Vebjørn Isaksen

Detecting Hateful and Offensive Language with Transfer-Learned Models

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Vebjørn Isaksen

Detecting Hateful and Offensive Language with Transfer-Learned Models

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Abstract

Social media services have seen an increase in unintended content, such as abusive and hateful expressions. As a result, automatic detection and removal of such content have become a vital task to reduce the number of inappropriate instances on online platforms. In recent years, the field has proposed various advanced language models that can be pre-trained on an unlimited amount of online text to obtain general language understanding. The language models have already surpassed state-of-the-art in nearly all language tasks tested so far, but the task of hate speech detection is still untested.

This thesis explores the effects of transferring knowledge from huge pre-trained language models to classifiers undertaking the downstream task of separating offensive, hateful, and normal language. Distinguishing hate speech form non-hate offensive language is found a challenging task, as hate speech not always includes offensive slurs and offensive language not always express hate. This thesis focuses on solving this task through three main parts. First, a thorough literature review was carried out to obtain insights into feature and model selection and also to investigate previous solutions to the task. Secondly, a preliminary study was performed to experiment with different features and models and in addition, to acquire baseline results from traditional machine learning approaches. Finally, four deep learning systems based on the Bidirectional Encoder Representations from Transformers (BERT) language model were implemented and tested against two datasets containing tweets labelled as either "Hateful", "Normal" and "Offensive". The results indicate that the attention-based models profoundly confuse hate speech with offensive and normal language. However, the key finding from the experiments is that the pre-trained models with either general or domain-specific language understanding outperform state-of-the-art results in terms of accurately predicting hateful instances. This finding demonstrates the massive potential of transferring knowledge from language models, but more research in the field of hate speech detection is required to create a practical usable system.

Sammendrag

Sosiale internettjenester har sett en økning i uønsket innhold slik som fornærmende eller hatefulle ytringer. Som et resultat har automatisk filtrering av slikt innhold blitt en vesentlig oppgave for å redusere antallet upassende meldinger på nettplattformer. I løpet av de siste årene har feltet presentert flere avanserte språkmodeller som kan trenes opp med store mengder data for å oppnå generell språkforståelse. Disse modellene har allerede utklasset tradisjonelle metoder på flere språkoppgaver, men har foreløpig blitt lite brukt til å detektere hatytringer.

Målet med forskningen i denne oppgaven var å undersøke effektene av å overføre kunnskap fra enorme, forhåndstrente språkmodeller, til klassifiserere som foretar seg oppgaven å separere språk som kan være støtende, hatefullt eller nøytralt. Å separere støtende og hatefulle ytringer på nett er sett på som en utfordrende oppgave ettersom ikke all netthat innholder støtende ord og støtende meldinger heller ikke alltid ytrer hat. Denne oppgaven fokuserer på å løse denne utfordringen gjennom tre hoveddeler. Først ble en studie av relevant litteratur gjennomført for å få innsikt i egenskaper og modeller brukt i tidligere løsninger. Deretter ble en forhåndsstudie gjennomført for å eksperimentere med ulike egenskaper og modeller, samt samle resultater fra tradisjonelle maskinlæringsalgoritmer. Til slutt ble fire systemer basert på språkmodellen Bidirectional Encoder Representations from Transformers (BERT) implementert og testet med to datasett som inneholdt tweets annotert som enten "Hateful", "Normal" og "Offensive". Resultatene indikerer at modellene ofte forveksler hatytringer med støtende og normalt språk. Det viste seg imidlertid at modellene er betydelig bedre til å nøyaktig klassifisere hatefulle ytringer enn systemer fra tidligere forskning. Dette funnet demonstrerer det massive potensialet som ligger i å overføre kunnskap fra språkmodeller, men det er likevel behov for mer forskning innenfor feltet for å lage et system som er brukbart i praksis.

Preface

This Master's thesis in Computer Science was a part of a Master's degree at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. I want to thank my supervisor Björn Gambäck, for providing exciting and relevant papers about the field, and for always answering my concerns around the thesis. His supervision throughout the last year has been a great support. I would also like to express gratitude to my fellow students Håkon Hukkelås for pointing out the paper about BERT, Stian Steinbakken for discussions about BERT and Stian Hanssen for interesting discussions about the results.

Vebjørn Isaksen
Trondheim, 22nd June 2019

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 15–17, 41

API Application Programming Interface. 1, 19, 53

BERT Bidirectional Encoder Representations from Transformers. 2–5, 19, 37, 43, 53–58, 60, 61, 63–69, 71–74, 77, 78, 80–82, 84–86

BOW Bag of Words. 9, 10

CNN Convolutional Neural Network. 16, 17, 39–41, 79, 80

CPU Central Processing Unit. 21

DNN Deep Neural Network. 16

GBDT Gradient Boosted Decision Tree. 39

GPU Graphics Processing Unit. 21

LR Logistic Regression. 14, 40, 50–52, 83

LSTM Long Short-Term Memory. 17, 18, 35, 39–41, 43, 79, 80

MLP Multilayer Perceptron. 15, 41

NB Naïve Bayes. 11, 50, 80, 83

NLP Natural Language Processing. 3, 4, 7, 8, 10, 11, 17–20, 27, 33, 35, 37, 43, 53, 55, 58, 72, 73, 79, 80, 83, 84

NLTK Natural Language Toolkit. 20, 47, 60

REST Representational State Transfer. 19

RNN Recurrent Neural Network. 17, 18, 40, 41, 58, 79, 80

SVM Support Vector Machine. 12, 40, 50–52, 80, 83

TF-IDF Term Frequency-Inverse Document Frequency. 9–11, 79

VSM Vector Space Model. 9

Glossary

ANN Artificial Neural Network is a framework for different machine learning algorithms. 15

API An Application Programming Interface (API) is a particular set of functions and procedures that a software program can utilize to access features or data of another software program. 1

BERT Bidirectional Encoder Representations from Transformers (BERT) is a language model developed by Google AI Language that is trained on a large amount of plain text from Wikipedia and BookCorpus. 2

BOW Bag of Words is a popular way to represent text as numerical vectors. 9

CNN Convolutional Neural Network is a variant of deep neural networks often used in the field of computer vision. 16

DNN Deep Neural Network is a set of neural networks with two or more hidden layers. 16

GBDT Gradient Boosted Decision Tree is a machine learning algorithm. 39

LR Logistic Regression is a supervised learning algorithm. 14

LSTM Long Short-Term Memory networks are a type of recurrent neural networks with enhanced memory to remember long-term dependencies. 17

NB Naïve Bayes is a set of supervised machine learning algorithms based on Bayes' theorem. 11

NLP Natural Language Processing is subfield in computer science that helps computers understand and process the natural languages of humans. 3

NLTK Natural Language Toolkit is a Python library for natural language processing. 20

RNN Recurrent Neural Network is a variant of deep neural networks that allows recurrence and thus can remember previous decisions. 17

SVM Support Vector Machine is a supervised learning algorithm. 12

TF-IDF Term Frequency-Inverse Document Frequency is a technique used to capture keywords in text documents. 9

VSM Vector Space Model is a vector representation of text documents. 9

1 Introduction

Social network platforms today are enormous, and billions of people are expressing their opinions through them. In June 2018, Facebook reported that 2.23 billion active users access their platform each month (Facebook, 2018). Twitter rarely share how many tweets that are posted each day but in their last update (Twitter, 2014) five years ago, they stated that 500 million tweets were posted every day. Managing unintended content among all user-generated content online is a challenging task. Some of the biggest firms invest heavily in automatic detection of abusive and offensive language, as well as hate speech. In June 2017, Instagram announced a new tool against offensive language in comments (Systrom, 2017). A year later, they upgraded this tool to filter comments that attack a person's appearance or character (Systrom, 2018). Google's Jigsaw incubator has developed a cutting-edge Application Programming Interface (API) called Perspective[1] which gives text input a percentage of how likely it is to be perceived as toxic. However, these and other existing tools share a common flaw of not distinguishing between offensive and hateful language. One important reason to keep these two separate is that hate speech is considered a felony in many countries. As a result, the companies behind the platforms have in recent years received increasing pressure from legal instances to detect and remove inappropriate content. The task of separating offensive and hateful language has shown to be demanding and is not considered a solved problem. However, with the recent scientific breakthroughs in the field and the concept of transfer learning, we can take huge steps in the right direction. This thesis will focus on the task of distinguishing offensive and hateful tweets by utilizing a pre-trained language model together with a deep learning approach.

1.1 Background and Motivation

The majority of the tweets on Twitter or posts on Facebook are harmless and often posted purposefully, but some of them can utter hatred towards a targeted individual or

[1] https://www.perspectiveapi.com

minority group and members. These posts are intended to be derogatory, humiliating or insulting and are defined as hate speech by Davidson et al. (2017). Different from offensive language, hate speech is usually expressed towards group attributes such as religion, ethnic origin, sexual orientation, disability or gender (Founta et al., 2018b). Although there is no formal or legal definition of hate speech, the field of research seems to agree that the above descriptions are suitable when defining hateful language. Hate speech is seen as a social problem that can have significant adverse effects as it easily can be posted online and reach a large number of people. Hate speech is not a new phenomenon, but the availability of social media can explain the increased use of hate speech as it is a useful tool for spreading such speech.

Today, most of the online communities have different approaches for removing content that violates their terms and conditions. They often depend on their users to report disturbing content, which is then reviewed by moderators who decide if the content should be removed or not. This manual approach scales poorly and is quickly becoming infeasible as the amount of user-generated data grows larger every day. Although systems that can detect unintended content automatically solve the scalability problem, it is crucial that online communities preserve freedom of speech. This is one reason why the separation between offensive and hateful language is crucial as not all instances of offensive speech violate a platform's terms and conditions.

The legal aspect the online communities face may be the most significant factor why the field has received increased attention lately and why firms invest heavily in automatic detection systems. There are different efforts from governments around the globe when confronting the problem of hate speech. Some countries demand the removal of disturbing content within a certain time limit while others have a very high threshold before hate speech is considered a crime. However, online communities share the apparent responsibility to keep their platforms safe and encourage civil interactions between users. Offensive and hateful language can be difficult to distinguish due to the many nuances of natural language, and many existing automatic detection systems merge the two forms. To have a system that can detect pure hate speech in real-time as comments and posts are posted to social network platforms is vital as such speech intimidates people and can be discouraging for people who want to speak publically. Such a system can give great value to companies trying to ensure safe environments on their platforms.

This thesis investigates whether transferring knowledge from the Bidirectional Encoder Representations from Transformers (Devlin et al., 2018), or **BERT** for short, language model has an effect when distinguishing hateful, offensive and normal language. BERT is a colossal language model trained on some of the Internet's largest available text corpora. It can be fine-tuned with a domain-specific text corpus which may increase the performance of the final model. BERT has shown state-of-the-art results in many Natural Language Processing (NLP) tasks, including the competitive Stanford Question Answering Dataset (SQuAD v1.1). This thesis will fine-tune the pre-trained BERT language model with data containing hateful and offensive language and investigate if a model with transferred knowledge can outperform the state-of-the-art in two accessible datasets widely used in the field of hate speech detection.

1.2 Goals and Research Questions

Goal *Investigate how to accurately detect and distinguish hateful, offensive, and normal language.*

The goal of this research is to improve the detection of true hate speech and to separate it from offensive and normal language. With the concept of transfer learning and a language model pre-trained on a large corpus, this may be done by fine-tuning the model with domain-specific data and investigate how well the model can tell the difference between hateful, offensive and normal language. The goal is set to be achieved through theoretical literature studies, a preliminary study, and practical experiments as described in the research questions below.

Research question 1 *Which features and representations of text, as well as models, are effective when detecting hateful, offensive, and normal language?*

For effective hate speech detection, the choice of features and text representation, in addition to model selection, is important. To gain insights into how to best make these choices, a review of previous literature of related studies and a preliminary study will be conducted. From this review, the findings will be thoroughly evaluated to gain an understanding of which features, text representation, and methods that are most effective when detecting hate speech. The survey and preliminary study will also investigate the difference between offensive and hateful language to understand better how to separate them in the experiments.

Research question 2 *How well does a deep learning model based on a large pre-trained language model distinguish between hateful, offensive, and normal language?*

To possibly improve the current state of hate speech detection, the recently released pre-trained language model BERT together with a deep learning model will be applied to the task of detecting hate speech. BERT is trained with two separate pre-training tasks where each task inputs a vast amount of raw text data. This strategy has shown remarkable results in many NLP tasks, though the task of hate speech detection is yet to be thoroughly tested. The experiments designed to answer this question will use the originally released BERT language models which are pre-trained on two huge corpora to gain general language understanding. The experiments will investigate if the models successfully can distinguish between hateful, offensive, and normal language.

Research question 3 *What are the effects of further pre-training a language model with hate speech corpora?*

This research question is closely related to Research Question 2 and will explore the effects of further pre-training the original BERT language models with raw text data from the hate speech domain. The intention of this extended training is for the model to gain more domain-specific language understanding and to possibly improve the overall accuracy when separating hateful, offensive, and normal language.

1.3 Research Method

Several research methodologies were used to answer the research questions, as well as the overall goal for the research in this thesis. For answering the first research question, the research method followed an exploratory approach, namely, secondary research. A qualitative analysis of previous research was conducted by reviewing existing literature in the field of hate speech detection. This was to get theoretical insights into how existing solutions try to solve the problem and to find relevant features and representations of text that could be utilized in the experiments. The experiments were constructed primarily to answer the second and third research question, with implementing and testing a fine-tuned pre-trained language model on two popular datasets in the field of hate speech containing offensive, hateful and normal language. However, the results from the experiments were also be compared to already existing solutions' results mainly found in the literature review in Chapter 4 and in the preliminary study described in Chapter 5. To sum up, the work in this thesis consisted mainly of exploratory research with the intention of to gain experience and insights into how well pre-trained language models can distinguish between hateful, offensive, and normal language.

1.4 Contributions

The exploratory work conducted in this thesis contributes to the area of hate speech detection. The experiments were designed to focus on separating hateful, offensive, and normal language with the use of two popular hate speech datasets. The main contributions can be listed as follow:

C1 A thorough literature review of existing studies focusing on transfer learning in the field of hate speech detection.

C2 An overview of how to implement the BERT language model to a downstream task such as hate speech detection.

C3 A preliminary study with baseline results on datasets separating between hateful and offensive language.

C4 The implementation of two systems based on the BERT language model pre-trained on a large corpora, as well as the implementation of two systems further pre-trained with domain-specific data from BERT's checkpoints.

C5 Experiments with the above-mentioned four systems on two hate speech datasets.

1.5 Thesis Structure

Chapter 2 introduces relevant technologies and methods used in the work of this thesis or related work.

Chapter 3 presents various datasets used in related work and this thesis. It will mainly focus on the two datasets used in the experiments.

Chapter 4 explores previous literature and gives an overview of existing related work in the field of hate speech detection.

Chapter 5 gives a brief overview of a preliminary study. The study's purpose was to investigate how different features, representations, and machine learning models can detect offensive, hateful and normal language.

Chapter 6 provides a detailed overview of the overall architecture of the implemented system.

Chapter 7 presents details around the experiments, including experimental plan, setup, and results.

Chapter 8 evaluates the results obtained in the experiments and discusses the research questions.

Chapter 9 lastly concludes the thesis and presents suggestions for future work in the field of hate speech detection.

2 Background Theory

This chapter consists of relevant theory within the fields of Natural Language Processing NLP and Machine Learning, along with basic concepts of methods and techniques. It will present the technology, architectures, tools, and libraries used in the thesis and in the papers presented in the related work described in Chapter 4.

2.1 Transfer Learning

Humans are experts on transferring knowledge obtained from one task to another. For instance, imagine you can play guitar, then you will use a lot of your knowledge about playing guitar if you were to learn how to play bass. This is an inherent ability humans have and use every day to easier encounter new tasks. Transfer learning in the field of machine learning is something that has been around for many years and is used to improve the performance of models drastically, for example, in computer vision. Even in NLP, word embedding techniques like word2vec (Mikolov et al., 2013b) and GloVe (Pennington et al., 2014) are examples of transfer learning. However, transfer learning in NLP has not proven to improve the performance of models as much as in other areas of machine learning.

A new task that improves learning through transferring knowledge from a related task already learned, is utilizing the core concept of transfer learning. Figure 2.1 shows a comparison between traditional machine learning and learning with transfer. In NLP, for example, a language model trained on one or more tasks with a large corpus of raw text can be fine-tuned to solve many of the subtasks in NLP. In this example, the task used to train the language model is commonly referred to as source task(s) while the target task is the task-specific problem to be solved. Torrey and Shavlik (2010) propose three ways transfer learning may improve the learning rate of a model. Firstly, the initial performance is higher at the start of the training due to the transferred knowledge from the source task. Secondly, transfer learning may use less time to get fully trained compared to models starting training from scratch. Lastly, the final achievable performance of a model with transfer is higher than models without transfer. However,

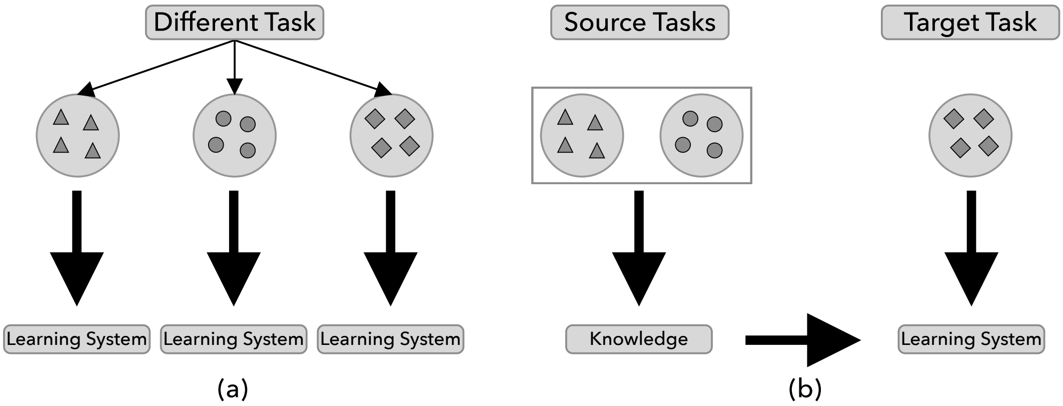

Figure 2.1: Two different learning processes with (a) traditional machine learning and (b) learning with transfer. Illustration from Pan and Yang (2010) with permission.

if the data used in the target task is very different from the data the source task was trained on, then the overall learning may be worse. Transfer learning is undoubtedly exciting, especially with the recent progress that shows it successfully applied to many NLP tasks. This thesis will investigate previous solutions with transferred knowledge and the effects of applying transfer learning to the task of hate speech classification.

2.2 Natural Language Processing

If you try to input raw text into a computer system, the result would probably not be as you expected. This is because the raw text needs some processing and representation for the computer to interpret it correctly. Natural Language Processing — or NLP for short — is the field in computer science that presents methods and techniques for making natural language readable and understandable for computers. This section will introduce a few relevant theories and methods widely used in the field of NLP.

2.2.1 Refining Written Text

Textual communication between humans online tend to be mediocre in terms of grammatical and lexical correctness. Words are regularly misspelled, periods and commas are omitted and sentences can be messy. This generally results in a lot of noise that makes it complicated for the computer to extract the necessary information. This is why preparing written text for representation is an important step in NLP. The most favored steps in preprocessing written text are tokenization, stemming, lemmatization and removal of common words like “I” and “not”. The latter technique is referred to as removal of stop-words and these terms are usually the most frequent in a document. They are generally filtered out because to reduce the dimensionality of the vectors. Tokenization is when text gets split into a sequence of characters, called tokens. In this process, certain characters and symbols can be thrown away. A token can be any sequence of characters like “me”, “dog” or “:-)” and a document is just a sequence of tokens. The goal of stemming and lemmatization is to reduce different grammatical forms of a word to a common base form. Stemming is using rules to reduce a word into its root form, e.g. “buses” to “bus”. Lemmatization is using a dictionary to retrieve the base of a word, commonly called a *lemma*. Stemmers and lemmatizers are built from many different algorithms and rules, where some of the more well-known are the Porter (Porter, 1980) and Lancaster (Paice, 1990) stemmers and the WordNet lemmatizer[1]. Tokens can further be refined by converting them into lowercase, performing spell-checks and discarding characters such as question or exclamation marks. All the above is done so the computer more efficiently can create a good numerical representation used as input to machine learning algorithms.

2.2.2 Representing Text

A vast number of different techniques can achieve numerical representation of unstructured text documents. The main reason for this process is to make a group of tokens, commonly called a document, mathematically computable. This section will present some of the different ways to represent text numerically.

Vector Space Model

The Vector Space Model (VSM) is a traditional text representation model which is used to compute similarities between documents. The main idea of this model is to construct numerical vectors that can be compared in terms of their similarity by utilizing concepts such as Cosine similarity. A popular VSM is the Bag of Words (BOW) model which represents a set of every word appearing in a collection of documents. Each document then gets represented by a set of terms and whether or not these terms occur in the document. This representation uses binary vectors to represent a document, while other forms of document representation count the occurrences of each term or using a weighting form called Term Frequency-Inverse Document Frequency (TF-IDF). The TF-IDF model is described in the next section. The BOW model is a common way to represent text as

[1] https://wordnet.princeton.edu/

- **D1**: People say nothing is impossible, but I do nothing every day.
- **D2**: Every day I think about nothing.

