Master's thesis

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

Mathias Wahl
Sondre Grav Skjåstad

Detecting Hate Speech in Norwegian Texts Using BERT Semi-Supervised Anomaly Detection

Master's thesis in Computer Science
Supervisor: Herindrasana Ramampiaro
June 2021

NTNU
Norwegian University of
Science and Technology

Mathias Wahl
Sondre Grav Skjåstad

Detecting Hate Speech in Norwegian Texts Using BERT Semi-Supervised Anomaly Detection

Master's thesis in Computer Science
Supervisor: Herindrasana Ramampiaro
June 2021

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

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis presents a novel solution to hate speech detection that combines several artificial intelligence methods to identify hateful content within short Norwegian texts. With the explosive growth of the internet and the ever-increasing adoption of social media and discussion forums, user-generated hateful utterances have become progressively more frequent. While freedom of speech is a constitutional right in Norway, discriminatory or hateful utterances are illegal and should therefore be removed.

Previous solutions to hate speech detection, both for English and non-English languages, have primarily used text classification approaches. While impressive results have been achieved using these methods, they face several drawbacks. Among these is the necessity for large, annotated corpora tailored to hate speech detection, which is not available for less used languages such as Norwegian.

This thesis contains a methodical literature review, a novel approach to hate speech detection, and an exhaustive experiment evaluating its viability and performance. The novel approach is called BSSAD and contains a convolutional neural network that uses a pre-trained, bidirectional encoder model to produce word embeddings. By using a bidirectional encoder, the resulting word embeddings are able to reflect context, making better use of the contents of smaller corpora.

The experiment results show that the new approach significantly outperforms previous solutions for hate speech detection in Norwegian. This indicates that introducing pre-trained BERT models yields more valuable word embeddings from which the BSSAD model is able to benefit.

Automatic hate speech detection in its infancy faces many challenges. While there are numerous advanced approaches for the English language, applying those to less-used languages like Norwegian yields inferior results due to a lack of specialized corpora. The BSSAD approach presented in this thesis combines recently developed state-of-the-art methods such as BERT models and anomaly detection to surpass previous approaches, showing great potential for future development based on this combination.

Sammendrag

Denne avhandlingen presenterer en ny løsning for deteksjon av hatefulle ytringer som kombinerer flere metoder innen kunstig intelligens for å identifisere hatefullt innhold i korte norske tekster automatisk. Internettet har historisk sett opplevd en eksplosiv fremvekst, og samtidig som bruken av sosiale medier og diskusjonsforum stadig vokser, vokser også tilfellene av brukergenerert hatefullt språk. Selv om ytringsfrihet er stadfestet i Norges Grunnlov, er det ulovlig å sette fram diskriminerende eller hatefulle ytringer, og slike ytringer bør derfor oppdages og fjernes.

Tidligere løsninger for deteksjon av hatefulle ytringer, både for Engelsk og andre språk, er stort sett basert på klassifiseringsmetoder. Selv om imponerende resultater er blitt oppnådd med disse metodene, medfører de et antall ulemper. En av disse er at de krever store, annoterte tekstkorpora som er tilpasset deteksjon av hatefulle ytringer, som ofte ikke er tilgjengelig for mindre brukte språk som Norsk.

Denne avhandlingen inneholder en metodisk gjennomgang av tidligere studier, en ny tilnærming til deteksjon av hatefulle ytringer, og et omfattende eksperiment som evaluererer hvorvidt løsningen er hensiktsmessing og dens ytelse. Den nye tilnærmingen, kalt BSSAD, inneholder et konvolusjonelt nevralt nettverk som drar nytte av en ferdig opplært, bidireksjonell enkoder modell for å produsere ordvektorer. Ved å bruke en bidireksjonell enkoder vil de resulterende ordvektorene kunne reflektere kontekst, og dermed dra mer nytte av innholdet i mindre tekstkorpora.

Resultatene av eksperimentet viser at den nye tilnærmingen overgår tidligere løsninger for deteksjon av hatefulle ytringer på norsk. Dette indikerer at bruken av opplærte BERT modeller kan produsere mer verdifulle ordvektorer som BSSAD modellen kan utnytte.

I tidilige stadier av utviklingen er det mange utfordringer knyttet til automatisk deteksjon av hatefulle ytringer. Det finnes et utvalg avanserte tilnærminger på Engelsk, men disse yter ikke like godt for mindre brukte språk som Norsk grunnet mangel på spesialiserte korpora. Metoden som er presentert i denne avhandlingen forbigår tidligere løsninger ved å kombinere BERT modeller og avviksdeteksjon, og viser dermed at det ligger stort potensiale i denne kombinasjonen for videre utvikling i feltet.

Preface

This thesis was written as part of the Master of Science in Computer Science at the Norwegian University of Science and Technology (NTNU) in the Spring semester of 2021. We would like to thank our supervisor Heri Ramampiaro for guidance and help with any questions we had during all stages of the thesis. We would also like to thank Vilde Arntzen for valuable discussions on the topic of hate speech detection in Norwegian.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Hate speech is becoming a growing concern with the continuously increasing prevalence of social media. While freedom of speech is a concept that should be guarded as much as possible, potential victims of hateful utterances should also be protected. The internet has, over the years, become an arena where some find it easy to hide behind the anonymity it provides while sharing hateful comments directed at these victims. Being able to detect such hateful comments is the first step of filtering them out and addressing them appropriately. Given the amount of continuously generated information on the internet, this is an enormous task that is not practical nor feasible to perform manually. Thus, the need for automatic detection of hate speech becomes evident.

Recent progress has been made in the area of hate speech detection using machine learning, a subgroup of artificial intelligence, with methods that have proven helpful in the semantic interpretation and classification of texts. This is especially useful within the area of hate speech detection, where sentiment plays an important role. This paper focuses on researching state-of-the-art applications of machine learning in attempts to solve the hate speech detection problem.

The following chapter starts by presenting the motivation behind this thesis. The following section addresses the problem specification, including the goal, scope, and research questions. Next, the contributions of the thesis are listed before an overview of the thesis structure is provided.

1.1 Motivation

The problem of hate speech detection is at its forefront when considering the English language. Although some approaches are language agnostic, research in the field when using non-English languages is comparatively undeveloped. However, the issue of hate speech is just as crucial for these languages. Unfortunately, the barrier of entry for research in non-English languages is higher because of the insufficient amount of available evaluation resources such as pre-trained word embeddings and annotated corpora. As a result, researchers attempting hate speech detection in non-English languages often have to begin by constructing a new, suitable corpus. Creating a decent-sized corpus with accurate annotations is time-consuming and may be challenging to justify. Moreover, the resulting corpora are often inferior in quality compared to their English counterparts. Therefore, approaches that are either language-agnostic or that perform well on smaller corpora will benefit the field.

A critical problem faced within hate speech detection is complexity. This problem is rooted in the fact that language is unstructured and complex. In order to make use of modern machines' computing capabilities, the data must be translated to a format that makes sense to a computer. One such format is word embeddings, which are numerical representations of each word in a text. Several different word embeddings are available with different sizes and techniques. With such a complex data source as language, the nuances between different word embeddings can significantly affect the works that rely on them. Moreover, the same word can convey different meanings when used in different sentences, highlighting the importance of context. This is especially relevant in hate speech detection, as hateful content has been known to appear concealed by, for instance, using metaphors or avoiding obscene language. Recently, solutions have been developed that can reflect this context in the produced word embeddings. Language models like BERT can consider the meaning of a word in the context of the surrounding text, offering additional value to the resulting word embeddings compared to traditional, static word embedding techniques.

Numerous approaches have been taken to detect hate speech. One such approach recently presented by Jensen (2020) is ADAHS, which was the first of its kind to be implemented and tested on both English and Norwegian corpora. Where previous methods usually employ multiclass text classification techniques, ADAHS uses anomaly detection to detect hateful content. This approach defines only one class, namely the normal class, and focuses on detecting outliers in the corpus, representing hateful utterances. Defining classes assumes similarity between the entries within. This assumption is unfortunate when applied to hate speech due to constant change in abusive language to avoid attention, as stated by Nobata et al. (2016). Thus, anomaly detection is advantageous as it assumes no similarity between the anomalous data points. The purpose of presenting ADAHS was not to provide an optimized solution but rather to present a novel way of detecting hate speech by rephrasing the problem to use anomaly detection. Its promising results are therefore a motivating factor for using recently developed methods that are not

yet thoroughly explored.

Many current approaches are in their infancy and leave topics such as optimizations for future work. Therefore, examining factors for optimization is a promising path of progression. There are many factors to consider when working with machine learning, such as features, types of models, hyperparameters, and corpora. Further investigating these factors and how they affect performance may lead to valuable discoveries.

1.2 Problem Specification

This section presents the overall goal and the following Research Questions (RQ) for the thesis. The scope describing the focus of the thesis is also included.

Goal

The thesis aims to look at existing hate speech detection approaches in languages other than English and improve upon them by implementing new state-of-the-art methods. Corpora as sources for datasets are not as abundantly available for non-English languages, and the datasets that do exist are not as extensive. Therefore, it is necessary to develop a solution that will perform well on smaller datasets. To improve on existing solutions, the factors that affect them need to be understood to then be able to configure the solutions optimally.

Scope

The scope of this thesis includes conducting a literature review, developing a novel solution and an experiment with accompanying discussions. The literature review regards the general field of hate speech detection and a more specific review of Norwegian hate speech detection. The novel machine learning method for hate speech detection is developed by combining previous methods and applying it on a Norwegian dataset. The experiment is created and executed using an experiment plan with a pre-defined set of configurations. Finally, the results are evaluated and discussed, and compared to previous implementations.

Research Questions

The following research question is presented to solve the overall goal explained above:

> **Research Question**
> How can existing approaches for hate speech detection in languages other than English be improved?

Hate speech detection is at its forefront using the English language. However, hate speech detection is equally relevant for other languages. Solving the problem can be done by either implementing language-specific solutions for non-English languages or language-agnostic solutions to cover a multitude of languages. This thesis aims to solve the problem by researching related works, implementing a solution, running experiments on the solution, and comparing it to previous solutions. The research question is divided into three parts presented below.

RQ1 *How can recently developed techniques in the field, such as BERT, be integrated to provide state-of-the-art results?*

The field of hate speech detection in concurrence with Natural Language Processing (NLP) is in constant development. Providing state-of-the-art results requires using state-of-the-art techniques such as BERT. Current solutions for non-English languages are not at the forefront of the field and utilize less sophisticated techniques, whereas BERT is a new method with advanced features. However, using recently developed techniques comes with the risk of being neither sufficiently documented nor thoroughly tested.

RQ2 *How can the performance be improved for approaches using smaller datasets?*

The absence of appropriately large datasets is a common problem for hate speech detection in less used, non-English languages. Instead of spending resources on expanding the existing datasets or creating new ones, it may instead be beneficial to create solutions that can perform well on smaller datasets. Discovering techniques to improve performance on smaller datasets will benefit hate speech detection for other languages.

RQ3 *How can we determine what factors affect the results?*

Examining the variable factors is essential to improve existing approaches. As time is a resource, not every factor can be considered, and a set of promising factors needs to be selected. Each of these factors can be evaluated individually, and a plan can be made to examine how they affect performance.

1.3 Contributions

Through pursuing the previously stated goal of improving hate speech detection in Norwegian using state-of-the-art techniques, this thesis provides the following contributions to the field:

I *A literature review including surrounding elements, in addition to a specific review of Norwegian hate speech detection.*

II *A novel approach to hate speech detection building on and combining state-of-the-art techniques.*

III *A thorough evaluation of the developed method and comparison of results to previous methods.*

1.4 Thesis Overview

The structure of the thesis is as follows:

Chapter 2 provides theory to introduce concepts related to the field of hate speech detection in addition to theory for specific concepts discussed in the thesis.

