HateXplain [32]	Multi-Class	Hateful - 5935 Offensive - 5480 Normal - 7814	19229	Gab and Twitter
Dynamically Generated [50]	Binary-Class	Hate - 22262 Not Hate - 18993	41255	Synthetically Generated
White Supremacy Forum [16]	Binary-Class	Hate - 1119 Not Hate - 8537	9656	Stormfront
Waseem and Hovy [51]	Multi-Class	Sexist - 3383 Racism - 1972 Neither - 11559	16914	Twitter
ETHOS [36]	Binary-Class	Hate - 443 Not Hate - 555	998	YouTube and Reddit
Founta [18]	Multi-Class	Hateful - 4948 Abusive - 27037 Normal - 53790 Spam - 14024	99799	Twitter

Table 2.2: Summary of widely used hate speech datasets

Chapter 3

Methodology

The following chapter presents the methods employed throughout the project, detailing what aspects of hate speech detection were investigated, the experimental design of each experiment and the rationale behind the decisions taken. Methods of data collection, experimental design and analytical approaches are detailed here.

3.1 Dataset Collection and Composition

Whilst there exists many methods and platforms to retrieve data, I decided to use existing datasets. This was because the gathering of the required amount of data to investigate hate speech detection is time-consuming and warrants a larger period of research time to carry out. Secondly, the data would then also need to manually annotated, a process which requires a large amount of time and resources, which weren't available for this project. Despite this, many hate speech datasets which have been reliably annotated exist online. Datasets which were considered for use are shown in Table 2.2

1. **Curated Hate Speech (Auxiliary Dataset)** - A collection of existing hate speech datasets from various sources which have been compiled into one [35]. It provides examples of posts and comments from a range of social media platforms. Its purpose was for use in training word embedding models (Word2Vec, Doc2Vec and FastText) in order to provide domain-specific vector representations.

2. **HateXplain (Primary Dataset)** - A benchmark hate speech dataset including annotator rationale as well as target community annotations [32]. This serves as the main dataset used throughout the project and was used to train and test the various hate speech detection models.

3.1.1 Curated Hate Speech

The Curated Hate Speech dataset is composed of 726,120 text examples from social media. The dataset was already filtered through a pre-processing pipeline to allow for immediate use by fellow researchers. The pre-processing steps were as followed:

1. Removal of unwanted characters (hyperlinks, emojis, symbols)
2. Expansion of word contractions and acronyms
3. Correction of grammatical errors
4. Number-to-text conversion
5. Punctuation removal
6. Correction of misspelt profanities
7. Data augmentation

The decision to use such a dataset for training embedding models is due to its large size and varied vocabulary specific to the domains of hate speech and social media. The size helped to facilitate a comprehensive understanding of the language and provided a sizeable vocabulary of word vectors to learn. These word vectors also included terms commonly found on social media, including hateful slurs, which allowed for contextually relevant embeddings to be learnt through the training process. Whilst the dataset contained labels pertaining to whether each document is considered hate speech, these were ignored, as only the textual content was needed as part of the unsupervised training process.

Pre-processed Text	**Post-processed Text**
.so YOU BITCH NIGGAS MUST NOT SAY SHIT.....YOU HEARD?	so you bitch niggas must not say shit you heard
I have never been called a kike or a “dirty jew” until @user began his campaign. His hateful rhetoric... @URL	i have never been called a kike or a until began his campaign his hateful rhetoric

Table 3.1: Pre-processed vs post-processed documents from Curated Hate Speech

3.1.2 HateXplain

HateXplain is composed of 19,229 posts gathered from Gab and Twitter (now known as X). Gab is a far-right microblogging site where it’s much more common to come across hateful posts and thus is unlikely to employ hate speech detection tools. Tables 3.2, 3.3 and 3.4 show the most used words in the documents for each label.

AMT was used for the annotation process. Each post contained three annotations, and the most annotated label for each document was used as the final label.

Word	Frequency
nigger	2338
white	1181
kike	1045
jew	776
like	686
faggot	474
fuck	437
muslim	419
get	402
people	390

Table 3.2: Ten most used words in documents labelled as hate

Word	Frequency
white	882
bitch	661
like	637
retarded	631
ghetto	479
woman	474
people	408
muslim	382
faggot	378
jew	377

Table 3.3: Ten most used words in documents labelled as offensive

Word	Frequency
white	1995
woman	862
people	836
like	724
muslim	618
black	533
get	476
one	462
immigrant	459
raped	449

Table 3.4: Ten most used words in documents labelled as normal

Label	**Twitter**	**Gab**	**Total**
Hate	708	5227	5935
Offensive	2328	3152	5480
Normal	5770	2044	7814
Total	8806	10423	19229

Table 3.5: Distribution of labels across Gab and Twitter

3.2 Data Pre-processing

Real world data from social media contains a large amount of noise. To improve the accuracy and performance of hate speech detection systems, the data must be formatted in such a way that provide models with the best opportunity to learn patterns within the data. This section provides an overview of the pre-processing steps used on the HateXplain dataset.

3.2.1 Text Cleaning

The first stage of pre-processing the data was to clean the text. This involves multiple techniques to remove unwanted words, characters, symbols and any other textual data considered unimportant.

1. **Lowercasing**: All words within the dataset are lowercased to ensure consistency throughout the dataset and ensure compatibility with all models.

2. **URL Removal**: URLs are removed as they provide no benefit in detecting hate speech.
3. **Masking**: This was done to hide the names of users mentioned in posts to provide anonymity. Any numbers used within posts are also masked to provide consistency across the dataset. Specific numbers also do not give any indication to the hatefulness of a post.
4. **Non-word and White-space Removal**: This was done to remove any erroneous characters.
5. **Emoji Removal**: Whilst emojis provide annotators with additional context as to how a post should be interpreted, they provide little benefit to the models being trained on.

3.2.2 Tokenization

The next step was tokenization, which is the transformation of sentences into individual units. There are many ways to perform tokenization, including at the character, word or subword level. Word level tokenization was selected due to its simplicity and compatibility with NN-based feature extraction techniques.

3.2.3 Word Contraction and Acronym Expansion

Word contractions as well as common acronyms used on social media were expanded. This was done to provide a standardised format of text across both HateXplain and the Curated Hate Speech datasets, allowing for more meaningful document vectors to be derived.

3.2.4 Punctuation Removal

This was done to simplify the data and provide consistency, reducing the feature space and making data easier to process for both traditional and NN approaches. The presence of punctuation added unnecessary complexity to text data and provided little benefit to the task of hate speech detection. It also allowed models to focus on words which were considered more important in detecting hate speech.

3.2.5 Data Split

The HateXplain dataset was split into training and test subsets. The dataset was split 80/20 and stratified sampling was used, allowing for proportional, representative samples for each set.

3.3 Experiment 1

The objective of the first experiment was to perform a comparative analysis of various feature extraction techniques, as a means to improve the hate speech detection capabilities of ML-based NLP approaches. In addition, the aim was to identify the most effective vector representations of textual data, contributing to the development of more accurate and reliable hate speech detection systems. This section describes the methods used to accomplish this and the rationale behind each of them.

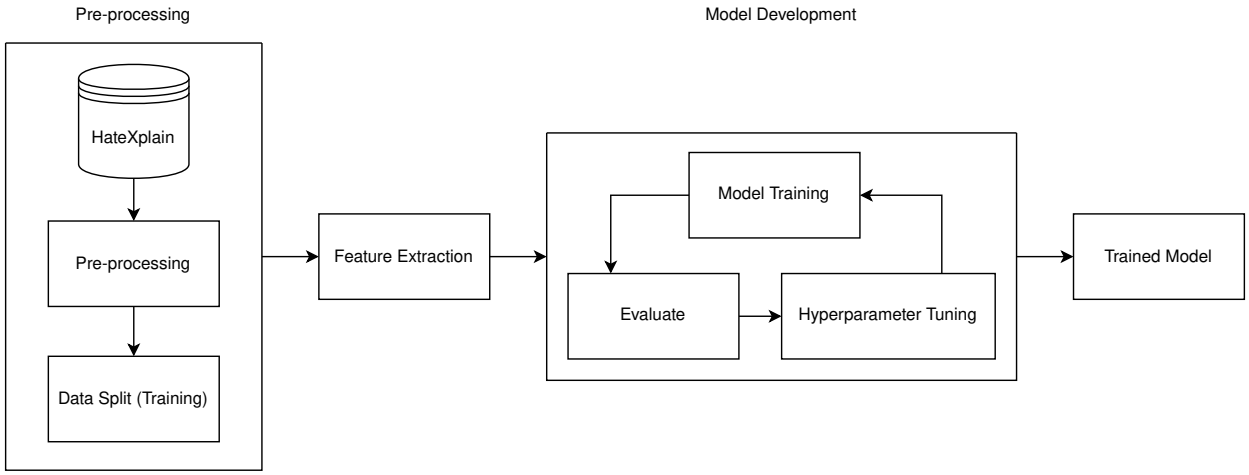


Figure 3.1: Model creation process for Experiment 1

3.3.1 Feature Extraction Rationale

Five feature extraction techniques were used, as discussed in Sections 2.1 and 2.2. They are split into three categories, dictated by their structure, which also determined how they were employed within the experimental design:

- Traditional: **TF-IDF over Bag of Words**.

Considered a traditional method, it involves no training or any use of NNs. TF-IDF was selected due to its well-regarded status in the field of NLP and text analysis. Furthermore, much of the literature uses TF-IDF as a baseline approach in hate speech detection models, and it has demonstrated good performance.

- Static Word Embeddings: **FastText**, **Doc2Vec** and **Word2Vec**.

These models were trained from scratch using the auxiliary dataset. They were selected due to their usage in existing literature and known performance levels in the hate speech domain. Furthermore, all three embeddings provide a unique approach to word embedding generation, from Word2Vec's context-based vectors, FastText's sub-word vector representations and Doc2Vec's document vectors.

- Deep Contextualised Word Embeddings: **ELMo**.

This model differs from others in that it is able to generate multiple contextualised word vectors for a single word. A pre-trained model was used as opposed to training one from scratch. This is because a significant amount of text data would be needed to train an ELMo model, and pre-trained models have shown to be effective in hate speech detection [23].

3.3.2 Model Selection Rationale

As discussed in Section 2.1, three different NLP models were used: **SVM**, **RF** and **LR**. Using three different models allowed for a comprehensive and robust evaluation into the effectiveness of each feature extraction technique and helped to determine which feature extraction technique is most consistent.

I used traditional NLP models as the foundation for this experiment, as they've been shown to produce performance comparable to that of deep learning approaches in related literature. Traditional approaches also have the benefit of computational efficiency when compared to deep learning approaches, making them a well-suited option for developing multiple models. The use of these particular ML models lies in the unique characteristics of each, with RF being an ensemble method, SVMs being a discriminative classifier and LR being a simple, linear classifier. The range of different methods allowed for robust evaluation.

3.3.3 Experimental Design

The experiment can be split into three stages: **Pre-processing**, **Feature Extraction** and **Model Development**.

3.3.3.1 Pre-processing

Two datasets were used: **HateXplain** and **Curated Hate Speech**. Curated Hate Speech was already pre-processed (steps in Section 3.1.1) and wasn't utilised in any way by the TF-IDF and ELMo features. The pre-processing pipeline for HateXplain has also been outlined in Section 3.2. Here, I clarify the difference in the pre-processing pipeline for TF-IDF features.