Terms	**D1**	**D2**
people	1	0
say	1	0
nothing	2	1
is	1	0
impossible	1	0
but	1	0
i	1	1
do	1	0
every	1	1
day	1	1
think	0	1
about	0	1

Figure 2.2: Bag of Words vector representation with two documents.

numerical vectors as it provides good accuracy and efficiency, as well as being simple to implement. A clear disadvantage of this model is that it is not able to represent the word order. This can result in two equal vectors representing two documents with a completely different context as long as the same words appear in both. In NLP, a collection of documents is commonly known as a *corpus.* As BOW represents each document by every word in the corpus, sparse data with mostly zeros and vocabularies with potentially millions of terms are possible scenarios for larger corpora. The latter scenario shows that the BOW model faces scalability challenges. Figure 2.2 illustrates the BOW model on two documents.

Term Frequency-Inverse Document Frequency

When analyzing documents, it is important to find keywords that can describe the subject of the document. Havrlant and Kreinovich (2017) state that the TF-IDF heuristic is one of the most widely used techniques for keyword detection in documents. The IDF part of the weighting scheme was first presented by Spärck Jones (1972) as “term specificity”. TF-IDF is a multiplication of two statistics. The first part is term frequency (TF), and it represents how many times a word occurs in a specific document; hence it is an important keyword for that document. However, if a term frequently appears in many documents, this word might not be that distinguishing after all. Stop-words can be such terms. To account for this, the second part, inverse document frequency (IDF), downscales the weight for the less informative terms and increases the weight on terms that occur in a smaller portion of the document set. To sum up, a word's TF-IDF score is higher when it appears many times in few documents, lower when the word appears in many documents or fewer times in one document, and lowest when the word occurs in nearly all of the documents in the corpus.

2.3 Machine Learning Models

Some computer programs can use their experiences to learn and improve their performances on a specific task. Such programs are called machine learning models, and they have shown to be a crucial component in the field of text classification because of their simplicity and versatility. They usually perform on labeled data, also known as supervised learning, but can also be used in cases where data is unlabeled, known as unsupervised learning. Hate speech detection approaches typically use labeled data and thus fall under the category of supervised machine learning. The following sections will briefly present three conventional machine learning models used in the field of text classification. They are all successfully adapted to a variety of NLP tasks with satisfying performance.

2.3.1 Naïve Bayes

Naïve Bayes (NB) is a set of methods that are simple, yet powerful supervised learning algorithms based on Bayes' theorem. They all share the "naïve" assumption that every pair of features is conditionally independent given the value of the class variable. The Naïve Bayes classifiers differ mainly by the decision rule they chose to use. Given class variable y and a set of features x_1 through x_n, Bayes' theorem states that:

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} \tag{2.1}$$

Described in words, Bayes' theorem states that the probability of class y given the features x_1 through x_n is equal to the probability of class y times the opposite probability divided by the probability of the features x_1 through x_n. Applying the naïve assumption of conditional independence between feature pairs and removing the $P(x_1, ..., x_n)$ term as it is a constant independent of y yields:

$$P(y|x_1, ..., x_n) = P(y) \prod_{i=1}^{n} P(x_i|y), \tag{2.2}$$

which is the equation used by Naïve Bayes classifiers to calculate the probability of a class given a set of features. The class with the largest probability is assigned to the set of features or a document in a text classification task. Although the naïve assumption made by the classifiers is highly simplified, the models have proven to perform well on real-world situations, and they are the preferred classifiers for spam filtering. Naïve Bayes classifiers are also a popular choice when classifying text documents because they are suitable for classification with discrete values (e.g., word counts). They require only a small amount of training data, are easy to understand and can be fast compared to more complex neural network models. One problem the classifiers can stumble upon is that if the naïve assumption is not met, they will probably perform poorly. However, for hate speech and offensive language detection, Naïve Bayes classifiers are a popular model choice that has achieved great results.

2.3.2 Support Vector Machine

Support Vector Machine (SVM) is a set of supervised learning algorithms used for both classification and regression. They are based on a simple concept and has proven useful for text classification tasks. Given data points in an n-dimensional space (n is the number of features) that belong to one of the classes, the concept involves finding a hyperplane with the largest margin to the nearest data points of each class and thus separates them. These nearest data points are called support vectors, and an optimal hyperplane has the maximum margin between the support vector(s) of each class and the plane. An illustration of a SVM hyperplane two-dimensional feature space is shown in Figure 2.3. The careful reader should notice that for SVMs to work, they require the data to be linearly separable. However, SVMs can be used when data is not linearly separable by applying functions commonly known as kernels. These kernels transform the input space into a higher dimensional space so that the data become linearly separable. The option to use different kernels on different input spaces makes SVMs versatile. After finding support vectors of each class, the algorithm uses only them and the hyperplane to determine the class of new instances. As a result of this, the rest of the data points can be omitted, which makes SVMs quite memory-efficient. They also perform well in high-dimensional feature spaces but can be subject to overfitting the training data, especially if the data contains noise. Basic SVMs are designed for use in binary classification, but by utilizing techniques like any-of or one-of, they can be used for multi-class prediction.

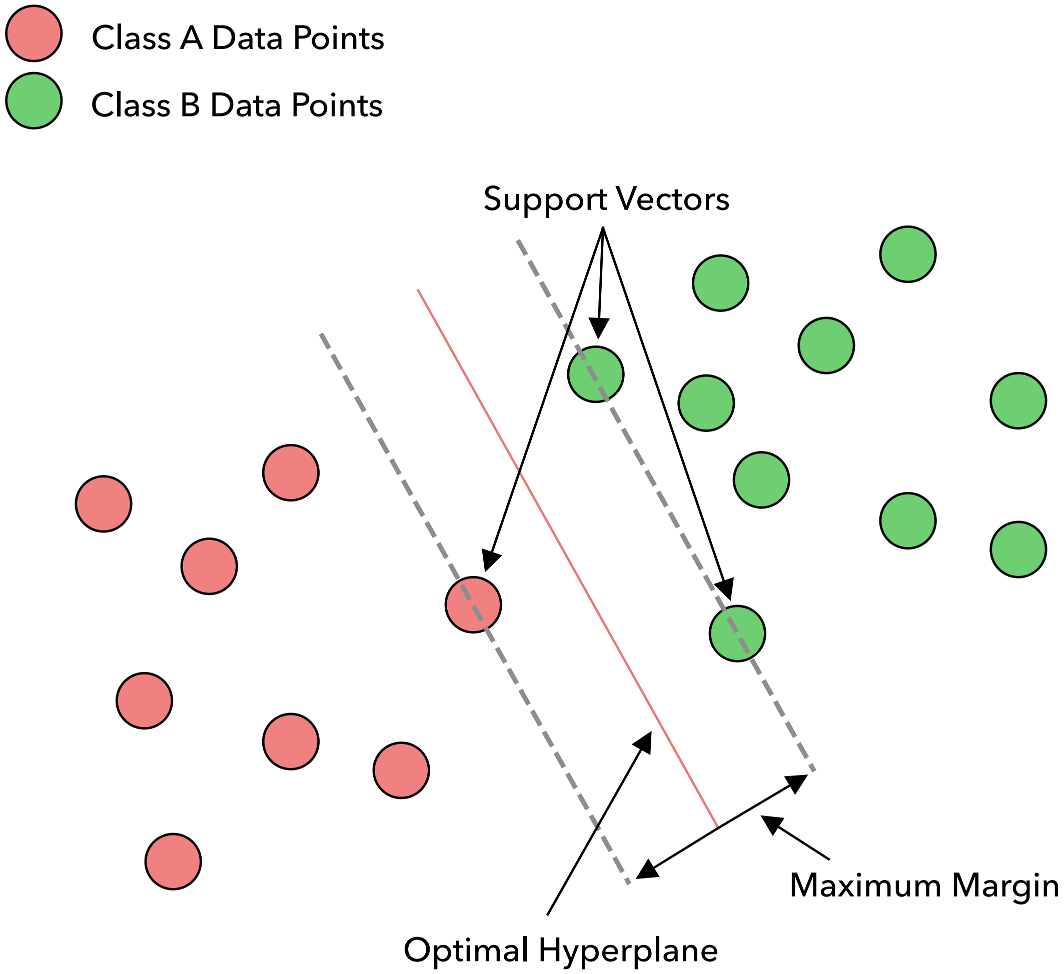

Figure 2.3: SVM hyperplane in two-dimensional space.

2.3.3 Logistic Regression

Logistic Regression (LR) is yet another supervised machine learning algorithm used for binary classification. As with Naïve Bayes, LR uses straightforward statistics to predict the class of some input variables. LR is simple to implement and often used as a baseline for a variety of classification problems, including document classification. LR uses a prediction function called the sigmoid function, shown in Equation 2.3.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

This function is also known as the logistic function and is an S-shaped curve that maps any real-valued number into a number in the range $[0, 1]$. The output of the logistic function can be read as the probability where values closer to 1 or 0 indicate the classifier's confidence towards one of the two classes. Given a threshold (generally 0.5 in binary classification), the LR algorithm uses the probability to determine which class the input belongs to. When LR is used for machine learning, a parameterized version of Equation 2.3 is used, shown in Equation 2.4. Here, the input vector x has n features and for each x_i, there is a corresponding coefficient parameter θ_i. The θ vector is found using maximum likelihood approaches, like gradient descent, and optimized based on the training data. The dot product between the θ and x vector is called the decision boundary and can be a line or a hyperplane in higher-dimensional feature spaces. Regularization can be applied to prevent overfitting and to improve the generalization performance of a LR model, i.e., how well it performs on new, unseen data.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.4}$$

2.3.4 Deep Learning

Machine learning consists of several subcategories, including supervised and unsupervised methods discussed in the last section. Deep learning is another subfield of machine learning which, unlike supervised and unsupervised learning, automatically extract features during the training session. Thus, deep learning is able to learn representations of data through multiple abstraction levels. Deep learning has been around for many years, with varying interest in the field through time. Recently, deep learning has been embraced by the field and successfully applied to many machine learning tasks probably due to its scalability, flexibility, and the fact that more data is available. It requires a sufficient amount of data for deep learning models to perform well, and they are often likely to overfit training data easily and require a lot of computing power. However, enough data is rarely a problem these days, and techniques like regularisation can help prevent overfitting. In addition, with the massive amounts of computational power available today, deep neural networks are becoming more and more used to solve machine learning problems. These three factors, data availability, regularisation, and computing power, are probably essential to the recent massive interest in the field. Deep learning networks exist in many shapes and forms, from simple linear artificial neural networks to more advanced and complex non-linear deep neural networks. This section will briefly present popular types of neural networks commonly used in natural language processing.

Artificial Neural Networks

Artificial Neural Network (ANN) is a set of networks that consist of connected nodes, commonly called artificial neurons or just neurons for short. In the earlier days of ANNs, these neurons were known as perceptrons. The simplest type of ANN is the feedforward neural networks. They have directed graphs which means that the information in the network only flows in one direction. Feedforward networks have one input layer, at least one hidden layer and one output layer, as shown in Figure 2.4. The hidden layer is located between the input and output layer, and Multilayer Perceptron (MLP) can have several hidden layers. Each neuron in the network receives input and based on an *activation function*, defines an output. The activation function is the non-linear transformation a network performs and are an important feature of ANNs. This feature allows the network to learn both linear and non-linear functions. A network's learning is based on the *backpropagation algorithm* which iteratively adjusts the weights between each node using gradient descent. Gradient descent is a popular method for optimizing a

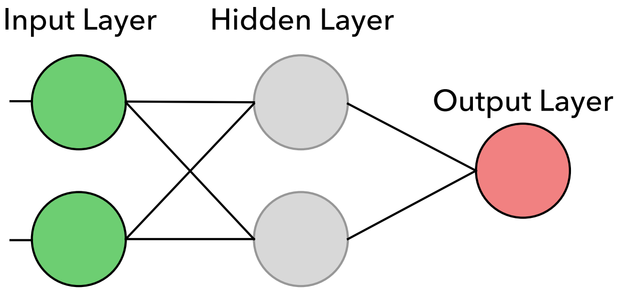

Figure 2.4: The architecture of a fully connected feedforward neural network with one input layer, one hidden layer and one output layer.

function and is commonly used in machine learning and deep learning. Its optimization strategy is to find weights that minimize a loss function. The loss function is based on the difference between a predicted value and the actual label. A popular loss function used for both binary classification and multi-classification is the *Cross Entropy*. Cross-entropy works by measuring the difference between two probability distributions. If the cross-entropy value is substantial, it means that the gap between the two distributions is vast while if the cross-entropy value is small, it means the two distributions are similar in value. Feedforward networks described in this section solve many linear and non-linear machine learning problems and are popular as they are easy to understand and implement. However, more complex types of ANNs exist and are the topic of the next section.

Deep Neural Networks

A Deep Neural Network (DNN) is a variant of neural networks that consist of multiple layers of nodes where each extra layer increases the complexity of the networks and allows them to represent more complex functions. DNNs are usually designed with two to ten hidden layers, but they can consist of hundreds of hidden layers, making them deep. The deeper the networks are, the more resources a computer needs in order to process all data flowing through the network.

Convolutional Neural Network (CNN) (LeCun, 1989) is a feedforward network where the information flows in only one direction, as with the ANNs. Unlike traditional ANNs, CNNs consist of one or more *convolutional* units which perform linear transformations, known as a convolution operation, on the input before passing it to the next layer. A convolutional network may also include pooling layers. These are layers that reduce the

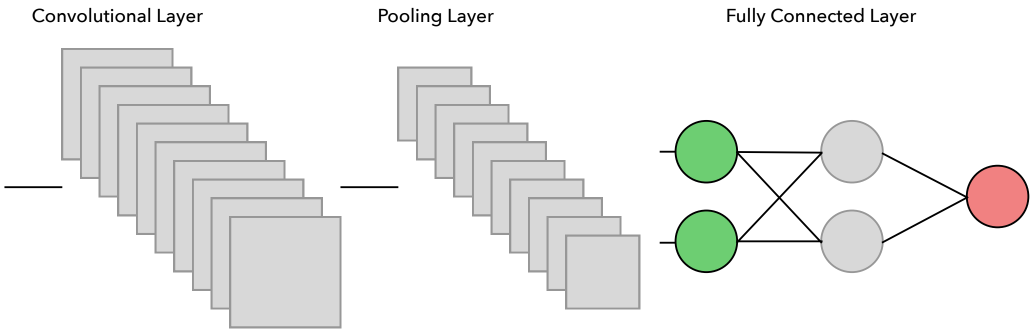

Figure 2.5: Simplified architecture of a Convolutional Neural Network with one convolutional layer, one pooling layer, and a fully connected layer often used for classification.

dimensions of the input either locally or globally. Convolutional or pooling layers are great features of the network as they allow the network to capture more local information, for example, surrounding words in a sentence or pixels in an image. They also reduce the dimension of data, i.e, making the models less complex, which gives faster training sessions and less chance of overfitting. A typical CNN consists of convolutional and pooling layers fully connected to a ANN as seen in Figure 2.5. CNNs are particularly powerful for image recognition and classification due to the digital structure of images, but they are also successfully applied to tasks within the natural language processing field.

Recurrent Neural Network (RNN) (Rumelhart et al., 1986) is another set of artificial neural networks that allow information to flow in cycles. These kinds of networks contain loops that sequentially pass the information from one step to another. Looking at Figure 2.6, we can see that RNNs can be structured as multiple copies of the same network put in sequence and thus the architecture of RNNs makes them very suitable for sequential data such as text. RNNs are said to have internal memory due to the loops in the network. As a consequence of this, future outputs are dependent on past and present decisions, which make the network remember past sequential information that is preserved in the network's hidden state. However, recent inputs have a more noticeable effect on the network than older ones. RNNs are applied with great success to many tasks in NLP, but their shortcoming of forgetting long-term dependencies may leave out valuable information. Therefore, a more complex network based on a recurrent architecture with long short-term memory was developed.

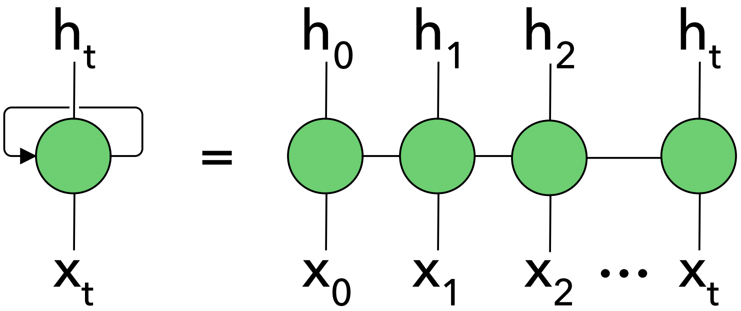

Figure 2.6: Simplified architecture of a Recurrent Neural Network with one input layer, one hidden layer, and one output layer. The unfolded version on the right of the equal sign shows the steps, or time steps, of a recurrent neural network.

Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks are a variant of RNNs that were designed to address the key problem of traditional RNNs. The special feature of LSTM networks is that they contain a more complex node. This node is composed of a cell, an input gate, an output gate, and a forget gate. The cell can store temporal information that has been written into it, and the network can read the stored information. The flow of information into and out of the cell is regulated by the three gates, which also ensure that the information stored in the cell is relevant. This gated cell allows the recurrent network to access information outside the normal flow of information in the network. LSTMs have greatly expanded what we can achieve with RNNs, especially on tasks with long-term dependencies. They are, not surprisingly, a popular model choice when tackling tasks within the field of NLP and have shown great performance.

Attention and Transformers

When humans read a sentence or look at an image, they do not perceive the whole scene in its entirety at once but focus on smaller parts until they have systematically understood the context. Neural networks, on the other hand, base their decision making on the entire input at once. For long input sequences, this may lead to the network forgetting earlier parts of the chain once the whole sequence is processed. Neural networks with attention mechanism may overcome this problem by carefully focus on certain parts of the sentence. The overall idea with the attention mechanism is to learn a vector that captures global information over all the items of an input sequence.

In the field of NLP, the attention mechanism was first proposed in two papers from Bahdanau et al. (2014) and Luong et al. (2015) which used the technique to improve the quality of Neural Machine Translation (NMT) systems. These systems use deep learning models, commonly referred to as sequence-to-sequence (seq2seq) models. Seq2seq models take as input a sequence of items and output another sequence of items, for example, an English sentence as input and a French sentence as output. A typical seq2seq model consists of an encoder and a decoder where the encoder compiles the information from the input sequence into a vector, commonly known as a context vector. The context vector is sent to the decoder, which iteratively produces the output sequence. Both encoders and decoders tend to be RNNs, but other sequential network architectures are also suitable. The context vector summarizes all the information about one sequence into the final hidden state, with the more recent observations more dominant than the earlier ones. The attention mechanism helps distribute information over several hidden states. In a seq2seq model with attention, the encoder passes all the previous hidden states to the decoder instead of just the last one. The decoder then uses the hidden states to decide which parts of the input sequence that are relevant at each decoding time step. The decoder chooses which hidden state to use and which to ignore by weighting the hidden states. Models with attention have proven successful in many NLP tasks, especially in NMT systems, and the attention mechanism has gained a lot of interest in the field lately.

Vaswani et al. (2017) proposed in the famous paper "Attention Is All You Need" a novel network architecture, named the Transformer, which is solely based on attention mechanisms. This architecture, as opposed to traditional attention-based models, does not include any recurrence or convolutions, which allows models to be more parallelizable and use less time during training sessions. The architecture focuses only on attention and thus avoids the problem of long-range dependencies within the input and output sequences. Vaswani et al. (2017) developed a model based on the Transformer architecture which achieved state-of-the-art results on several translation tasks, but more importantly, the paper created a lot of interest for attention-based models and how they could be utilized in other NLP tasks. The Transformer architecture is vital for the work in this thesis as it is a fundamental concept for BERT, which is a trained Transformer Encoder stack.

2.4 Tools and Libraries

Many open-source libraries (set of code previously written) and tools are developed for working with text data. They are easily accessible, often have understandable documentation, and make the job more convenient rather than reinventing the wheel over and over again. This section will briefly describe the tools and programming libraries available when working with text and how they were used in this thesis. The programming language of choice for the experiments is Python mainly because of its extensive support for (machine learning) libraries.

2.4.1 Twitter Developer and Tweepy

Twitter provides an application programming interface API for third-party developers to access content on their platform, e.g., tweets. A Twitter developer account is required to take advantage of the Twitter API. This account has some level of access to either get, insert, delete, or update information about tweets, users, or timelines on the Twitter platform using the API's endpoints. This kind of API style is commonly called REST, which is an acronym for representational state transfer. Working directly with the Twitter

REST API can be cumbersome, so it is beneficial to have a layer on top of the Twitter REST API for easy access to its data. Tweepy is a Python library that provides a ton of handy functions for accessing the Twitter REST API easily. Tweepy was used to retrieve tweets based on a long list of tweet IDs. As mentioned, these tweets have substantial metadata attached to them that can be helpful features for hate speech detection.