Chapter 3 provides an overview of related research and state-of-the-art works.

Chapter 4 presents the proposed solution and describes in detail its inner workings, including an overview of the model architecture.

Chapter 5 presents an experiment plan and setup, in addition to the results of the executed experiment.

Chapter 6 contains an evaluation and discussion of the model and its results.

Chapter 7 discusses the results gathered as well as contributions and possible further work.

CHAPTER 2

Background Theory

This chapter introduces key concepts to lay a foundation for understanding the topics addressed in the following chapters. First, the definition of hate speech and its intricacies will be discussed. Next, the theory appropriate for the thesis will be presented. This theory begins with general concepts before introducing more specific topics directly related to the proposed solution and experiment.

Parts of the following chapter were produced as part of the specialization project preceding this thesis (Wahl & Skjåstad, 2020). While some of the following sections include parts from the other project, most of them have been adapted and built upon to better fit this thesis' specific focus better.

2.1 Definition of Hate Speech

The term "hate speech" is a broad one. To the best of our knowledge, no universal definition has yet been made to describe it. Therefore, in the context of hate speech detection, a pragmatic interpretation of the term is beneficial. Generally, the consensus in previous works seems to be that hate speech can be recognized as any communication that disparages a person or a group based on some characteristic such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other characteristic (Nockleby, 2000). Despite the generally accepted description, the problem of labeling utterances as hateful or non-hateful is deceptively complicated.

Firstly, and importantly, hate speech is often context dependant. A text can be labeled one way when considered on its own and another when considered a part of a bigger picture. For instance, the labeling could be affected by an ongoing discussion, a reference in the text, or the author's background. As an example, the phrase "Go back where you came from" might be innocent enough in the

context of giving someone directions. However, in the context of immigration, it can undoubtedly be perceived as a hateful utterance. When it comes to hate speech detection, the importance of context is a double-edged sword: On the one hand, it makes the problem far more complex, making it challenging to solve automatically. On the other hand, it allows for creative and clever solutions that can use other available data to indicate the probability for a particular text to be hateful. Several variations of such solutions have been implemented and shown to improve results for hate speech detection. Pitsilis et al. (2018) show improvements with a solution that utilizes a user's tendency to have written hateful messages in the past when labeling hateful texts.

Secondly, hate speech is a subjective matter. One annotator might find something offensive that another does not. Moreover, there is the issue of broad definitions: it can be challenging to distinguish between hate speech and offensive language. Including offensive words in a text might make it offensive, but that does not inherently mean it contains hate speech. While some solutions show promising results, they may fall short by having too broad definitions, as stated by Davidson et al. (2017).

Furthermore, hate speech can be rather sophisticated and challenging to detect without human intuition. While certain words or variations of them can be indicative of hate speech, it is no guarantee. Moreover, hate speech can consist of grammatically correct and sophisticated sentences that, on the surface, do not seem alarming. Here, it is again in the context that one must root the classification: who the recipient is, who the author is, at what time the comment was produced, and where it was produced.

To summarize, the problem of detecting hate speech is a complicated one. Without a universal definition, it is hard to set criteria for what constitutes hate speech. Even when someone claims an utterance to be hateful, others might disagree. Whether a given comment is perceived as hateful or not often depends on surrounding factors such as related content and sources and the intended recipient. Thus, classifying content as either hateful or non-hateful is a tedious and complicated process that requires automation. In the following section, the field of machine learning is introduced. In this field, algorithms can be found that show promise concerning the automation of hate speech detection.

2.2 Machine Learning

Machine learning is a broad subject within computer science with an abundance of applications. There exist many variations of methods, each with its strengths and weaknesses. Choosing the correct method is usually done based on previous work. New machine learning-based methods are also discovered through exploration, where different methods are tested on the given application. This section describes the main methods of machine learning that are relevant for automatic hate speech detection.

2.2.1 Machine Learning Supervision

Supervised machine learning uses a labeled dataset as input to a function and maps it to a specified output. The key concept for supervised learning is that a prediction model is created based on a pre-labeled dataset (Goodfellow et al., 2016). Each data point in the dataset has a corresponding label used to train the prediction model. After this process, the trained model can be used by providing new, unlabeled data points as input. The trained model then outputs the predicted label. Supervised machine learning has two types of results, one by classification the other by regression. Classification is used to put data points in one of a collection of classes, whereas regression is used for continuous values, predicting a corresponding value for a given input.

Unsupervised learning is more of an exploratory approach and differs from supervised learning by only using unlabelled datasets. The goal is to detect previously undetected patterns and structures. A common usage of unsupervised learning is clustering, where the data is split into different groups or clusters. Clustering has the benefit of not requiring the previous labeling, which can be demanding work. However, it can be hard to infer meaning from the results.

Semi-supervised learning uses a combination of supervised and unsupervised methods. An example of this may be that not all data points are labeled (Goodfellow et al., 2016). Semi-supervised learning can be effective for incomplete datasets or where some labels are hard to define or assign.

2.2.2 Classification

Classification is the problem where the learning model categorizes the results into different classes. The classes can be binary or multi-class. The classification works best when the classes are balanced in the training dataset. However, lack of such balance within datasets is a common issue for hate speech detection because the natural frequency of hateful utterances is much lower compared to non-hateful ones (Burkal & Veledar, 2018).

Each piece of information that represents a class is known as a feature (Goodfellow et al., 2016). These features make up a representation of the given dataset and are the input of the classification model. For instance, features may include age, name, gender, or any information relating to a person for a dataset of people. Designing a suitable and descriptive set of features, also known as feature engineering, is a central task within machine learning. In the case of hate speech detection, feature engineering is not thoroughly explored. However, works such as Waseem and Hovy (2016) and Nobata et al. (2016) have looked into the incorporation of other features than the text itself with varying results.

2.2.3 Anomaly Detection

Anomaly detection is a method for finding anomalies or outliers in a dataset. It can be utilized for both supervised, unsupervised and semi-supervised learning. Two definitions of data are relevant for anomaly detection, namely normal and anomaly. Normal data is comprised of the most regular data points, which should conform to the majority of a given dataset. Anomalies are rare data points or data in abnormal patterns which can appear in the dataset. Supervised anomaly detection uses datasets where each data point is labeled as either normal or anomaly. Semi-supervised anomaly detection uses a partially labeled dataset or a dataset containing only normal data points. Unsupervised anomaly detection does not have any labels, and the model attempts to identify data points or patterns in the dataset which do not conform to the normal distribution (Chandola et al., 2009).

2.3 Classic Methods for Data Mining in Text

Machine learning is a broad and intricate subject that encompasses a large variety of applications, one of them being text analysis. The foundation for textual machine learning comes from concepts such as text data mining, natural language processing, and information retrieval. It is essential to understand and discuss these more classic concepts to confer and present different solutions to the task of detecting hate speech in text. Text in this context can be considered as unstructured data. Unstructured data is unsuitable for direct computer processing, thus requiring conversion into structured data.

2.3.1 Preprocessing

An information retrieval approach to preprocessing text makes use of five transformations: *lexical analysis*, *stopword elimination*, *stemming*, *keyword selection*, and *thesauri* (Baeza-Yates & Ribeiro-Neto, 2011). *Lexical analysis* entails removing symbols, numbers, and punctuation while also converting the text to lower or upper case. Additionally, it involves tokenizing the text, converting each word in the sentence to some corresponding token. Thus, tokenization represents texts as lists of unique tokens. *Stopword elimination* is the process of removing words that do not contribute to adding meaning to the text. Such words appear in nearly all texts, with common examples including “the”, “a”, and “is”. There are numerous lists of these stopwords available online which can be used. *Stemming* attempts to reduce words to their “base” form by removing conjugation and plurality. *Keyword selection* is used for selecting useful words that have distinct values, thus disregarding less valuable words. *Thesauri* finds similarities between and synonyms for different words, allowing for a deeper understanding of texts by relating them to other ones that are semantically similar. These methods are not straightforward and therefore need to be adapted to the task at hand. This thesis is concerned with

hate speech detection, where the texts in question usually come from internet sites such as social media. Because of this, they often contain spelling errors that have to be taken into consideration. The approach also differs for different languages, such as Norwegian, where words are often compounded.

2.3.2 Feature Extraction

Machine learning methods require text to be represented as structured data. Converting textual data to categorical numerical vectors is known as feature extraction. A feature is one representation of text, a simple example being each term converted into a number. The process of choosing and creating favorable features is known as feature engineering. The following paragraphs will describe some well-known text representations such as TF-IDF, N-grams, and Bag-of-words.

TF-IDF Term Frequency and Inverse Document Frequency are two well-known methods of defining the importance of a term in a document. In this context, a term is a word or a string in a document, a document is a complete text, and each document is part of a collection of documents. Different terms will have different degrees of importance in a document, and how often these terms appear will affect the overall meaning or sentiment. However, using only the frequency of words in a text will not yield accurate results, as the most common words usually do not convey the most meaning. For this reason, term frequency is often normalized. One common method for this is shown in Equation 2.1, where $tf_{i,j}$ is some normalized version of $f_{i,j}$, which is frequency of the term i in the document j. Normalization allows for potentially meaningful terms not to be dwarfed by exceedingly common ones (Baeza-Yates & Ribeiro-Neto, 2011). To further rate a term, it is pertinent to consider it in the context of the collection of documents it belongs to, which IDF attempts to achieve. As opposed to TF, IDF assigns higher weights to rarer terms occurring in fewer documents in the collection. It does this by taking the total number of documents (N) and dividing by the number of documents the term appears in (n_i), then normalizing the quotient logarithmically, as shown in Equation 2.2 where idf_i is the IDF of term i. (Baeza-Yates & Ribeiro-Neto, 2011). TF and IDF are usually combined as TF-IDF which highly weights frequent terms in the document while also being rare in the document collection.

$$tf_{i,j} = \begin{cases} 1 + log f_{i,j}, & \text{if } f_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$idf_i = log \frac{N}{n_i} \tag{2.2}$$

N-grams Words or terms in a text do not always have a distinct and singular sentiment. The context of the surrounding sentence may change the meaning of a word. An example is the word “black”, where followed by the word “people” may imply racial or ethnic sentiment, while followed by the word “out” may imply loss

of consciousness or a power outage. A way of providing this context is the use of n-grams. N-grams are an ordered sequence of characters or words with a length of n (Bengfort et al., 2018). However, providing context to words is not the only use of n-grams. As mentioned, it can also be used for characters. A use case for this is in informal texts where words may have many different variations caused by abbreviations, slang, and spelling errors. In these cases, character n-grams are able to detect the canonical spelling of a word, which could allow understanding to be applied to misspelled words (Schmidt & Wiegand, 2017).

Bag-of-words Bag-of-words is a method for representing text by adding each distinct word to an unordered list, or "bag", together with each word's corresponding frequency in the text. Bag-of-words is a rudimentary way of representing a text that can find similarities between texts and classify texts. Bag-of-n-grams is an alternate version of bag-of-words where a text is represented with some variation of n-grams and their frequency (Bengfort et al., 2018). Using bag-of-n-grams may allow the representation to include some contextual and more nuanced information.

2.4 Natural Language Processing

Natural language processing (NLP) is a branch within Artificial Intelligence concerned with allowing computers to process, understand, and analyze natural language in text or audio. Bird et al. (2009) describe natural language as "a language that is used for everyday communication by humans; languages such as English, Hindi, or Portuguese. In contrast to artificial languages such as programming languages and mathematical notations, natural languages have evolved as they pass from generation to generation and are hard to pin down with explicit rules." NLP is a large field that covers a variety of use cases. However, with regards to hate speech detection, the use case derives meaning from short texts.