TF-IDF features also included stop word removal and lemmatization. Stop word removal removed common words that offered little semantic meaning. Lemmatization reduced inflections to common base words. As TF-IDF features produced large and sparse vectors, it benefitted from dimensionality reduction and reduced noise. By reducing the number of semantically similar words and allowing for greater focus on hate-related, content words, this improved the quality of extracted features.

3.3.3.2 Feature Extraction

For **TF-IDF**, document vectors were created, where each document was represented by the TF-IDF scores for the words within them.

For **ELMo**, a pre-trained model was loaded. Word embeddings were generated for each word within a document, and were averaged by taking the mean to obtain a document vector. Subsequently, vectors for documents within the training and test subsets were produced.

$$\text{Document Vector} = \frac{\sum_{i=1}^{n}(w_i)}{n} \tag{3.1}$$

Here, w_i represents each word embedding in a document.

For **Word2Vec**, a model is trained from scratch. The Curated Hate Speech dataset was used to train the model, and optimal hyperparameter values were selected (discussed in Section 4.2). Word2Vec models allowed for a selection of hyperparameters such as:

- **Vector Size**: Dimensionality of each word embedding and thus the resulting document vector
- **Window**: How many words around the target word are considered during the training process
- **Minimum Count**: The minimum number of times a word must be seen within the training data for a resultant word embedding to be derived
- **Skip Gram**: A binary value that determines whether to use the CBOW (0) or SG (1) approach to learning word embeddings.
- **Epochs**: Number of passes on the training dataset

Each document from the training and test subsets were inputted into the model, where an embedding representation for each word within the document is generated. Document vectors were derived by taking the mean average of all word embeddings.

For **FastText**, the same methods of Word2Vec apply, albeit with a FastText model trained from scratch.

For **Doc2Vec**, similar methods to Word2Vec and FastText apply. We trained a model from scratch using the Curated Hate Speech dataset and selected appropriate hyperparameters. Document vectors were inferred directly from the model and no averaging techniques were utilised.

3.3.3.3 Model Development

A key step of the model development process lies in **hyperparameter optimisation**. Each model has its own set of hyperparameters which need to be optimised. This is important,

as it results in the most optimum performance being achieved given the constraints of a model and its underlying dataset. In the domain of hate speech detection, improvements in detection can have a tangible impact on social media platforms and wider society.

Optimal hyperparameters for each model were found using the **grid search** approach. This method searches over the entire space of the given hyperparameter values and determines the best combination in a brute force manner. Whilst it's been shown to find the most optimal values, this is generally only effective for small search spaces. As the search space increases, grid search becomes increasingly slow and ineffective [46]. Existing literature which has examined hate speech detection is used to establish a starting point for hyperparameter values, reducing the potential search space.

Due to the drawn out process of using grid search, only a subset of the training data was used, whilst all training data was used to train the final model. K-fold cross-validation was conducted as part of evaluating the grid search process to ensure reliability in the derived hyperparameters. Each set of hyperparameter values were then empirically evaluated against one another using the macro F1 score. (More details of the evaluation strategy used during the development phase can be found in Section 3.6).

After optimal hyperparameters were selected for each model, they're trained on using the full set of training data.

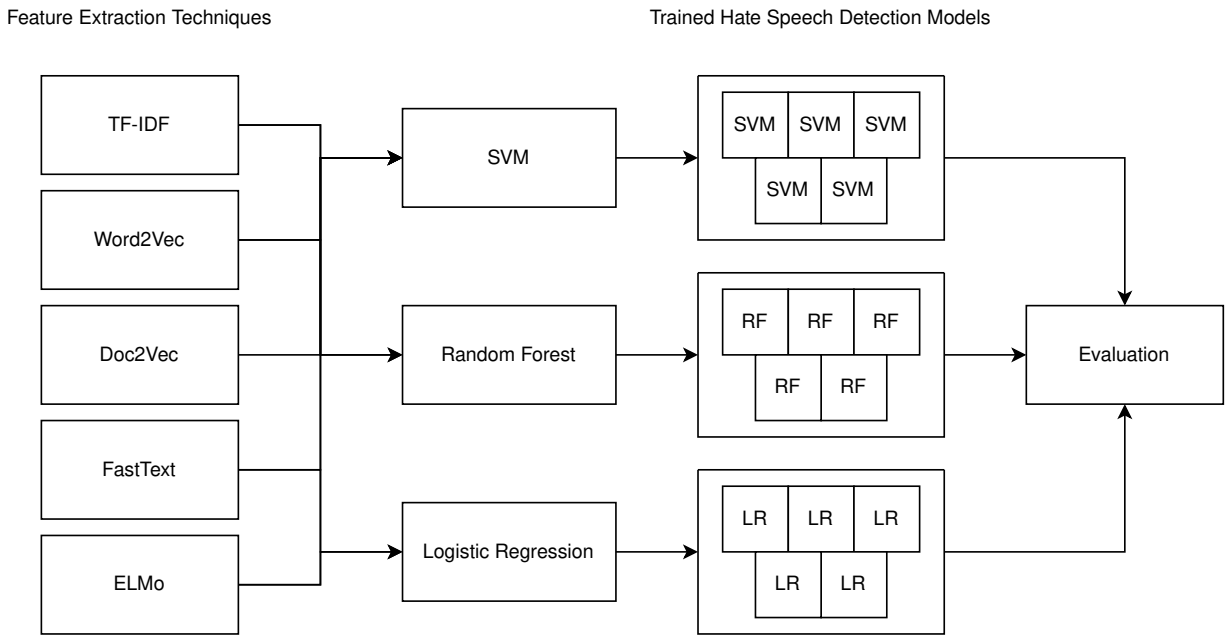


Figure 3.2: Summary of Experiment 1

3.4 Experiment 2

The second experiment was built on top of the previous one, using the best performing traditional models and carrying out a comparative analysis with deep learning models

for hate speech detection. The aim of the experiment was to determine which approach produces state-of-the-art results and to evaluate the disparity in performance. By determining the most optimal approach to hate speech detection, this experiment provides guidance on the best course of action when implementing hate speech detection systems on a smaller, restricted scale or an expensive, larger scale. This section outlines the methods used to achieve this.

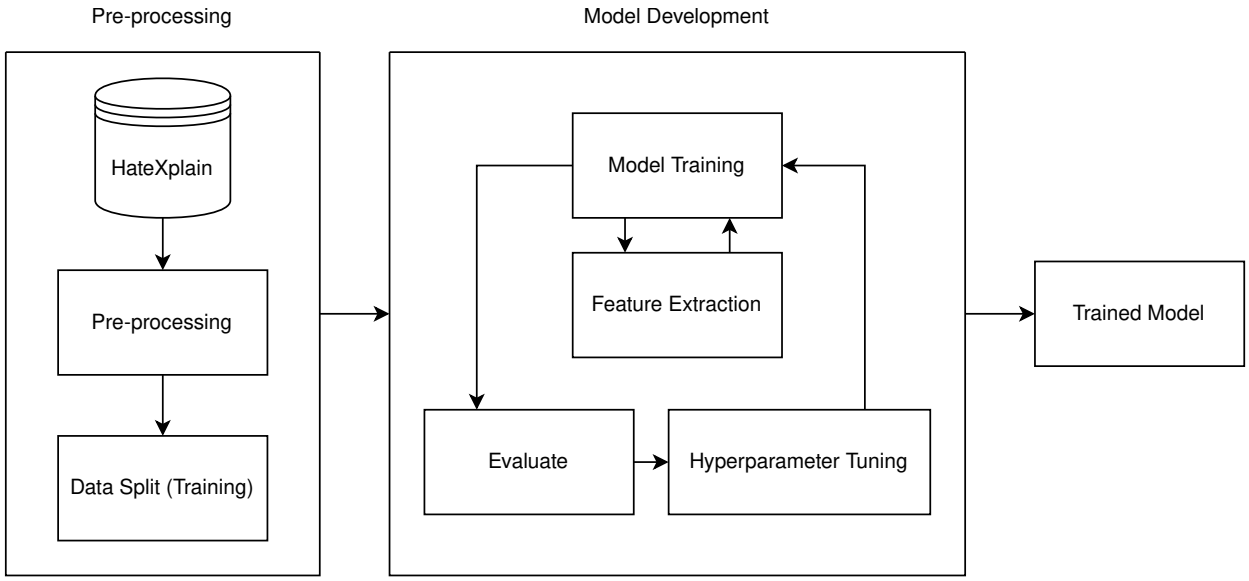


Figure 3.3: Model creation process for Experiment 2

3.4.1 Model Selection Rationale

Three traditional models were included in this experiment (an SVM, RF and LR) and three additional deep learning-based NLP models were developed for this experiment: **BiLSTM**, **BERT** and **RoBERTa**. Using a range of different models to represent deep learning approaches allowed for a comprehensive evaluation.

The decision to utilise BiLSTMs models is rooted in related literature. Whilst many deep learning models have been employed in hate speech detection, BiLSTMs perform well due to their ability to capture long term dependencies between words in both directions. In addition, there's a tendency for literature to focus more on BiLSTMs compared to counterparts like LSTMs or RNNs, as it's been well established that BiLSTMs provide better performance in detecting hate speech.

Many different pre-trained LLM's exist today, such as BERT, GPT and LLaMA. The choice to use BERT over others is due to its extensive use in hate speech detection tasks across the literature, allowing for comparative analysis. The use of other pre-trained LLMs in the domain of hate speech detection is few and far between. Where BERT and GPT models were both used, BERT produced better performance [52]. Using a pre-trained

LLM over training one from scratch is preferred in this case due to the computational costs required to so.

RoBERTa shares the same underlying architecture as BERT but with an enhanced training procedure over more data and for a longer duration, leading to performance improvements [30]. This model was implemented to explore whether the improvements can be observed in the domain of hate speech detection.

3.4.2 Experimental Design

The experiment is split into two stages: **Pre-processing** and **Model Development**.

3.4.2.1 Pre-processing

The pre-processing pipeline followed the same steps outlined in Section 3.2. The intuition behind keeping stop words and not performing additional pre-processing (as opposed to Subsection 3.3.3.1) is due to the ability of deep, NNs to capture longer-term dependencies between words and better understand contextual nuances in language. As can be seen in Table 3.6, the presence of stop words alters the sentiment of the sentence.

The data splitting strategy followed a similar process to Subsection 3.2.5, with a slight difference. A validation dataset is derived from the training set, and was used to find optimal hyperparameters as part of the development process. 20% of the training data was used for validation.

Stop word removal	Stop word inclusion
'hate' 'muslims'	'i' 'do' 'not' 'hate' 'muslims'

Table 3.6: Comparison of sentences with and without stop words

3.4.2.2 Model Development

The BiLSTM model was trained from scratch, whilst BERT and RoBERTa utilised pre-trained models and were fine-tuned. This section provides a brief overview of these steps (development steps for RoBERTa follow the same as BERT).

Like traditional methods, relevant features need to be extracted from the data in order to be understood by models. This process differs with deep learning models, where feature extraction is integrated into the development process, as opposed to existing as its own distinct step.

When creating a BiLSTM model, **feature extraction** was performed in order to convert textual data into an appropriate dense vector format. This can take the form of learnt word embeddings from an external model (such as Word2Vec) or the BiLSTM can derive

embeddings itself through the training process. The latter was preferred as it utilised the full training capabilities of the BiLSTM to determine how well it learnt word embeddings. An **embedding layer** is used, mapping the sequence of word indexes derived from the training data to dense vectors, which are iteratively updated through the training process [10].