2.4.2 Natural Language Toolkit

Steven Bird, Ewan Klein, and Edward Loper are leading the community-driven Natural Language Toolkit (NLTK) (Bird et al., 2009) project which began in 2000. The NLTK library, together with the Standford CoreNLP[2] and CMU Twitter NLP[3], is a leading text processing platform that provides access to more than 50 corpora and lexical resources. Various contributors help the library evolve and make sure it stays up-to-date. NLTK contains tools for tasks mentioned earlier in the chapter like tokenization, stemming, stop-word removal, and more. After retrieving the tweets, NLTK was used to tokenize and remove common words and special characters. NLTK can help with the burden of classifying text and provides implementations of popular machine learning algorithms. However, the optimization and efficiency of these implementations can be insufficient, and other more machine learning specialized frameworks are preferable.

2.4.3 Scikit-learn

Scikit-learn, commonly referred to as sklearn, is a tool for data mining and data analysis. It features machine learning algorithms for classification, regression and clustering such as Naïve Bayes, Support Vector Machines, K-means, and many more. As with many favored libraries, scikit-learn is community-driven and open-source and depends on contributors to evolve. As of 2018, Pedregosa et al. (2011) have the leadership of the popular framework and ensure that it stays up-to-date and well-maintained. Some parts of the experiments in this thesis are implemented with the help of methods and modules from the sklearn framework. After extracting the features and creating feature vectors from the datasets, sklearn makes the model selection an easy task as it is quite simple to try several machine learning models and observe how they perform.

Some concepts from the sklearn framework that were applied in the experiments were transformers, pipeline, and grid search. Datasets transformations can be done by using a set of transformers, to clean the data, scale features, or reduce the dimensionality of the

[2] https://stanfordnlp.github.io/CoreNLP/

[3] http://www.cs.cmu.edu/ ark/TweetNLP/

vectors. If a dataset requires a series of these transformers to be applied, the pipeline class is designed purposefully for this effort. It works by applying a series of transformers followed by an estimator, which makes it a useful tool for the machine learning workflow. The last concept, grid search, is designed to deal with the fact that many of the machine learning algorithms have a huge parameter space. Instead of tweaking the parameters by hand (which would be an exhaustive task), grid search helps to tune the parameters to their very best. Even this tuning can be comprehensive, so the grid search class provides the functionality of inserting a range to the parameters that should be used to limit the possible values. All in all, scikit-learn is a useful framework when working with machine learning.

2.4.4 Pytorch

One of the most popular open-source machine learning libraries for Python is called Pytorch, originally released with a paper from Paszke et al. (2017). Pytorch is easy to grasp, fast, flexible, and commonly referred to as a research-first library. It is often used for deep learning implementations and contains a lot of useful high-level features. One compelling benefit with Pytorch compared to, for example, Scikit-learn, is its ability to use the power of a machine's one or more graphics processing unit (GPU) that supports CUDA. This allows for quicker matrix and vector operations to be computed compared to running only on the central processing unit (CPU), hence very suitable for deep learning neural networks, which require a lot of computations. Building deep neural networks is intuitive and straightforward with this framework and, along with the GPU support, this was one of the main reasons why this library was selected for the work in this thesis.

2.5 Performance Metrics

In statistical analysis, there exist several different ways to measure the performance of a test. When measuring the performance of a classification model, the notations true positives (TP), true negatives (TN), false positives (FP) and false negatives (FP)

		Prediction	
		Positive	**Negative**
Actual	**Positive**	True positive	False negative
	Negative	False positive	True negative

Table 2.1: Two by two confusion matrix often used for binary classification.

describe the outcome of a classifier's prediction. The terms positive and negative indicate the classifier's prediction on an instance, while true and false indicate whether or not that prediction resembles the actual class. Table 2.1 shows a typical confusion matrix commonly used for binary classification. For multiclass classification, additional entries for each class are added to the confusion matrix. A simple evaluation of a classifier's performance is to calculate its accuracy, shown in Equation 2.5.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

However, achieving a high accuracy does not equal a great model due to a possible imbalance in the dataset. To compensate for this imbalance, three types of measures that are popular in information retrieval, classification and natural language processing are *precision*, *recall* and F_1*-score*. Precise evaluation of a classifier's performance is an important part of understanding how the model performs on a certain dataset and to distinguish weak and strong classifiers. If a task involves predicting multiple classes, three different ways of computing the average precision, recall and F_1-scores exists: *micro*, *macro* and *weighted* averaging. Micro averaging will sum the contributions of each class to compute the overall average while macro averaging will compute the average for each class unweighted. Weighted average calculates metrics for each label and uses the support of each label to calculate their weighted average.

2.5.1 Precision

The precision metric can be calculated by dividing the true positive instances with true and false positive instances. If a model achieves high precision, it means that most of the predicted positive instances were correctly classified. Precision is defined in Equation 2.6.

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

2.5.2 Recall

Recall can be calculated by dividing the true positive instances with all of the true positive and false negative instances. With high recall, a model has identified most of the actual positive instances correctly. Recall is defined in Equation 2.7.

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

2.5.3 F_1-score

Both recall and precision should be examined when evaluating a model, and several different metrics have been developed using them, such as the F_1-score. Using statistical terms, the F_1-score can be seen as a harmonic mean of the precision and recall values. So if an algorithm achieves perfect precision and recall, the F_1-score will be 1 or between 0 and 1 otherwise. Combining the precision and recall yields the F_1-score as denoted in Equation 2.8.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.8}$$

3 Data

As stated in Section 2.3, machine learning consists of models that are able to learn from their experiences. In supervised learning, these experiences consist of labeled data which is used to train a model. Datasets should be of proper size for the models to generalize the training data well. Creating and labeling such datasets can be a large and time-consuming task; therefore the work in this thesis will use existing datasets. When creating a dataset, the creators need to decide which labels that best describe the different data samples and how to annotate them correctly. In NLP, many existing datasets containing hate speech are publicly available for use and consist of data from several sources online, mainly Twitter. While searching for datasets, it became clear that there only exist two datasets that make the distinction between hateful and offensive language, others have two classes, either hate and none hate or offensive and non-offensive. Both of these datasets consist of tweets from Twitter but differ in size and how the classes are distributed over the datasets. Section 3.1 will present some existing datasets used in prior research. Section 3.2 will go further into the development and content of the selected datasets for the experiments in this thesis.

3.1 Hate Speech Datasets

One important factor when attacking the problem of hate speech detection is how the datasets are constructed. Which labels are used to annotate the samples? Are the labels distributed well over the dataset? What kind of hate speech knowledge do the annotators possess and what definition of hate speech do they use? Almost every relevant dataset available is labeled by humans, which results in different approaches taken when creating and annotating the datasets. Some researchers use expert annotators (Waseem and Hovy, 2016), others use majority voting among several amateur annotators using platforms like CrowdFlower (Davidson et al., 2017). A combination of both experts and amateurs annotators has also been studied by Waseem (2016) with promising results. In the latter research, Waseem compares expert annotations against majority voted annotation by amateurs and finds that the experts perform equally well classifying binary class, but underperform on multi-class classification. Waseem showed that amateurs are more likely to annotate samples as hate speech than experts due to the amateurs' subjective idea of what hate speech is defined as. However, when the amateurs fully agree that a sample is hate speech, they outperform the expert annotators. This can greatly reduce the annotation work for the expert annotators by asking them only to consider the samples not fully agreed upon by the amateurs.

The findings in the research above demonstrate that annotating a corpus as hate speech or not is a difficult task. This is in line with the work of Ross et al. (2016) who researched how to annotate a dataset containing Twitter threads reliably. To measure reliability, they used Krippendorff's α. Their approach involved asking two groups to answer three different questions. Only one group was given Twitter's definition of hateful conduct. The first question was simply to decide if a tweet was hateful or not, and the second question was to determine whether or not a tweet should be banned from Twitter. Finally, the groups were asked to rate a tweet as offensive on a scale from 1 (not offensive) to 6 (very offensive). The group that was given a definition of hateful conduct partially combined their own subjective meaning of hate speech with the definition from Twitter, although this did not improve the reliability. Ross et al. conclude the paper by claiming that hate speech is a vague concept that needs a standard definition with better guidelines for annotators. They also raise the question if hate speech is best suited as a classification problem or a regression problem. So instead of a binary yes-or-no outcome, a score between zero and one can be assigned to a sample representing how hateful it is.

Chatzakou et al. (2017) had a goal of detecting users with bullying or aggressive behavior on Twitter. They created a dataset were around 10k tweets distilled from a larger set of 1.6 million tweets. The larger set consisted of 1 million random tweets and 650k tweets collected using 309 different hashtags related to hateful speech. The hashtags were used to increase the chance of collecting tweets showing aggressive or bullying behavior. After some preprocessing, the tweets were ready to be labeled. Chatzakou et al. did some experiments to find the optimal batch size of tweets for each worker to work with. They found an optimal batch size of 5-10 tweets which gave a total of 1500 batches for five CrowdFlower workers to label as either "Aggressive", "Bully" or "Normal". These labels are not often seen in hate speech datasets, although they are an important type of hateful language. Founta et al. (2018b) did use similar labels as initial annotations in their construction of a large dataset, but chose to merge them with other types or eliminate them as the annotators selected them only a few times. Section 3.2 will discuss this further.

Due to the lack of a large corpus of messages containing explicit harassment, Golbeck et al. (2017) set out to build one themselves. They first tried to get blocked comments or messages from media houses but were not able to collect enough data and thus turned to Twitter. Like Chatzakou et al., hateful search words and hashtags were used to identify tweets containing the level of harassment the authors searched for. The first 35000 tweets retrieved from this query process were to be labeled as either "Harassing" or "Non-Harassing". Following strict coding guidelines, two coders were used to label the data. If they disagreed on whether a tweet was harassing or not, a third coder would be used to break the tie. All coders were trained on the coding guidelines for three weeks, and the tweets in the corpus took three months to annotate. The final dataset contains 5495 positive examples and 29505 negative examples, thus being one of the larger hate speech datasets in the field. However, the dataset does not contain any offensive language and is therefore not relevant for the work in this thesis.

Most of the dataset above contain either tweets or comments, de Gibert et al. (2018) wanted to create a dataset on sentence-level. Their source was Stormfront, a white supremacist online forum. A total of 10568 sentences were obtained from the forum, separated into ten batches and manually annotated by four coders. Sentences with less than three words or more than 50 words were discarded, the rest were to be labeled as either "Hate", "Non-Hate", "Relation" or "Skip". The "Relation" label is for a sentence that not contain hate speech on their own but together with other sentences. As there are many types of hate speech, de Gibert et al. constructed labeling criteria and guidelines. The final annotated dataset is unbalanced with 1287 positive examples and 8537 negative examples. However, this unbalance is not necessarily a bad feature as it represents the distribution of real-world examples where there is a lot more content without hate speech than with.

Over the years, several hate speech datasets have been produced with data from online sources. Table 3.1 shows an overview of some existing datasets in the field. Datasets can either be published to the public or available upon request and vary in labels, size, and language. However, unlike other NLP tasks, the task of hate speech detection lacks a shared benchmark dataset that can be used to measure the performance of different machine learning models. The lack of a benchmark dataset is a major limitation when comparing state-of-the-art. In a survey on hate speech detection by Schmidt and Wiegand (2017), the authors argue that the community would benefit from a benchmark dataset for the task of detecting hate speech. Until a benchmark dataset is created and agreed upon, the field needs to utilize the existing datasets to create the best possible classification models.

Authors	Source	Annotation labels	Type	Samples	Annotators	Accessible
Waseem and Hovy (2016)	Twitter	Racism, Sexism, Neither	Tweets	16907	1	Yes
Waseem (2016)	Twitter	Racism, Sexism, Neither	Tweets	6909	4 or more	Yes
Chatzakou et al. (2017)	Twitter	Bully, Aggressive, Normal	Tweets	~10k	5	No
Golbeck et al. (2017)	Twitter	Harassing, Non-Harassing	Tweets	35000	2-3	Yes
Ross et al. (2016)	Twitter	Hate, Non-Hate	Tweets	451	2	Yes
ElSherief et al. (2018)	Twitter	Hate, Non-Hate	Tweets	27330	3 or more	Yes
Gao and Huang (2017)	Fox News	Hate, Non-Hate	Comments	1528	2	Yes
de Gibert et al. (2018)	Stormfront	Hate, Non-Hate	Sentence	9916	4	Yes
Davidson et al. (2017)	Twitter	Hate, Offensive, Neither	Tweets	24783	3 or more	Yes
Founta et al. (2018b)	Twitter	Hateful, Offensive, Spam, Normal	Tweets	80000	5-20	Yes

Table 3.1: An overview of some existing datasets in the field of hate speech detection.

3.2 Selected Datasets

As stated above, not many existing datasets have a clear separation between offensive and hateful language. The two datasets presented in this section are the only two datasets found in the literature study that separates between offensive and hateful speech. Even for humans, choosing a label (e.g., hateful, abusive, cyberbullying, offensive) to describe a particular language is a difficult task (Chatzakou et al., 2017). Castelle (2018) states that hate speech is linguistically nuanced and therefore, detecting such speech is one of the hardest tasks in the artificial intelligence field. Some studies, such as Waseem and Hovy (2016), choose to merge offensive and hateful language and treat them as one class when detecting abusive language. This makes it difficult to decide if the classifiers truly are classifying hate speech or "just" text containing derogatory terms. Without any formal definition of hate speech other than the social platforms' privacy policies and definitions made in other studies, it can be difficult for classifiers to separate hate speech from other forms of abusive language. Nevertheless, the fact that hate speech targets minority groups in a potentially harmful manner seem to be a consensus in the field (Davidson et al., 2017) where offensive language is often considered as text that contains slurs or unflattering terms. This section will present the development approaches and dataset statistics for the two datasets selected for the experiments in this thesis.

3.2.1 Davidson et al. (2017)

Davidson et al. (2017) focus on creating a dataset for the task of classifying tweets with both a hateful and an offensive class. They claim that previous work using supervised learning yields low precision because the detection methods tend to classify all texts containing slurs as hate speech. The dataset consists of 24783 English tweets and their label along with some information including the number of annotators and which label the annotators chose for the specific tweet. The authors are anonymized as the dataset only contains the plain text of each tweet. The inter-annotator agreement score between the annotators was 92%. The number of CrowdFlower annotators range from three to nine, and majority voting was used when deciding the final class for a tweet. "Hate Speech", "Offensive Language" or "Neither" were the labels the annotators could choose

Class labels	Normal	Offensive	Hateful
Number of tweets (%)	4163 (17)	19190 (77)	1430 (6)

Table 3.2: Label distribution of retrieved tweets from Davidson et al. (2017).

# of Annotators	Hate Speech	Offensive Language	Neither	Class	Tweet
3	0	3	0	1	bitch plz whatever

Table 3.3: Random sample from the Davidson et al. (2017) dataset showing how the data are represented in the dataset.

from to best describe a tweet. The label distribution can be seen in Table 3.2, while an example of a row from the dataset can be seen in Table 3.3. The majority of the tweets in this dataset are labeled as either offensive or hateful language (83%). This can be explained by the process Davidson et al. (2017) used when collecting the data. They searched for tweets with known hate speech phrases collected from a hate speech repository[1], which resulted in around 33K tweets. They then extracted the time-lines from each author of these tweets, which resulted in a corpus of 85.4 million tweets. The final 25K tweets are random samples from this corpus. Some of the 25K tweets were not annotated as there was no agreement between the annotators. As a result, the final number of tweets in the dataset is 24783.

Davidson et al. tried several machine learning classifiers to observe which performs best on the dataset. They landed on a model using Logistic Regression with L2 regularization because it has performed well in previous work. The classifiers were to predict each tweet with either the class “Hateful”, “Offensive” or “Neither”. Davidson et al. were able to minimize the number of offensive samples classified as hateful due to their multi-class framework. However, almost 40% of human-coded hate speech is misclassified by the classifier as either “Offensive” or “Neither” (as in neither “Hateful” or “Offensive”). Taking a closer look, they find that certain terms are useful when differentiating between hate speech and offensive language. While these terms surely matter when detecting hate speech, Davidson et al. also specify that datasets should include hateful text without any offensive language. Another fact that the researchers point out is that the annotators tend to make decisions based on their own subjective bias when labeling a dataset. Sexist language is more commonly considered offensive while racist or homophobic slurs are recognized as hateful. This observation is consistent with prior findings from Waseem and Hovy (2016). Davidson et al. round off saying that in future work these biases should be corrected as they are likely to enter the classification models.

[1] https://hatebase.org/

3.2.2 Founta et al. (2018b)

Recently, Founta et al. (2018b) set out to create a large and highly accurate annotated dataset. They created a huge dataset compared to already existing hate speech datasets, containing over 100 thousand annotated tweets. They performed comprehensive studies to select which labels that best describe abusive behavior on Twitter. After collecting and processing data, at least five CrowdFlower workers annotated tweets as either "Normal", "Spam" or "Inappropriate". If they mark a tweet with the "Inappropriate" label, the workers were asked to select the most describing label for the tweet, either "Hateful", "Offensive", "Abusive", "Aggressive" or "Cyberbullying". Each label was defined using previous acclaimed work and provided to the workers beforehand, for example, hate speech was defined by the authors as "Language used to express hatred towards a targeted individual or group, or is intended to be derogatory, to humiliate, or to insult the members of the group, on the basis of attributes such as race, religion, ethnic origin, sexual orientation, disability, or gender." The annotation process resulted in "Offensive" and "Abusive" as the most popular categories, followed by "Hateful" and "Aggressive". Cyberbullying rarely occurred according to the annotators.

After the first round of annotating the tweets in the dataset, Founta et al. created a subset of the dataset containing only the tweets labeled as "Inappropriate". This subset was then used in a second round with even more workers (10–20) to strengthen the confidence in the results from round 1. After two rounds, the results from round 2 were quite similar to the ones in round 1. However, the "Abusive" label was slightly more popular in round 2. Comparing the results from the two rounds and calculating the Cosine Similarity between two labels, Founta et al. find that some of the labels correlate, especially the "Abusive" and "Offensive" label. The "Hateful" category does not seem to correlate with any of the other "Inappropriate" labels. After carefully examining the correlations, they chose to omit the "Cyberbullying" label completely and merge the "Offensive" and "Aggressive" class into the "Abusive" class. They retained the "Hateful" label separate as its correlation with the other labels is not significant enough. With the four labels, "Normal", "Spam", "Hateful" and "Abusive", Founta et al. selected some annotators based on basic demographic information to annotate the final dataset.

The outcome of the final round was similar to the preliminary rounds with the "Normal" category as the most popular while the two "Inappropriate" categories "Abusive" and "Hateful" covers around 20% of the dataset. The dataset can at first appear quite imbalanced with a fraction of only 4% hate speech, but this is not necessarily bad as it is a fair representation of the real world where hate speech is only a small portion

Class labels	Normal	Offensive	Spam	Hateful
Number of tweets (%)	53790 (54)	27037 (27)	14024 (14)	4948 (5)

Table 3.4: Label distribution of tweets from Founta et al. (2018b).

out of all user-generated data online. Davidson et al. (2017) and Burnap and Williams (2015) report that they found 5% and 11% hate speech in their data collection process. Table 3.4 shows the label distribution for Founta et al.'s dataset. Not only did Founta et al. provide a large, high quality and publically available hate speech corpus, they also detailed their annotation methodology for others to follow. As future work, they plan on enriching the dataset with more boosted data to decrease the imbalance in the dataset.

4 Related Work

In this chapter, other research in the field of hate speech detection will be presented. This includes work related to implementation and comparison of models, construction of a hate speech dataset, as well as approaches using the concept of transfer learning. The chapter will finish with a brief overview of the current state-of-the-art methods to the task of hate speech detection. Some of the challenges regarding hate speech detection that will be explored in this chapter include the lack of a precise definition of hate speech, why hate speech should be separated from offensive speech and how to produce datasets that correctly represent the nuances of the natural language.

4.1 Transfer Learning in NLP

Transfer learning has recently gained interest in the field of NLP, probably due to the advantages described in Section 2.1. This section will widely present work related to transfer learning in several NLP tasks. This section will begin with a presentation of an early transfer learning technique used in NLP, namely word embeddings. Then, more recent and complex language models will be surveyed.

4.1.1 Word Embeddings

Word embeddings are numerical vector representations of words that use a word's context so that similar words have similar vector representations. These vector representations of words can be trained on a large amount of unlabeled data and used as initializations for the word vectors of deep learning models to transfer knowledge between two different tasks. The use of word embeddings as a representation of words is an early example of transfer learning in NLP that achieved higher accuracy on certain tasks, such as named entity recognition (Turian et al., 2010).

Mikolov et al. (2013b) created a word embedding model, commonly referred to as *word2vec*, that uses the Skip-gram model (Mikolov et al., 2013a) to train distributed representations of words and phrases. The model consists of a shallow two-layer neural network that can be trained to capture the linguistic contexts of words. The model produces a high-dimensional vector space based on the corpus given as input. In the vector space, words with similar contexts will be positioned closer to each other by the model. At the time of release, the word2vec model architecture was very computational efficient, which allowed it to be trained on much larger corpora than previous models, thus outperforming them by a great margin.

Pennington et al. (2014) later created a word embedding model named GloVe, short for global vectors. GloVe is a count-based method that also is able to capture the meaningful linear substructure from prediction-based methods such as word2vec. The model architecture consists of a global log-bilinear regression model that is suitable for creating a vector space with meaningful substructure. GloVe and word2vec are fundamentally similar as they both capture local word-word co-occurrence statistics from a corpus. However, GloVe also captures global word-word co-occurrence statistics by training on the non-zero entries in a global word-word co-occurrence matrix. This can be computationally expensive when training a large corpus, but it is only necessary to train this once. GloVe outperformed word2vec and other previous word embedding models on tasks like word analogy and named entity recognition.