Frameworks serving different purposes have been created to solve NLP-related problems. For this thesis, it is pertinent to look at frameworks for representing and analyzing hate speech texts. A popular method of representing texts is using word embeddings. Word embeddings are a way to map high dimensional words of a text into a less complex continuous vector space (Bengfort et al., 2018). Using only a basic mapping such as TF-IDF will be solely based on word similarity, while word embeddings can capture and identify semantic similarity. There are many systems, techniques, or models to apply word embeddings. Some of the most used and popular ones are Word2Vec, fastText, GloVe, and BERT.

Word2Vec is a prediction based method, commonly using a Continuous-Bag-of-Words (CBOW) architecture (Bengfort et al., 2018). This approach assumes similarity by removing a word and checking the probability of another being in the same context. This implies semantic similarity by words like "walking" and "running" being similar as they can replace each other while maintaining the same sentiment.

GloVe (Global Vectors) is an unsupervised method that uses a global corpus to find a word-to-word co-occurrence vector representation (Pennington et al., 2014). Like Word2Vec, it only considers the local context of words.

fastText employs a different approach from GloVe and Word2Vec by using character representations (Athiwaratkun et al., 2018). These representations allow for noncanonical words, also known as Out of Vocabulary (OOV) words, to be included in the word embeddings. As mentioned, hate speech detection deals with informal text, and it can therefore be useful to consider out of vocabulary words.

BERT is a bidirectional language representation technique that can represent words as homonyms, meaning a single word having multiple meanings based on context (Devlin et al., 2018). An example of such a word is "saw", which would result in different representations when used in the sentences "I saw something over there" and "A saw is a cutting tool".

2.5 Popular Base Learners

Base learner is a term often used when discussing ensemble learners. According to Zhou (2009), "Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem". The multiple learners that make up the ensemble by themselves are known as base learners. These base learners serve different purposes, and each has its strengths and weaknesses when applied to different tasks. Some of the most popular learners are logistic regression, naïve Bayes classifiers, support vector machines, and decision trees.

Logistic regression (LR) originates from the field of statistics and can be used as a supervised machine learning model. It is used to predict whether an object belongs to one class or another. It was originally designed for binary classification but has been expanded to allow for multi-class prediction as well (Collins et al., 2002).

Naïve Bayes classifiers are also from statistics and are based on applying Bayes' theorem (Bayes, 1763). The classifiers use a set of simple probabilistic models to classify objects while assuming independence between features.

Support Vector Machines (SVMs) are supervised machine learning models that can be used for both classification and regression. It is used for data with binary classes. However, it can be used for multiple classes by performing multiple binary class classifications. An essential part of SVM is the mapping of the input to a high-dimensional vector space. This mapping is one of the reasons it is suitable for text classifications, as text is high dimensional by nature (Joachims, 1998).

Decision trees are based on dividing the classification into multiple stages and making rules for each stage (Safavian & Landgrebe, 1991). To classify an object, the tree has to be traversed from the root node to a leaf node taking paths based on rules set on each node. An advantage of this approach is intuitiveness, as it is

easier for an analyst to observe how decisions are made.

2.6 Deep Learning

Deep learning is a subgroup of machine learning that has gained traction over the years. As opposed to classical supervised and unsupervised learning, deep learning techniques require no manual selection and engineering of features beforehand, a task that proves challenging in many cases. Instead, deep learning techniques can automatically extract features by building representational hierarchies of abstract concepts, which increase in complexity with higher levels. According to Goodfellow et al. (2016), the power of deep learning lies in this hierarchy, which allows models to learn complicated concepts by building them out of simpler ones. The hierarchies are deep, giving deep learning its name.

Artificial Neural Networks (ANNs) make up one type of the models used in deep learning and are based on the biological neurons in the human brain. In the brain, neurons are interconnected through synapses, along which signals are sent from neuron to neuron. Analogous to this, ANNs contain layers of nodes, or neurons, that are highly interconnected through weighted links. A node that receives one or more signals will process the input and signal other nodes with which it is connected. This output is based on a non-linear function of the collected input. The simplest variants of neural networks are feed-forward, meaning that the signal travels in only one direction, originating in the input layer and propagating through the network towards the output layer, forming a directed acyclic graph.

Deep Neural Networks (DNNs) are neural networks that contain one or more hidden layers between the input and output layer. The depth of a network refers to the number of hidden layers within it, with deeper networks having more layers. The increased number of layers and accompanying nodes makes the networks more complex, which in turn allows the network to model complex data with fewer units than a similarly performing shallow network (Bengio, 2009). The networks discussed in the following paragraphs are all examples of DNNs.

Convolution Neural Networks (CNNs) are deep neural networks that have one or more convolutional layers as part of the hidden layers. Like the other hidden layers, these convolutional layers receive input, transform the input, and provide an output to the next layer based on the transformed input. In convolutional layers, however, the transformation is a convolution operation. These convolution operations are used in order to extract features based on patterns in the data. For each layer, several filters are created, which essentially are numerical matrices with variable size and content, depending on the function of the filter. When performing the convolution operation, the filters are moved over a larger matrix, representing the input data. The dot product between the filter and a section of the matrix is calculated to output a transformed result that can be used to detect patterns. CNNs are popularly used within image classification, as the pixels that make up the image are easily represented through numerical matrices. However, CNNs

can also be applied to text by representing the text as matrices by, for example, concatenating vector representations of each word.

In contrast to feed-forward networks, **Recurrent Neural Networks** (RNNs) contain cycles within the graph, allowing the output from some level in the network to be fed back as input to a previous level. These cycles allow the network to maintain a type of memory in the form of an internal state. Thus, the output of an RNN will depend on its internal state, which in turn depends on its previous inputs. Because of this, RNNs are especially well suited to handle sequential data such as time series, text, or audio.

Long Short-Term Memory (LSTM) networks are variants of RNNs that are designed to improve short-term memory by addressing the vanishing gradient problem. The problem relates to backpropagation, which is used to train and optimize neural networks. For a given prediction of the model, gradients are calculated for each node based on the results from a loss function. The internal weights of the network are adjusted based on these gradients, allowing the model to learn. However, the gradient of a given level is calculated with respect to the gradient of the layer before and can diminish drastically over the course of the backpropagation process, resulting in little or no learning in the shallow layers of the network. This applies to RNNs' backpropagation through time, where each time step in an RNN represents a layer in the neural network on which backpropagation is applied. Here, the earlier parts of the sequence of input data, like the shallow layers of the network, are not well adjusted for, meaning that long-range temporal relationships are lost. LSTM networks solve this by introducing Long Short-Term Memory cells within the network that allow LSTM based models to detect long-term dependencies in, for example, texts, making them even more useful for solving NLP-related problems.

Transformer networks, introduced by Vaswani et al. (2017), are deep neural networks that build upon the concept of attention. In short, attention allows computation of context by representing the relationship between each entry in a sequence and all the entries of another sequence.

Transformers are designed for the purpose of sequence-to-sequence modeling tasks such as translation and consist of one encoder and one decoder section. The encoder section takes a sequence as input, for example, a sentence to be translated, and transforms the sequence into embeddings, which are numerical representations of the input sequence. The decoder section takes the embeddings as input and returns the output sequence of the model. Both sections include modules that make use of self-attention. Self-attention is similar to attention, but instead of using two different sequences, the relationship is represented between each entry in a sequence and every other entry within the same sequence. Therefore, self-attention is useful for representing contextual relationships within an input sequence like a sentence.

To appreciate the benefit of employing self-attention, consider the following example using the sentence "a blue sky" as an input sequence. First, an attention matrix is computed reflecting the relationship between all pairs of words within

the sentence. The corresponding attention matrix for the example can be seen in Figure 2.1. Each value in the matrix represents the relationship between the relevant terms. For instance, the word “blue” has a stronger contextual relation to “sky” than it has to “a”. Each row in the attention matrix can subsequently be assigned to each word as their attention vector. In the example, the attention vector for the word “sky” is the last row in Figure 2.1.

	a	blue	sky
a	[-,	0.4,	0.9]
blue	[0.4,	-,	0.7]
sky	[0.9,	0.7,	-]

Figure 2.1: Example attention matrix

Transformer models take the entire sequence as input and calculate the attention vector for each entry with respect to the entire input sequence. This process differs from RNNs, where each entry in a sequence is processed sequentially. Not having to process the sequence entries sequentially brings important benefits to transformer models. Firstly, with sequential processing, context is only detected with respect to the previously processed entries. Thus, sequential processing allows only for interpreting unidirectional context, risking the loss of potentially valuable relations between entries. Transformer models can detect contexts bidirectionally by considering all other entries when calculating the attention vectors. Moreover, each attention vector is independent of the others. This independence means that the vectors can be calculated concurrently, making better use of hardware capacity when compared to RNNs. Finally, by considering the entire sequence simultaneously, the transformer models do not face the short-term memory issues that LSTM architectures attempt to solve.

2.7 BERT

Bidirectional Encoder Representation from Transformers, or BERT, is a language representation model based on the transformer network architecture. It was first introduced by Devlin et al. (2018) and has since been referenced in more than 20 000 works related to NLP.

As explained in Section 2.6, transformers are sequence-to-sequence models used for tasks like translating texts from one language to another. The model is largely divided into two sections, one encoder, and one decoder. The encoder takes as input a sequence, for instance, a text, and produces embeddings for every word in the sequence simultaneously. The word embeddings are numerical vectors that encapsulate the meaning behind the word. As such, two words that have similar meanings will produce similar word vectors. The decoder takes as input the word

embeddings from the encoder and produces the output sequence of the model.

In short, the BERT language model is constructed by stacking multiple encoders from the transformer network architecture. The stack includes either 12 or 24 encoders for the base and large BERT models, respectively. Recall that because they employ self-attention, transformer encoders can capture the context of a sequence bidirectionally. Because BERT is based on transformer encoders, it is also bidirectional in this regard. Moreover, BERT benefits from how the encoders calculate each word embedding independently. This independence allows for parallelization and utilization of hardware to speed up the encoding process.

BERT is publicly available, and a multitude of pre-trained models can be found on the web. The purpose of pre-training is for the model to understand language and context. For this, the model is trained on two unsupervised tasks simultaneously, namely Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The goal of MLM is for the model to predict masked tokens in sentences where parts of the sentence are removed. For instance, when inputting to the model "Please fill out the [MASK]", the target output of the model is [MASK]="blanks". In order to make these predictions, BERT learns to interpret bidirectional context within the sentence. In the case of NSP, the model takes two separate sentences and predicts whether the second sentence follows the first. Thus, NSP teaches the model to interpret context both within one sentence and across multiple sentences. By solving both of these tasks, the resulting model acquires a good understanding of language and context. Such pre-trained models can produce valuable word embeddings that include information about the context in which a word appears. Because of this acquired understanding of context from training on the MLM task, BERT is well equipped to handle Out of Vocabulary (OOV) words by analyzing the surrounding text of the word in question. Moreover, because it employs self-attention through the encoders, BERT models are able to detect the different meanings behind homonyms. As explained in Section 2.4, homonyms are words that, while spelled identically, carry different sentiments. An example is the word "bark", which carries different meanings in "the bark of the dog" and "the bark on the tree". Because of these advantages, one common way of using BERT is to extract the resulting word embeddings as input for another custom-made model trained for an NLP-related task.

The pre-trained BERT models are trained using massive unlabeled text datasets. In the pre-trained state, the models have a good understanding of language and can provide valuable word embeddings. However, it is not uncommon to employ a secondary fine-tuning of the model. When fine-tuning, the model is further trained in a supervised fashion to better fit specific tasks. Pre-trained BERT models are available in over 100 languages at the time of writing, making BERT readily available for use even for less common languages. Moreover, an effort has been made to create a multilingual BERT model, mBERT[1], that supports 100 languages, including English, Chinese, Spanish, German, Arabic, and Urdu.