For BERT, as a pre-trained model was used, it's already capable of extracting features from input sequences. The training data was tokenized in a format specific to BERT, with the addition of a **[CLS]** token. The training data was fed into the pre-trained BERT model, which produced contextualized embeddings for each token in the input sequence. The **[CLS]** token is a document level, vector representation of the entire input sequence, which was subsequently fed into the classification layer to obtain label predictions [27]. The features extracted from input sequences were adjusted throughout the fine-tuning process as weights for the pre-trained model and classification layer were updated. A visual representation of this process can be seen in Figure 3.4.

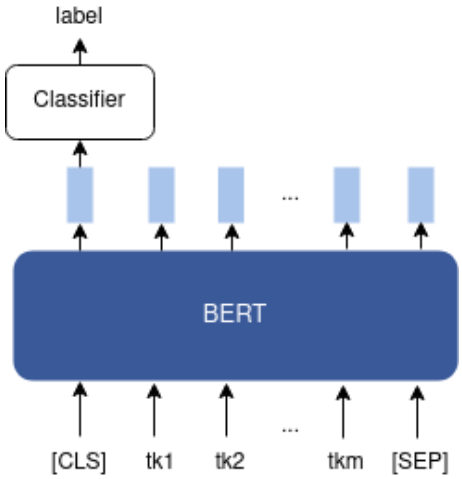


Figure 3.4: Fine-tuning BERT for text classification [27]

After formatting data in such a way that it can be fed into the models, optimal **hyperparameters** were selected. For a BiLSTM model, this included hyperparameters related to the structure of the NN, as well as those related to the training procedure:

- **Layers**: Number and types of layers implemented as part of the model. In this case, the majority of layers will be BiLSTMs, although other layers exist (e.g. regularization, dense).
- **Units**: The number of neurons within each layer. Higher amounts can increase the model's potential to learn complex patterns, but at the expense of computation time and the potential to overfit.
- **Activation Function**: Transforms the training data through each layer, adding non-linearity to help learn complex patterns.

- **Regularization**: Techniques which can be implemented to prevent overfitting. This can be in the form of additional layers (dropout and regularization layers) or regulations in the training process (early stopping).
- **Learning Rate**: The amount of adjustment made to model weights with respect to the loss gradient. Too large or too small of a learning rate can lead to poor training performance. This can be dynamically adjusted through the use of optimizers.
- **Batch Size**: The number of samples fed into the model at a time during the training process.
- **Epochs**: The number of passes made on the entire training dataset. Too few or too many epochs can lead to under or overfitting.

The same process is applied to BERT, with a different set of hyperparameters due to differences in architecture. Certain hyperparameters remain applicable, such as regularization, learning rate, batch size, and number of epochs.

The development process for deep learning models takes a long time, so performing a grid search wasn't feasible. Instead, hyperparameter values were taken from relevant literature and used as a starting point, which were then manually adjusted during development through empirical evaluation. Sets of hyperparameters were locally evaluated on a per-epoch basis, with validation accuracy and loss used. The validation loss indicated whether the models were overfitting to the training data. Once the loss began to increase, early stopping was employed to select the epoch with the best validation accuracy and loss.

The final BiLSTM model was trained on the entire HateXplain training set, and the final BERT and RoBERTa models were fine-tuned using the same set.

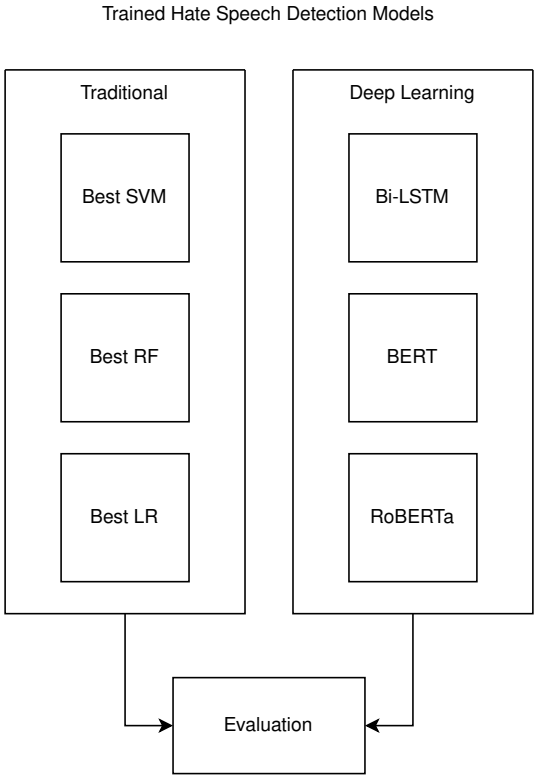


Figure 3.5: Summary of Experiment 2

3.5 Experiment 3

The third experiment examined the impact of differing datasets on hate speech detection; specifically, the impact of binary vs multi-class classification. The aim was to highlight how the classification of data impacts the ability of models to accurately detect hate speech. Whilst metrics provide insight, it should also be considered how these categorisations apply to real world incidents, and how the ambiguities in defining hate speech provide reasons to favour one classification approach over the other.

3.5.1 Experiment Rationale

There exists a wide range of hate speech datasets, and they provide alternative ways of classifying hate speech. Some will categorise social media posts as hateful or not, whilst others will opt for a more fine-grained approach through sub-categorising hate speech instances. The approach desired is based on the level of granularity needed for the hate speech detection model. Some social media platforms may prioritise the identification of hate speech, whilst others may seek the type of hate speech being expressed. This experiment provides an analysis into the effectiveness of both approaches by utilising the HateXplain dataset in both a binary and multi-class setup. Although research on hate speech detection has been done, where dataset labels for different categories have been combined [20, 45], there has been no direct research investigating the impact of

classification approaches using the same dataset.

3.5.2 Model Selection Rationale

This experiment built on from the last, where the three best performing models from the traditional and deep learning paradigms were used to compare binary and multi-class setups. The multi-class models for detecting hate have already been established from prior experiments, and binary models were developed for this experiment. The method of selecting models in this way made the most sense, as it's not the underlying model architecture or approach being evaluated, but the impact of the underlying dataset used to train them. Thus, using the best performing models from prior experiments gave a fair baseline to work from.

3.5.3 Experimental Design

The experimental design will focus on the creation of the Binary HateXplain dataset as opposed to model development, as the methods for this were identical to previous experiments.

3.5.3.1 Binary HateXplain

To convert the original dataset into a binary format, it was necessary to devise a strategy for merging the existing labels (hateful, offensive, and normal). I decided to merge the "offensive" and "normal" categories, as I wanted the detection of hate speech to be as stringent as possible. It's common on social media to come across offensive posts which aren't considered hate speech (in accordance to the definition of hate speech established in the introduction of this dissertation). All hate speech instances in the original dataset were used, and instances of "offensive" and "normal" posts were proportionally sampled from their original categories. This was done to match the composition of the original dataset and avoid the impact of class imbalance on model performance.

3.5.3.2 Model Development

Pre-processing, feature extraction, and model development steps followed what has been laid out in Sections 3.3 and 3.4. The only difference was the data splitting strategy, where training and test subsets were split in a 90/10 ratio instead of 80/20, to maximise the number of training samples given the smaller dataset.

Hate	Not Hate	Total
5935	5935	11870

Table 3.7: Binary HateXplain data distribution

Despite having different labels compared to the original HateXplain dataset, the same hyperparameter values found in the previous experiments were used for the binary, hate speech detection models.

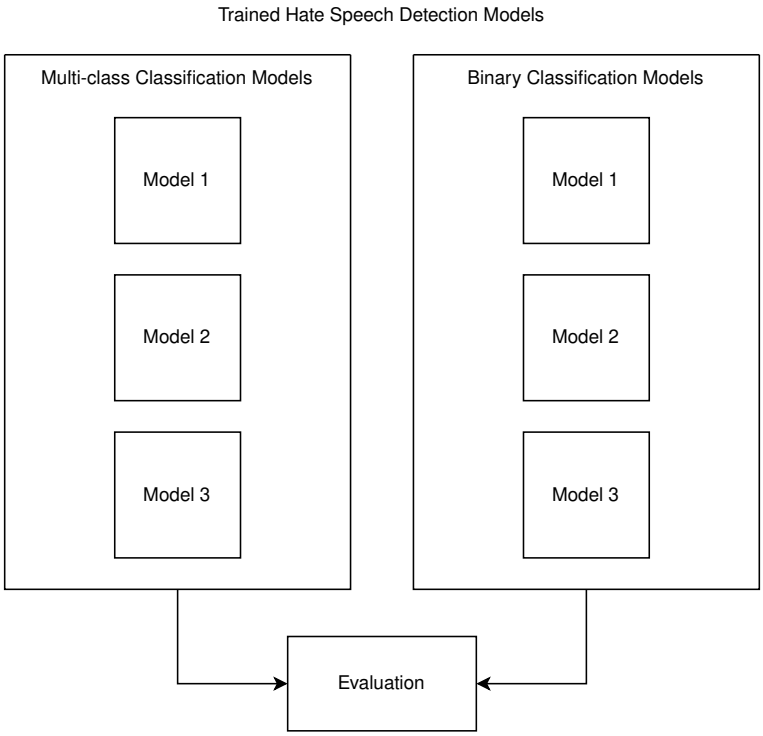


Figure 3.6: Summary of Experiment 3

3.6 Evaluation

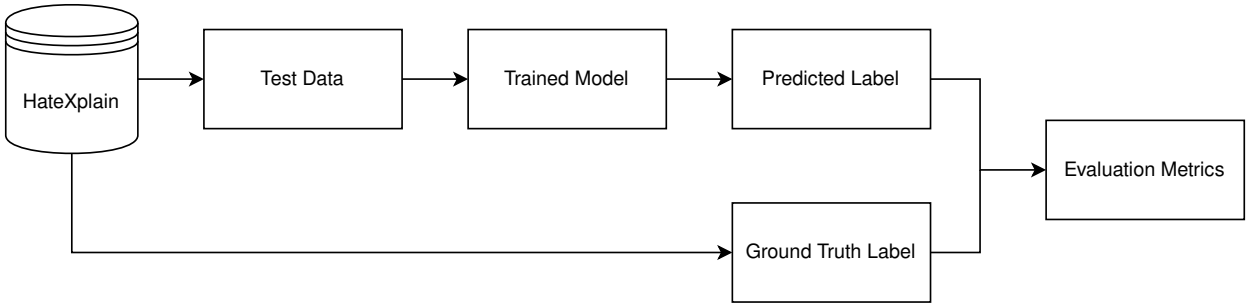


Figure 3.7: Outline of the evaluation process

The evaluation of the hate speech detection models serves as an indicator of their overall effectiveness. A range of different performance-based metrics are used:

1. **Accuracy**: The overall correctness of each model, determined by the proportion of correctly predicted labels against all predicted labels. The calculation is shown in Equation 3.2.

2. **Precision**: A class dependent metric which measures the ability of a model to correctly predict labels for a given class. This gives an indicator to prediction ability. The calculation is shown in Equation 3.3.

3. **Recall**: A class dependent metric which measures the ability of a model to correctly capture instances of a given class. This gives an indicator to class coverage. The calculation is shown in Equation 3.4.

4. **F1 Score**: A balanced measure between precision and recall, giving a harmonic average between the two for a given class. The calculation is shown in Equation 3.5.

5. (**Macro Average**) Precision, Recall and F1 Score: Metrics which assess the performance of the entire model. Computed by taking the mean average of the respective measure for all classes. Macro-averaged precision, recall, and F1 score treat all classes equally, giving an unweighted average.

In addition, we calculate the **Cross Entropy Loss (CE)**, a measure of how well a model's label predictions compare to the ground truth labels. The value is sought to be minimised, to enure confidence in model predictions. The calculation is shown in Equation 3.6.