In the task of hate speech detection, pre-trained word embedding models are not present in many existing solutions. However, Kshirsagar et al. (2018) recently fine-tuned a 300-dimensional GloVe Common Crawl Embeddings (840B Token) to detect hate speech in three popular hate speech datasets. The transformed word embedding model, named TWEM, consists of a combination of pre-trained word embeddings and max/mean pooling from a simple, fully-connected transformer. Although Kshirsagar et al. did minimal hyper-parameter tuning and text pre-processing, the transferred knowledge from the pre-trained word embedding model in combination with the pooling showed promising results.

4.1.2 Language Models

Given a sequence of words, a language model is a statistical model that is able to predict the next word of the sequence. N-grams is the most fundamental language model that by grouping word in a continuous sequence of length two (bigrams), three (trigrams) and observe how frequently these groupings appear in a document, tries to predict the next word. This can be taken a step further by utilizing neural networks to create neural language models. These models can be trained on a task, such as machine translation, with large amounts of training data available online to create contextualized word vectors.

Furthermore, a pre-trained model can be applied to a variety of downstream NLP task to improve learning by a considerable margin even with smaller labeled datasets. A pre-trained model can also be fine-tuned with domain-specific data as the data the model is pre-trained with often is different from the data in the target task.

Inspired by the successfully applied transfer learning in computer vision, McCann et al. (2017) developed context vectors (CoVe) by training a deep LSTM encoder from an attentional sequence-to-sequence model. The encoder is trained on the task of machine translation to contextualize word embedding vectors. These context vectors are created by feeding GloVe word embeddings to a standard, two-layer, bidirectional, LSTM network. GloVe captures the global co-occurrences while the network is able to capture contextual information from the sentence as a whole. For downstream tasks like question answering and classification, McCann et al. propose to concatenate each vector in GloVe with its corresponding vector in CoVe. The study shows that when these context vectors are used as initialization vectors for a classification, the model improves the accuracy in many NLP tasks such as sentiment analysis (SST, IMDb), question classification (TREC), entailment (SNLI), and question answering (SQuAD). However, one major limitation of CoVe is that it is bounded by the number of machine translation dataset available to the supervised pre-training of the language model.

To overcome the above-mentioned limitation, Peters et al. (2018) introduced a new type of language model that is pre-trained in an unsupervised way. ELMo is short Embeddings from Language Model and are word representations that are deeply contextualized. ELMo uses a bidirectional Language Model (biLM) to create the word representations, and they are deep because they are a function of all the internal layers of the biLM. The language model is learning to predict the next word by using the history of input received. This task is done in two phases, in the first phase the history contains all words before the target word while the second phase, the history includes all the words after the target word. Both the forward and backward phase is modeled by a multi-layer LSTM where the networks' hidden layers are concatenated together to create contextualized embeddings. These context vectors can then be used to transfer textual understanding to downstream NLP tasks. Peters et al. state that ELMo representations alone considerably improves the state-of-the-art in language understanding problems. ELMo outperformed the above mentioned CoVe in all of the task where direct comparisons were possible. However, both CoVe and ELMo require carefully engineered custom architectures for each task, which makes it difficult and cumbersome to find a suitable architecture for every task.

Howard and Ruder (2018) proposed a language model called Universal Language Model Fine-tuning (ULMFiT) that introduced the concept of fine-tuning the language model.

ULMFiT performs three steps to achieve state-of-the-art results in several classification tasks, first by training the language model on general domain data, then fine-tuning to language model with a task-specific corpus and finally training the classifier. The first step is straightforward and similar to other pre-training of language models where large corpora such as Wikipedia are used to obtain general textual knowledge. Step two involves fine-tuning the language model on task-specific data. ULMFiT introduces two techniques called *discriminative fine-tuning* and *slanted triangular learning rates*, both dynamically changing the learning rate in order to learn task-specific features. In the third step, the pre-trained language model is augmented with two additional linear layers to fine-tune the classifier by using *concat pooling* and *gradual unfreezing*. ULMFiT brought methods to effectively fine-tune a language model to various tasks, solving the custom architecture problem from previous language models. The ULMFiT model also enables classifiers to learn using far fewer labeled examples than before, making it possible to predict on tasks where labeled data is limited.

Inspired by the idea of an unsupervised language model from ELMo, OpenAI's Generative Pre-training Transformer (GPT) created by Radford et al. (2018a) is a language model that expands the amount of text the language model can be trained on. GPT is built by using two existing ideas: a combination of unsupervised pre-training (Dai and Le, 2015) and transformers (Vaswani et al., 2017) with attention. However, GPT is only utilizing the decoder part of a transformer as the language model and thus discards the encoder part. The decoder is built to mask future tokens and therefore makes a reasonable basis for a language model where the task is to predict future words. GPT stack twelve decoder layers where each has masked self-attention as described by Vaswani et al. (2017). This type of language model is capable of learning from a massive corpus with unlabeled data by predicting future words. Also, no task-specific model architecture is needed, for example, a multiple choice task would need three transformer stacks, one for each choice, in order to work properly. Since GPT's pre-training technique only consists of predicting future words, a limitation is that it is unidirectional and only learns left-to-right context. ELMo used LTSM to capture both the left and the right context, but the transformer is not able to perform backward language modeling. However, the GPT language model was able to beat state-of-the-art in 9 out of 12 tasks studied. An improved version of GPT, named GPT-2, was recently released by Radford et al. (2018b). This model introduced a few modifications and additions to the original GPT such as more extensive vocabulary and context size. Given a starting word or paragraph, the model also showcased some impressive text generating without any task-specific training. Even without fine-tuning the model for downstream tasks, such as summarization or translation, GPT-2 showed promising results. However, the exploration of what GPT-2 is capable of has just started, and due to concerns of it being used to generate unintended or malicious content, Radford et al. have not released the largest pre-trained language model to the public.

Bidirectional Encoder Representations from Transformers (Devlin et al., 2018), or BERT for short, is a direct descendant of GPT although instead of using a stack of transformer decoders, BERT uses a stack of transformer encoders. While GPT only trains a forward language model, BERT is a bidirectional language model which means the model considers context on both left and right of the target token. Devlin et al. state that this bidirectional nature of BERT is the single most important new contribution. BERT's language model is not trained on the traditional language model task of predicting future word; instead, Devlin et al. trains the language model on two separate tasks. The first procedure is what they call "masked LM" (MLM) where 15% of the tokens in each input sequence are randomly masked with either a placeholder, a random word or the original word. Without knowledge about which words it will be asked to predict, the language model is to only predict the missing words rather than constructing the entire input. The second task is similar to the traditional language model task, but instead of predicting future tokens, the language model is asked to predict whether a sentence is the next sentence of another sentence. So given two sentences, A and B, there is a probability of 50% that B does follow A and a probability of 50% that B does not follow A. This task allows BERT to better understand the relationship between sentences which is essential knowledge in most NLP tasks. Devlin et al. pre-trains the language model Wikipedia and a book corpus of 7000 books. Like GPT, BERT only requires a few parameters to be fine-tuned for downstream tasks. The paper presents state-of-the-art results of BERT fine-tuned on eleven NLP tasks, including many of the same tasks done by ELMo and GPT. BERT has received a lot of interest in the field due to these results, and with the release of two pre-trained language models, $BERT_{BASE}$ and $BERT_{LARGE}$, BERT can be used as a language model for other language understanding tasks such as hate speech detection.

4.2 Machine Learning Model Selection

Due to the massive amount of user-generated content online and the motivations for the task of detecting hate speech automatically, both an academical and industrial interest has arisen in the field. Many challenges remain unresolved, and poor consistency in features indicative of hate speech across different datasets is one of them. In the paper

"All You Need is 'Love': Evading Hate Speech Detection", Gröndahl et al. (2018) compare five modern state-of-the-art model architectures with three datasets well-known within the field. The models' features are either character-based or word-based and are set to predict either two or three classes. Gröndahl et al. show that when the models are trained with one dataset and tested against another, they perform poorly. They also state that hate speech detection is independent of model architecture after examining that all models perform equally well if they are trained and tested within one dataset. In addition, when each two-class model was tested against offensive language, the models tended to identify the offensive samples as hate speech. This substantiates the claim that consistency in features indicative of hate speech is a real problem. Given the lack of any academical or legal definition of hate speech, annotation in datasets differ, which results in models assigning ordinary offensive text as hate. This is referred to as a false positive, as described in Section 2.5.

Lee et al. (2018) did a similar comparative study, surveying the most frequently studied classical machine learning models as well as recurrent neural networks. They used the recently published dataset from Founta et al. (2018b) containing 100k tweet IDs distributed on four labels: "Hateful", "Offensive", "Spam" and "Normal". Lee et al. state that the dataset has not been studied to its potential and their goal was to achieve a reliable baseline for this dataset. They used five popular machine learning algorithms: Naïve Bayes, Logistic Regression, Support Vector Machine, Random Forest, and Gradient Boosted Trees. They also included the two neural network based models Convolutional Neural Networks and Recurrent Neural Networks. Each algorithm was used to create multiple classifiers with different types of features and parameters. Each algorithm was implemented twice, with either word- or character-level features; however, the results showed that neither of them is an improvement over the other. As for the total metric scores for the word-level representations, the models performed more or less equally well. The lowest F_1-score was 73%, and the highest was 80.5% yielding a difference of 7.5%. This gap agrees with the conclusion reached by Gröndahl et al. (2018) that the architecture used does not remarkably improve the model. Lee et al. also surveyed how well the neural network models perform given some context to the tweets as they assumed a reference to the context would result in better results. In some of the labels, spam, and hateful, the results showed some improvements; however, the results did not show any gain in performance in the total perspective. As a conclusion, Lee et al. created a baseline for this recent dataset and expected that ensemble models of variant models and features could improve the results further.

4.3 Hate Speech Detection Approaches

As mentioned, the field of hate speech detection has gained a lot of interest in the recent decade, and several approaches to detect hate speech have been presented. However, hate speech detection is still considered as one of the most challenging machine learning problems and is far from being a solved problem. Nobata et al. (2016) mention some challenges within hate speech, e.g., that the abusive language with time evolves new slurs and clever ways to avoid being detected. As a result, they performed a longitudinal study over one year to see how trained models react over time. The model employed different features such as n-grams, word embeddings, and other linguistic and syntactic features. The experiments performed by Nobata et al. focus on features rather than model selection and how well the models perform with different sets of features. The data were collected from two domains, finance, and news, and contained some noise according to the authors. All features combined yielded the best performing model; however, looking at individual features, the character n-grams performed best. Waseem and Hovy (2016) also found that their model performed well with character n-grams as features. They experimented with extra-linguistic features such as gender and location, although such demographic features brought little improvements to a model's performance.

Over the last few years, transferring knowledge from word embeddings to be used as input to neural networks has been a common technique used by researchers in the field. Gambäck and Sikdar (2017) experimented with character n-grams in combination with word embeddings from word2vec. They trained four CNNs, one character based with 4-grams, one with word vectors from word2vec, one with randomly generated word vectors and finally one with a combination of character n-grams and word embeddings from word2vec. All models were trained on data from Waseem (2016). The best performing neural network, with an F_1-score of 0.783, was the model with transferred knowledge from word2vec word embeddings. Adding the character n-grams to this system boosted the precision, but lowered the recall yielding a lower F-score. Gambäck and Sikdar, along with Wulczyn et al. (2017), propose a LSTM neural network as a possible deep learning approach to detect hateful language for future work.

Badjatiya et al. (2017) experimented with a lot of different traditional machine learning models and neural networks, including LSTMs. In addition, word embeddings from GloVe were used to possibly increase the overall performance, as discussed in Section 2.1. Their best performing model was a LSTM neural network with random word vectors where the output of the neural network was used as input to a Gradient Boosted Decision Tree (GBDT). This system yielded an F_1-score of 93%, increasing the state-of-the-art by roughly 18 F_1 points. However, these results have shown to be difficult to reproduce correctly (Mishra et al., 2018; Fortuna et al., 2019). Pavlopoulos et al. (2017a) and Pavlopoulos et al. (2017b) tested word embeddings from both GloVe and word2vec in their RNN based neural network and found this model to be effective for comment moderation. The conclusion is based on a comparative study where their recurrent neural network was tested against CNNs and other types of RNNs. Actually, the RNN based neural network with transferred knowledge from word embeddings outperformed previous state-of-the-art methods based on character and word n-gram features.

Pitsilis et al. (2018) showed similar findings with their ensemble RNNs, although without the use of word embeddings. Instead, they converted tweets into vectors using word-based frequency. This approach is advantageous because it is independent of the language used in the message. They fed standard vectorized word uni-grams to multiple LSTM networks and used two mechanisms for aggregating the classifications, namely *Voting* and *Confidence*. Majority voting is the preferred mechanism, but if all classifiers disagree, then the classifier with the highest confidence is given preference. The approach taken by Pitsilis et al. showed the potential of ensemble methods to detect hate speech and outperformed previous state-of-the-art with an F_1-score of 0.932.

Park and Fung (2017) created a hybrid system that tried to capture features from two input levels. Their system includes two CNNs where one is character based, and the other is word based. These two parts are connected at a deeper level of the network where the classification is performed. Park and Fung created multiple datasets by combining samples from Waseem and Waseem and Hovy. Their goal was to explore a two-step classification approach where the first step is to classify a sample as either abusive or normal. Given that the sample was classified as abusive, the second step would further classify the sample as either "Sexist" or "Racist". The two-step approach was compared to one-step multi-class classification, although the results were quite similar. However, the hybrid CNN did outperform an LR and SVM model in the one-step multi-class classification. Park and Fung showcased the potential of two-step classification and will continue to explore this approach. They propose training the two classifiers with separate datasets, e.g., a large general one for the first classifier and a more task-specific dataset for the second classifier. This approach is not unlike the idea behind language models described earlier in Section 4.1.2.

Meyer and Gambäck (2019) proposed an optimised architecture where they combined components with CNNs and LSTMs into one systems. One part of the system used character n-grams as input while the other part used word embeddings as input. The output of each part was merged to produce the final classification. Meyer and Gambäck used the dataset from Waseem and Hovy and obtained greater results than previous comparable solutions. They state that different convergence rates in the system's components may have reduced the performance and suggests utilizing dynamic convolutions as future work.

As mentioned, the task of hate speech detection does not have a commonly accepted benchmark dataset. Most of the research discussed so far in this section use the popular dataset from Waseem and Hovy or the slightly modified dataset from Waseem. These datasets use the labels "Sexist", "Racist" or "Neither" and one can argue if these labels are sufficient enough to represent hate speech. Another dataset widely used in recent research is the one from Davidson et al. They separate hateful language from offensive and normal language and thus make the task harder than separating hateful language from just normal speech. Zhang et al. (2018) used this dataset, along with others, to test the performance of their model, which consists of a combination of LSTM and CNN. However, they merged the offensive class with the normal class, so the final dataset only contained two classes, namely hateful and normal language. The system works by feeding word embeddings from word2vec into the CNN to produce input vectors for the LSTM network with GRU cells which perform the final classification. The model outperformed the state-of-the-art on 6 out of 7 datasets and shows again that deep neural networks are good at detecting hate speech.

Founta et al. (2018a) also used the dataset from Davidson et al., but chose not to merge the offensive samples with the normal ones and thus took on the challenge of separating hateful and offensive language. Their system consists of two networks, one RNN based with text input and one feed-forward ANN with metadata as input. The two networks run in parallel with a concatenation layer and finally a classification layer. Founta et al. tried many combinations of these two network with different features and training strategies. Their best performing model achieved an F_1-score of 0.89, which is slightly below the baseline LR model from Davidson et al. Unfortunately, the paper did not include metrics for each class, which would be interesting to analyze to see how well the model classified true hate speech samples.

Kshirsagar et al. (2018) created a model that surpasses Davidson et al.'s baseline. The model, named Transformed Word Embedding Model (TWEM), consists of pre-trained word embeddings as input to multiple MLP layers. The simple network architecture allows for few parameters and minimal feature preprocessing compared to previous methods. The model achieved a total F_1-score of 0.924, where the baseline model achieves 0.900. However, the increase in F-score is only due to better performance on the "Normal" and "Offensive" class, actually performing worse on the "Hate" class with F_1-score of

0.49, which is two F-score points below the baseline. This agrees with Malmasi and Zampieri (2018), who state that a key finding of their study is the noticeable difficulty of distinguishing hateful language from profanity. They implemented a few supervised machine learning algorithms along with more advanced ensemble classifiers and tested them on the dataset from Davidson et al. (2017). The classifiers were tested with different sets of features, including n-grams and word representations based on clustering. The best accuracy was obtained by an RBF kernel SVM meta-classifier yielding a F_1-score of 79.8%. The hardest class to classify was the “Hate” class which was often confused with the “Offensive” class.

Malmasi and Zampieri did an extensive analysis of the experiments and the results in an attempt to explain why it is so difficult to separate hateful language from offensive language. The study shows that the tweets with the highest probability of being tagged as hate usually are targeted at a specific social group. The key finding in the report was that the standard features are not sufficient enough to successfully distinguish hateful and offensive language. Malmasi and Zampieri state that context and features with the ability to obtain a deeper linguistic understanding of documents may be required to improve the performance. They also mention that the variability of labels in the annotated data can be a reason for the classifiers’ weak performance on the “Hate” class.

However, a recent study by Gaydhani et al. (2018) showed remarkable performance by their model on a combination of datasets from Davidson et al. and Waseem (2016). They used n-grams as features and fed the TF-IDF values of these into typical machine learning classifiers like Support Vector Machine, Naïve Bayes and Logistic Regression. Logistic Regression outperformed the other two models obtaining a final F_1-score of 96%. The proposed model is near perfect misclassifying only 0.035% of the true hate speech samples. However, these results have shown difficult to reproduce even though their code repository is available at GitHub[1]. Analyzing the final train and test data supposedly used by the model shows that 74% of the test data is either duplicate or already in the training data which results in a highly biased test set as pointed out by another user in the repository’s “issues” page.

Many of the studies mentioned in this section used word embeddings to transfer knowledge to their models. 2018 was the year of language models, and more hate speech detection approaches with this type of transfer learning are being published at the time of writing. Zampieri et al. (2019) presented results and findings from the recent SemEval-2019 Task 5 and 6 competition which consisted of three subtasks. Pérez and Luque (2019) and Indurthi et al. (2019) were the top teams in Task 5, and they used

[1] https://github.com/adityagaydhani14/Toxic-Language-Detection-in-Online-Content

ELMo together with LSTM neural network to achieve great results. Liu et al. (2019) used BERT Base with default parameters to deliver some of the best results in Task 6. Zampieri et al. observed that state-of-the-art deep learning models, in particular, BERT, were used by the top teams in all subtasks endorsing the rise of advanced transfer learning in NLP.

5 Preliminary Study

This chapter will present a preliminary study done prior to the work in this thesis. The purpose of the study was to investigate how different classical machine learning models perform on the two datasets described in Section 3.2. The study also gave insights into various techniques for preprocessing and feature selection and in general, a good understanding of the field of hate speech detection. The first section will present how the classifiers were implemented while the second section will present the experiments and results for the preliminary study.

5.1 Implementation

This section will present how the study was implemented from collecting the data to classifying tweets. The implementation consists of three different machine learning algorithms trained on one dataset and tested with another to get an insight into how well the models generalize and if they are able to distinguish hateful and offensive language. The first section will describe the two different datasets and how they are created, while the final sections will present how the data were collected and prepared for the classifiers, as well as how the classifiers were implemented with modules and functions from the scikit-learn framework.

5.1.1 Data

The datasets used in the preliminary study are the same two as used in this thesis, both described in Section 3.2. The notation dataset **D** will be used for the dataset from Davidson et al. (2017) while the notation dataset **F** will be used for the dataset from Founta et al. (2018b). This preliminary study did some experiments with training with dataset **F** and testing with dataset **D** to investigate how well a model would perform on data outside the original dataset. However, these results are not relevant to the work in this thesis and will be omitted. The models' performance on both datasets will be presented separately and serve as a baseline for the models implemented in this thesis.

5.1.2 Collecting Tweets

Dataset **F** provided a comma-separated values (CSV) file of tweet IDs and their corresponding labels. So to make use of this dataset, it was necessary to utilize the Twitter API to retrieve the tweets. The Twitter API requires some keys and tokens which can be provided by creating a Twitter developer account and initiate a Twitter platform app. To access the Twitter API, a Python package called Tweepy[1] was used, which makes this process less cumbersome. Tweepy offers functions for handling Twitter's rate limits[2]. Tweepy includes a function called `status_lookup()` which takes in a list with a max length of 100 tweet IDs and returns a JSON array with tweets as JSON objects. These JSON objects contain the tweets and their attached metadata. Since tweets in this dataset had a fair chance of being deleted (due to the possible abusive content), the JSON objects were fetched in chunks of one hundred at a time and stored in a large JSON array. However, only the original tweet text and label were relevant for use later in the study. Dataset **D** provided a CSV file with the tweets and some additional annotator information, but again, only the original tweet and their labels were used.