[1]https://github.com/google-research/bert/blob/master/multilingual.md

2.8 Evaulation Methods of Models

When attempting to solve a task, it is vital to be able to compare to previous workds and other solutions. Thus, it is pertinent to use a measurable metric for evaluating the results. In machine learning, researchers often attempt to solve the same task while using separate methods. However, comparing the results of this research is not a straightforward task, as there are many factors to consider. The researchers may have used different datasets, made dissimilar assumptions, or they may have used different approaches to defining hate speech, such as using binary labelling or a more nuanced grading system.

The simplest metric to measure the results of a classification solution is the percentage of correctly classified data. However, this metric can be misleading, as using an unbalanced dataset may skew the results. This scenario can be exemplified by considering a classifier predicting when a game of roulette will land on the number 0 with the binary set of classes, namely "yes" and "no". A model that always predicts "no" might have an accuracy of about 97%. The model, while at first glance appearing quite successful, has completely failed at achieving its goal. Because of this, more sophisticated metrics are often used, such as *precision*, *recall*, *F_1-score* and *ROC/AUC*.

Precision and Recall

Precision and recall are two metrics commonly used for evaluating both information retrieval and classification results. The metrics can be used to evaluate the performance of a model classifying texts as hateful using a test set. The model will be tested on the test set and return the number of texts classified as hateful. Note that this is a mix of true and false positives. The precision can then be calculated as the number of correctly classified texts out of the total number of texts classified as hateful. Recall denotes the number of correctly classified texts out of the total number of actually hateful texts (Bengfort et al., 2018). Thus, a strict model which is very good at classifying the most hateful texts will have high precision for classifying hateful texts. However, it will have low recall if it does not classify most of the less obvious comments as hateful.

F_1-score

F_1-score, also called F-score or F-measure, is used to get a balance between precision and recall. It is calculated by taking the harmonic mean of precision and recall and, producing a number between 0 and 1. An F_1-score of 0 means that no objects have been classified correctly, while an F_1-score of 1 means that all classified objects are classified correctly. F_1-score is commonly used in the field of text classification as a metric for evaluating model performance (Baeza-Yates & Ribeiro-Neto, 2011).

ROC/AUC

A receiver operating characteristics (ROC) graph is a visualization of the performance of a classification (Fawcett, 2006). ROC graphs have been used for evaluating

machine learning models since 1989 (Spackman, 1989) and have in later years been discovered as useful metrics when working with unbalanced classes. The ROC graph is two-dimensional, with the true positive rate on the y-axis and the false positive rate on the x-axis. The performance will be shown by the ROC curve, where a random guess would result in a straight diagonal line. Some analysis can be gathered visually from the graph. However, a single number is easier to compare. The area under the curve (AUC) can be calculated to a number between 0 and 1, where 0 means all classifications are wrong, and one means all classifications are correct.

CHAPTER 3

Related Work

This thesis is concerned with the problem of hate speech detection in Norwegian. This topic is somewhat of a niche area, as the majority of prior research on the topic is focused on the English language. In addition to research for English and Norwegian, it is also pertinent to examine solutions made for different languages and language agnostic solutions that might contain transferable insights.

The following chapter begins by presenting an overview of well-known methods for hate speech detection using classification. It continues by describing commonly used features within hate speech detection. Anomaly detection is then explored before an in-depth review of the ADAHS approach by Jensen (2020). Finally, a discussion of the Norwegian dataset used by ADAHS and in this thesis' experiment is presented.

As with Chapter 2, the following chapter extends upon the specialization project preceding this thesis (Wahl & Skjåstad, 2020). The chapter has been adjusted where relevant to fit the topics of this thesis.

3.1 Classification Methods

Previous solutions to hate speech detection have historically tended toward training classifiers in a supervised fashion on manually annotated corpora (Fagni et al., 2019; Schmidt & Wiegand, 2017). These classification methods are divided mainly into two groups, namely the classic and deep learning methods (Zhang & Luo, 2019).

The **classic methods** implement the use of algorithms such as support vector machines (SVMs), linear regression (LR), naïve Bayes (NB), decision trees (DT), and Random forests (RF). According to Schmidt and Wiegand (2017), LR and SVMs are among the most used classical methods.

Waseem and Hovy (2016) used LR when experimenting with a variety of features in order to detect hate speech. Sharma et al. (2018) published a constructed dataset fetched from Twitter with multiple annotated classes and applies hate speech detection on the corpus using SVMs, RFs, and NB, achieving accuracies of 72 to 76%. Burnap and Williams (2015) used SVMs, RFs, and Bayesian LR for implementing three separate models, in addition to building an ensemble classifier comprised of a combination of all three. Here, the optimal results we obtained using the ensemble classifier. Davidson et al. (2017) tests a variety of models, including LR, naïve Bayes, decision trees, RFs, and linear SVMs. They found LR and linear SVMs to perform significantly better than the rest, achieving F_1-scores up to 0.90. Furthermore, the classical methods are included as baseline models in works that explore deep learning models, such as Fagni et al. (2019), Zhang and Luo (2019) and Del Vigna et al. (2017).

Deep learning methods have become increasingly popular in many recent state-of-the-art works. As described in Section 2.6, these methods make use of neural networks in order to automate the process of extracting features from the input. Frequently used deep learning methods in recent works include CNNs and RNNs, with LSTM networks being the most widely used variant of the latter. Gambäck and Sikdar (2017) uses CNN for creating four separate classification models, using combinations of character-level n-grams and word vectors. Fagni et al. (2019) employs a set of deep learning models including a CNN in an ensemble, achieving F_1-scores that outperform a set of baseline methods based on SVMs. Similarly, Zhang and Luo (2019) found that an extended CNN-based model outperformed baseline SVMs and, indeed, other state-of-the-art solutions at the time with its highest achieving F_1-score at 0.96. Pitsilis et al. (2018) utilize LSTM networks when exploring user history-based features for detecting hate speech. de Gibert et al. (2018) implemented classifiers using LSTM, CNN, and SVM, with the LSTM based classifier achieving the best performance of the three. Furthermore, Del Vigna et al. (2017) uses LSTM and SVM to create two classifiers that perform hate speech detection in Italian, observing that both models perform better when classifying on datasets with a higher inter-annotator agreement.

Overall, the solutions that implement deep learning methods seem to perform better than those that use classic methods. However, when exploring the implementation of BERT in classification models, Isaksen and Gambäck (2020) found that when using an RNN, learning did not improve when compared with a shallow, two-layered network. Furthermore, the classical methods still perform reasonably well and are therefore still included in many solutions as baselines for comparatively evaluating deep learning models.

3.2 Features

One central topic for all text classification problems, and indeed classification in general, is features. Classifying text in hate speech detection is no exception. Schmidt and Wiegand (2017) constructed a summary of the state-of-the-art of

hate speech detection that includes an overview of several types of features that will be discussed in this section. Furthermore, new features are subject to continuous development and testing in several bodies of work. Several additions to the summary will therefore also be included where deemed fitting.

Simple Surface Features are features that might be included in all manner of text classification tasks and include well-known features such as bag-of-words-based vectors. Many of the proposed solutions to the hate speech detection problem use either token or character-level n-grams. Waseem and Hovy (2016) found that token and character n-grams performed well on their own and that performance degraded when adding additional features, author sex being the exception. However, Nobata et al. (2016) found that including additional features improved performance, even though n-grams was the single most predictive feature.

Including character-level n-grams is beneficial over tokens in that they are less sensitive to spelling variations that occur either by mistake or intentionally. For example, a malicious user might produce the phrase *kill yrslef a$$hole* in an attempt to escape automatic detection of offensive or hateful content. While such spelling variations pose challenges when using token-leveled approaches, character-leveled variants are more easily able to detect similarities between the alternative spellings of a given word (Schmidt & Wiegand, 2017). Nobata et al. (2016) shows that, when using only token and character-based n-grams as features, the character-based methods provide the best results. Furthermore, Mehdad and Tetreault (2016) found character n-grams to be more predictive than word level n-grams in a systematic comparison between the two.

Other simple surface features that are not based on tokens or characters can also benefit hate speech detection. Nobata et al. (2016) include features based on the occurrences of URLs, the inclusion of politeness words and words not recognized by the English dictionary, capitalization, non-alpha characters, and average length of words, showing that the inclusion of these features enhances performance.

Word Generalization is a solution to the problem of data sparsity and high dimensionality, which is prone to occur when dealing with short texts. The general concept of word generalization is to establish some connection between similar words to determine commonalities between a set of words or phrases. Generally, this is achieved through word clustering or word embeddings.

When employing word clustering, the resulting cluster IDs can be allocated to each word and added as features. Algorithms for this purpose include Brown Clustering (Brown & Huntley, 1992), which allocates each word with exactly one cluster, and Latent Dirichlet Allocation (Blei et al., 2003), which provides a distribution metric for each word, indicating to which degree the word belongs to each cluster.

In later contributions, however, it has become more popular to use word embeddings for similar purposes. Word embeddings are distributed word representations based on neural networks that present words as n-dimensional vectors. These vectors can be valuable foundations for features because different words that are

semantically related may end up having similar vector representations. Popular word embeddings include Word2Vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) and fastText (Mikolov et al., 2017).

Sentiment Analysis is the task of detecting the sentiment polarity of a text. Because it is reasonable to suggest that hate speech often contains negative sentiment (Schmidt & Wiegand, 2017), these polarities could be useful to incorporate as auxiliary features. Gitari et al. (2015) utilize sentiment analysis in multi-step approaches, wherein the first step includes the use of a classifier to detect negative polarity. Additionally, Van Hee et al. (2015) presents a single-step solution that uses the frequency of positive, negative, and neutral words as features.

Lexical Resources are used to benefit from the assumption that hateful messages might include specific words such as insults, curse words, slurs, or some widely used word variation. Typically, to obtain these descriptive words, publicly available lexical resources such as hate speech-related word lists are used. The occurrence of such words is a popular choice for baseline or feature when attempting to classify hateful content. Burnap and Williams (2015) and Nobata et al. (2016) both make use of publicly available lists of hate-speech-related terms in order to improve their results. However, while being popular inclusions for features, such occurrences are usually not sufficiently descriptive alone and serve better as additions to more descriptive features such as word or character-level n-grams, as reflected in the findings of Nobata et al. (2016).

Linguistic Features make use of syntactic information of language in order to improve results. This syntactic information includes Part-of-speech (POS) tagging of words and creating typed dependency relationships. POS tagging categorizes words grammatically to identify words such as verbs, nouns, and adjectives. This categorization provides additional context that can be used as features. Building upon POS, Burnap and Williams (2015) found that by employing typed dependency relationships, which are able to capture long-distance connections between non-consecutive words, the number of false negatives was reduced by 7 percent over baseline BoW features. Such typed dependencies hold advantages over simple POS tagging in sentences such as “leave them alone” and “send them away”. The POS representations are the same in these sentences, but the dependency tuples (them, home) and (them, alone) are quite different, and the former might be more common among hateful utterances.

Meta-Information contains data about the context around a given text and can consequently be a valuable source for features in hate speech detection. An example of valuable contextual information is user history, such as the user’s sex or the frequency of which a user has produced hateful utterances in the past. Previous works that utilize such user history include Pitsilis et al. (2018), Waseem and Hovy (2016) and Unsvåg (2018).

In addition to text, modern social media also includes images, videos, and audio content. Such content is frequently commented on, and these comments can be potential sources for hate speech. Therefore, *Multi-modal Information* about non-

textual content is also included as feature sources in works such as Zhong et al. (2016) and Hosseinmardi et al. (2015), who both use information about photos posted Instagram for this purpose.

Finally, *Knowledge-Based Features* make use of world knowledge to improve the understanding of the context surrounding the sentence in question. Dinakar et al. (2012) utilizes world knowledge in order to perform hate speech detection focusing on anti-LGBT utterances. However, such approaches require manual coding and can result in solutions that only work for certain confined areas of hate speech. It is presumably for these reasons that similar solutions have been infrequent, Dinakar et al. (2012) being the only one that employs knowledge-based features to the best of the authors' knowledge.