These metrics will be used to compare models with variations in: feature extraction technique, approaches to ML-based NLP, and dataset composition. In addition, a comparative analysis with models from related literature will be performed. In particular, a comparison with research from Mathew et al. [32], from which the HateXplain dataset was created.

In this project, evaluation can be split into two parts:

- **Development Phase Evaluation**: This is carried out during the development of models and was used to determine the most optimal hyperparameters. Results from this phase are presented in Chapter 4.

- **Testing Phase Evaluation**: This is carried out after all hate speech detection models were developed and is considered the primary form of evaluation. Results from this phase are presented in Chapter 5.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.4}$$

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.5}$$

$$CE = -\sum_{i=1}^{C} y_i \log(p_i) \tag{3.6}$$

(TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.)

(CE is the cross entropy loss, C is the number of labels, y_i is a binary indicator whether the label i is correct for the current observation and p_i is the predicted probability of the observation being in class i).

Chapter 4

Development

This chapter presents the development process in full and puts the methods laid out in section 4 into practice. It details practical aspects of each experiment, such as: languages, libraries, hyperparameters values and code implementation. The programming language of choice for the entire project was Python.

4.1 Data Pre-Processing

The implementation of the pre-processing stage primarily made use of Pandas [33] and NLTK [8]. Pandas was used to manage and manipulate the dataset. The NLTK library was largely responsible for carrying out the pre-processing steps (text cleaning, tokenization, punctuation removal and in the case of TF-IDF, lemmatization and stop-word removal). Word contraction and acronym expansion were performed using a publicly available contractions' dictionary [35]. Figure 4.1 shows the code implementation used to preprocess the dataset, and Figure 4.2 shows a sample of the contractions dictionary used for expanding contractions and acronyms. Table 4.1 presents an example of a final pre-processed piece of text.

The data splitting process for training and test sets was implemented using scikit-learn [40]. Table 4.2 shows the label distribution for both training and test datasets.

```python
def preprocessing_data(text):
    text = ' '.join(text)

    #Text Cleaning
    text = clean(text, no_emoji=True)
    #Tokenization
    tokens = word_tokenize(text)
    #Contraction and Acronym Expansion
    tokens_expand = expand_contractions(tokens, contractions_dict)
    #Remove Stop Words
    nltk_stop_words = set(nltk.corpus.stopwords.words('english'))
    tokens_no_stop_words = [word for word in tokens_expand if word not in nltk_stop_words]
    #Lemmatization
    lemma = nltk.wordnet.WordNetLemmatizer()
    tokens_lemmatize = [lemma.lemmatize(word) for word in tokens_no_stop_words]
    # Remove punctuations from each word
    tokens_final = [word for word in tokens_lemmatize if word not in string.punctuation]

    return tokens_final
```

Figure 4.1: Data pre-processing code implementation

Pre-processed Text	**Post-processed Text**
In fact many of the people destroying our country are fucking niggers and illegal spics with 8 fucking kids or more. So stop and ask yourself wtf are you even talking about moron. LMAO	'in' 'fact' 'many' 'of' 'the' 'people' 'destroying' 'our' 'country' 'are' 'fucking' 'niggers' 'and' 'illegal' 'spics' 'with' '⟨number⟩' 'fucking' kids' 'or' 'more' 'so' 'stop' 'and' 'ask' 'yourself' 'what' 'the' 'fuck' 'are' 'you' 'even' 'talking' 'about' 'moron' 'laughing' 'my' 'ass' 'off'

Table 4.1: Pre-processed vs post-processed document from HateXplain

```
{
"there's":"there is",
" w8  ":"  wait ",
" wam  ":"  wait a minute ",
" wtg  ":"  way to go ",
" wk  ":"  week ",
" w/end  ":"  weekend ",
" wkd  ":"  weekend ",
" wb  ":"  welcome back ",
" wbu  ":"  what about you? ",
" whatcha  ":"  what are you ",
" wtf  ":"  what the fuck ",
" wth  ":"  what the heck?",
" wyd  ":"  what ya doing? ",
" sup  ":"  what's up? ",
" wu  ":"  what's up? ",
```

Figure 4.2: Sample data from the contractions and acronyms dictionary

Label	**Training**	**Test**	**Total**
Hate	4748	1187	5935
Offensive	4384	1096	5480
Normal	6251	1563	7814
Total	15383	3846	19229

Table 4.2: HateXplain data distribution for training and test subsets

4.2 Experiment 1

4.2.1 Feature Extraction

Gensim [43] was used to implement Word2Vec, Doc2Vec and FastText models, scikit-learn was used for TF-IDF features and AllenNLP [19] was used to implement the pre-trained ELMo model.

The Gensim models shared a similar development process regarding model training. The Curated Hate Speech dataset was loaded and formatted in a way that separates documents from one another, and optimal hyperparameters are selected.

Hyperparameter	**Word2Vec**	**Doc2Vec**	**FastText**
Vector Size	300	300	300
Window Size	2	3	3
Minimum Count	3	3	3
Skip Gram	0	0	0
Epochs	5	5	5

Table 4.3: Hyperparameter values for word embedding models

Existing literature [34, 3] was used to provide a basis for initial values. However, many intermediary training and evaluation steps were performed in order to come to the above parameters. I also took into consideration the time-taken to train each model from scratch, where Skip Gram often took long and produced no significant difference in hate speech detection performance. Word2Vec was quickest to train, whilst FastText took longest, likely due to the generation of various n-gram embeddings.

In the case of **ELMo**, the original, pre-trained model[1,2] was used instead of medium and small variants. This increased the time taken to derive document embeddings but led to slightly improved performance. Documents had to be processed in batches, due to excessive memory consumption when processed all at once.

[1]ELMo options: `https://s3-us-west-2.amazonaws.com/allennlp/models/elmo/2x4096_512_2048cnn_2xhighway/elmo_2x4096_512_2048cnn_2xhighway_options.json`

[2]ELMo weights: `https://s3-us-west-2.amazonaws.com/allennlp/models/elmo/2x4096_512_2048cnn_2xhighway/elmo_2x4096_512_2048cnn_2xhighway_options.json`

4.2.2 Model Development

All three traditional models are implemented using scikit-learn. Optimal hyperparameters for each feature extraction technique are found. Grid search is performed for each model.

Figure 4.3 shows a snippet of code of the hyperparameter tuning process for SVM's using Word2Vec as features. A subset of 2000 samples were used to speed up grid search. Five splits in the training data were made as part of cross-fold validation, and the data was shuffled. The best set of hyperparameters were determined by the macro F1 score, which was the mean average over the five folds.

A total of fifteen independent grid searches were carried out, one for each combination of feature extraction technique and model. Tables 4.4, 4.5 and 4.6 present the hyperparameter values for SVM, RF and LR models. The test dataset was used to carry out the primary evaluation.

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'kernel': ['linear', 'poly', 'rbf'],
    'gamma': [0.1, 1, 10, 'auto', 'scale'],
    'class_weight': [None, 'balanced'],
    'shrinking': [False, True]
}


svm_model = SVC(random_state=42)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid_search = GridSearchCV(svm_model, param_grid, cv=cv, scoring='f1_macro', n_jobs=-1)
grid_search.fit(X_train_w2v[:2000], y_train[:2000])

print("Best Hyperparameters: ", grid_search.best_params_)
✓ 7m 12.8s
Best Hyperparameters:  {'C': 10, 'class_weight': 'balanced', 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False}
```

Figure 4.3: Grid search for SVM code implementation

Hyperparameters	**Word2Vec**	**FastText**	**TF-IDF**	**Doc2Vec**	**ELMo**
C	10	10	1	1	10
Kernel	RBF	Poly	RBF	RBF	RBF
Gamma	Auto	10	Auto	Auto	Scale
Class Weight	Balanced	None	None	Balanced	None
Shrinking	False	True	True	False	False

Table 4.4: SVM hyperparameters for different embedding techniques

Hyperparameters	**Word2Vec**	**FastText**	**TF-IDF**	**Doc2Vec**	**ELMo**
Num. of Trees	200	300	200	100	200
Criterion	Gini	Entropy	Gini	Entropy	Gini
Max Features	None	Log2	None	Log2	Log2
Class Weight	None	None	Balanced	None	None

Table 4.5: Random Forest hyperparameters for different embedding techniques

Hyperparameters	**Word2Vec**	**FastText**	**TF-IDF**	**Doc2Vec**	**ELMo**
C	0.1	1	0.1	10	1
Penalty	L2	None	L2	L1	L2
Solver	SAGA	LBFGS	SAGA	Liblinear	SAGA
Max. Iterations	100	1000	1000	100	1000
Class Weight	Balanced	Balanced	None	Balanced	None

Table 4.6: Logistic Regression hyperparameters for different embedding techniques

4.3 Experiment 2

The three traditional hate speech detection models used as part of this experiment were SVM, RF and LR (all with TF-IDF as features).[3]

4.3.1 Model Development

Keras [14] was used to implement BiLSTM whilst Transformers [53] and PyTorch [39] were used to implement BERT and RoBERTa. Development was conducted in a Google Colab environment, using Nvidia A100 [1] and V100 [2] GPUs.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(15383, 70, 300)	8,041,500
dropout (Dropout)	(15383, 70, 300)	0
bidirectional (Bidirectional)	(15383, 70, 128)	186,880
dropout_1 (Dropout)	(15383, 70, 128)	0
bidirectional_1 (Bidirectional)	(15383, 128)	98,816
dense (Dense)	(15383, 64)	8,256
dropout_2 (Dropout)	(15383, 64)	0
dense_1 (Dense)	(15383, 3)	195

Figure 4.4: Summary of the BiLSTM model

Optimal hyperparameters were found by using the paper published by Mathew et al.

[3]These were found to be the best performing models. Evaluation results can be found in Chapter 5.

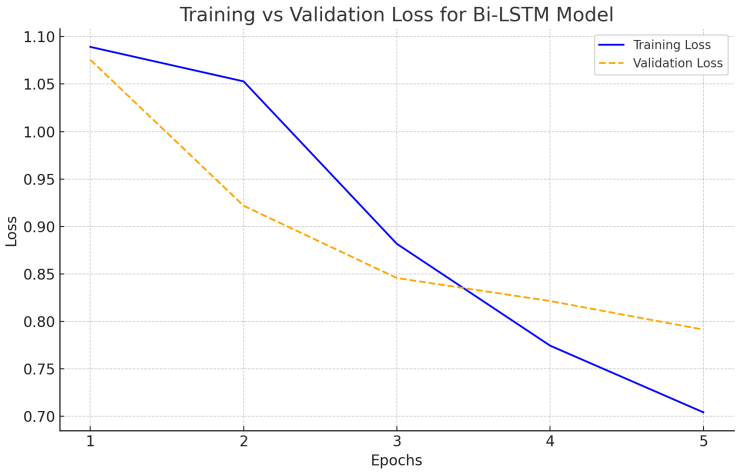
[32] as a basis, and adjusting values through the process of empirical evaluation. The validation accuracy and losses were used as part of the empirical evaluation, and adjustments were made until the highest validation accuracy was achieved. Structural (in the case of BiLSTM) and training-related hyperparameters were adjusted. Figure 4.4 provides a summary of the BiLSTM model structure.

A validation set is used to guide the development of deep learning models. Figures 4.5a, 4.5b and 4.5c detail the training process and the number of epochs used. Table 4.7 details the hyperparameter values. The validation set was re-combined with the training set for final model training. The test dataset was used to carry out the primary evaluation.

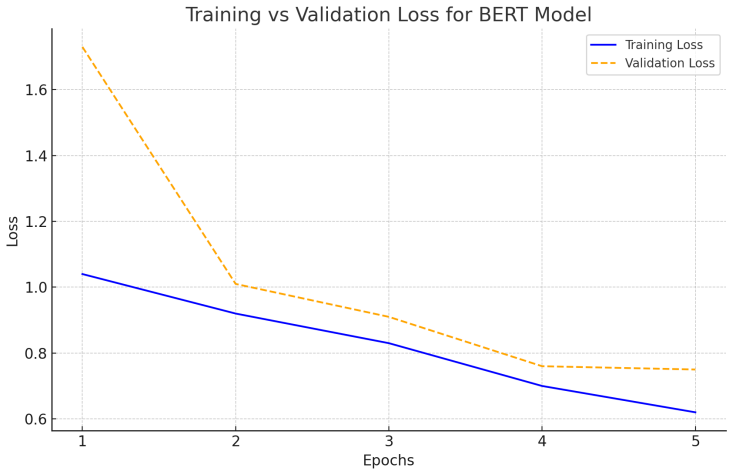
Hyperparameters	**BiLSTM**	**BERT**[1]	**RoBERTa**[2]
Layers	8	N/A	N/A
Units	64	N/A	N/A
Activation Function	Softmax	GELU	GELU
Regularization	Dropout, Early Stopping	Early Stopping	Early Stopping
Dropout	0.5, 0.3 (last)	N/A	N/A
Learning Rate	1e-4	5e-5	5e-5
Batch Size	128	32	16
Epochs	5	5	3

[1] bert-base-uncased model with 12-layer, 768-hidden, 12-heads, 110M parameters.
[2] roberta-base model with 12-layer, 768-hidden, 12-heads, 125M parameters.

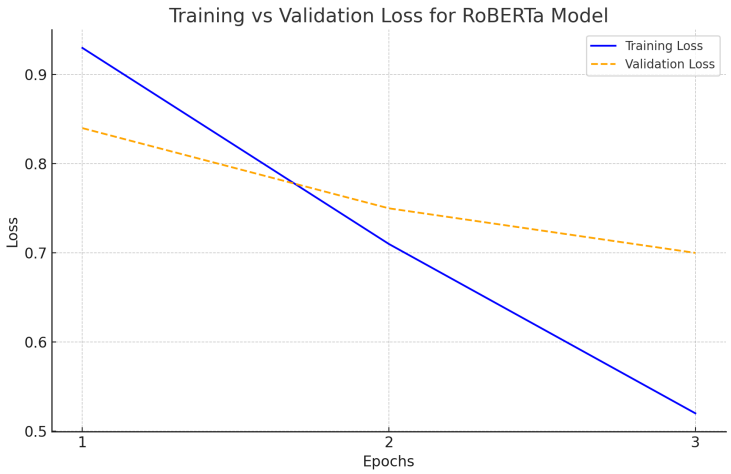
Table 4.7: Hyperparameters for deep learning models



(a) BiLSTM model



(b) BERT model



(c) RoBERTa model

Figure 4.5: Training process for different models

4.4 Experiment 3

The three hate speech detection models used for this experiment are RoBERTa, SVM (with TF-IDF) and LR (with TF-IDF).[4]

4.4.1 Binary HateXplain

The implementation details follow what was laid out in Section 4.1. However, the data splitting strategy was ignored, and a new dataset was created from the original HateXplain. This was done by taking all documents labelled as ”hate speech” and proportionally sampling documents belonging to the ”offensive” and ”normal” classes. Pandas was used to split the documents into separate data frames based on the labels. The samples were then combined into one dataset, and labels were transformed to either a 0 (hate speech) or 1 (not hate speech). Figure 4.6 shows the code implementation for this.