5.1.3 Pre-processing Data

Since text data online can be from several sources, cleaning online user-generated text is a problem specific task. There are many choices to be made and trade-offs when processing natural language, which makes it a complicated task. It can be beneficial to start with a simple processing approach that produces a simple vocabulary and gradually increase the complexity to see if it results in a better performing model. Tweets tend to contain a lot of Internet slang, which results in several abbreviations per tweet, emoticons, and links to websites. Threads are also commonly used on Twitter, which results in users mentioning each other in the tweets. Cleaning data is an essential task as it lays the foundation for feature extracting. Removing too much can result in the loss of valuable features while retaining too much can result in a complex vocabulary. As described in Section 2.4.2, the NLTK framework provides many different tokenizers that can be used in various domains. For this study, the `TweetTokenizer` was applied to the tweets from both datasets. This tokenizer features the options for preserving the casing, emoticons, and user mentions. A decision was made about using the tokenizer's default settings except for the "preserving the casing" parameter which was set to false. Reducing all letters to lowercase is a common strategy to avoid duplicate word entries with the same

[1] http://www.tweepy.org/

[2] https://developer.twitter.com/en/docs/basics/rate-limiting.html

meaning and to reduce the dimensionality of the features. However, some proper nouns are derived from common nouns, and reducing them to lowercase can result in the loss of proper nouns. Finally, stop words were filtered out using NLTK's list of English stop words as they might not be distinguishing enough in the tweets. Scikit-learn requires a specific folder structure to load the data correctly, so after cleaning the data, each tweet was written to a plain text file and stored in one of the three folders: "Hateful", "Offensive", "Normal".

5.1.4 Feature Extraction

A machine learning algorithm cannot interpret text data, so constructing numerical feature vectors that can be used as input to the machine learning algorithms is necessary. This process is commonly known as feature engineering and is a crucial step in order to get well-performing models. Careful feature extraction is important for the classifiers to generalize beyond the training data and thus be able to predict new, unseen data. Scikit-learn includes a function called `CountVectorizer` which builds a dictionary of features and transforms the tweets into feature vectors. It represents the tweets as a matrix of token counts, i.e., as a bag of words, described in Section 2.2.2. The function accepts a wide range of parameters and customization such as n-gram range and minimum document frequency. However, the default values are reasonable. The feature vectors consist of word-level unigrams (individual words) with a minimum document frequency of two. Experiments with other n-grams such as bigrams and trigrams were tried, but none of them gave a significant increase in model performance. After creating the bag of words representation, the TF-IDF weighting scheme was applied to weight the feature vectors. This is to highlight less common terms that occur in many tweets and are likely to be less informative.

5.1.5 Classifiers

After the process of feature engineering, the feature vectors are ready to be used as input to different classifiers. When loading in data, due to the specific folder structure described above, scikit-learn is able to infer the label of each tweet as well as the three classes used for prediction. Scikit-learn provides implemented versions of popular machine learning algorithms with high functionality. Three supervised learning algorithms, described in Section 2.3, were chosen to be implemented: Naïve Bayes (NB), Support Vector Machine (SVM) and Logistic Regression (LR). They all have built-in support for multiclass classification and were trained to classify tweets as either "Hateful", "Offensive" or

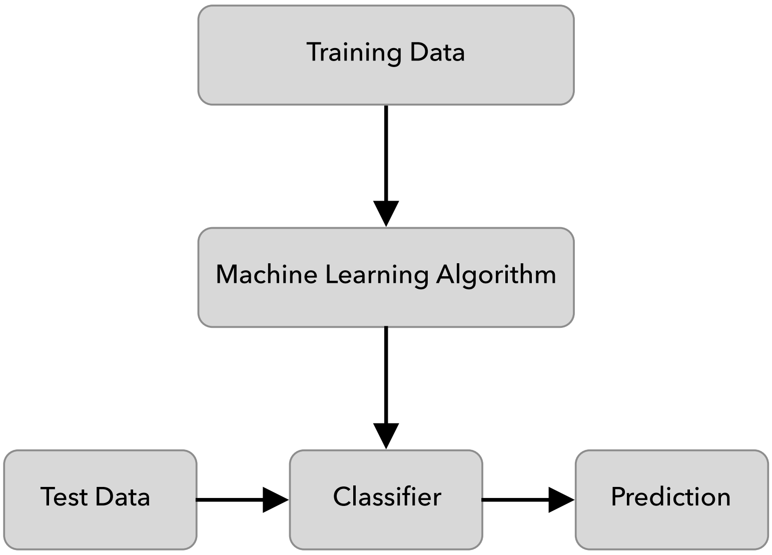

Figure 5.1: Overall architecture for each of the three model implementation.

“Normal”. While Naïve Bayes has multiclass classification inherently, the Support Vector Machine and Logistic Regression use a one-vs-all or one-vs-rest approach. This approach is a commonly used strategy for multiclass classification and the default parameter for the algorithms. Selecting the right parameters can have a significant impact on a model’s performance.

Although this study does not aim at achieving the best performing model, the classifiers were tested with different parameters until a satisfying set of parameters was found. For the SVM, the hinge loss function with L2 regularization was implemented, which results in a linear SVM. Linear SVM requires scaling of the feature vectors, so scikit-learn’s normalizer module was used to normalize the feature vectors. LR was implemented with the liblinear solver with L2 regularization. A multinomial version of the NB was used where the distribution of each feature is assumed to be multinomial as opposed to, for instance, Gaussian. This assumption works well for discrete features as word counts in text and hence an appropriate choice for both datasets. More advanced methods for selecting the optimal hyperparameters can be achieved by using a grid search. Nevertheless, the models from scikit-learn perform well out of the box with little parameter tuning as we shall see in Section 5.2.2. Figure 5.1 presents the overall architecture of each of the three model implementation.

To evaluate the performance of a model, several metrics were produced, including the ones explained in Section 2.5. For each of the models, built-in functions from scikit-learn were used to calculate the accuracy, confusion matrix, precision, recall, and F_1-score. The latter three are computed for each class and then used to calculate micro, macro, and weighted averaged scores. All these metrics provide valuable insights into the models' overall performance for each class.

5.2 Experiments and Results

In order to carry out an experiment and receive some results, it is crucial to have a reasonable hypothesis that one could prove true or false. This study hypothesized that a classifier should be able to distinguish hateful, offensive, and normal language. The intention behind the hypothesis was to get an insight into how model selection affects the performance on each dataset and create a baseline for the work in this thesis. By training three models, the statement from Gröndahl et al. (2018) that model architecture is less important than the type of data and annotation process, could be further investigated. This experiment also gains many insights into challenges in the field of hate speech detection, which is valuable for the work in this thesis. The first section will explain the experiment plan while the second section will present the results.

5.2.1 Experiment Plan

Datasets **D** and **F** will be used to test the performance of three classical machine learning algorithms. This experiment is divided into two parts; the central part is to check how well models perform on each dataset while the second part is to observe if model architecture affects the performance. In addition to examining the overall performance, how well the models distinguish hateful language from normal and offensive language is an interesting part of this experiment to observe. This observation gives insights into how the models classify hate speech compared to human coders. When the models are trained on such large dataset which can contain noise, overfitting the training data is a possibility. Both datasets **D** and **F** are split into a training set and a test set. Both datasets were split into two sets, the training set of size 80% of the original dataset and test set of size 20% of the original dataset.

5.2.2 Results

As this experiment is divided into two parts, the following two sections will present the results obtained from each part. The metrics precision, recall and F_1-score, described in Section 2.5, were used to evaluate the models' performance. The results from the

Model	Normal			Offensive			Hateful			Total (macro)		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1
Naïve Bayes	**0.84**	0.74	0.79	0.89	**0.96**	0.92	0.35	0.11	0.17	0.69	0.61	0.65
Support Vector Machine	0.80	**0.91**	0.85	**0.93**	**0.96**	**0.94**	**0.61**	0.08	0.14	**0.78**	0.65	**0.71**
Logistic Regression	0.83	0.90	**0.86**	**0.93**	0.95	**0.94**	0.46	**0.22**	**0.30**	0.74	**0.69**	**0.71**

Table 5.1: Precision, recall and F_1-score of each class for the three models performing on dataset **D** (scores in bold are for the best performing model for each metric and class).

		Predicted		
		Normal	**Offensive**	**Hateful**
Actual class	**Normal**	737	81	2
	Offensive	106	3669	73
	Hateful	41	185	64

Table 5.2: Confusion matrix for the LR model performing on dataset **D**.

three classifiers' performance on datasets **D** and **F** are presented in Table 5.1 and 5.3, respectively. Table 5.2 and Table 5.4 show the confusion matrices for the best performing model which were the LR model on both datasets.

5.2.3 Performance on Dataset D

Looking at Table 5.1, we can see that all three models overall perform well for the "Normal" and "Offensive" class while the recall score on the "Hateful" class is quite bad. The SVM model performs best on the "Offensive" class while the LR model clearly performs best on the "Hateful" class. The SVM and LR models outperform the NB model with a total F_1-score of 0.71, which is a notable increase over the NB model's F_1-score of 0.65. The results in the "Hateful" columns show that the models are not able to classify hate speech in this dataset. With F_1-score of 0.31, the LR was the model with

Model	Normal			Offensive			Hateful			Total (macro)		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1
Naïve Bayes	0.93	0.92	0.92	0.77	0.87	0.82	0.18	0.08	0.11	0.63	0.63	0.63
Support Vector Machine	0.92	**0.99**	0.95	**0.88**	0.86	0.87	**0.80**	0.11	0.19	**0.87**	0.65	**0.74**
Logistic Regression	**0.94**	0.98	**0.96**	**0.88**	**0.89**	**0.88**	0.59	**0.21**	**0.31**	0.80	**0.69**	**0.74**

Table 5.3: Precision, recall and F_1-score of each class for the three models performing on dataset **F** (scores in bold are for the best performing model for each metric and class).

		Predicted		
		Normal	**Offensive**	**Hateful**
Actual class	**Normal**	8208	135	34
	Offensive	260	2491	56
	Hateful	127	205	270

Table 5.4: Confusion matrix for the LR model performing on dataset **F**.

the highest score on the hateful class. However, this low F_1-score is not sufficient enough to correctly classify hate speech. Taking a closer look at the confusion matrix computed for the LR model predicting on datasets **D** in Table 5.2, we see that the model predicts tweets as less hateful than the annotators. 64% of true hate speech is misclassified as "Offensive" and 14% as "Normal" and the model was only able to classify 22% of the true hateful samples correctly. This pattern occurs in the confusion matrix for the other two models as well. Similar findings were shown by both Davidson et al. (2017) when they tested an LR model and Malmasi and Zampieri (2018) when they tested an RBF kernel SVM model on dataset **D**.

5.2.4 Performance on Dataset F

Table 5.3 shows the results of the three models performing on dataset **F**. We immediately see by the bold scores that the LR model outperforms the other two models in almost all classes except a few where the SVM model is better. However, as with dataset **D**, the overall performance of the SVM and LR models are similar with F_1-scores of 0.74. For the "Normal" and "Offensive" class, the models are able to classify the examples, but for the "Hateful" class, the models' ability to predict the class is poor. With an F_1-score of 0.31, the LR model is not able to recognize hateful samples even though it achieves nearly twice as good scores as the other two models on this class. Lee et al. (2018), as described in Section 4.2, created a baseline for the newly released dataset **F** testing several models with different complexity. They obtained a similar macro-averaged F_1-score of 0.30 when using a recurrent neural network model. Looking at the confusion matrix in Table 5.4, we see that the same pattern occurs as with dataset **D**. That is, 55% of the true hate speech samples are misclassified as either "Normal" or "Offensive". They are most often recognized as "Offensive" by the model with 34% of the true hate speech samples predicted as offensive. This is also the case in the offensive row where 9% of the true offensive samples were classified as "Normal" and only 2% as "Hateful".

5.2.5 Performance of Different Models

The second part of this experiment was to examine the performance of different models to see if model architecture does matter when classifying hate speech. Looking at the F_1-score for each class, the models perform highly similarly with few exceptions. Table 5.1 shows that LR outperforms both of the other models with F_1-score of 0.86 for the "Normal" class, 0.94 for the "Offensive" class and 0.30 for the "Hateful" class. However, the performance of the SVM and LR models are overall quite similar even though the LR model is better at classifying hateful samples. The similar F_1-score of 0.71 can be explained by the SVM model's high precision on the "Hateful" class. This is similar for dataset **F** in Table 5.3 where both the SVM and LR models achieved an F_1-score of 0.74 due to the SVM's high recall on the "Hateful" class. Further investigation in the three models' confusion matrices computed for each dataset shows that they tend to learn the same features, hence model architecture does not seem to affect the models' ability to learn the features significantly. This is in line with the conclusion from Gröndahl et al. (2018).

6 Architecture

As pointed out in Section 4.3, many previous attempts at classifying hate speech with transfer learning use word embeddings from either word2vec or GloVe. Word embedding techniques only capture the semantic relations among words, whereas language models are more complex and can capture the meaning of a word in a sentence, i.e., the word's context. The work in this thesis will focus on such language models and explore the effect of transferring knowledge from a substantial pre-trained language model to a classifier predicting hateful and offensive expressions from online communities. Despite many available language models, as discussed in Section 4.1.2, BERT is the language model of choice for the work in this thesis. BERT has shown excellent results on a wide variety of NLP tasks (Devlin et al., 2018), but has only been initially tested on the task of hate speech detection. This chapter gives a detailed overview of the system architecture implemented in this thesis. The first section will present the preprocessing techniques while the second section will explore BERT in more detail. The final section concludes the architecture with an overview of the final classifier.

6.1 Collecting Data

The two datasets used for the work in this thesis are the same used in the preliminary study and are introduced and discussed in Section 3.2 and 5.1.1. The two datasets were selected as they were the only ones found in the literature review that separate between hateful and offensive language. As in the preliminary study, the notation dataset **D** and **F** will be used throughout the rest of the thesis to represent the datasets from Davidson et al. (2017) and Founta et al. (2018b), respectively. Both datasets were collected early in the preliminary study through the Twitter API, as discussed in greater detail in Section 5.1.2. As the authors of dataset **F** only provide tweet IDs for researches to retrieve tweets through the Twitter API, some tweets may for several reasons not be retrievable, e.g., a tweet or the user account behind a tweet may have been deleted. This is, in fact, the case for many of the tweets in dataset **F**; of the 99799 provided tweet IDs, 68299 tweets were retrievable. The label distribution of the retrieved samples compared to the original

Dataset version	Normal	Offensive	Spam	Hateful
Original	53790 (54)	27037 (27)	14024 (14)	4948 (5)
Available	41784 (61)	14202 (21)	9372 (14)	2941 (4)

Table 6.1: Label distribution of retrieved tweets from Founta et al. (2018b). Number in parenthesis represents the percentage of the total dataset.

label distribution for dataset **F** is shown in Table 6.1. Although more data is preferable for a deep neural network, the data retrieved should still be sufficient enough for a model to learn the features. As for dataset **D**, the authors provided the anonymized tweets directly in a comma-separated values (CSV) file along with some annotation data.

6.2 Preprocessing

Neural networks, as with other machine learning algorithms, are not capable of taking raw text as input, so some text preprocessing of the data is necessary. This process is a difficult task, as the natural language contains a lot of noise, especially online. Twitter has set a strict limit of 280 characters per tweet, which often leads to the authors making use of abbreviations and internet slang. Additionally, many tweets contain retweeted content, mentions of other users, URLs, hashtags, emojis, etc. Although most of these are commonly removed from the tweets, preprocessing is a domain-specific task that should be done with care so that essential features do not get lost in the process. As language models can capture context between words and prefer complete sentences, only a simple preprocessing technique was used to clean the data. The NLTK library, introduced in Section 2.4.2, provides a flexible Twitter-aware tokenizer, `TweetTokenizer`, that was used to remove URLs, numbers, mentions and an 'RT' symbol often used to mark that a tweet is a retweet of another tweet. However, the tokenizer was only used to clean tweets and not to tokenize them, and this is done later by another tokenizer specialized for BERT. Stop words were *not* removed even though that is a commonly applied technique in previous solutions. This choice was made to keep as much context as possible in each tweet for the language model to capture. As BERT takes sentences as input, the goal of the preprocessing was to create complete sentences out of the tweets without too much noise.

The specialized tokenizer for BERT, named `BertTokenizer`, follows two steps to convert the cleaned data to input that BERT is able to understand:

- **Basic tokenization** performing both text normalization and punctuation splitting.

The first convert all whitespace characters to spaces, lowercase the input, and strip out accent markers. The latter add whitespace around all punctuation characters, which is any character that is not a letter, number or space.

- **WordPiece tokenization** performing whitespace tokenization, i.e., split each word on whitespace, to the output of the above procedure. Then, WordPiece tokenization is applied to each of these tokens to format the input correctly for BERT.

The WordPiece tokenization step above is a technique to segment words into subword-level and used for several NLP tasks. For example the different tenses "looking", "looks" and "looked" will all be segmented into "look ##ing", "look ##s" and "look ##ed". This segmentation allows the model to not treat these words completely different as any other count-based technique possibly would. The model will also observe the segmented part "look" more and possibly learn more about it. Out of vocabulary (OOV) words are never possible with word segmentation like WordPiece because any word that does not occur in the vocabulary is segmented into subword units. BERT's English vocabulary consists of 30,522 segmented words that are learned beforehand. Some of the entries in this vocabulary are placeholders that can be replaced with words not already in the vocabulary. ElSherief et al. (2018) created a list of keywords commonly used as hate speech, and most of those keywords were placed in the unused placeholders in the vocabulary. This modified vocabulary was only used in the process of further training BERT from its checkpoints, as described in Section 6.5 below.

After WordPiece segmentation, the preprocessing of the data is finished. As discussed in Section 2.2.1, other preprocessing techniques, like stop-word removal, stemming and, lemmatization, can be used to refine the data further. However, as the BERT language model takes sentences as input, most of the preprocessing in this approach focused on creating complete sentences.

6.3 BERT Model Architecture

BERT is a recently released language model that is the first unsupervised, deeply bidirectional system for pre-training a general-purpose understanding of language. Section 4.1.2 introduced the language model along with other recent work that BERT is built upon. BERT's language models can be pre-trained from scratch using only a plain text corpus or fine-tuned with a domain-specific corpus. Although pre-training is a one-time procedure, it is relatively expensive requiring a large amount of crawled text and computational power. Along with the paper, Devlin et al. (2018) released several pre-trained models for researchers to use. Among the released pre-trained models, the experiments used these two models:

- **BERT Base**, Uncased: 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT Large**, Uncased: 24-layer, 1024-hidden, 16-heads, 340M parameters

The two models used in this thesis were trained on the English Wikipedia and BookCorpus[1] for 1M update steps. BERT Base consists of a stack of twelve encoder layers with 768 hidden units and twelve attention heads while the larger model, BERT Large consists of a stack of twenty-four encoder layers with 1024 hidden units and 16 attention heads. Attention and transformers are discussed in Section 2.3.4. Both of these models are lowercased and pre-trained checkpoints that can either be trained with more data or fine-tuned with task-specific data. Both of these approaches were implemented and tested in the experiments.

Model Input and Output

After preprocessing, it is important to construct an input representation that BERT accepts. BERT takes a sequence of words as input, just like any encoder of the transformer. The models are trained with sequence length up to 512, but when fine-tuning, the sequence length can be shorter to save substantial memory. Each encoder in the stack applies self-attention and then passes the results through a simple feed-forward network. The output of the feed-forward network is finally handed over to the encoder above. As we can see in Figure 6.1, the first token is a special `[CLS]` token. The token's purpose will become clear later on, but it is necessary to add it to each input sequence. BERT is able to represent both a single sentence or a pair of sentences, for example, a question and an answer. Thus BERT can be used for sentence pair classification tasks and another special token, `[SEP]`, that separates these sentences needs to be added between two sentences. However, for single sentence classification tasks, the token only needs to be added at the end of each input sequence. A final input sequence may look like this:

[CLS]	i	do	nothing	every	day	[SEP]

We know by now that a computer does not understand plain text as input, so some numerical representation of the sentences is needed. Most language models pass each

[1] https://yknzhu.wixsite.com/mbweb

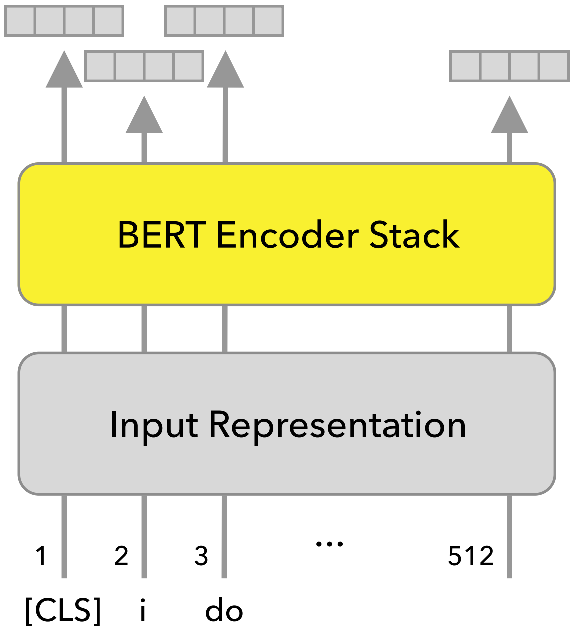

Figure 6.1: Simplified model architecture of BERT's input representation.

input token through a token embedding layer to achieve a numerical representation. BERT solves this by passing each token through three different embedding layers and summing the results from each layer. Each of these three layers converts an input sequence of tokens to a vector representation of size $(n\,, 768)$, where n is the number of tokens in the input sequence.