3.3 Anomaly Detection

One approach that has recently been employed for detecting hate speech is anomaly detection. This method is used to detect unexpected, deviating, or rare behavior. Using anomaly detection for textual data is not a topic that is thoroughly explored. In the context of hate speech detection, the anomalies would be hateful utterances. According to Burkal and Veledar (2018), hate speech utterances make up only 10% of comments on social media and news sites such as Facebook, NRK, and TV2. Chandola et al. (2009) presents a structured overview of research in the field of anomaly detection for a variety of domains. This section will introduce different aspects of anomaly detection presented by this overview and how it could be applied to hate speech detection.

When working with anomaly detection, it is essential to make the distinction between noise and anomalies. In the context of hate speech detection, noise could be a comment which consists of a string of random characters. This comment would appear very different from regular text and would be of no benefit to analysis. Another distinction is between novelties and anomalies. Novelty detection aims to discover previously undetected samples or patterns in the data.

Chandola et al. (2009) establishes some main challenges for general anomaly detection that also apply to hate speech detection. One of these challenges is concerned with defining normal behavior. As text is very high dimensional, it is hard to create objective definitions. The lack of objectivity may cause an increase in false negatives and false positives. For general anomaly detection, anomalies are often the result of malicious actions, which in this context translates to users actively trying to circumvent the detection by masking their hateful comments as normal ones. Another concern is change over time. Something considered non-hateful now might be considered hateful in the future due to an event or gradual language evolution. Additionally, the unavailability of labeled data for semi-supervised and unsupervised solutions is a valid concern. Finally, how to deal with noise is also a problem with no straightforward solution which has to be considered.

3.3.1 Types of Anomalies

Point anomalies, contextual anomalies, and collective anomalies are three classes of anomalies discussed by Chandola et al. (2009). Any of the three types can be applied to hate speech detection in texts.

Point anomalies can be defined as data points that are anomalous when compared to the dataset as a whole. Using point anomalies is the most straightforward approach and can be applied to hate speech detection by transforming the text into vectors and comparing for similarity.

Contextual anomalies use the context around a data point to decide if it is an anomaly or not. This method is commonly used for time-based or spatial data, which can be applied to social media as posts are usually paired with a timestamp and sometimes a location. Contextual anomalies can be exemplified by looking at temperatures throughout the year in Norway. A point with the value of 0°C would be considered quite normal during the winter months. However, a day with 0°C during the summer would be out of the ordinary and therefore considered a contextual anomaly. This method is hard to apply to hate speech detection as topics in social media are all connected. A controversial event in world news might provoke a lot of hateful comments. Attempting to identify one of these comments as hateful using contextual anomalies would fail as it would be similar to the other related comments. This example is a problem that collective anomalies could solve.

Collective anomalies are a group of related data points that are anomalous to the dataset as a whole. This approach could detect a wave of hateful comments. However, it would also detect other trends or waves happening.

3.3.2 Data Collection and Learning Supervision

As with other machine learning techniques, a well-built dataset is a necessity when applying anomaly detection. However, data collection is not as straightforward as collecting as many data points as possible. It must also be considered if the data should be labeled or not. Labeling the data is the most costly part of the process. It could therefore be pertinent to only label some of the data. Anomaly detection can be performed in a supervised, semi-supervised or unsupervised fashion, each designed to work with three different types of datasets, respectively.

Unsupervised models use the most basic dataset where the data points are unlabelled. An unsupervised model has to infer what data points are anomalies within the dataset with no given knowledge of the data. This technique is used when it is too costly to label the dataset or when the goal is to explore potential unknown outliers or tendencies. Most unsupervised techniques will run in a single iteration over the dataset and return the detected anomalies. This approach is not particularly suitable for real-time detection because, for a new data point to be processed, the entire dataset must be reconsidered. By comparison, a trained model will be able

to make predictions for new data points without having to reconsider the entire dataset and is better equipped for real-time scenarios.

Supervised models require that the data points in the dataset are labeled as either normal or anomalies. Labeling all of the data points can be difficult and also very time-consuming. Furthermore, as the anomalous data points are rare by nature, it is hard to gather enough data to represent all the possible anomalous cases accurately. An interesting idea to combat this is to inject artificial anomalous data to improve the dataset (Fan et al., 2004).

Semi-supervised models are models that use datasets that are partially labeled. Traditionally, semi-supervised settings use datasets where the labeled portion of the data exclusively consists of normal data points. This approach can be useful where it is easy to identify what normal behavior is, while the anomalous data may be hard to define due to them being too sparse. It could also be that the anomalies are dynamic. Another less popular semi-supervised setting includes both normal anomalies in the labeled portion of the dataset. For either approach, it is crucial to consider that the unlabelled part of the dataset may be contaminated with anomalous data (Ruff et al., 2020).

3.3.3 Anomaly Detection for Text

According to Chandola et al. (2009), anomaly detection for text is primarily used to detect novel topics, events, or news stories in a collection of documents or news articles. In the context of hate speech detection, this would translate to detecting the topic of hate speech in a collection of social media comments. In terms of data, textual anomaly detection faces challenges of sparsity, high dimensionality, and temporal change. The anomaly detection techniques used for textual data are comprised of Mixture of Models, Statistical Profiling using Histograms, Support Vector Machines, Neural Networks, and Clustering Based anomaly detection (Chandola et al., 2009).

As pointed out by Jensen (2020), the application of anomaly detection to hate speech detection is a recently conceived approach. Their work is the very first to conduct in-depth research on the potential for such solutions. In it, a semi-supervised, deep anomaly detection solution to automatically detect hate speech in Norwegian is presented. For this, a CNN model is used along with pre-trained word embeddings based on fastText and GloVe.

Hendrycks et al. (2018) employed semi-supervised deep anomaly detection with datasets containing images and texts. By training anomaly detection models against an auxiliary dataset of outliers, the models were able to generalize and detect unseen anomalies, which was found to improve performance. Ruff et al. (2019) introduce a new anomaly detection method for text classification employing the pre-trained word embeddings GloVe, fastText, and BERT. Additionally, an SVM-based model was used as a baseline. The solution is found to be capable of learning distinct, diverse contexts from unlabeled text corpora.

3.4 ADAHS

The master thesis conducted by Jensen (2020) is highly relevant to discuss in depth. This thesis hypothesizes that the problem of hate speech detection can be looked at from an anomaly detection perspective and then attempts to prove that hypothesis correct. Another focus of the thesis is to attempt to solve the problem for Norwegian and possibly in a language-agnostic way. The thesis' contributions consist of a literature review, creating a Norwegian dataset, developing an anomaly detection method for detecting hate speech, and experimentation using the implemented method.

Background

The goal of the thesis is: "Investigate how to accurately detect hate speech in text using anomaly detection techniques". This goal is further split into several research questions to elaborate on precisely what the author wants to achieve. The author is not attempting to create a solution that competes with other state-of-the-art solutions, but rather to explore a new way to solve the problem, laying the foundation for future research.

Previous methods of hate speech detection using classification methods have been steadily improving over the years. However, they face some difficult challenges. These challenges are the basis of motivation for using anomaly detection, as first suggested by Gröndahl et al. (2018).

Classification methods perform best when given a balanced dataset, meaning that the classes in the dataset are more or less evenly distributed. For hate speech detection, that would mean that there would be an equal number of hateful and non-hateful messages in the dataset. This imbalance is reflected in the real-life distribution of the two types, where the amount of non-hateful messages greatly exceeds the number of hateful ones. Creating a balanced dataset would require either tweaking the collection to gather more hateful data or discarding non-hateful data to match the number of hateful data. The first suggestion could cause the method to work with data gathered with that method but would fail with a normal distribution. The second suggestion is not cost-effective as a large quantity of non-hateful data would need to be gathered and labeled just to be discarded. While this is a problem for classification methods, anomaly detection builds on the assumption that anomalies are underrepresented in the dataset.

Another concern with classification methods is that they assume similarities within a class. This assumption is a problem for topics such as language where the data is changing over time, which could mean that a new hateful message might not share many similarities with historical ones. Anomaly detection does not face this issue as it compares new data points only to the normal data class. According to Ruff et al. (2020), the assumption of similarity between data holds for the normal class but is invalid for anomalies.

Using machine learning methods generally requires a significant amount of data. In addition, classification methods require that the data is labeled with the correct classes. This process of collecting and labeling data is a costly effort. Not having to label all of the data points or simplifying the labeling process could be highly beneficial. The thesis suggests using a semi-supervised approach which means that all of the data would not have to be labeled.

The thesis outlines several challenges with hate speech detection using anomaly detection. As mentioned in the motivation, the detection of anomalies is based on dissimilarity to the normal data. This assumption means that the model needs to have a robust and extensive representation of the normal behavior in the domain. Achieving such a representation is especially difficult when working with text, as it is a high-dimensional data type, and it is impossible to represent all permutations of a textual topic.

The output of the resulting anomaly detection model is binary, where a data point is identified to be either normal or anomalous. These binary values can be found by scoring the data on the degree that it is anomalous and selecting all data above a scoring threshold as anomalous. Selecting this threshold to an appropriate value is also a challenge.

Solution

A solution is presented in the form of a deep semi-supervised Anomaly Detection Approach to Hate Speech detection called ADAHS. The ADAHS model is based on and extends a general semi-supervised anomaly detection method presented by Ruff et al. (2020) called Deep SAD and the implementation of Context Vector Data Description (CVDD) by Ruff et al. (2019). The method is rooted in the assumption that similarity holds between normal data instances but not for anomalous instances. The ADAHS solution uses a deep, CNN-based architecture in addition to pre-trained word embeddings. GloVe was used as the pre-trained word embeddings for the English dataset, while fastText was used for the Norwegian dataset.

The English and Norwegian datasets were preprocessed and cleaned to match the vocabulary of the word embeddings. Methods such as removing punctuation, adding and removing white spaces where appropriate, reducing all text to lowercase, and removing unwanted tokens such as names or usernames were utilized. In addition, a list of common misspellings was created to replace many of the OOV words with the canonically spelled ones.

Ruff et al. (2020) describes two different settings for semi-supervised anomaly detection. The first using only normal data for training, the second using both labeled normal and anomaly data for training, ADAHS making use of the latter. As the number of normal instances greatly outweighs the number of anomalous instances, most training data will be normal. One of the challenges faced is choosing the ratio of normal to anomalous instances and what impact the ratio has on the model's

performance.

Two baseline methods were selected to assess the performance of the ADAHS model. As opposed to the semi-supervised setting of ADAHS, the two baseline methods are unsupervised. The first method is a One-Class SVM model by Schölkopf et al. (2001) and the second is the CVDD by Ruff et al. (2019). The baseline methods also use the same word embeddings as ADAHS.

Results

The experimental results of the thesis are given in three parts. The first part contains the results from using the Norwegian and the English dataset to tune the parameters, achieving the optimal settings for the rest of the experiments. The second part is testing with different amounts of labeled data and how pollution influences the performance. Finally, the third part is using the baseline methods on the same data to compare the results.

It is underlined that the goal of the experiments is not to find the optimal parameters to get the most accurate results but that some tuning may help evaluate the model. Therefore, two parameters were tuned. The first one was tested on five different values based around an assumption from Ruff et al. (2020). The second parameter was tested on three different values.

The experiment using the semi-supervised approach was split into two scenarios. The first scenario was to test with varying amounts of labeled data. The second scenario entailed testing with varying amounts of pollution. In this context, pollution refers to anomalous data that is added to the set of unlabelled data.