```
hate_df = df_hatexplain[df_hatexplain['final_label'] == 0]
offensive_df = df_hatexplain[df_hatexplain['final_label'] == 2]
normal_df = df_hatexplain[df_hatexplain['final_label'] == 1]

num_samples_offensive = int(len(hate_df) * (len(offensive_df) / (len(offensive_df) + len(normal_df))))
num_samples_normal = len(hate_df) - num_samples_offensive

offensive_sampled = offensive_df.sample(n=num_samples_offensive, random_state=42)
normal_sampled = normal_df.sample(n=num_samples_normal, random_state=42)

df_hatexplain_binary = pd.concat([hate_df, offensive_sampled, normal_sampled])
df_hatexplain_binary['final_binary_label'] = df_hatexplain_binary['final_label'].apply(lambda x: 1 if (x == 1 or x == 2) else 0)
df_hatexplain_binary = df_hatexplain_binary.sample(frac=1, random_state=42).reset_index(drop=True)
```

Figure 4.6: Code implementation used to create the Binary HateXplain dataset

The data splitting process for training and test sets used scikit-learn [40].

Category	**Training**	**Test**	**Total**
Hate	5341	594	5935
Not Hate	5342	593	5935
Total	10683	1187	11870

Table 4.8: Distribution of data in training and test sets for Binary HateXplain

4.4.2 Model Development

SVM, LR and RoBERTa models were developed for binary classification. The same libraries and hyperparameters were used for development. Six hate speech detection models were evaluated (three multi-class and three binary classification). The test dataset was used to carry out the primary evaluation.

[4]BERT was excluded from model selection despite its performance, as it’s considered similar to RoBERTa.

Original HateXplain		Binary HateXplain	
Text	Class	Text	Class
thanks for coming to my ted talk p.s. calling me ching chong with not make me laugh	Offensive	thanks for coming to my ted talk p.s. calling me ching chong with not make me laugh	Not Hate

Table 4.9: Original HateXplain vs Binary HateXplain class labels

Chapter 5

Evaluation

This chapter presents the evaluation and discussion of each experiment's results during the testing phase, according to Section 3.6. All results were obtained using the test subset of HateXplain and Binary HateXplain respectively.

5.1 Experiment 1

FE Technique	Model	Performance			
		Accuracy	Precision (Macro)	Recall (Macro)	F1 (Macro)
Word2Vec	**SVM**	**0.62**	**0.61**	**0.59**	**0.59**
	RF	0.54	0.53	0.51	0.49
	LR	0.60	0.59	0.58	0.57
FastText	SVM	0.58	0.57	0.55	0.54
	RF	0.5	0.49	0.47	0.45
	LR	**0.59**	**0.58**	**0.56**	**0.56**
TF-IDF over BoW	SVM	**0.67**	0.66	0.64	0.64
	RF	**0.67**	0.66	0.64	0.64
	LR	0.66	0.66	**0.65**	**0.65**
Doc2Vec	**SVM**	**0.57**	**0.56**	**0.53**	**0.53**
	RF	0.47	0.44	0.43	0.40
	LR	0.54	0.53	0.51	0.50
ELMo	SVM	**0.61**	**0.60**	0.58	0.57
	RF	0.55	0.52	0.51	0.48
	LR	**0.61**	0.59	**0.59**	**0.58**

Table 5.1: Model performance results across various feature extraction techniques

The first experiment was concerned with the impact of feature extraction techniques on hate speech detection. The results for the performance of each feature along with the implemented model can be seen in Table 5.1. A mean average of F1 scores for each FE technique was used to determine the best performer, as seen in Table 5.2. Due to the slight imbalance in labels, more emphasis was placed on macro F1 scores than accuracy when determining which models yielded better performance. Macro F1 was used instead of other F1 averages due to considering all label predictions as equally important.

A summary of key observations from the experiment:

1. Out of all the implemented feature extraction techniques, TF-IDF yielded superior performance.
2. Doc2Vec features yielded surprisingly poor results.
3. ELMo's contextualised word embeddings provided little difference in hate speech detection compared to the static embeddings used by Word2Vec.
4. The impact of embeddings being domain-specific or agnostic didn't impact hate speech detection performance, although this wasn't explicitly tested for.
5. SVM and LR models were most effective at detecting hate speech.

Feature Extraction Technique	**Mean F1 (Macro)**
Word2Vec	0.55
FastText	0.52
TF-IDF over BoW	**0.64**
Doc2Vec	0.48
ELMo	0.54

Table 5.2: Mean F1 scores for various feature extraction techniques

TF-IDF was considered the simplest solution, yet was able to outperform all NN-based approaches. There are several reasons why this occurred. Firstly, features within the underlying dataset could never be accounted for, and what's observed with one dataset may not be observed with another. The strength in TF-IDF features lied in their simplicity, and the mere presence or absence of keywords was enough to learn patterns and differentiate hateful sentiments. Furthermore, the complexity of deriving document vectors via NNs could be to the detriment of the features, as the features derived would not have been representative of the text, and may also have led to overfitting during model development.

Word embedding models such as: W2V, FastText and ELMo were still able to produce respectable results. Document vectors were created from averaging the word embeddings for all words within a document. This approach was superior to Doc2Vec, which

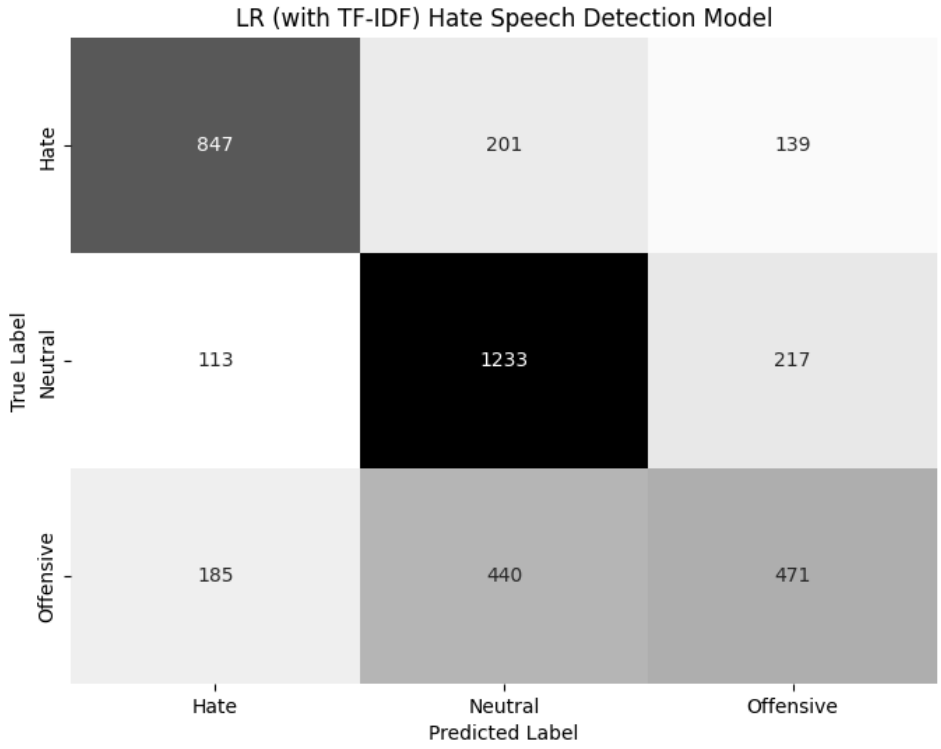
performed poorly in contrast. This was likely because the Doc2Vec model was unable to derive meaningful document vectors due to a lack of training data. Furthermore, the task of hate speech detection seems to be more sensitive to word presence than the nuanced combination of words within a sentence. Therefore, averaging word embeddings would result in document vectors whose labels are easier to predict.

As referenced in Table 3.6, perhaps the importance of embedding approaches in capturing contextual nuances was overstated, and word presence would've sufficed.

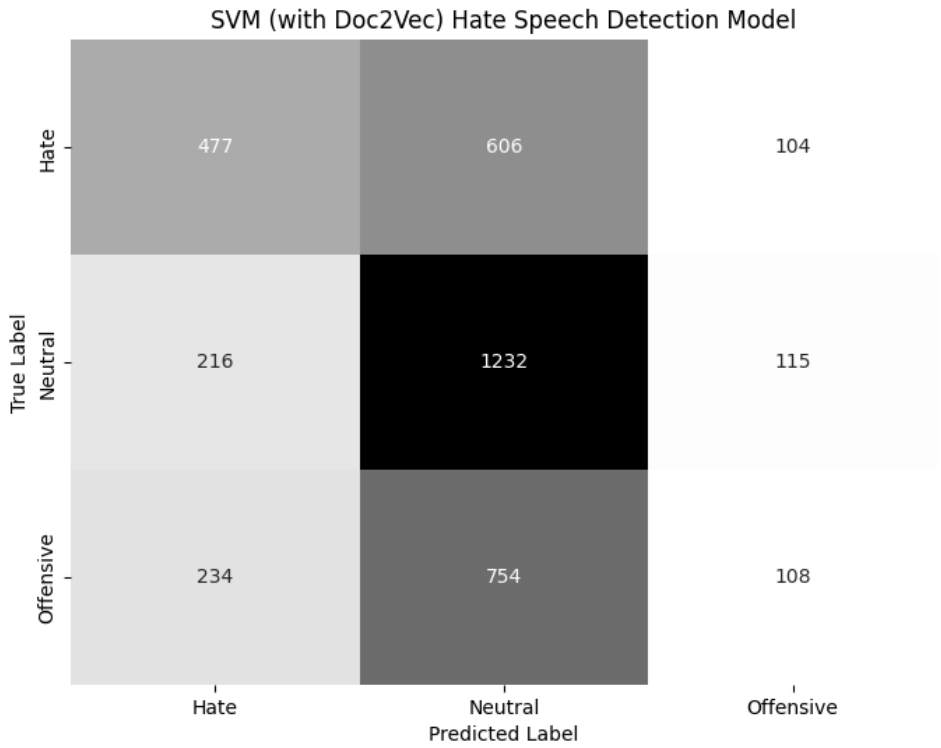
Figure 5.1b presents a confusion matrix for Doc2Vec features. This gives a clearer indication as to where the model struggled. There was a tendency to favour predictions for neutral labels, resulting in a large rate of false positives and false negatives. The confusion matrix for the LR model (TF-IDF) (Table 5.1a) shows the same tendency to favour predictions for neutral labels, albeit with less false positives. This would suggest a common issue for all models and likely stems from the composition of the dataset.

ELMo embeddings produced document vectors with performance comparable to Word2vec. Whilst it was expected that ELMo would've produced superior document vectors which would've led to better hate speech detection performance, this is understandable as a pretrained, domain-agnostic ElMo model was used. In comparison, the Word2Vec model was trained from scratch on a hate speech corpus. ELMo vectors were better equipped to capture the context-sensitive nature of words, whilst Word2Vec vectors were more suited to the domain of social media and hate speech.

To summarise, despite the simplicity of traditional NLP features, they are still effective in hate speech detection when combined with traditional models. Whilst the use of word embeddings have been favoured in recent years, their popularity within the field does not always lead to favourable results, and we should consider simpler solutions to a problem before employing NNs.



(a) Confusion matrix for LR (with TF-IDF)



(b) Confusion matrix for SVM (with Doc2Vec)

Figure 5.1: Confusion matrices for hate speech detection models

5.2 Experiment 2

Approach	Model	Performance			
		Accuracy	**Precision (Macro)**	**Recall (Macro)**	**F1 (Macro)**
Baseline[1]	*CNN-GRU*	*0.63*	*N/A*	*N/A*	*0.61*
	BiRNN	*0.60*	*N/A*	*N/A*	*0.58*
	BERT	*0.69*	*N/A*	*N/A*	*0.67*
Traditional	SVM (TF-IDF)	0.67	0.66	0.64	0.64
	RF (TF-IDF)	0.67	0.66	0.64	0.64
	LR (TF-IDF)	0.66	0.66	0.65	0.65
Deep Learning	BiLSTM	0.65	0.63	0.63	0.62
	BERT	0.70	0.70	0.69	0.69
	RoBERTa	**0.73**	**0.72**	**0.71**	**0.71**

[1] Results from Mathew et al. [32]

Table 5.3: Performance comparison of traditional and deep-learning based NLP models

The second experiment was concerned with the comparison of traditional and deep learning-based models. Whilst the prior experiment provided insight as to how the feature extraction impacts detection capabilities, this experiment is concerned with whether the best traditional hate speech detection models can be improved upon using what's considered state-of-the-art at the current time. Table 5.3 provides a summary of the results.

A summary of key observations from the experiment:

1. Deep Learning approaches were able to demonstrate a tangible improvement in performance over traditional in some cases
2. Traditional approaches provide competitive performance with deep learning approaches, and better them in some cases.
3. Traditional and deep learning approaches developed for this experiment were both able to outperform baseline models.