- The *token embedding* layer transforms each WordPiece token into a vector representation of a fixed length. This fixed length is 768 and 1024 for BERT Base and BERT Large, respectively. Our example sentence above would be of shape $(7\,, 768)$ after passing through this embedding layer, assuming BERT Base is used.

- *Segment embeddings* are mostly relevant for sentence pair classification where BERT needs to distinguish the tokens in each input pair. This layer consists only of two vector representations, namely zeros and ones. The vector that consists of zeros is assigned to the first input sequence, i.e., every token left of the `[SEP]` token. The other vector with ones is assigned to the second input sequence, that is, every token right of the `[SEP]` token. In cases with only one input sentence, the whole vector is filled with only zeros. The length of these token vectors is the same as for token embeddings.

- The final layer consists of *positional embeddings*. BERT's design allows for input sequences up to length 512. Without too much detail, Devlin et al. inform that BERT uses learned positional embeddings which means that this layer works like a lookup table of shape $(512\,,768)$ for BERT Base. The first row is a fixed, learned positional vector embedding that is always assigned to the first token of the input sequence. The second row is always assigned to the second token, so on and so forth.

These three vector representations are summed element-wise to construct a single vector representation. The final vector is used as an input representation for BERT's encoder stack. The architecture up until now is no different than a typical transformer's encoder; however, the model output is where BERT separates itself from a traditional transformer.

Each token position in the input sequence outputs a hidden vector with length of 768 for BERT Base and 1024 for BERT Large. Each encoder outputs hidden vectors that can be used as contextualized word embeddings that can be fed into an existing model. This means that BERT can be used in the same way as the ELMo language model discussed in 4.1.2. Devlin et al. did achieve results not far behind the fine-tuning approach on some of the NLP tasks with contextualized word embeddings. The paper also explores different ways the hidden vectors can be used to achieve the best embeddings. However, it is the fine-tuning approach that is the focus of the experiments. For this, only the hidden vectors from the final encoder in the stack are relevant. As a matter of fact, it is only the hidden vector in the first position that is used for the downstream task of sentence classification. As we can see in Figure 6.1, the hidden vector from the special `[CLS]` token mentioned above is the only output relevant for single sentence classification tasks. This output vector can then feed into a final classifier, such as a simple feedforward neural network discussed in Section 2.3.4.

6.4 Final Classifier

For a single sentence classification task, only the final hidden state vector at the first position is relevant. By design, this output vector corresponds to the special `[CLS]` word embedding. This vector can be used as input to any classifier. Although Devlin et al. achieved great results with their models only using a single-layer neural network, the final systems used in the experiments are slightly modified with a more complex architecture. An additional linear layer of size 2048 was added to increase the complexity of the model. An RNN model, introduced in Section 2.3.4, was also implemented and tested, but later

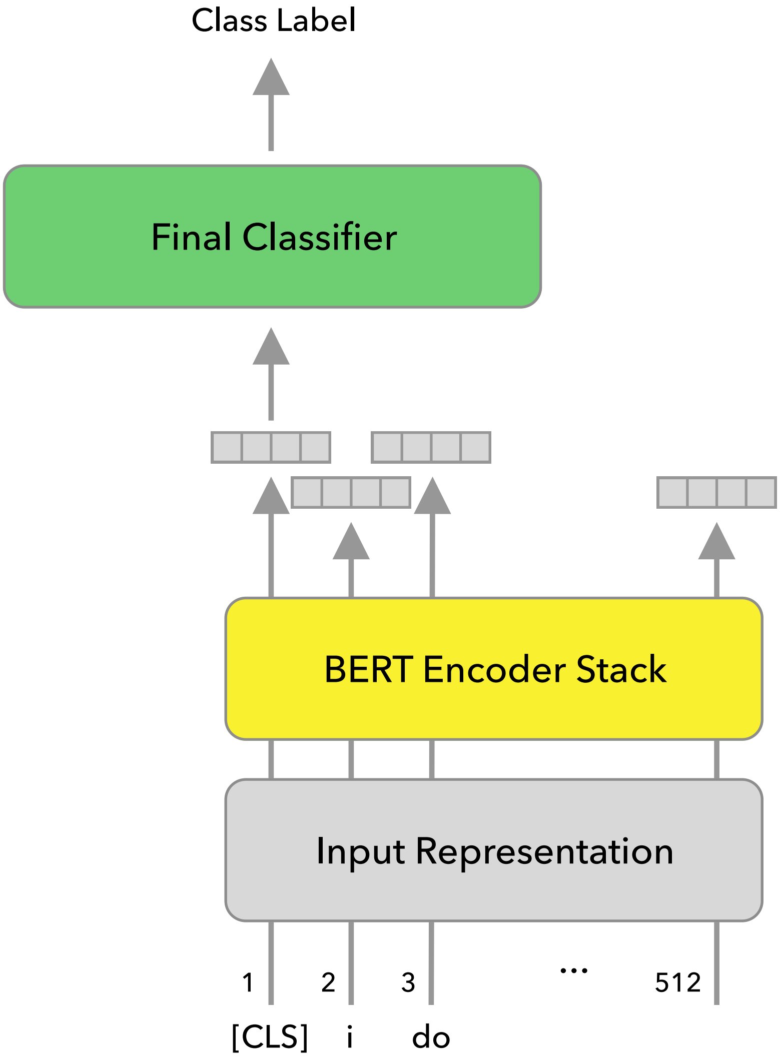

Figure 6.2: Overall system architecture. The final classifier includes a feedforward neural network with one input layer, one hidden layer, one output layer and one softmax layer.

omitted as the learning did not improve.

For fine-tuning, only the number of labels needs to be added as a new parameter, three and four for the systems used in this thesis. All BERT parameters and the final classifier network parameters are fine-tuned jointly to maximize the systems' ability to predict the correct class. The logits from the final linear layer are then passed through a standard Softmax layer to calculate the final label probabilities. Between BERT's pooled output and the first linear layer, and between the first and second linear layers, dropout is utilized to regulate the systems to reduce the risk of overfitting by the model. During training, dropout is randomly applied to some connections between the neurons to prevent them from co-adapting too much. In addition, cross entropy is used to calculate the classification error of each sample. A specialized version of the Adam optimizer is used to update the whole network's weights iteratively based on the training data. An extended version of Figure 6.1 is shown in Figure 6.2, which gives an overall overview of the entire architecture of the systems implemented for the experiments.

6.5 Further Language Model Training

Starting from BERT's Wikipedia and BookCorpus checkpoint, it is possible to further train the language model with domain-specific corpora. This will likely be beneficial to the final performance of the whole system as the domain language is quite different from, for example, general Wikipedia articles. This technique of using unlabelled data from the same domain as the target task to train the language model further using the original pre-training objective(s) was first seen in the ULMFiT paper by Howard and Ruder (2018). This additional step helped the model improve the performance significantly, even with less than 1000 labeled training examples. Since the taken approach in this thesis only uses two datasets, we can see from Table 3.1 that there are still a lot of datasets from the target domain available. Remember, the pre-training only requires the raw text, and so the labels are irrelevant. All available datasets, except the two used for the target task, in Table 3.1 were collected and used to further train BERT on domain data.

One of BERT's pre-training objectives is next sentence prediction in which the model predicts whether one sentence follows another sentence or not. This is further discussed in Section 4.1.2. As a result, the input format for further training BERT is a single file with untokenized text and one sentence per line. Each document, i.e., a tweet, is separated by a blank line. NLTK's `sent_tokenizer` was used to split documents into sentences of at least one word. Since tweets rarely consist of multiple complete sentences due to Twitter's 280 character limit, some tweets were split in the middle to construct

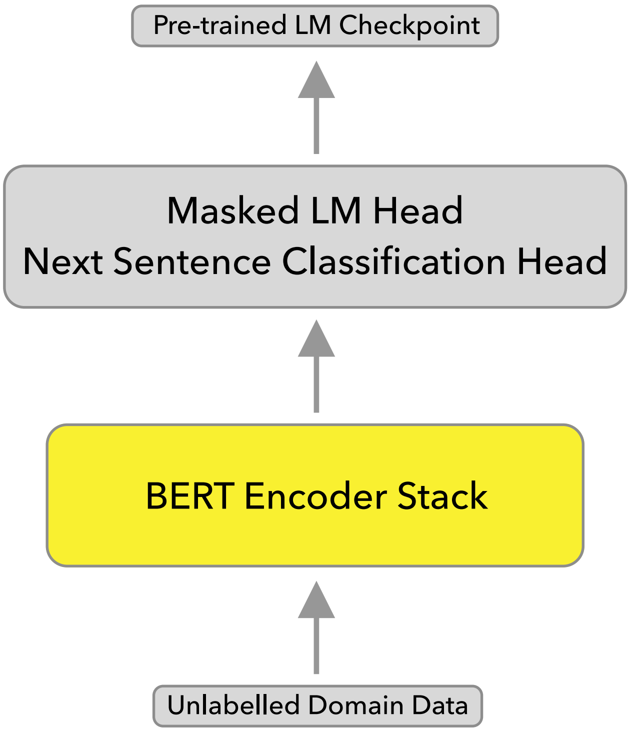

Figure 6.3: Simplified architecture of the pre-training process.

two sentences instead of discarding them. Other datasets were formatted more easily, for example, the Stormfront forum data from de Gibert et al. (2018) contained a large folder where each text file was a sentence. As mentioned, when further training BERT only a single file with all documents is required. Therefore, all text data from the datasets were merged into one file yielding one large text file with nearly 170,000 lines. This file was then used to further train two language models from BERT Base and Large checkpoints on the two original pre-training objectives, masked LM and next sentence prediction as described in Section 4.1.2. The pre-training process is illustrated in Figure 6.3. The output of this process, two language models, trained on Wikipedia, BookCorpus, and domain data was used in the experiments to investigate the effect of further training the language model with domain-specific data.

7 Experiments and Results

As a part of the research in this thesis, some experiments were carried out with an aim to answer questions related to the research questions and the overall goal of the thesis. This chapter presents the experiments conducted to investigate if a large pre-trained language model can separate between hateful, offensive, and normal language in two popular hate speech datasets. The first section presents the overall plan for carrying out the series of experiments. The second section provides the reader with an overview of each experiment, together with the information necessary to reproduce individual experiments. The final section presents detailed results obtained from each experiment.

7.1 Experimental Plan

In order to keep a series of experiments effective and in a structured fashion, an experimental plan is crucial. The plan for this research's experiments consists of two parts, one for each dataset described in Section 3.2. In both parts, the two pre-trained language models BERT Base and BERT Large will be used to investigate the effect of transferring knowledge from a language model to the downstream task of hate speech detection. These experiments are related to Research Question 2 as described in Section 1.2. In addition to the two original pre-trained language models from Devlin et al. (2018), two language models further trained with domain-specific data from BERT Base and Large Wikipedia and BookCorpus checkpoints will be used to investigate the effect of utilizing domain data to improve the models' domain-specific language understanding. This strategy relates to Research Question 3, but also to Research Question 1 as these experiments explore different features. All four language models are connected with a final deep learning classifier, as described in Chapter 6. The difference between BERT Base and BERT Large is the size of the network as presented in Section 6.3. The four systems can be summarized as:

- **BERT Base** pre-trained on Wikipedia and BookCorpus.
- **BERT Large** pre-trained on Wikipedia and BookCorpus.

- **BERT Base*** pre-trained on Wikipedia and BookCorpus and fine-tuned with unlabelled domain-specific text data.
- **BERT Large*** pre-trained on Wikipedia and BookCorpus and fine-tuned with unlabelled domain-specific text data.

Each of the four systems' performance will tested with the two selected datasets from Davidson et al. (2017) (dataset **D**) and Founta et al. (2018b) (dataset **F**). Dataset **F** originally annotates tweets as either "Hateful", "Offensive" "Spam" or "Normal", so to correctly compare the results with results from previous research, experiments with dataset **F** with original annotations were carried out. However, when separating hateful, offensive and normal language, the "Spam" class is redundant and omitted in the second part of the experiments. This part uses dataset **F** without the "Spam" class which will be denoted as dataset **F***. Dataset **D** is only annotated with either "Hateful", "Offensive" or "Normal" and thus the four systems are only tested once with dataset **D**. Each dataset is used to experiment with two different pre-training approaches in order to investigate the effect of transferring knowledge from language models with general and domain-specific language understanding. The analysis of these experiments may answer questions related to the research questions formulated in the introduction of this thesis. In addition to the experiments presented in this chapter, several other experiments were conducted during the exploration phase to gain insights into which preprocessing techniques, classifier architectures, and hyperparameters that worked best with the language models.

7.2 Experimental Setup

In all of the experiments, only a few hyperparameters differentiate the four systems. This section will present data, hyperparameters, and other configurations necessary to run the experiments. As described in Section 6.3, BERT Base and Large only differ in network size. They consist of 110M and 340M parameters, respectively, and thus require a sufficient amount of memory. Therefore, all experiments were run on devices with at least 64GB RAM, the same amount recommended by the creators of BERT. The input sequence length and batch size during training are factors that affect memory usage a lot. The two original language models were pre-trained with a sequence length of 512 and a batch size of 256. The two fine-tuned language models were fine-tuned with a sequence length of 128 and a batch size of 32. All four language models were trained with the Adam optimizer; however, choosing a more memory efficient optimizer may reduce memory usage as the Adam optimizer is quite memory hungry. The learning rate for the Adam optimizer that was found to be optimal was 3e-5 for the fine-tuning process and 2e-5 for the classification process. These learning rates were found after doing an exhaustive search with parameters suggested by Devlin et al. (2018). Other parameters shared by the four systems are a dropout probability of 10% on all layers, the number of training epochs which was 3 and an evaluation batch size of 8. Dropout and early stopping are applied to reduce the risk of overfitting the model. When the number of epochs surpasses five, there is a clear tendency that the difference between the training and validation error increases, which usually is an indication of overfitting. Both datasets were split into a training set containing 80% of the total samples and a test set containing the remaining 20% of the samples. The better alternative here, cross-validation with multiple folds, was not implemented due to framework limitations. As the datasets were fairly imbalanced, Scikit-learn's stratified splitting function was used to ensure that the classes are equally balanced in each set. Shuffling of the samples was also applied each run. All text data in the experiments were lowercased as the cased versions of BERT are recommended for tasks such as named entity recognition. The fine-tuning of the language models took around 3 hours on two Nvidia V100 GPUs with 32GB RAM each, while classification with BERT Base and Large took on average around 1 hour and 2 hours, respectively.

7.3 Experimental Results

This section will present the results of each experiment. The results from the two datasets, dataset **D** and **F**, will be presented separately. The results on dataset **D** from each of the four systems will be given in one table while the results on dataset **F** will be given in two tables, one for each multiclass classification. The slightly modified dataset **F** without the "Spam" class will be denoted as **F***. The performance of the systems will be measured by the metrics described in Section 2.5, namely Precision, Recall, and F_1-score. These metrics will be calculated for each class as this gives more detailed insights into how the models classify each sample. The *macro* averaged and weighted averaged total for each metric will also be presented for comparison reasons although *micro* averaged total may be more suitable for unbalanced datasets. Each of the four systems will be denoted as BERT Base, BERT Large, BERT Base* and BERT Large* where the asterisk denotes the further pre-trained language models described in Section 6.5. The following two sections will present the results obtained by the four models on each of the two selected datasets.

7.3.1 Dataset from Davidson et al. (2017)

The first dataset the four models were tested on was from Davidson et al. (2017) and contained a total of 24783 tweets. The dataset is quite unbalanced with 77% of the tweets being annotated as “Offensive” and only 6% being labeled as “Hateful”. The collection approach taken by the authors is the reason for this unbalance. This approach consisted of crawling tweets from authors known for hateful expressions, as discussed in Section 3.2. As we can see in Table 7.1, all four models perform more or less equally in almost all metrics. The obvious observation here is that all four models are able to correctly classify tweets as “Normal” and “Offensive” fairly well. BERT Large is the model that performs best with a final macro averaged F_1-score of 0.759, with the other three models not far behind. This is in line with Devlin et al. (2018) who found that BERT Large outperforms BERT Base across all tasks tested.

Out of the four models, BERT Large is also the model that obtains the best scores for the “Hateful” class with precision, recall and F_1-score of 0.520, 0.364 and 0.428,

		Model			
		BERT Base	BERT Large	BERT Base*	BERT Large*
Normal	Prec.	0.867	**0.889**	0.883	0.883
	Rec.	**0.906**	0.888	0.893	0.888
	F_1	0.886	**0.888**	**0.888**	0.885
Offensive	Prec.	**0.941**	0.938	0.929	0.932
	Rec.	0.953	0.959	**0.965**	0.961
	F_1	0.947	**0.948**	0.947	0.946
Hateful	Prec.	0.497	**0.520**	0.477	0.460
	Rec.	0.343	**0.364**	0.213	0.259
	F_1	0.406	**0.428**	0.294	0.331
Micro avg.	Prec.	0.910	**0.913**	0.909	0.908
	Rec.	0.910	**0.913**	0.909	0.908
	F_1	0.910	**0.913**	0.909	0.908
Macro avg.	Prec.	0.768	**0.782**	0.763	0.758
	Rec.	0.734	**0.737**	0.690	0.703
	F_1	0.751	**0.759**	0.725	0.729
Wgt. avg.	Prec.	0.903	**0.905**	0.895	0.897
	Rec.	0.910	**0.913**	0.909	0.908
	F_1	0.906	**0.909**	0.902	0.902

Table 7.1: Precision, recall, and F_1-score of each class for the four models performing on dataset **D** (scores in bold are for the best performing model for each metric and class).

respectively. Out of all the examples the model predicted as hateful, 52% of them were correctly classified. Only 36% of the total true hateful tweets were classified correctly, yielding a low recall score. The two models with general language understanding, BERT Base and Large, outperform the two models with domain-specific language understanding on the "Hateful" class. On this class, BERT Base* obtains a F_1-score of 0.294 compared to BERT Large's F_1-score of 0.428. This gap in F_1-scores is unexpected as the intention of further training the language models with domain-specific data was to increase the hateful language understanding.

7.3.2 Dataset from Founta et al. (2018b)

The dataset from Founta et al. (2018b) is nearly three times the size of the one from Davidson et al. The label distribution is also more balanced with roughly half of the samples labeled as "Normal" and the other half distributed between the "Offensive", "Spam" and "Hateful" labels. Although only 6% of the tweets are annotated as "Hateful", this is a fair representation of the real world where only a small portion of the content posted online is hate speech. This dataset is first tested with all the original labels even though "Spam" samples are not relevant for the research in this thesis. The "Spam" samples are included to correctly obtain results that can be compared to previous research such as the baseline from Lee et al. (2018). To compare the models' performance with the traditional machine learning models' performance carried out in the preliminary study, a slightly modified version of this dataset without the "Spam" samples is also produced. This dataset, denoted **F***, is more relevant for the work in this thesis which focuses on the task of separating hateful, offensive, and normal language.

Dataset F

The best scores for each metric are more spread across the four models when performing on this dataset, and there is no clear difference between the models. We observe from Table 7.2 that all four models would obtain the same F_1-score of 0.67 if two decimal precision were used. As with the previous dataset, the models are able to correctly classify tweets as "Normal" and "Offensive" quite well while misclassifying most of the true "Hateful" and "Spam" tweets. The best F_1-scores for the "Normal" and "Offensive" class are 0.869 and 0.884, respectively. Although they are both obtained by BERT Base*, we observe that the other models are only a few thousandths points away from the same score. The only telling difference between the models is the scores on the "Hateful" class. BERT Base is the clear winner on this class with the highest precision, recall and F_1-score.

		Model			
		BERT Base	BERT Large	BERT Base*	BERT Large*
Normal	Prec.	0.850	0.860	0.855	**0.863**
	Rec.	**0.887**	0.871	0.883	0.868
	F_1	0.868	0.865	0.**869**	0.865
Offensive	Prec.	0.848	**0.850**	0.848	0.845
	Rec.	0.919	0.920	**0.924**	0.923
	F_1	0.882	**0.884**	**0.884**	0.882
Hateful	Prec.	**0.562**	0.513	0.543	0.500
	Rec.	**0.279**	0.269	0.270	**0.279**
	F_1	**0.373**	0.353	0.361	0.358
Spam	Prec.	**0.567**	0.556	**0.567**	0.555
	Rec.	0.473	0.536	0.495	**0.542**
	F_1	0.516	0.546	0.529	**0.548**
Micro avg.	Prec.	0.811	0.810	**0.812**	0.809
	Rec.	0.811	0.810	**0.812**	0.809
	F_1	0.811	0.810	**0.812**	0.809
Macro avg.	Prec.	**0.707**	0.695	0.703	0.691
	Rec.	0.640	0.649	0.643	**0.653**
	F_1	**0.672**	0.671	**0.672**	0.671
Wgt. avg.	Prec.	0.798	0.801	0.800	**0.802**
	Rec.	0.811	0.810	**0.812**	0.809
	F_1	0.804	0.805	**0.806**	0.805

Table 7.2: Precision, recall, and F_1-score of each class for the four models performing on dataset **F** (scores in bold are for the best performing model for each metric and class).