The English dataset has six classes of anomalies data: toxic, severe toxic, obscene, threat, insult, and identity hate. Different experiments are done by adding no labeled data, labeled normal data, labeled data from one class, or labeled data from all classes. The experiments are done using both fastText and GloVe. Experimenting with the English dataset reveals that not adding any labeled data results in poor performance, almost comparable to random guess. The same applies when adding labeled normal data, which performs even worse using fastText. The scores are calculated using AUC, and the best performing configurations were using fastText. The best configurations were adding 5% labeled toxic anomalies and when adding 10% labeled anomalies from all classes.

The Norwegian dataset has five classes of increasing severity of hate speech. The tests were done with two different splits of the dataset. The first combined classes 4 and 5 as anomalies, only making up 1.84% of the entire dataset. The second combined classes 3, 4, and 5 as anomalies, making up 5.65% of the dataset. The inadequate number of anomalies means that the experiment cannot use the desired 5% and 10% added anomalous data. This test shows that the model performs poorly when no labeled data is added. It also shows that performance slightly improves with added labeled normal data. The best configurations were using all

of the available anomaly data.

The pollution test is done by adding 1% or 5% of anomalous data to the training set. The test on the English dataset generally shows that adding 1% decreases the performance slightly, and adding 5% decreases the performance slightly more. The test on the Norwegian dataset shows slight deviations, both increasing and decreasing performance by adding pollution.

The unsupervised baseline tests were performed using both the English and the Norwegian datasets. The OC-SVM tests result poorly, with the bests AUC scores being 67.6% and 50.4% using the English and Norwegian datasets, respectively. The CVDD test shows slightly improved performance, with the best AUC scores being 70.9% and 55.2% using the English and Norwegian datasets, respectively.

Discussion

There were a number of general takeaways from the results of the thesis. Firstly, all methods perform better on the English dataset than the Norwegian dataset. The methods also generally perform poorly in the unsupervised setting. Including pollution seems to yield slightly worse performance. Lastly, adding labeled anomalous data to the training set improves performance significantly.

One reason for the Norwegian dataset's unsatisfactory performance is the size of the dataset. When using only classes 4 and 5 as anomalies, they make up only 150 comments of the total of 8192 in the test set. This is a very small number when working with deep learning. With such a small dataset, it is possible that the deviating results from the pollution tests were caused by random chance. The author mentions that even with strict annotating guidelines, there was much inter-annotator disagreement. One suggestion could be to use binary classes when annotating the dataset. This approach may make it easier for annotators to label the hateful data, allowing for labeled data to be collected more effectively.

While generally producing better-performing models, the English dataset faces its own set of challenges. For instance, the dataset contains different classes of anomalies, which may overlap. This overlap is unfortunate because it is potentially unclear what classes are actually represented as anomalies.

Comparing the results using the Norwegian and English datasets is difficult for a number of reasons. Firstly, there are differences associated with the languages that might impact the results. For instance, it is conceivable that the English pre-trained models perform better than the Norwegian equivalents. Secondly, the datasets have different approaches when it comes to the annotation process. Moreover, the datasets are built from different types of sources. The Norwegian dataset is gathered from three different social media sites with set topics, while the English dataset contains more direct hate, profanities, and cursing.

Overall, the thesis shows that anomaly detection has potential in the field of hate

speech detection. The semi-supervised tests outperform the unsupervised baseline methods. This is best shown using the English tests, as the Norwegian baseline test performances barely exceed random guess. There are many potential factors to improve, such as using better datasets, using more advanced word embeddings, tuning parameters, and testing other anomaly detection techniques. Further experimentation with this approach could potentially lead to state-of-the-art performance.

3.5 Norwegian Corpus

One of the contributions of the thesis described in the last section is a new, annotated Norwegian corpus on which to perform hate speech detection. The corpus was created in a cooperative effort between Andreassen Svanes and Gunstad (2020) and Jensen (2020). Corpora are the basis on which the machine learning algorithms are trained and are subsequently significant for the models' performances. There are no current standard corpora for hate speech detection in Norwegian, so it is valuable to examine one of the latest contributions in this area. The corpus consists of 41 891 short texts with accompanying classes as defined by the theses' multiclass definitions. The data was sourced from the social media sites Facebook and Twitter, as well as from comments fetched from the politically themed news site *Resett*[1].

3.5.1 Sources

To obtain the comments from Resett, a web crawler was used to fetch the data from the 1000 most recent articles. These were mainly comments on potentially sensitive subjects such as immigration, environment, and politics. After removing duplicates, preprocessing, and discarding texts with less than ten characters, the final contribution to the corpus from Resett was 6000 comments.

The data from Twitter was collected by using the Twitter Search API for developers. This search query required search words to yield results. Because using common everyday Norwegian terms resulted in a minuscule amount of hateful data instances, the final set of search words includes only inflammatory words to increase the number of non-neutral occurrences in the dataset. After preprocessing, such as removing duplicates and non-Norwegian occurrences, a total of 24 510 short texts from Twitter were included in the constructed corpus.

From Facebook, 11 400 comments were fetched and added to the corpus. These were gathered from public pages of newspapers and public persons. The pages were manually visited, and heavily debated posts on immigration, environment, and politics were sought out. Then, the public comments were anonymized, filtered, and preprocessed before they were put in the final corpus.

[1]https://www.resett.no/

3.5.2 Annotation Process

The annotation of the corpus was done manually and on two different occasions. This is because the part of the corpus sourced from Resett was gathered and annotated in another project, preceding the master theses. The annotation was performed jointly by the authors of the theses and external annotators. For both occasions, the same grading system containing five categories from neutral to hateful was used. The five categories were Hateful, Moderately Hateful, Offensive, Provocative, and Neutral. The classes are categorized from 1 to 5, reflecting severity, where Neutral is marked as 1 and Hateful as 5. The annotation process was the same in both stages and was based on guidelines provided by de Gibert et al. (2018). These annotation guidelines were set in place such that all annotators would have the same understanding of hate speech. This included concrete definitions and examples for each of the classes, which themselves were based on previously formulated definitions provided by works such as Sanguinetti et al. (2018) and Sharma et al. (2018).

3.5.3 Discussion

The data used to build the corpus was gathered from three different sources, all of which are different domains with slight differences regarding writing patterns and composition. As shown by the results of Gröndahl et al. (2018), models tend only to perform well when tested on the same type of data on which they are trained. This issue is somewhat mitigated by the fact that all three data sources were annotated under the same circumstances and by the same annotators. Additionally, including data from different sources is beneficial for making a less case-specific solution applicable in a more general sense. However, the unequal distribution of data provided by each source could be an issue when training the classifier if the data from each source is sufficiently different from the others. For instance, the portion of the corpus sourced from Twitter makes up roughly 59% of the entire dataset. If it is assumed that the distribution of sources similar in the training set and the structure or content of the Twitter data somehow differs from the rest of the corpus, the trained model will be better equipped to classify the Twitter data than the data gathered the other two sources.

As part of the annotation process for the data fetched from Twitter, all tweets containing phrases such as “+”, “pluss”, “abo” and “DN+” were auto-annotated as neutral because the authors found that these were often neutral tweets that were created by newspaper accounts. Although they might have manually sifted through these tweets after automatically annotating them, this is not made clear. If this is not the case, it is conceivable that some non-neutral comments were falsely annotated as neutral because they include a “+” character or the word “pluss”. This erroneous labeling could contribute to degrading the performance of any model that is trained using the dataset.

As the Twitter API required the inclusion of a search word, a list of offensive

words was created to uncover tweets that contained them. However, the inclusion of offensive words by itself is not an indication of hateful or non-hateful content. Using only offensive words means that the results exclude all the hateful comments that do not contain these offensive words. Given that the data fetched from Twitter makes up 59% of the dataset, and if the same distribution of hate speech is assumed in the three parts, then most of the hate speech in the dataset will contain offensive words. This could make it more challenging to detect hate speech from other sources, which may not contain any offensive words.

3.6 Summary

In this chapter, works related to hate speech detection have been presented. These make use of both classic and deep learning methods in order to detect hateful content within texts. Throughout the works, various features have been employed that are useful for classification tasks. Some of the most important features for classification, both in general and when considering the hate speech detection problem, have been introduced and discussed. One more recently emerging technique in the field of hate speech detection is that of anomaly detection. Using this technique, recent works have been able to achieve promising results by regarding hate speech as anomalies. One such solution is presented in Jensen (2020). Along with a new, annotated corpus in Norwegian for the purpose of hate speech detection, the thesis presents a novel solution based on anomaly detection called ADAHS. This solution shows potential for hate speech detection both when tested on English and Norwegian datasets. Thus, the combination of the background theory presented in Chapter 2 and the related work discussed in this chapter lay the foundation for the remainder of this thesis, beginning with the proposed solution presented in Chapter 4.

CHAPTER 4

The BSSAD Solution

This chapter introduces and describes the BERT Semi-Supervised Anomaly Detection approach to hate speech detection (BSSAD). The approach is built upon and extends the concepts and architecture of the ADAHS approach, presented in Jensen (2020), which in turn is an extension of the Deep SAD and Context Vector Data Description (CVDD) approaches, presented in Ruff et al. (2020) and Ruff et al. (2019), respectively. The method interprets hate speech detection as an anomaly detection problem, where hateful content is considered anomalous data and neutral content is considered normal data. BSSAD uses a pre-trained BERT model to produce context-dependent word embeddings. It employs a semi-supervised setting in which the model is trained given an adjustable amount of labeled data. Furthermore, it employs a deep convolutional neural network (CNN) to extract important features for learning. This chapter presents the preprocessing techniques applied to the dataset, followed by an outline of the semi-supervised setting. Then, the architecture and behavior of the proposed model are described in detail. Lastly, a set of configurable hyperparameters are presented that can be tuned in order to optimize performance, along with their respective functionalities.

4.1 Preprocessing

For training and testing the BSSAD model, this thesis makes use of the dataset discussed in Section 3.5, which contains Norwegian short texts. However, unstructured data such as user-generated text is difficult for machines to comprehend. For computers to make sense of the data, it is necessary to process the texts and convert them to a format that the computer understands. One such format is word embeddings, where each word is converted to a set of numerical representations usable to the computer. Such conversion of texts to make them interpretable to computers without compromising the contents within is an important task and a central area within natural language processing (NLP).

Before converting the text in the dataset into word embeddings, preprocessing techniques are applied to clean the data. Below, these techniques and any others employed as part of the BSSAD approach are introduced and described.

As part of lexical analysis, the text is lowercased, and all occurrences of '\n' are replaced by whitespace. The whitespace on either end of each text is also removed. Whitespace is added after '.' to avoid occurrences where '.' is erroneously followed by another character. This correction was added after observing many such occurrences in the dataset. Not adding the whitespaces would result in an incorrect concatenation of words when stripping punctuation from the text. An example is ' [...] her.dette [...]' that, after removing punctuation, would result in a perceived single word 'herdette' instead of the presumably correct 'her dette'. Then, the text is stripped for punctuation and special characters, not including the Norwegian characters 'æ', 'ø' and 'å'. Next, the special tokens '@user' and 'navn', referring to user entities, are removed. Finally, redundant whitespaces are also removed.

Common misspellings are corrected in the dataset to minimize the amount of out of vocabulary (OOV) word occurrences. Traditionally, misspellings have been essential to correct when employing pre-trained, static word embeddings with well-defined word vocabularies. Left untreated, misspellings are equivalent to OOV words and lose valuable information as static pre-trained models have no corresponding word vectors. BERT models are better equipped to handle OOV words by inferring context, so misspellings are not equally detrimental. Nevertheless, BERT models are not immune to this issue, and OOV words might still result in the loss of valuable information. Because of this, correction of common misspellings is carried out. The correction process is based on the vocabulary of a pre-trained fastText model for Norwegian Bokmål. Meanwhile, the word embeddings used by the BSSAD model is produced by a more advanced pre-trained BERT model for Norwegian Bokmål[1]. While there might be slight differences between the vocabularies of the fastText and BERT models, the corrections are considered good enough for use with both.