4. Pre-trained LLMs possess the greatest capabilities in the domain of hate speech detection, warranting their status as state-of-the-art and usage in real-world hate speech detection systems.

The introduction of deep learning models presented an opportunity to provide improvements in hate speech detection over the traditional. This would be expected due to the ability of deep, NNs to capture complex patterns and nuances in language. However, results from this experiment suggest the answer is not so clear.

The results for BiLSTMs are competitively placed, with a macro F1 score of 0.62 and an overall accuracy of 0.65. This is a clear improvement over the CNN and BiRNN used in related literature. The reason for such an improvement is difficult to pin down, but could be attributed to differences in programming language implementations, more optimal hyperparameters being used or more beneficial data pre-processing steps being taken. Despite this, the results of the BiLSTM model does not improve on the traditional models established in Experiment 1. In line with the summary from the prior experiment, traditional models have established themselves as capable hate speech detection tools, even with the advent of transformative, deep learning technologies in the field of NLP. All traditional models presented here also outperform the baseline models, which are based on deep learning architecture.

Both BERT and RoBERTa models were able to produce state-of-the-art performance, both in the field of hate speech detection for multi-class classification and on the HateXplain dataset. The BERT model was able to improve upon the results of baseline BERT (although performance increase wasn't significant). RoBERTa produced a small improvement over BERT and was the best performing hate speech detection model. These results were expected, given the emergence and popularity of LLM-based architectures within the field of NLP.

Whether the disparity in performance between traditional and deep learning approaches is large enough depends on the situation hate speech occurs and how much we value marginal gains. Within this experiment, both traditional and deep learning models have demonstrated effectiveness. LLMs are better equipped to deal with hate speech detection, especially on social media; they're trained on larger amounts of data and thus can handle larger vocabulary and manage phrases not encountered during model development. With this comes increased computational requirements and cost. This is reaffirmed by the use of LLMs trained from scratch specifically for hate speech detection, which have not been focused on in this project due to the lack of resources and cost to develop them. Such models like HateBERT exist [11], and have been shown to outperform general BERT models.

5.3 Experiment 3

Classification	Model	Performance			
		Accuracy	**Precision (Macro)**	**Recall (Macro)**	**F1 (Macro)**
Multi-class	SVM (TF-IDF)	0.67	0.66	0.64	0.64
	LR (TF-IDF)	0.66	0.66	0.65	0.6
	RoBERTa	0.73	0.72	0.71	0.71
Binary	SVM (TF-IDF)	0.80	0.80	0.80	0.80
	LR (TF-IDF)	0.80	0.80	0.80	0.80
	RoBERTa	**0.83**	**0.83**	**0.83**	**0.83**

Table 5.4: Performance comparison of multi-class and binary classification setups

The third experiment was concerned with the impact of dataset classification on a model's ability to accurately detect hate speech, and the trade-offs made as a result of each approach. Whilst raw metrics provide some insight, the usefulness of such models is heavily dependent on the environment they're employed in and how robust we want our hate speech detection models to be. Results are provided in Table 5.4.

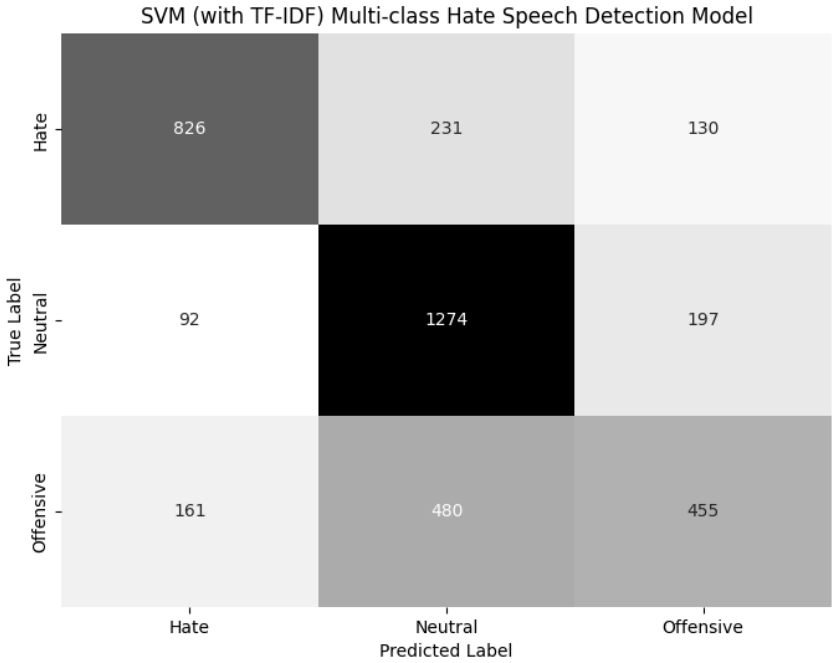
A summary of key observations from the experiment:

1. In all cases, a binary classification setup to hate speech detection results in a better performing model than a multi-class setup.
2. The disparity in performance between traditional and deep learning-based models for a binary setup is less than that of a multi-class setup.
3. LLMs still provided the best performance.

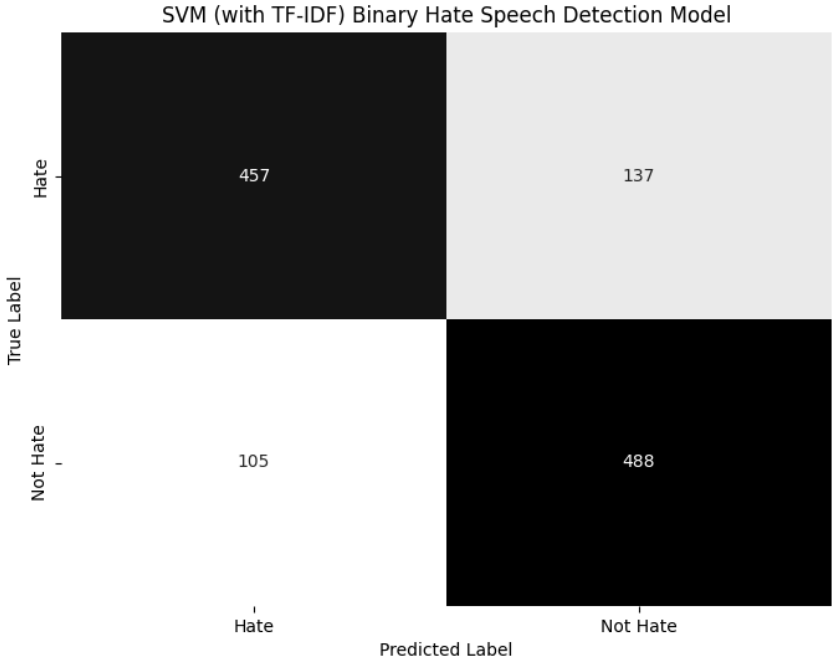
The binary hate speech detection models were able to perform better in the classification tasks compared to their multi-class counterparts. Despite fewer data being used to train the binary models, there was still enough to derive useful features and patterns. Whilst the improvement in classification performance isn't surprising, the experiment highlights how big an impact adding another category for classification has on model performance. The result can be explained intuitively, as fewer labels means less margin for error in predictions and a simpler decision-making process.

Figures 5.2a and 5.2b for SVMs illustrate that classification was more challenging in the multi-class setup compared to the binary setup. The inclusion of offensive labels posed difficulties for all models, leading to an increase in both false positives and false negative predictions. Most incorrect predictions for neutral labels were due to conflation with offensive labels and vice versa. This issue was mitigated in the binary setup, where neutral and offensive labels were combined, simplifying the classification task. It was

hypothesized that multi-class models would've struggled most with classifying hateful and offensive texts due to their inclusion of similar words types and profanity. As this wasn't the case, it can be inferred that neutral and offensive documents share a similar composition.



(a) Confusion matrix for Multi-class SVM



(b) Confusion matrix for Binary SVM

Figure 5.2: Confusion matrices for SVM (with TF-IDF) hate speech detection models

Traditional models were able to produce performance similar to LLMs in a binary setup. The difference in accuracy and F1 score was insignificant, at 0.03 This is likely due to the simpler nature of the classification problem, but once again highlights the effectiveness of traditional approaches as tools in this domain and presents traditional approaches as candidate to be used in real-world hate speech detection systems. RoBERTa was able to achieve state-of-the-art performance, with a macro F1 score of 0.83, an improvement over its multi-class counterpart by 0.12. Although, the increase in performance over traditional models isn't justified in the additional computational resources used.

Results from this experiment show that it can be beneficial to separate the detection of hate speech from other related categories such as offensiveness, profanity, cyberbullying, trolling etc. Nonetheless, when deploying hate speech detection tools in the real world applications, both multi-class and binary approaches have benefits and drawbacks. A binary approach on the surface provides greater accuracy in hate speech detection, yet they're unable to produce a fine-grained analysis of other problematic posts which have a desire to be moderated. In addition, the nature of having two labels to classify posts means a higher level of false positives being flagged, leading to over-moderation and negatively impacting the user experience. Conversely, deploying a multi-class solution to hate speech detection could lead to a higher incidence of false negatives, leading to many hate speech instances being unpunished. Yet, they still provide greater insight and analysis on the types of violations occurring on platforms and can allow for tailored moderation based on the severity of a post (e.g. warnings would be suitable for offensive posts whilst account bans would suffice for hate speech.)

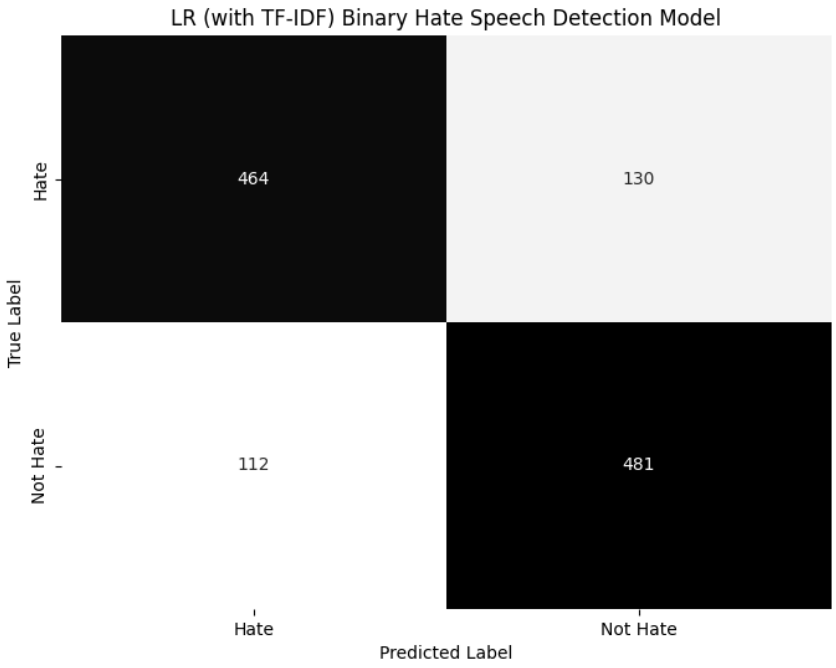


Figure 5.3: Confusion matrix for LR hate speech detection model

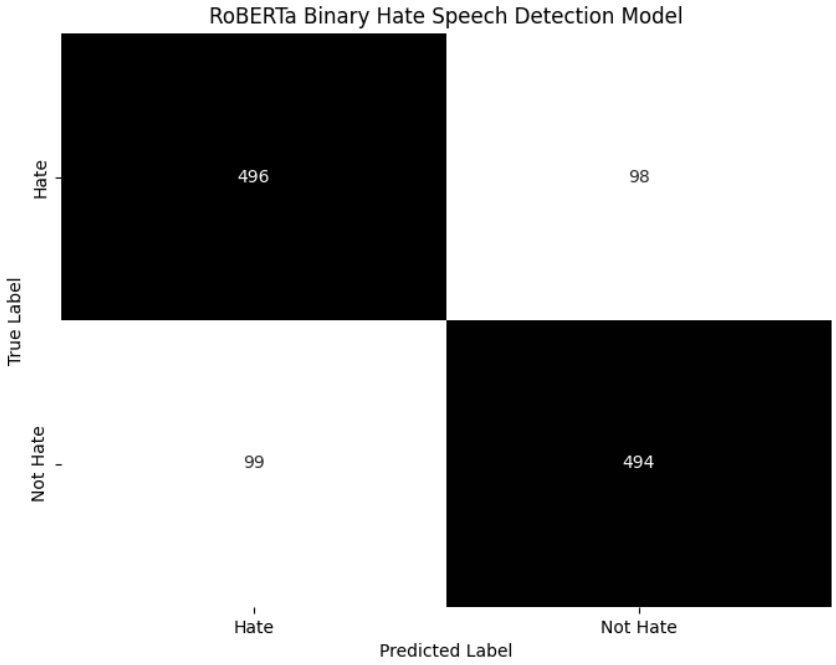


Figure 5.4: Confusion matrix for Binary RoBERTa hate speech detection model

Chapter 6

Conclusion

This chapter summarises the work that has been carried out and what's been achieved. In addition, I reflect on some of the limitations and challenges encountered in the development and evaluation stages, as well as suggest potential improvements. Finally, ideas for future research are given.

6.1 Achievements

In line with the aims and deliverables set out in Chapter 1, I summarise what has been achieved.

- Explored and evaluated various feature extraction techniques and their impact on hate speech detection.
- Explored and evaluated traditional and deep learning-based NLP approaches to hate speech detection.
- Explored and evaluated how dataset classification through both multi-class and binary setup impacts hate speech detection.
- Developed multiple hate speech detection models using a variety of NLP approaches.