However, this model perform similar to the other three with a total macro averaged F_1-score of 0.672. This is the highest total F_1-score which were obtained by BERT Base and Base*. We observe again that these models are not able to recognize true hate speech and misclassify most of the hateful samples as either “Normal” or “Offensive”. This is represented by the poor recall scores in the “Hateful” column in Table 7.2.

Dataset F*

Removing the “Spam” class from the original dataset, we immediately see an increase in the models’ scores for all three classes as shown in Table 7.3. The increase is most noticeable for the “Normal” class going up from an F_1-score of 0.869 to 0.962; however, this is expected as the “Normal” class was highly confused with the “Spam” class in the

		Model			
		BERT Base	BERT Large	BERT Base*	BERT Large*
Normal	Prec.	0.956	0.956	0.956	**0.957**
	Rec.	**0.968**	0.967	0.967	0.964
	F_1	**0.962**	0.961	0.961	0.960
Offensive	Prec.	**0.865**	0.861	0.860	**0.865**
	Rec.	0.920	**0.926**	0.921	0.919
	F_1	**0.892**	**0.892**	0.889	0.891
Hateful	Prec.	0.573	**0.574**	0.531	0.485
	Rec.	**0.299**	0.264	0.264	0.284
	F_1	**0.393**	0.362	0.353	0.358
Micro avg.	Prec.	**0.923**	0.922	0.921	0.919
	Rec.	**0.923**	0.922	0.921	0.919
	F_1	**0.923**	0.922	0.921	0.919
Macro avg.	Prec.	**0.798**	0.797	0.782	0.769
	Rec.	**0.729**	0.719	0.717	0.723
	F_1	**0.762**	0.756	0.748	0.745
Wgt. avg.	Prec.	**0.915**	0.914	0.912	0.911
	Rec.	**0.923**	0.922	0.921	0.919
	F_1	**0.919**	0.917	0.916	0.915

Table 7.3: Precision, recall, and F_1-score of each class for the four models performing on dataset **F*** (scores in bold are for the best performing model for each metric and class).

previously presented results in Table 7.2. The increase is less notable for the “Hateful” class although BERT Base outperformed the other models by quite a margin with a final F_1-score of 0.393. BERT Base is surprisingly the model that performs best overall on dataset **F*** beating the other three models on nearly every metric and with a final macro averaged F_1-score of 0.762.

Remarkably 97% of the tweets labeled as “Normal” were correctly classified by the model. Only 30% of true hateful samples in the dataset were correctly classified meaning 70% of the total samples were misclassified as either “Normal” or “Offensive”. Again, the models seem to recognize true hate speech as less hateful than the annotators. This trend is also seen the other way around with only a small portion of the normal and offensive samples classified as more toxic by BERT Base. From Table 7.3 we observe that the two models trained with domain-specific data, BERT Base* and BERT Large*, performed worse on the “Hateful” class than the other two models with F_1-scores of 0.353 and 0.358, respectively. Again, this is an interesting observation as more training with domain-specific data has shown to increase the performance of models in previous solutions.

8 Evaluation and Discussion

The first part of this chapter interprets and evaluates the experiments presented in Chapter 7. The second part discusses the findings in light of the research questions formulated in Section 1.2. Relevant comparisons to previous literature, important results, and limitations of the experiments will also be discussed throughout the whole chapter.

8.1 Evaluation

The experiments in this thesis consisted of four system setups where each of them used a different version of BERT as a backbone. Two hate speech datasets were used together with a slightly modified version of one of the datasets, to test each system. The experiments were carried out by following the experimental plan presented in Section 7.1. This section will evaluate several aspects of the experiments and the results obtained by the four systems on the three datasets.

8.1.1 Datasets

During the literature review presented in Chapter 4, only two datasets were found to separate between offensive and hateful language. Other datasets either merge these two types of abusive language or chose only to use one of them as their negative class. This separation between offensive and hateful language is essential to include in any dataset as models should be trained to preserve freedom of speech and correctly moderate unintended content. Although we in the experiments can see likeness and trends in the results obtained for each dataset, a comparison of these results should be avoided as they can lead to misconclusions. Also, the datasets are produced differently with varying size, annotators, type of data, and label distribution, which make any comparison of results unreliable. The lack of a shared, precise definition of hate speech results in annotators receiving different annotation guidelines, and thus, two assumably similar datasets may contain different interpretations of, for example, the hate class.

As a reminder, the datasets form Davidson et al. (2017) and Founta et al. (2018b) are denoted **D** and **F**, respectively. The modified dataset without the “Spam” class is denoted **F***. The main difference between the two datasets used in the experiments is the size and label distribution. The size of dataset **F** allows for more training samples than dataset **D** although systems transferring knowledge from pre-trained language models have shown that even small datasets can achieve similar performance (Howard and Ruder, 2018). The four models’ overall performance on datasets **D** and **F** are the same despite the fact that the latter dataset allows for more language model fine-tuning. The label distribution in dataset **F** is more realistic than dataset **D**, where a large portion of the samples is labeled as “Offensive”. However, this unbalance of dataset **D** does not seem to affect the models’ performance noticeably. The reason is probably that dataset **D** contains a sufficient amount of class samples for the models to learn the other two classes. This ability to learn with a few training examples is, in fact, one of the main advantages of using language models instead of traditional word embeddings. The unbalance of dataset **D** could be resolved with one of several resampling techniques, but was chosen not to for comparison reasons. Enriching datasets **F** and **F*** with more nuances of hate speech samples could also improve the models’ ability to separate hateful language from offensive and normal language. This sample boosting of the original version of datasets **F** and **F*** is mentioned as future work by the authors of the dataset, Founta et al.

8.1.2 Language Model Selection

Although datasets without the distinction between offensive and hateful language were irrelevant for testing the models in the experiments, they were used as unlabelled data to further pre-train two BERT language models. This additional training is intended to give the language models domain-specific language understanding and has shown to increase the overall performance in other NLP tasks (Devlin et al., 2018). However, the results obtained from the experiments show that the two models with domain-specific language understanding performed worse or equal to the language models with general language understanding. As we can see in Table 7.1, the worst performance of the two extended BERT models was on dataset **D**. BERT Base* and Large* obtained macro-averaged F_1-scores of 0.725 and 0.729, respectively, while the original BERT models obtained F_1-scores of 0.751 for BERT Base and 0.759 for BERT Large. The difference between these scores is a result of the models’ performance on the “Hateful” class as the performance on the “Normal” and “Offensive” classes are near identical for all four models. BERT Large outperforms the other three models on the “Hateful” class with a F_1-score of 0.428. This is in line with the original BERT paper where Devlin et al. (2018)

found that BERT Large outperformed BERT Base on several NLP tasks. BERT Large's increased performance may be due to the larger model size, as increasing the models' size has in recent years shown to lead to improvements on large-scale tasks.

However, this is not the case for the results obtained by BERT Large on dataset **F***. Looking at Table 7.3, we observe that the smaller model BERT Base outperforms BERT Large on nearly every metric. The most compelling difference can again be seen in the "Hateful" row, where BERT Base achieved an F_1-score of 0.393 compared to BERT Large's F_1-score of 0.362. This difference is a result of better recall obtained by BERT Base, which means that the model correctly predicted more true hate speech instances than BERT Large. The datasets are randomly shuffled before divided into two fractions, which add some randomization to each run, and this may explain why BERT Base was the best performing model on dataset **F***. Cross-validation would be the preferred training and testing strategy, but due to framework limitations, this strategy was not feasible to implement.

Surprisingly, there is no telling difference when comparing the two models with general language understanding to the two models with domain-specific language understanding. Further training with large domain-specific corpora is expected to be beneficial and increase the performance on downstream tasks like hate speech detection. However, the results from the experiments do not reflect this assumption, and it seems like all four models are able to capture similar features, thus performing equally well, as seen in, for example, Table 7.2. As described in Section 6.5, next sentence prediction is one of two pre-training objectives performed by BERT during training. So in order to further pre-train the language model, it is necessary to obtain documents containing at least two sentences. Obtaining such documents became a limitation, as the domain-specific data used in the experiments mostly consist of tweets, as seen in Table 3.1 on page 28. Tweets often consist of only one single-sentence and omitting every single-sentence tweet would lead to a much smaller training corpus. So in order to include single-sentence tweets in the training corpus, they were split at the middle word to construct two sentences. This strategy is not optimal and may be one of the reasons why BERT Base* and Large* did not perform as expected. After the experiments were carried out, an update to the framework's repository was released where several improvements to the pre-training process were suggested. Example code to pre-generate the input data from raw documents following the methodology used in the original BERT paper was one of the improvements. This algorithm may not solve the problem with single-sentence documents, but it is certainly recommended as future work to investigate this new methodology and explore if further pre-training with domain-specific data can improve the overall performance.

8.1.3 Misclassifications

Generally, the results from each dataset indicate that it is hard to separate hateful language from offensive and normal language. This was also the key finding stated by Malmasi and Zampieri (2018) and Davidson et al. (2017) when testing their models' performance on dataset **D**. Analysing the confusion matrices for the best performing model on each dataset, shown in Table 8.1, Table 8.2 and Table 8.3, we see indeed that the "Hateful" row contains most misclassifications. The models highly confuse true hate speech with offensive and neutral speech. For dataset **D**, most of the annotated hateful samples are confused with the "Offensive" class, and this may be due to the skewed dataset where the "Offensive" samples dominate. With datasets **F** and **F***, there is roughly an equal distribution of misclassifications between the "Offensive" and "Normal" class. This indicates that neither of the tested models using features from the pre-trained language model is capable of distinguishing hateful language from offensive and neutral language with acceptable accuracy.

To investigate BERT Base's predictions on dataset **F*** deeper, some correctly and incorrectly classified instances were sampled. The sampled instances with their annotated and predicted labels are summarized in Table 8.4 on page 76. Instances 1, 2, and 3 from this table are all annotated as "Hateful", but predicted as "Normal" by the model. None of these samples strikes as clear hate speech and are perhaps mislabelled by the human coders and correctly predicted by the model. Instances 4, 5, and 6 were predicted to be offensive, but were all annotated as "Hateful". Instance number 5 is tricky to annotate as it depends on what the author means are "nasty as hell": either the referred people or their opinions. In the case of the latter one, this sample should most likely be annotated as "Offensive" while the former one is perhaps recognized as "Hateful". Instance 6 was found four times in the original dataset with different authors, and the text resembles a news headline. The four instances were annotated two times as "Hateful" and two times as "Offensive" by the human coders. The model predicted the correct label on one of the instances but chose the opposite label for the other three instances. As stated by

		Predicted		
		Normal	**Offensive**	**Hateful**
Actual class	**Normal**	740	79	14
	Offensive	76	3680	82
	Hateful	16	166	104

Table 8.1: Confusion matrix for the BERT Large model performing on dataset **D**.

		Predicted			
		Normal	**Offensive**	**Hateful**	**Spam**
Actual class	**Normal**	7413	223	56	665
	Offensive	150	2612	68	11
	Hateful	224	198	164	2
	Spam	937	46	4	887

Table 8.2: Confusion matrix for the BERT Base model performing on dataset **F**.

		Predicted		
		Normal	**Offensive**	**Hateful**
Actual class	**Normal**	8087	212	58
	Offensive	154	2614	73
	Hateful	216	196	176

Table 8.3: Confusion matrix for the BERT Base model performing on dataset **F***.

Chatzakou et al. (2017), annotation is even hard for humans and instance 6 is a good example that the gold standard is not perfect even though the dataset from Founta et al. (2018b) is thoroughly constructed. Although cases where the model predicts an instance as more hateful than the annotators are rare, instances 7 and 8 show two examples of these cases. Instance 8 is correctly annotated as "Normal" and the model is probably confused by the phrase "hate it". Instance 7 is a perfect example of how thin the line between offensive and hateful expressions is and thus, how hard it is to separate them correctly. The model tends to predict instances containing clear racist or homophobic slurs as hate speech, although this is not always the case. However, obvious hate speech, shown in instances 9 and 10, appears more straightforward for the model to understand and accurately predict.

8.1.4 Comparison to State-of-the-art

Even though the experiments in this thesis were developed mainly to explore the effect of transferring knowledge from language models, it can be beneficial to compare the obtained results to previous solutions in the field of hate speech detection. As presented in Chapter 4, several studies have been performed on the dataset from Davidson et al. (2017). The dataset from Founta et al. (2018b) was recently released, and as a result, there exist far fewer results on this dataset. Luckily, Lee et al. (2018) created a baseline for this dataset presenting several detailed results from a chunk of machine learning algorithms' performance. In addition to this baseline, the results from the preliminary

#	Text	Annotated as	Predicted as
1	*the guy in the White House had an entire election campaign selling fear. But I digress...*	Hateful	Normal
2	*It's all going according to the Putin/Trump plan, now they'll never believe we colluded!*	Hateful	Normal
3	*This is a beast and way more than I was expecting. Looks over engineered and exactly wh. . .*	Hateful	Normal
4	*Oh god, labmate just pointed out that one of the algorithms we use is called UPARSE. UP ARSE. Fucking nerds, man.*	Hateful	Offensive
5	*THIS yes what IS wrong with u people omg nasty as hell*	Hateful	Offensive
6	*ISIS message calls Trump 'foolish idiot'*	Hateful, Offensive	Hateful, Offensive
7	*Stop claiming these Bitches you n*ggas look stupid*	Offensive	Hateful
8	*I feel like the Grand Canyon April Fooled tf out of us today and I am not amused Never go to the west rim hate it*	Normal	Hateful
9	*Ayala is a traitor to the rule of law fire her*	Hateful	Hateful
10	*tired of u feminist bitches bc this is just disgusting*	Hateful	Hateful

Table 8.4: 10 sampled examples predicted by BERT Base on dataset **F***.

System	**Prec.**	**Rec.**	**F1**
BERT Large	**0.91**	**0.91**	0.90
Davidson et al. (2017)	**0.91**	0.90	0.90
Founta et al. (2018a)	0.89	0.89	0.89
Kshirsagar et al. (2018)	–	–	**0.92**

Table 8.5: Comparison of systems performing on dataset **D**. All scores are weighted average as confirmed by Kshirsagar et al. (2018).

study presented in Chapter 5 will be compared against experimental results from the best performing model on each of the three datasets.

Table 8.5 shows the results obtained on dataset **D** by BERT Large compared to results from other previous solutions. Although dataset **D** is a popular used dataset in the field, some researchers, like Zhang et al. (2018), chose to merge the "Offensive" and "Normal" classes into one non-hate class and thus, the results from these papers are not considered to be comparable to the results carried out in the experiments. All four systems in Table 8.5 perform equally well with F_1-scores around 0.90. BERT Large is outperformed by Kshirsagar et al. (2018)'s Transformed Word Embedding Model (TWEM) which is discussed in Section 4.3. BERT Large outperforms the solution from Founta et al. (2018a) and obtains similar results as the baseline from Davidson et al. (2017).

Lee et al. (2018) tested several machine learning algorithms on dataset **F** intending to create a baseline for this dataset. Table 8.6 shows two selected models from this baseline together with the two extended BERT models, BERT Base* and BERT Large*. We observe from the table that the two BERT models and Lee et al.'s word-based RNN-LTC model performs similarly on this dataset. However, BERT Base* achieves an F_1-score of 0.361 on the "Hateful" class, which is an increase compared to the RNN-LTC model's F_1-score of 0.302. This indicates that BERT Base* is undoubtedly better at separating

System	**Prec.**	**Rec.**	**F1**
BERT Base*	0.800	0.812	**0.806**
BERT Large*	0.802	0.809	0.805
Lee et al. (2018) (CNN word)	0.789	0.808	0.783
Lee et al. (2018) (RNN-LTC word)	**0.804**	**0.815**	0.805

Table 8.6: Comparison of systems performing on dataset **F**. All scores are weighted average. The last two F_1-scores are reported wrong by Lee et al. according to Equation 2.8 on page 23. The correct score for these entries should be 0.798 and 0.809, respectively.

System	Prec.	Rec.	F1
BERT Base	0.80	**0.73**	**0.76**
Naïve Bayes	0.63	0.63	0.63
Support Vector Machine	**0.87**	0.65	0.74
Logistic Regression	0.80	0.69	0.74

Table 8.7: Comparison of systems performing on dataset **F***. The three traditional machine learning systems are from the preliminary study. All scores are macro-averaged.

hateful language from the other types of language compared to other solutions. The RNN-LTC model outperformed BERT Base* on the “Spam” class resulting in the similar total weighted average scores.

Dataset **F*** is without the “Spam” class, and the experimental results on this dataset are compared to the three baseline systems tested in the preliminary study. This is because no other research was found during the literature review. The macro-averaged scores from all systems are shown in Table 8.7. Out of the four tested models, BERT Base was the best performing model with an F_1-score of 0.76. This score is high enough to outperform the three other tested models comfortably. Again, BERT Base’s performance on the “Hateful” class is compellingly better than the best performing Logistic Regression model from the preliminary study. BERT Base obtain an F_1-score of 0.393 while the LR model achieves an F_1-score of 0.310. The improved performance on the “Hateful” class on both datasets **F** and **F*** implies that models transferring knowledge from pre-trained language models are able to distinguish the nuances of abusive language more accurately.

8.2 Discussion

Section 1.2 formulated an overall research goal, along with more specific research questions aimed at reaching this goal. The goal of this thesis was to investigate how to accurately distinguish between hateful, offensive and normal language. Three different research questions were formulated as sub-goals, and these will be addressed in this section.

Research question 1 *Which features and representations of text, as well as models, are effective when detecting hateful, offensive, and normal language?*

There have been numerous attempts at creating predictors that can detect hate speech and other types of abusive speech accurately. The various approaches use task-specific preprocessing techniques, input features, models, etc., and therefore hard to generalize which features and models that work best for every task. However, studying the different approaches is important to gain insights into the possible techniques and models that can be utilized. Previous studies on abusive language detection were found during the literature review and are summarized in Section 4.3. The set of features and text representations examined in previous solutions varies while supervised learning is the preferred classification method. In the early studies on hate speech detection, most features were constructed by looking at simple surface features in the language such as bag-of-words and TF-IDF, which are described in Section 2.2.2. Burnap and Williams (2015) and Waseem and Hovy (2016) used a set of features containing bag-of-words combined with unigrams or n-grams, and these features are found highly predictive on their own. However, Nobata et al. (2016) reported that models combining both character and token-level n-grams achieved the highest performance in their experiments. This observation agrees with Meyer and Gambäck (2019) who used character n-grams and word embeddings as input to their dual-stranded CNN-LSTM system and achieved great results. Gaydhani et al. (2018) achieved remarkable results on separating hateful and offensive language by combining n-grams with TF-IDF. The models implemented in the preliminary study also use TF-IDF to form numerical vectors, although the models from Gaydhani et al. greatly outperform the models in the preliminary study. However, the results from Gaydhani et al. have shown challenging to reproduce as discussed in Section 4.3.

Although surface features have shown great results, syntactic features and word order is ignored with these kinds of features. In addition, they are highly dependent on predictive words in the documents to work well. This is not always the case with hate speech as hateful expressions can be uttered without offensive slurs. Several recent works use word generalization as a technique to counter this problem. Distributed representations of words or word embeddings, presented in Section 4.1.1, is such a technique commonly used by researchers. The first popular embeddings technique in NLP, word2vec, was proposed by Mikolov et al. (2013b). Gambäck and Sikdar (2017) used word embeddings from word2vec to capture semantic information and found that the CNN model using these word embeddings outperformed the other three models tested including a model using character n-grams. This is the opposite of Badjatiya et al. (2017) who used word embeddings from GloVe, but found that an LSTM model using randomly generated word vectors outperformed the models using transferred knowledge from GloVe. However, Pavlopoulos et al. (2017a) and Pavlopoulos et al. (2017b) also used word embeddings from Glove and word2vec and found their RNN based model to work well for comment moderation.

Word embeddings are capable of capturing many different properties of a word in a document such as a word's context. However, these word embeddings are static in the way that the context of a word is always the same regardless of the overall context where it occurs. Contextualized word embeddings address this problem of polysemy by capturing a word's semantics in different contexts. Language models are trained on predicting the next word given a sequence of words and the LSTM neural network architecture is commonly used to learn this ability. The ELMo language models created by Peters et al. (2018) use the hidden states of these trained LSTM models to generate deep contextualized word representations for each token. ELMo and other language models are further discussed in Section 4.1.2. These techniques for generating numerical vectors are recently discovered, and few existing solutions using them are proposed. Pérez and Luque (2019) and Indurthi et al. (2019) used ELMo embeddings to obtain some of the best results on the hate speech dataset from Task 5 sub-task A in the recent SemEval-2019 competition. Similarly, for the offensive detection Task 6 sub-task A, among the top-10 ten teams, seven of them chose to use BERT as their language model. Liu et al. (2019) was the top-performing team and used BERT Base with default parameters. This is in line with BERT Base used in the experiments where the model achieves some of the best results on each dataset.