As mentioned in Section 2.3.1, one central part in text preprocessing is tokenization. In this project, words are converted into integer tokens, representing texts as ordered lists of numbers. For this conversion, a pre-trained BERT tokenizer for Norwegian Bokmål is used. This tokenizer is designed to encode and decode tokens appropriate as inputs for the pre-trained BERT model for Norwegian Bokmål employed in the first layer of the BSSAD model to generate word embeddings. Finally, Section 2.3.1 mentions other common preprocessing techniques such as stopword elimination, stemming, and keyword selection. Using these techniques removes parts of the texts that could carry valuable information, which could negatively affect the training potential of the data. Because of this, such techniques are not included as part of the data preprocessing.

Before being fed to the model, the data is collected in batches of multiple texts that are given simultaneously. By default, these batches contain 64 texts, but the batch size (b) is adjustable. When dividing the data into batches, if there are

[1]https://huggingface.co/nbailab

not enough data samples to fill the last of the batches, the final, non-full batch is dropped. Then, the texts are padded so that all texts within a single batch are of equal length. This padding is done by concatenating padding tokens at the end of each text in the batch. After the texts are padded, they are stacked and then transposed to achieve better performance and integration using CUDA.

4.2 Semi-Supervised Setting

BSSAD utilizes a semi-supervised setting when training the model. To achieve this, a portion of the data samples supplied to the model is labeled as either normal or anomalous. The model is given no knowledge of the labels of the remaining data. The number of labeled samples can be adjusted and is subject to testing as part of this thesis' experiments.

For the semi-supervised setting, assume we have n sample-label pairs (x_1, l_1), $(x_2, l_2), ..., (x_n, l_n)$ with $l \in \{-1, 0, +1\}$ where $l = -1$ denotes normal data, $l = -1$ anomalous data and $l = 0$ represents unlabeled data. The dataset used in this thesis is fully labeled, and so to achieve such a semi-supervised setting, a large portion of the labels must be kept inaccessible for the model during training. This is achieved by setting $l = 0$ for the relevant samples before being used by the model. However, the original labeling of the dataset is kept in order to evaluate the model's performance.

The amount of labeled normal and anomalous data samples is denoted by γ_a, γ_n, and γ_l. Here, γ_a refers to labeled anomalous samples and γ_n to labeled normal samples. Additionally, γ_l describes both labeled anomalous and normal samples in the case where they have equal values. That is, whenever $\gamma_l = k$, it follows that $\gamma_a = \gamma_n = k$. The values of γ_a and γ_n describe the desired percentage of labeled anomalous and normal samples in relation to the entire dataset.

It is not always the case that there are enough samples to meet the desired ratio. In accordance with the nature of anomaly detection problems, this issue mainly concerns anomalous data. In such situations, the solution is limited to provide the number of labeled anomalies available.

4.3 Model Architecture

Below is an overview of the architecture of the proposed BSSAD solution. BSSAD implements a modified iteration of the ADAHS solution presented by Jensen (2020), with the inclusion of BERT embeddings. ADAHS, in turn, was an iteration based on Deep SAD, presented by Ruff et al. (2020) and CVDD, presented by Ruff et al. (2019). As such, the BSSAD architecture shares commonalities with CVDD, Deep SAD, and ADAHS.

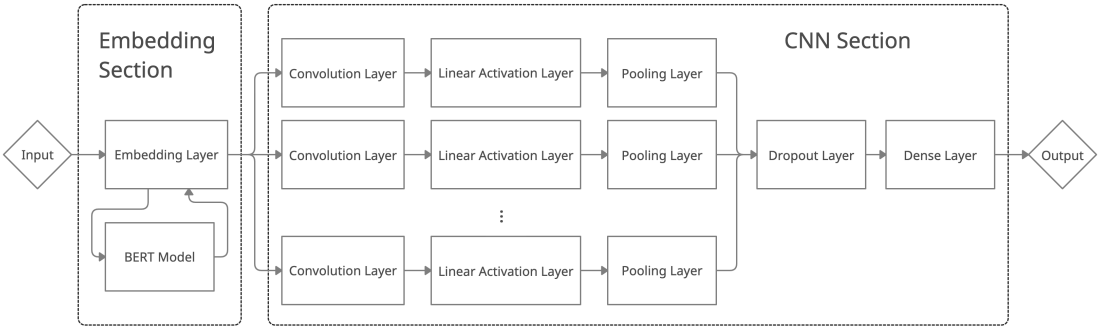


Figure 4.1: Overview of the BSSAD architecture

Fundamentally, the BSSAD solution can be divided into two main segments; one embedding segment and one convolutional neural network (CNN) segment. The embedding segment takes the input of the model, which is the cleaned and tokenized batches of texts described in 4.1, and generates word embeddings using a pre-trained BERT model. The second segment includes a CNN that performs convolution operations on the stacked word embeddings to extrapolate features and patterns that can be used for learning. Figure Figure 4.1 shows an overview of the architecture of the BSSAD solution, and depicts the architecture's two main sections.

The BSSAD model employs a CNN and consists of a collection of layers. The word embeddings are produced in the first layer of the network and subsequently fed into a convolutional layer. This layer performs convolution operations on the stacked embeddings. Next, in a linear activation layer, the rectified linear unit (ReLu) activation function is applied to the output of the convolutional layer. Then, max pooling is used to reduce dimensionality in a pooling layer. A dropout layer is implemented for increased robustness before a final, fully connected dense layer applies linear transformation to produce the output of the model.

The first step for the model is to convert the cleaned, tokenized, and batched text into word embeddings. This conversion is accomplished in the first layer of the network and is imperative for the model's ability to learn and perform well. BSSAD supports the use of NoTraM[2], which offers pre-trained BERT models for Norwegian Bokmål. Specifically, BSSAD employs the `NB-BERT-base`[3] model, which is trained on a vast collection of Norwegian text from the last two centuries. The output of the last hidden state of this model is the word embeddings, which are extracted for further use in the BSSAD model. These word embeddings have a dimensionality of 768 × sentence length. Recall from Section 4.1 that the sentence length is fixed for each batch but may vary between different batches.

The convolution operations are performed by running filters over matrices in search of internal patterns, as described in 2.6. Convolution is commonly used on images, where the matrix is based on the values of each pixel. When applied in NLP, the

[2]https://github.com/NBAiLab/notram
[3]https://huggingface.co/NbAiLab/nb-bert-base

matrices can be created by stacking the word embeddings for each word in a text. Given the word length l_w and the word embedding dimensionality d_e, stacking the word embeddings results in a $l_w \times d_e$ matrix. The filters that are run over the matrices can have variable sizes. We denote the filter size $w_f \times h_f$, where w_f and h_f is the width and height of the filter, respectively. When applied to images, it is common for both w_f and h_f to be of variable sizes. In NLP, however, the filters usually employs $w_f = d_e$, and varies h_f. As mentioned above, in the case of BSSAD, $d_e = 768$. In practice, this means that the filters consider the entire word embedding vectors of h_f words, which again can be described as considering the word embeddings of the h_f-grams in the text. Recall from 2.3.1 that the n-grams of a given text are the collection of the n consecutive words in said text. By applying filters that run over multiple words simultaneously, the convolution operation is able to capture some context to the words being analyzed, making it easier to reflect content and meaning in the outputs.

The convolution process can simultaneously employ several filters of different sizes to analyze different regions of the input text. The amount and sizes of these filters are determined on a configurable set of h_fs, denoted as the filter set (f). By default, $f = [1, 2, 3, 4, 5]$.

For each h_f if the filter set, 100 filters of size $h_f \times 768$ are used in the convolution operations to pick up complimentary features from the same regions in the matrices.

The output of the convolutional layer is fed to the linear activation layer. Here, the ReLu activation function is applied to the output of the convolutional layer, and the result is passed to the pooling layer. In this layer, max pooling is used to extract the highest value for each feature. This reduces the dimensionality of the output from the linear activation layer while retaining the highest valued features. The results of the max-pooling are then concatenated to a feature vector. The feature vector is fed to a dropout layer to prevent the model from overfitting and increase robustness. Here, dropout is performed by randomly dropping nodes during training with a probability p of $p = 0.50$. By employing dropout, overfitting is avoided by making sure that the model is resilient to the loss of any individual piece of data (Nielsen, 2015).

Lastly, the output vector of the dropout layer is fed to the final, fully connected layer. Here, linear transformation is applied to the vector to reduce its dimensionality to a representation dimension (d). The resulting vector is the model's final output, so the value of d dictates the dimensionality of the output space. While being adjustable, the default value for d is $d = 32$.

The output vector is used to calculate the anomaly score of the input text. This is done by defining a d-dimensional hypersphere as the mean from the previously encountered output vectors, and calculating the distance between this sphere and the new output vector from the model. The distance between the hypersphere and the output vector is denoted as the text's anomaly score. This method builds on the assumption that the distribution of the normal texts is uniform, whereas hateful texts are considered anomalies that do not conform to this distribution and

will thus generally lie further away from the hypersphere.

4.4 Optimisation

The BSSAD model has a set of configurable hyperparameters that can be tuned to optimize its performance. These hyperparameters include learning rate (lr), eta (η), weight decay (λ), batch size (b), representational dimension of the output space (d), and filter set (f). Below, each hyperparameter is explained in terms of its impact on performance and functionality.

Learning rate: The learning rate (lr) is a hyperparameter that influences how the model adjusts during training. When the loss is calculated for a given input to the model, the gradient of that loss is subsequently also calculated. The gradients are then multiplied by lr, resulting in the value that the weights are subtracted by. Thus, lr dictates how much impact the loss value has on the model, or in other terms, how much the model learns from each input. Higher values for lr will cause the model to make larger adjustments, risking perpetual over adjusting that prevents the model from performing optimally. Conversely, lower values for lr will cause the model to adjust less, requiring longer training to achieve optimal performance.

Eta: The hyperparameter eta is proprietary to the Deep SAD solution and its iterations and was introduced by Ruff et al. (2020). The purpose of eta is to adjust how the model weighs labeled normal and anomalous data provided during training. When $eta = 1$, the model weighs the normal and anomalous data equally. When $eta > 1$, the anomalous data is weighed more. Conversely, when $eta < 1$, the labeled normal data outweighs the anomalous data.

Weight decay: L_2 regularisation, or weight decay, is tuned using different values for λ. Weight decay is important to optimize in order to prevent overfitting. In short, overfitting occurs when the model specializes too much on the training data, meaning it performs well on that particular data but is unable to generalize enough to perform well on previously unseen data. Introducing weight decay prevents overfitting by penalizing higher complexity in the model. This is done by adding to the loss function an additional term dependent on the size of the weights. In turn, this incentivizes the model to reduce weights, reducing the impact of some layers, thus reducing the overall complexity of the model and consequently its tendency to overfit the training data.

Batch size: The batch size b is an integer value that dictates how many comments are processed by the network in a single pass-through. Choosing a larger batch size allows the machine to utilize more of its capacity and can shorten the experiments' running time. However, the benefits of choosing a high batch size are limited to the capabilities of the computer. Moreover, Even if the computer can handle large batches, the quality of the model may degrade as the batch size increases.

Representation dimension: The representation dimension d dictates the size of the output vector of the model. Larger values for d allows for representing more complex information in the vector. However, for exceedingly high values of d, there is a risk of making the output vector too complex, resulting in the inclusion of obscurities and noise. Conversely, having a too low value for d means complex but potentially valuable information cannot be represented. Therefore, an appropriate middle ground should be found through tuning.