In Experiment 1, several models were successfully developed and benchmarked against one another, employing both traditional features like TF-IDF, and NN-based features such as Word2Vec. The dual approach allowed for a comprehensive evaluation of each feature appropriateness to hate speech detection.

As part of Experiment 2, both traditional and deep learning-based models were evaluated against one another. Explanations for such results were laid out, and state-of-the-art performance was achieved using the HateXplain dataset.

As part of Experiment 3, both traditional and deep learning-based models were used

to provide a comparative analysis between classification approaches. Results gave an indication as to the suitability of the classification setup, depending on the environment in which the hate speech detection model is deployed.

Overall, each experiment contributed to the investigation and development of multiple hate speech detection models, made from combinations of traditional and NN-based approaches. This project provided a comprehensive overview of NLP approaches to hate speech detection. Furthermore, this project has provided insight as to how each approach and model could be applied to real world applications and social media platforms.

6.2 Limitations and Challenges

Whilst the initial aims and objectives were met, there were limitations which occurred due to the nature and scope of the project, as well as improvements which could've been made in the methods and evaluation strategy employed.

Firstly, the initial focus of the project was to investigate the impact and prominence of hate speech targeted towards footballers. The scope of this project had to be expanded, as there wasn't enough domain-specific data publicly available. The closest dataset which could be found was from Toraman et al. [47], which contained hate speech instances against athletes. The posts themselves were hidden behind tweet identifiers, requiring payment to the X (formerly Twitter) API in order to retrieve a limited number of them. Combined with the fact that many of these posts would've been taken down, it was unlikely to provide enough data for analysis.

I also planned on collecting my own data from various social media platforms in order to create a hate speech dataset representative of the current landscape (as of 2024). Again, this was impossible due to time constraints in collecting a large enough dataset. Furthermore, the X API is not a cost-friendly solution. Despite this, I was still able to pivot and produce an investigation on hate speech detection using publicly available datasets.

Limitations were also presented in the form of available hardware. With the advent and popularity of deep learning in NLP, the resources required to train models has increased, particularly LLMs. Whilst it would've been ideal to have been able to train an LLM from scratch with domain-specific data, it wasn't possible due to the time, cost and resources required to do so.

The deep learning models developed as part of this project provided challenges regarding training and fine-tuning. When initially attempted, I found that my computer wasn't capable of doing so, and so I resorted to purchasing a Google Colab subscription and utilising their GPUs. When attempting to derive document vectors using the ELMo model, my computer constantly ran out of memory, and so Colab had to be used again.

The use of publicly available datasets also comes with drawbacks. There naturally exists underlying biases in their contents, including annotator biases which impacts the labelling process, and the domain from which the posts are from (half of the posts were from Gab, a far right microblogging site).

An evaluation strategy I would've liked to include was portability, also known as cross dataset evaluation. This uses other datasets in the testing phase as a means of evaluation. This assesses a model's ability to predict labels on data with a completely different distribution and annotation process, mimicking how such a model would perform in the real world. This would've provided more generalisable results and allowed for more comparative analysis with a wider range of existing literature.

6.3 Future Work

Based on these challenges as well as the evolving use of social media, more work can be done. Here I present a list of work that can be done in the future:

1. Whilst a thorough analysis into the effectiveness of feature extraction techniques was done, I would like to extend this further by using more advanced features such as Sentence-BERT. I'd also like to explore the impact of these features when combined with deep learning approaches, and finally I'd like to investigate how well features can perform when combined, such as TF-IDF over Word2Vec embeddings.

2. I would still also like to extend this work to the topic of hate speech detection against footballers on social media, as there's a large difference in the frequency and types of abuse footballers receive compared to others. In addition to this, it would also be beneficial to explore user demographics such as: country of origin, frequency of hateful posts, age, gender etc.

3. With the changing landscape of social media, hate speech is no longer conveyed through just text but also includes audio, video and pictures. It would therefore be of interest to explore methods of hate speech detection of these various forms of media, which could be made possible with advancements in LLMs.

6.4 Final Thoughts

To conclude, this project has successfully achieved its aims and objectives and has provided important research in the domain of hate speech detection, whilst also laying the foundations for additional work to be carried out. The issue of hate speech, both online and in the real world, is not going away anytime soon. With the current state of global affairs and increasing user bases on social media platforms, this issue will only

continue to grow. Therefore, research in this domain must continually evolve to create the most optimal conditions for enhancing the effectiveness of hate speech detection tools and methods. As companies invest more time and infrastructure into this issue, the presence of hateful beliefs and posts can decrease, producing a healthier social media environment.

Bibliography

[1] "NVIDIA A100 Tensor Core GPU". [Online]. Available: https://www.nvidia.com/en-gb/data-center/a100/ [Accessed: 14 April 2024].

[2] "NVIDIA Tesla V100". [Online]. Available: https://www.nvidia.com/en-gb/data-center/tesla-v100/ [Accessed: 14 April 2024].

[3] T. Adewumi, F. Liwicki, and M. Liwicki. "Word2Vec: Optimal hyperparameters and their impact on natural language processing downstream tasks". *Open Computer Science*, 12(1):134–141, 2022. [Accessed: 12 April 2024].

[4] M. S. Adoum Sanoussi, C. Xiaohua, G. K. Agordzo, M. L. Guindo, A. M. Al Omari, and B. M. Issa. "Detection of Hate Speech Texts Using Machine Learning Algorithm". In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0266–0273, 2022. [Accessed: 25 March 2024].

[5] D. Alsagheer, H. Mansourifar, M. M. Dehshibi, and W. Shi. "Detecting Hate Speech Against Athletes in Social Media". 08 2022. [Accessed: 26 March 2024].

[6] D. Asogwa, C. Chukwuneke, C. Ngene, and G. Anigbogu. "Hate Speech Classification Using SVM and Naive BAYES", 03 2022. [Online]. Available: https://www.researchgate.net/publication/359971436_Hate_Speech_Classification_Using_SVM_and_Naive_BAYES [Accessed: 24 March 2024].

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Mathematiques. "A Neural Probabilistic Language Model". 10 2001. [Online]. Available: https://www.researchgate.net/publication/2413241_A_Neural_Probabilistic_Language_Model [Accessed: 15 March 2024].

[8] S. Bird, E. Klein, and E. Loper. *"Natural language processing with Python: analyzing text with the natural language toolkit"*. " O'Reilly Media, Inc.", 2009. [Accessed: 12 April 2024].

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching Word Vectors with

Subword Information". *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. [Accessed: 17 March 2024].

[10] J. Brownlee. How to Use Word Embedding Layers for Deep Learning with Keras, Feb 2021. [Online]. Available: `https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/` [Accessed: 4 April 2024].

[11] T. Caselli, V. Basile, J. Mitrović, and M. Granitzer. "HateBERT: Retraining BERT for abusive language detection in English". In A. Mostafazadeh Davani, D. Kiela, M. Lambert, B. Vidgen, V. Prabhakaran, and Z. Waseem, editors, *Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021)*, pages 17–25, Online, Aug. 2021. Association for Computational Linguistics. [Accessed: 16 April 2024].

[12] A. Chaudhary. A Visual Guide to FastText Word Embeddings. [Online]. Available: `https://amitness.com/2020/06/fasttext-embeddings/` [Accessed: 18 March 2024], Jun 2020.