Regarding model selection, early research in NLP used traditional machine learning algorithms as the ones presented in Section 2.3. Waseem and Hovy (2016) found a logistic regression model to be most effective when performing on their own dataset. An LR model was also the best performing model in the experiments carried out in the preliminary study. Today, these classical classification algorithms are mostly used as baselines and compared against deep neural network based algorithms. Lee et al. (2018) tested a lot of machine learning algorithms and found that the models based on deep learning outperformed the traditional algorithms such as SVM and NB. Other researchers chose only to implement the neural network based models as they are believed to improve the overall performance. For instance, Gambäck and Sikdar (2017) trained four CNN based models and achieved results above the baseline from Waseem and Hovy. Pitsilis et al. (2018) utilized an ensemble RNN based model without the use of word embeddings and outperformed the state-of-the-art on the dataset from Waseem and Hovy. Model selection is important when creating a hate speech predictor; however, Gröndahl et al. (2018) argue that model architecture is less important than the type of data and labeling criteria. They found that the tested models, which ranged from simple Logistic Regression to more complex LSTM, performed equally well when recreating several state-of-the-art solutions. This is not in line with the results from the preliminary study, which show that the LR model outperforms both the SVM and NB models. However, Gröndahl et al.'s results are consistent with the investigations conducted during the experiments in Chapter 7, where changes in the final classifier's complexity did not reflect any changes in the results.

Although neural networks require more training data than, for example, Naïve Bayes, several existing hate speech datasets contain a sufficient amount of samples to be effectively used for deep learning. Howard and Ruder (2018)'s ULMFiT language model also shows that with pre-trained language models, great results can be achieved even with small datasets. The task of hate speech detection lacks a benchmark dataset and Schmidt and Wiegand (2017) argue for a benchmark dataset representing all nuances of hate speech. Although the dataset from Founta et al. (2018b) contains nearly 100k samples, Table 8.4 showed that some classes might contain overlaps, duplicates and mislabelled samples.

Research question 2 *How well does a deep learning model based on a large pre-trained language model distinguish between hateful, offensive and normal language?*

The architecture and experiments presented in Chapter 6 and Chapter 7, respectively, were designed to answer this research question. More specifically, the implemented models BERT Base and Large were tested on two different datasets containing the three classes "Hateful", "Offensive" and "Normal". Although the four systems achieve F_1-scores close to existing solutions, the performance on the "Hateful" class is undoubtedly better for the two pre-trained models. This indicates that the language models' general language understanding obtained during pre-training is capable of recognizing more hateful examples than neural network based models. However, the models' recall on the "Hateful" class is poor and the systems confuse true hate speech with both offensive and normal instances. This agrees with the observations made by Malmasi and Zampieri (2018) and Davidson et al. (2017). Thus, using this system in production would mean that far too many hate speech utterances would slip through the system. Even though the model rarely classifies offensive or normal instances as hateful, it is important to preserve the freedom of speech and censoring a normal instance wrongly as hateful would not be acceptable. With that said, the models are able to correctly classify the offensive and normal samples on all three datasets with high accuracy. With similar accuracy on the "Hateful" class, these systems may be used for semi-automatic moderation where inappropriate content would need to be reviewed manually by a human.

Research question 3 *What are the effects of further pre-training a language model with hate speech corpora?*

Two other models, BERT Base* and Large* were implemented to help answer this research question. These models were further pre-trained from BERT's checkpoint with domain-specific corpora to obtain more domain-specific language understanding and thus improve the models' overall performance. Although the models achieved results above the original language models on some metrics, the overall performance of all four systems was similar without any remarkable improvements. As mentioned above, the generation process of the domain-specific corpora had issues with single-sentence documents, which resulted in documents containing a single-sentence split in the middle. As most of the existing datasets in the hate speech domain include tweets, this strategy had to be applied to a large portion of the samples; however, the final input mostly contained documents with several sentences. Recently, an update to the repository was released with code aimed at generating the pre-training data with an optimal strategy. This strategy may result in a more consistent and optimal generation of an input corpus that may allow for the model to gain better domain-specific language understanding and thus improve the overall performance. Also, collecting larger documents containing hateful and offensive utterances together with neutral language, such as the dataset from de Gibert et al. (2018), is preferable as opposed to only using tweets as pre-training data. Although not tested during the experiments, it may also be beneficial to pre-train BERT from scratch with only domain-specific documents.

9 Conclusion and Future Work

This chapter will provide a conclusion to the work carried out in this thesis, as well as discuss how this thesis contributes to the field of hate speech detection. The final section provides ideas for further exploration of transfer learning with language models and hate speech detection in general.

9.1 Conclusion

The field of hate speech detection has gained a lot of interest in recent years by not only NLP researchers, but also by some of the biggest social media platforms such as Facebook, Instagram, and Twitter. These platforms allow user-generated content and hence are responsible for handling and removing unintended content within a specific time limit. As these platforms report millions of users who use their service every day, an automatic detection and filtering system is highly necessary as manual moderation is not feasible. The research field continuously explores new ways of detecting abusive and harmful text content in order to assist in the search for practical tools. The work conducted in this thesis aimed at exploring the effects of transferring knowledge from large pre-trained language models to the task of separating hateful, offensive and normal language. The exploratory research consisted of a literature review, a preliminary study, and experiments.

An extensive literature review of related studies was performed and presented in Chapter 4. This literature review intended to gain insights into existing solutions in the field of hate speech detection. These insights focused on which features, text representations, and classification methods found useful when creating hate speech predictors. Word embeddings from word2vec or GloVe and n-grams were some of the more popular set of features used in state-of-the-art solutions. Regarding the choice of the classifier model, some studies implemented simple classical machine learning algorithms such as SVM, NB and LR while others chose to implement neural network based models and even some hybrid systems. However, Gröndahl et al. (2018) argue that model architecture is not as important as the labeling criteria and type of data in the datasets. The results from experiments conducted in this thesis agree with this statement to a certain extent, although language models have shown exciting and should be focused on in future research.

In addition to the literature review, Chapter 3 presented some of the most popular hate speech datasets in the field. The choice of the dataset used in the experiments is important as the models can only learn from the samples within a dataset. Due to the lack of a shared definition of hate speech, different datasets are constructed with different labeling criteria and classes. This makes comparisons of results across datasets not suitable. Therefore, the field should strive to create a benchmark dataset containing the nuances of abusive language. As Table 8.4 shows, annotating a dataset is a difficult task. By utilizing pre-trained language models, a benchmark dataset does not need to contain as many as possible samples, but instead contain correctly labeled samples.

To further explore the effects of applying language models to the downstream task of hate speech detection, four systems based on the recently release BERT language models were implemented. All four systems were trained on the English Wikipedia and BookCorpus to obtain general language understanding. In addition, two of the systems were further pre-trained with unlabelled domain-specific data to investigate the effect of this strategy. However, the results did not reflect any notable improvement with the extended language models. As discussed in the previous chapter, a flaw in the pre-training process was that a large portion of the data contained single-sentence documents, which is a problem as BERT is trained on a next sentence prediction task. All four models achieved F_1-scores close to or above state-of-the-art solutions on both datasets and their ability to correctly distinguish hate speech from offensive and ordinary language was considerably better than the compared solutions. However, the scores on the “Hateful” class is not sufficient enough to bring the systems into practical use cases as hateful expressions would pass through the system or more benign cases would be incorrectly censored. Language models bring a considerable potential to understanding all the nuances of hateful utterances, and further exploration of how to most effectively train and transfer knowledge from them is necessary.

9.2 Contributions

This thesis contributes to the growing field of hate speech detection intending to encourage further exploration of the massive potential of transfer learning in NLP, primarily through language models. A thorough literature review mainly focusing on previous solutions utilizing more or less complex transfer learning is presented in Chapter 4. This thesis also provides an architecture overview and implementations of one of the latest language models, namely BERT. In addition, investigation of the effect of further training BERT with domain-specific corpus was carried out. The implemented systems achieved similar overall results as previous state-of-the-art solutions; however, the ability to recognize and correctly predict hateful instances was noticeably better, but unfortunately not to such an extent that the problem of separating hateful and offensive language is solved. More research in the field is necessary to find effective tools able to separate the nuances of the natural language correctly.

9.3 Future Work

In any research, there is always room for improvement, and during the work conducted in this thesis, several new ideas came up to further improve the work. This section presents some suggestions inspired by these ideas for others to further research and explore the use of language models for transfer learning. In addition to the concrete suggestions below, general explorations with language models are encouraged in order to reveal their full potential. Although BERT was built upon recent work like ELMo and ULMFiT, BERT is not the only language model available. Several others are presented in Section 4.1.2 and a collection of some of them is made available as Pytorch models in an excellent open-source repository[1].

9.3.1 Multilingual BERT

Hateful and abusive expressions are not only existent in the English language, although most of the datasets and pre-trained language models are in English. Datasets in several different languages are available for researchers, for instance, a German dataset from Ross et al. (2016) and a Greek dataset from Pavlopoulos et al. (2017a). BERT's official repository[2] is continuously updated with new releases of pre-trained language models, and the repository's contributors recently made a multilingual language model available. This model is pre-trained with 104 different languages and can be used to create a language-independent hate speech predictor in the same way as the implemented language models discussed in Chapter 6.

[1] https://github.com/huggingface/pytorch-pretrained-BERT

[2] https://github.com/google-research/bert

9.3.2 Pre-train BERT from Scratch

The four models used in the experiments were all pre-trained on the English Wikipedia and BookCorpus to obtain general language understanding. Typically, the language that appears in Wikipedia articles and books are somewhat domain neutral and contain formal language, which is why these two corpora are selected as input data. Although the intention of using these neutral corpora is good, the language may be too different from the hate speech domain in terms of words and sentences. Therefore, it may be beneficial to collect documents from hate speech datasets and create one large corpus using the updated method for generating input data for BERT. This corpus can then be used as input data to pre-train BERT's encoders from scratch, which may be beneficial and improve the overall performance of the models. However, be prepared that pre-training is reasonably computationally expensive. Also, a new WordPiece vocabulary may be necessary to train as well.

9.3.3 Hyperparameter Optimization

Hyperparameters are values that are chosen and set before the learning process. As the research in this thesis consisted of exploring the effects of language models applied to the task of hate speech detection, extensive hyperparameter tuning was not carried through. Instead, the recommended number of epochs, values for learning rate, sequence length, etc. were tested manually to find the final values used in the experiments. If the highest possible scores are desirable, it may be beneficial to run a more exhaustive grid-search to find the optimal parameters that maximize the models' performance. In addition, cross-validation during the training process is favorable to reduce the probability of overfitting and create models that generalize well.

Bibliography

Badjatiya, P., Gupta, S., Gupta, M. & Varma, V. (2017). Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion* (pp. 759–760). Perth, Australia: International World Wide Web Conferences Steering Committee.

Bahdanau, D., Cho, K. & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473.*

Bird, S., Klein, E. & Loper, E. (2009). *Natural language processing with Python.* O'Reilly Media.

Burnap, P. & Williams, M. L. (2015). Cyber hate speech on Twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy and Internet*, *7*(2), 223–242.

Castelle, M. (2018). The linguistic ideologies of deep abusive language classification. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)* (pp. 160–170). Brussels, Belgium: Association for Computational Linguistics.

Chatzakou, D., Kourtellis, N., Blackburn, J., De Cristofaro, E., Stringhini, G. & Vakali, A. (2017). Mean birds: Detecting aggression and bullying on Twitter. In *Proceedings of the 2017 ACM on Web Science Conference* (pp. 13–22).

Dai, A. M. & Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems* (pp. 3079–3087).

Davidson, T., Warmsley, D., Macy, M. & Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM).*

de Gibert, O., Perez, N., Garcia-Pablos, A. & Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)* (pp. 11–20). Brussels, Belgium: Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

ElSherief, M., Nilizadeh, S., Nguyen, D., Vigna, G. & Belding, E. (2018). Peer to peer hate: Hate speech instigators and their targets. In *Twelfth International AAAI Conference on Web and Social Media.* Palo Alto, California, USA.

Facebook. (2018). Facebook company info. Retrieved October 15, 2018, from https://newsroom.fb.com/company-info/

Fortuna, P., Soler-Company, J. & Nunes, S. (2019). Stop PropagHate at SemEval-2019 tasks 5 and 6: Are abusive language classification results reproducible? In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 745–752). Minneapolis, Minnesota, USA: Association for Computational Linguistics.

Founta, A. M., Chatzakou, D., Kourtellis, N., Blackburn, J., Vakali, A. & Leontiadis, I. (2018a). A unified deep learning architecture for abuse detection. *arXiv preprint arXiv:1802.00385.*

Founta, A. M., Djouvas, C., Chatzakou, D., Leontiadis, I., Blackburn, J., Stringhini, G., Vakali, A., Sirivianos, M. & Kourtellis, N. (2018b). Large scale crowdsourcing and characterization of twitter abusive behavior. In *Twelfth International AAAI Conference on Web and Social Media.* Palo Alto, California, USA.

Gambäck, B. & Sikdar, U. K. (2017). Using convolutional neural networks to classify hate-speech. In *Proceedings of the First Workshop on Abusive Language Online* (pp. 85–90). Vancouver, BC, Canada: Association for Computational Linguistics.

Gao, L. & Huang, R. (2017). Detecting online hate speech using context aware models. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017* (pp. 260–266). Varna, Bulgaria: INCOMA Ltd.

Gaydhani, A., Doma, V., Kendre, S. & Bhagwat, L. (2018). Detecting hate speech and offensive language on Twitter using machine learning: An N-gram and TFIDF based approach. *ArXiv e-prints.* arXiv: 1809.08651

Golbeck, J., Ashktorab, Z., Banjo, R. O., Berlinger, A., Bhagwan, S., Buntain, C., Cheakalos, P., Geller, A. A., Gergory, Q., Gnanasekaran, R. K., Gunasekaran, R. R., Hoffman, K. M., Hottle, J., Jienjitlert, V., Khare, S., Lau, R., Martindale, M. J., Naik, S., Nixon, H. L., Ramachandran, P., Rogers, K. M., Rogers, L., Sarin, M. S., Shahane, G., Thanki, J., Vengataraman, P., Wan, Z. & Wu, D. M. (2017). A large labeled corpus for online harassment research. In *Proceedings of the 2017 ACM on Web Science Conference* (pp. 229–233). WebSci '17. Troy, New York, USA: ACM.

Gröndahl, T., Pajola, L., Juuti, M., Conti, M. & Asokan, N. (2018). All you need is “love”: Evading hate speech detection. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security* (pp. 2–12). AISec '18. Toronto, Canada: ACM.

Havrlant, L. & Kreinovich, V. (2017). A simple probabilistic explanation of term frequency-inverse document frequency (TF-IDF) heuristic (and variations motivated by this explanation). *International Journal of General Systems*, *46*(1), 27–36.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Howard, J. & Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146.*

Indurthi, V., Syed, B., Shrivastava, M., Chakravartula, N., Gupta, M. & Varma, V. (2019). Fermi at SemEval-2019 task 5: Using sentence embeddings to identify hate speech against immigrants and women on Twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval-2019)* (pp. 70–74). Minneapolis, Minnesota, USA: Association for Computational Linguistics.

Kshirsagar, R., Cukuvac, T., McKeown, K. & McGregor, S. (2018). Predictive embeddings for hate speech detection on Twitter. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)* (pp. 26–32). Brussels, Belgium: Association for Computational Linguistics.

LeCun, Y. (1989). Generalization and network design strategies. In *Connectionism in Perspective* (Vol. 19). Elsevier, Zürich, Switzerland: Citeseer.

Lee, Y., Yoon, S. & Jung, K. (2018). Comparative studies of detecting abusive language on twitter. In *Proceedings of the 2nd workshop on abusive language online (ALW2)* (pp. 101–106). Brussels, Belgium: Association for Computational Linguistics.

Liu, P., Li, W. & Zou, L. (2019). NULI at SemEval-2019 task 6: Transfer learning for offensive language detection using bidirectional transformers. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 87–91). Minneapolis, Minnesota, USA: Association for Computational Linguistics.

Luong, T., Pham, H. & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412–1421). Lisbon, Portugal: Association for Computational Linguistics.

Malmasi, S. & Zampieri, M. (2018). Challenges in discriminating profanity from hate speech. *Journal of Experimental & Theoretical Artificial Intelligence*, *30*(2), 187–202.

McCann, B., Bradbury, J., Xiong, C. & Socher, R. (2017). Learned in translation: Contextualized word vectors. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 6294–6305). Curran Associates, Inc.

Meyer, J. S. & Gambäck, B. (2019). A platform agnostic dual-strand hate speech detector. In *3rd Workshop on Abusive Language Online (ALW3) (forthcoming)*. Florence, Italy: Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26* (pp. 3111–3119). Curran Associates, Inc.

Mishra, P., Del Tredici, M., Yannakoudakis, H. & Shutova, E. (2018). Author profiling for abuse detection. In *Proceedings of the 27th International Conference on Computational Linguistics* (pp. 1088–1098). Santa Fe, New Mexico, USA: Association for Computational Linguistics.

Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y. & Chang, Y. (2016). Abusive language detection in online user content. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 145–153). WWW '16. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee.

Paice, C. D. (1990). Another stemmer. *SIGIR Forum*, *23*(3), 56–61.

Pan, S. J. & Yang, Q. (2010). A survey on transfer learning. (Vol. 22, *10*, pp. 1345–1359). Piscataway, NJ, USA: IEEE Educational Activities Department.

Park, J. H. & Fung, P. (2017). One-step and two-step classification for abusive language detection on twitter. In *Proceedings of the First Workshop on Abusive Language Online* (pp. 41–45). Vancouver, BC, Canada: Association for Computational Linguistics.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. & Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS-W*.

Pavlopoulos, J., Malakasiotis, P. & Androutsopoulos, I. (2017a). Deep learning for user comment moderation. In *Proceedings of the First Workshop on Abusive Language Online* (pp. 25–35). Vancouver, BC, Canada: Association for Computational Linguistics.

Pavlopoulos, J., Malakasiotis, P. & Androutsopoulos, I. (2017b). Deeper attention to abusive user content moderation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 1125–1135). Copenhagen, Denmark: Association for Computational Linguistics.

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

Pennington, J., Socher, R. & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Doha, Qatar: Association for Computational Linguistics.

Pérez, J. M. & Luque, F. M. (2019). Atalaya at SemEval 2019 task 5: Robust embeddings for tweet classification. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 64–69). Minneapolis, Minnesota, USA: Association for Computational Linguistics.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. & Zettlemoyer, L. (2018). 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)* (pp. 2227–2237). New Orleans, Louisiana: Association for Computational Linguistics.

Pitsilis, G. K., Ramampiaro, H. & Langseth, H. (2018). Detecting offensive language in tweets using deep learning. *arXiv preprint arXiv:1801.04433.*

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–137.

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018a). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf.*

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2018b). *Language models are unsupervised multitask learners.* Technical report, OpenAi.

Ross, B., Rist, M., Carbonell, G., Cabrera, B., Kurowsky, N. & Wojatzki, M. (2016). Measuring the reliability of hate speech annotations: The case of the european refugee crisis. In *Proceedings of the Workshop on Natural Language Processing for Computer Mediated Communication (NLP4CMC)* (pp. 6–9). Bochum, Germany.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Schmidt, A. & Wiegand, M. (2017). A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media* (pp. 1–10). Valencia, Spain: Association for Computational Linguistics.

Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*(1), 11–21.

Systrom, K. (2017). Keeping Instagram a safe place for self-expression. Retrieved October 16, 2018, from https://instagram-press.com/blog/2017/06/29/keeping-instagram-a-safe-place-for-self-expression/

Systrom, K. (2018). Protecting our community from bullying comments. Retrieved October 16, 2018, from https://instagram-press.com/blog/2018/05/01/protecting-our-community-from-bullying-comments-2/

Torrey, L. & Shavlik, J. (2010). Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques* (pp. 242–264). IGI Global.

Turian, J., Ratinov, L.-A. & Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 384–394). Uppsala, Sweden: Association for Computational Linguistics.

Twitter. (2014). The 2014 #YearOnTwitter. Retrieved October 15, 2018, from https://blog.twitter.com/official/en_us/a/2014/the-2014-yearontwitter.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30* (pp. 5998–6008). Long Beach, CA, USA.

Waseem, Z. (2016). Are you a racist or am I seeing things? Annotator influence on hate speech detection on Twitter. In *Proceedings of the First Workshop on NLP and Computational Social Science* (pp. 138–142). Austin, Texas: Association for Computational Linguistics.

Waseem, Z. & Hovy, D. (2016). Hateful symbols or hateful people? Predictive features for hate speech detection on Twitter. In *Proceedings of the NAACL Student Research Workshop* (pp. 88–93). San Diego, California: Association for Computational Linguistics.

Wulczyn, E., Thain, N. & Dixon, L. (2017). Ex machina: Personal attacks seen at scale. In *Proceedings of the 26th International Conference on World Wide Web* (pp. 1391–1399). WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N. & Kumar, R. (2019). SemEval-2019 task 6: Identifying and categorizing offensive language in social media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 75–86). Minneapolis, Minnesota, USA: Association for Computational Linguistics.

Zhang, Z., Robinson, D. & Tepper, J. (2018). Detecting hate speech on twitter using a convolution-gru based deep neural network. In A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai & M. Alam (Eds.), *The Semantic Web* (pp. 745–760). Cham: Springer International Publishing.

Vebjørn Isaksen

Detecting Hateful and Offensive Language with Transfer Learned Models

NTNU
Norwegian University of
Science and Technology