[13] B. Chiu and S. Baker. "Word embeddings for biomedical natural language processing: A survey". *Language and Linguistics Compass*, 14, 12 2020. [Accessed: 18 March 2024].

[14] F. Chollet et al. Keras, 2015. Available: `https://github.com/fchollet/keras` [Accessed: 14 April 2024].

[15] T. Davidson, D. Warmsley, M. Macy, and I. 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. [Accessed: 22 March 2024].

[16] O. de Gibert, N. Perez, A. García-Pablos, and M. 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, Oct. 2018. Association for Computational Linguistics. [Online]. Available: `https://aclanthology.org/W18-5102/` [Accessed: 22 March 2024].

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In *North American Chapter of the Association for Computational Linguistics*, 2019.

[18] A.-M. Founta, C. Djouvas, D. Chatzakou, I. Leontiadis, J. Blackburn, G. Stringhini, A. Vakali, M. Sirivianos, and N. Kourtellis. "Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior". 02 2018. [Online]. Available: `https://www.researchgate.net/publication/322886443_Large_Scale_Crowdsourcing_and_Characterization_of_Twitter_Abusive_Behavior` [Accessed: 27 March 2024].

[19] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. Peters, M. Schmitz, and L. S. Zettlemoyer. "AllenNLP: A Deep Semantic Natural Language Processing Platform". 2017. [Accessed: 12 April 2024].

[20] S. Gupta and Z. Waseem. "A Comparative Study of Embeddings Methods for Hate Speech Detection from Tweets". *Association for Computing Machinery: New York, NY, USA*, 2017. [Accessed: 25 March 2024].

[21] Z. S. Harris. "Distributional Structure". 1954. [Accessed: 15 March 2024].

[22] C.-W. Hsu and C.-J. Lin. "A Comparison of Methods for Multi-class Support Vector Machines". *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002. [Online]. Available: `https://ieeexplore.ieee.org/abstract/document/991427` [Accessed: 20 March 2024].

[23] M. Jain, P. Goel, P. Singla, and R. Tehlan. "Comparison of Various Word Embeddings for Hate-Speech Detection". In *Data Analytics and Management: Proceedings of ICDAM*, pages 251–265. Springer, 2021. [Accessed: 26 March 2024].

[24] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. "Bag of Tricks for Efficient Text Classification". In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, Apr 2017. [Accessed: 16 March 2024].

[25] C. Kingsford and S. Salzberg. "What are decision trees?". *Nature Biotechnology*, 26:1011–1013, 9 2008. [Online]. Available: `https://www.nature.com/articles/nbt0908-1011` [Accessed: 20 March 2024].

[26] C. Kwak and A. Clayton-Matthews. "Multinomial Logistic Regression". *Nursing research*, 51:404–10, 11 2002. [Accessed: 19 March 2024].

[27] M. Labiadh. Exploring BERT: Feature extraction Fine-tuning, Feb 2024. [Online]. Available: `https://medium.com/dataness-ai/exploring-bert-feature-extraction-fine-tuning-6d6ad7b829e7#` [Accessed: 4 April 2024].

[28] Q. Le and T. Mikolov. "Distributed Representations of Sentences and Documents". *31st International Conference on Machine Learning, ICML 2014*, 4, 05 2014. [Online]. Available: `https://www.researchgate.net/publication/262416109_Distributed_Representations_of_Sentences_and_Documents` [Accessed: 14 March 2024].

[29] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. "Handwritten Digit Recognition with a Back-Propagation

Network". In *Neural Information Processing Systems*, 1989. [Accessed: 19 March 2024].

[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". *ArXiv*, abs/1907.11692, 2019. [Accessed: 2 April 2024].

[31] J. Lu. "Hate Speech Detection Based on Multiple Machine Learning Algorithms". In *Proceedings of the 2023 International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2023)*, pages 244–252. Atlantis Press, 2023. [Accessed: 23 March 2024].

[32] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee. "HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection". In *AAAI Conference on Artificial Intelligence*, 2020. [Accessed: 25 March 2024].

[33] W. McKinney et al. "Data structures for statistical computing in python". In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010. [Accessed: 12 April 2024].

[34] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient Estimation of Word Representations in Vector Space". *Proceedings of Workshop at ICLR*, 2013, 01 2013. [Online]. Available: `https://www.researchgate.net/publication/234131319_Efficient_Estimation_of_Word_Representations_in_Vector_Space` [Accessed: 14 March 2024].

[35] D. Mody, Y. Huang, and T. E. Alves de Oliveira. "A curated dataset for hate speech detection on social media text". *Data in Brief*, 46:108832, 2023. [Accessed: 28 March 2024].

[36] I. Mollas, Z. Chrysopoulou, S. Karlos, and G. Tsoumakas. "ETHOS: an Online Hate Speech Detection Dataset". 2020. [Online]. Available: `https://arxiv.org/abs/2006.08328v2` [Accessed: 27 March 2024].

[37] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang. "Abusive Language Detection in Online User Content". pages 145–153, 04 2016. [Accessed: 22 March 2024].

[38] A. Parmar, R. Katariya, and V. Patel. *"A Review on Random Forest: An Ensemble Classifier"*, pages 758–763. 01 2019. [Accessed: 20 March 2024].

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. 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. [Accessed: 14 April 2024].

[40] 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. [Accessed: 12 April 2024].

[41] A. Perrier. Apply a simple bag-of-words approach. [Online]. Available: `https://openclassrooms.com/en/courses/6532301-introduction-to-natural-language-processing/8081284-apply-a-simple-bag-of-words-approach` [Accessed: 13 March 2024].

[42] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. "Deep Contextualized Word Representations". In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, Jun 2018. [Accessed: 17 March 2024].

[43] R. Řehůřek and P. Sojka. "Software Framework for Topic Modelling with Large Corpora". In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. [Accessed: 12 April 2024].

[44] B. Ross, M. Rist, G. Carbonell, B. Cabrera, N. Kurowsky, and M. Wojatzki. "Measuring the Reliability of Hate Speech Annotations: The Case of the European Refugee Crisis". 09 2016. [Accessed: 22 March 2024].

[45] H. Saleh, A. Alhothali, and K. Moria. "Detection of Hate Speech using BERT and Hate Speech Word Embedding with Deep Model", 11 2021. [Accessed: 24 March 2024].

[46] I. Syarif, A. Prugel-Bennett, and G. Wills. "SVM Parameter Optimization using Grid Search and Genetic Algorithm to Improve Classification Performance". *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 14:1502, 12 2016. [Accessed: 29 March 2024].

[47] C. Toraman, F. Şahinuç, and E. H. Yilmaz. "Large-Scale Hate Speech Detection with Cross-Domain Transfer". In *Proceedings of the Language Resources and Evaluation Conference*, pages 2215–2225, Marseille, France, June 2022. European Language Resources Association. [Accessed: 16 April 2024].

[48] United Nations. What is hate speech? [Online]. Available: `https://www.un.org/`

en/hate-speech/understanding-hate-speech/what-is-hate-speech [Accessed: 12 March 2024].

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is All You Need". 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf [Accessed: 1 April 2024].

[50] B. Vidgen, T. Thrush, Z. Waseem, and D. Kiela. "Learning from the Worst: Dynamically Generated Datasets to Improve Online Hate Detection". 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 1667–1682. Association for Computational Linguistics, Aug 2021. [Accessed: 27 March 2024].

[51] Z. Waseem and D. 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, Jun 2016. Association for Computational Linguistics. [Accessed: 27 March 2024].

[52] B. Wei, J. Li, A. Gupta, H. Umair, A. Vovor, and N. Durzynski. "Offensive Language and Hate Speech Detection with Deep Learning and Transfer Learning". 02 2021. [Online]. Available: https://www.researchgate.net/publication/353729289_Offensive_Language_and_Hate_Speech_Detection_with_Deep_Learning_and_Transfer_Learning [Accessed: 1 April 2024].

[53] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. "Transformers: State-of-the-Art Natural Language Processing". In Q. Liu and D. Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. [Accessed: 14 April 2024].

[54] H. Zahera and M. Sherif. "ProBERT: Product Data Classification with Fine-tuning BERT Model". 10 2020. [Accessed: 15 April 2024].

[55] Y. Zhao. RNN, LSTM, and Bidirectional LSTM: Complete guide: Dagshub, Dec 2023. [Online]. Available: https://dagshub.com/blog/rnn-lstm-bidirectional-lstm/ [Accessed: 18 April 2024].

[56] W. Zhou and J. Bloem. "Comparing Contextual and Static Word Embeddings with

Small Data". In *Conference on Natural Language Processing*, 2021. [Accessed: 18 March 2024].