Filter set: As explained in Section 4.3, the filter set f is a list of integers that dictate the amount and sizes of filters to be used in the convolutional layer. Each integer in the list represents the number of consecutive words, or n-grams, that is being processed by the corresponding filter. As an example, using $f = \{1, 2, 3\}$ would result in filters that consider 1-grams, 2-grams, and 3-grams, respectively. For each increase of f, another filter is added with greater size and complexity, along with a higher computational cost. Larger filters are able to capture the context over multiple words, making them beneficial for longer texts. However, the increase in size means that the filter can detect obscure and specific patterns that may not be useful for the model's purpose, decreasing its performance. On the opposite end, when using only small n-grams, some of the contextual meaning in the text may be discarded as the n-grams only contain shallow information.

CHAPTER 5

Experiment and Results

The following chapter consists of three parts. First, the experiment plan is presented, explaining the goal of the experiments and its overall structure. Then, the experimental setup is introduced, explaining the base configurations and the data distribution of the dataset. The setup includes what values will be tested and why they were chosen. The configurations of all of the tests are provided for reproducibility and further evaluation of the results. The final part of the chapter presents the results of the experiments.

5.1 Experiment Plan

The experiment section of this thesis consists of the tests made to the solution after it was initially realized. In other words, the trial and error part of implementing BERT into the preceding solution will not be part of the results. After BERT was successfully implemented, it was essential to create a structured and scientifically supported plan moving forward. The plan is vital to stay on track and progress toward answering the research questions in the allotted time frame. The running time of the experiments was also an important consideration. Each test run took several hours, and the results were usually not available before the following working day. Therefore, running incorrect or irrelevant tests would cost crucial time. Another goal of the plan was to make the results be presentable and provide a clear understanding of the path from beginning to end, underlining why choices were made, where values were gathered and what can be learned from the results.

The experiment was performed as an exploratory study investigating how an initial implementation of BERT would perform and the impact of hyperparameter tuning on the performance. When tuning the hyperparameters, it was necessary to identify logical and relevant values to test. Thus, the values were chosen based on previously applied ones from other sources.

It was crucial to isolate the setup variations between runs to the values being evaluated throughout the testing. Hence, when examining different hyperparameters and configurations, an effort was made to keep the model consistent in all other respects. For this, a set of base configurations was defined. Instead of adjusting the configurations iteratively after every hyperparameter test, the base configurations were used throughout while only changing the value of the tested hyperparameter. The exception is in the third and final stage of the tests, wherein the optimal configurations from previous stages are used. This is done to more accurately compare the model's performance to preceding solutions while also testing the model's capabilities under optimal conditions.

Performing the hyperparameter tuning using base configurations was a compromise we had to make to preserve the experiment's integrity. One preferred approach would be to implement a grid search to identify the optimal parameters by iterating over possible combinations within a defined range for each parameter. However, given the temporal restrictions and the fact that each test run took several hours, such a setup was deemed unfeasible for this thesis.

When the hyperparameters were satisfactorily tuned, the goal was to evaluate the model's performance while providing various amounts of labeled anomaly and normal data. The reasoning behind this is twofold: firstly, it is done in order to be able to compare more results with ADAHS, and secondly, to identify the relative importance between the inclusions of known/unknown anomaly/normal data in the training of the model.

5.2 Experiment Setup

In this section, the details of the experiments are presented, including which hyperparameters and configurations were evaluated. Additionally, the background for choosing the tested values is provided. Thus, the following paragraphs serve as rationales for the test configurations and guides for reproducing the experiments.

5.2.1 Data Distribution

In accordance with the research goal, the model's performance was evaluated on Norwegian short texts. Thus, all experiments were performed using the Norwegian dataset presented in Jensen (2020) and Andreassen Svanes and Gunstad (2020). The text preprocessing methods described in Section 4.1 were applied to the dataset prior to use in the experiments. The resulting distribution of comments after preprocessing and selection is presented in Table 5.1.

When running the experiments, anomalies and non-anomalies are defined through a configurable collection of categories. Specifically, one anomaly class and one normal class are constructed by choosing which of the categories in the dataset make up each respective class. These two classes are then used as sources for sampling

Table 5.1: Distribution of comments

Category	Samples	Percentage
1 - Neutral	34083	82.8%
2 - Provocative	4734	11.5%
3 - Offensive	1563	3.8%
4 - Moderately hateful	509	1.2%
5 - Hateful	250	0.6%
Total	41139	100%

labeled data in the learning phase and evaluating the model's performance regarding detecting anomalies. We denote here that AC and NC represent the anomaly class and the normal class, respectively. Furthermore, we make use of notations on the form of AC = $\{4, 5\}$ to express that the anomaly class contains categories 4 and 5. According to Jensen (2020), the inter-annotator agreement scores for the dataset used in the experiments indicate that annotators have difficulty agreeing on the annotation of categories $\{4, 5\}$. Therefore, one test was set up having AC = $\{4, 5\}$ and NC = $\{1, 2, 3\}$. Furthermore, many hate speech detection solutions struggle to separate hateful and offensive utterances (Jensen, 2020). Because of this, an additional test was set up having AC = $\{3, 4, 5\}$ and NC = $\{1, 2\}$. Finally, for exploratory purposes, it was interesting to investigate the model's performance when all non-neutral categories were included as anomalies. Thus, a final test was planned having AC = $\{2, 3, 4, 5\}$ and NC = $\{1\}$.

The results of the tests, shown in Table 5.6, indicate that having AC = $\{4, 5\}$ yielded the best performance. Thus, this was used in the base configurations mentioned in Section 5.2.2. In the final phase of the experiments, described in Section 5.2.6, a more comprehensive examination was performed having AC = $\{4, 5\}$ and AC = $\{3, 4, 5\}$. Here, for each configuration of AC, the model's performance is tested when varying the number of labeled samples of AC and NC provided in the training phase.

5.2.2 Base Configurations

Initially, the experiment was run using the default settings of ADAHS. These settings included the following hyperparameters and configurations: Learning rate lr = 1e−5, eta $\eta = 10$, weight decay λ = 1e−6, batch size $b = 64$, representation dimention $d = 32$, and filter set $f = \{1, 2, 3, 4, 5\}$. The default supervised settings include $(\gamma_n, \gamma_a) = (0.10, 0.10)$, and having the anomaly class consisting of categories {4,5}. In order to ensure reproducability and consistency, the same seed was used in all experiments, arbitrarily set to 1337. Furthermore, in all experiments, the model is trained over 100 epochs. These configurations were regarded as base configurations and were employed in all tests except the semi-supervised setting tests.

5.2.3 Experiment Phases

The experiment to be conducted is split into three phases, with varying goals. The first phase aims to test the initial configuration of the proposed model. In the second phase, a range of values are tested for the hyperparameters to find an optimized configuration. The third test aims to measure the impact of the distribution of labeled normal and anomaly data to the model's training. This third and final phase is run using the optimized configurations discovered from phase 2.

5.2.4 Phase 1 - BERT Embeddings

The first part of the experiment is the first successful run with BERT implemented, as explained in Section 5.1. This test serves as a base test to identify the viability of relying on BERT as word embeddings in the context of hate speech detection in Norwegian. The test was run in three configurations, as reflected in Table 5.2. In the tests, the anomaly class consisted of categories $\{2, 3, 4, 5\}$, $\{3, 4, 5\}$, and $\{4, 5\}$, respectively. The reasoning for these configurations are as stated above in Section 5.2.1. In all other regards, the experiments were similar, and all employed the base configurations described in Section 5.2.2.

Table 5.2: BERT Embedding setup

Anomaly Classes
$\{2, 3, 4, 5\}$
$\{3, 4, 5\}$
$\{4, 5\}$

5.2.5 Phase 2 - Hyperparameter Tuning

In this second phase of experiments, configurations and hyperparameters were tuned and tested in parallel. These parameters were all tested using the base configurations. Testing all possible combinations of the chosen parameter value ranges would ensure optimal performance. However, as mentioned in Section 5.1, such a comprehensive testing phase was considered infeasible due to the time restrictions of this thesis. Instead, running the tests in parallel allowed testing a reasonably sized set of parameters, providing a best-effort performance increase for the BSSAD model. The parameters tested in parallel included eta η, weight decay λ, batch size b, learning rate lr, representational dimension of the output space d and filter sets f. The following paragraphs contain the setup and short explanations for the chosen values for each of the tests.

Learning Rate

One effective way to explore different values for learning rate lr is by annealing, a technique proposed by Smith (2017). This method relies on periodically altering lr over a run of several epochs. One variation on this is step decay, in which the learning rate is decreased by some factor Γ every number of epochs. For each epoch, the training loss can be calculated and plotted in a graph that can later be analyzed to locate areas of good learning rate values.

The first part of the lr test utilizes a learning rate scheduler to perform learning rate annealing. The test has an initial learning rate of 1e$-$8 and a *step value* of 20. A step value of 20 means that the learning rate will be changed every 20 epochs, where the amount of change is decided by the gamma value Γ. Γ is what the learning rate will be multiplied by after every 20 epochs. This test uses $\Gamma = 10$. This means the test will run with $lr =$ 1e$-$8 for the first 20 epochs, then $lr =$ 1e$-$7 for the next 20 epochs, then continue this process until $lr =$ 1e$-$4 for the last 20 epochs. The resulting training loss curve can be visually examined and used to identify where the training loss decreases the most. Such a decrease may indicate potentially good learning rate values.

The second part of the test is an experiment using static learning rate values. Following the previous test, the resulting graph showed promising results for $lr \approx$ 1e$-$5. A range of static values around the promising value was selected: $lr \in \{$1e$-$7, 1e$-$6, 1e$-$5, 1e$-$4, 1e$-$3$\}$. Goodfellow et al. (2016) explains that selecting a learning rate too low will cause slow learning and possibly lead to converging towards a high value of training loss. Selecting a large value may lead to a feedback loop where the weights may continuously increase to a numerical overflow. After executing the tests with static lr values, the resulting loss curves and AUC scores were analyzed to find a good value for lr.

Finally, Jensen (2020) achieved improved results by employing a two-phase learning rate setup. This setup involves defining a searching phase spanning the first 50 epochs, having $lr =$ 1e$-$4, and a fine-tuning phase in the final 50 epochs having $lr =$ 1e$-$5. Building on this concept, an additional two tests were designed with a similar setup. One of the two employed $lr =$ 1e$-$4 in the searching phase and $lr =$ 1e$-$5 in the fine-tuning phase. The other used $lr =$ 1e$-$5 in the searching phase and $lr =$ 1e$-$6 in the fine-tuning phase. Both cases considered the first 50 epochs to the the searching phase and the last 50 to be the fine-tuning phase. The different testing configurations for lr are reflected in Table 5.3.

Eta

As descriced in Section 4.4, the hyperparameter η dictates how the labeled normal and anomalous data is weighed during training. The values tested for η were chosen based on the earlier implementations of the Deep SAD model, including ADAHS. Ruff et al. (2020) used $\eta \in \{0.01, 0.1, 1, 10, 100\}$, while Jensen (2020) used $\eta \in \{0.01, 1, 5, 10\}$. Consequently, $\eta \in \{0.1, 1, 5, 10, 50, 100\}$ were initially

Table 5.3: Learning rate setup

	Starting lr	Γ	Step size	Ending lr
Annealing	1e−8	10	20	1e−3
Static lr	1e−7	–	–	1e−7
	1e−6	–	–	1e−6
	1e−5	–	–	1e−5
	1e−4	–	–	1e−4
	1e−3	–	–	1e−3
Two-phase lr schedule	1e−4	0.1	50	1e−5
	1e−5	0.1	50	1e−6

used as values for testing η as they are within the range of previous tests. After these values were tested, tendencies emerged, indicating that potentially better-performing alternatives existed outside the chosen set of values. Because of this, further exploration was conducted. For the following test phase, the values $\eta \in \{20, 30, 40, 60, 70, 80, 90, 250, 500, 1000\}$ were chosen. The second phase ensured that more potentially optimal values would be tested. An overview of the tests for η can be seen in Table 5.4.

Weight decay

When testing different values for λ, used in L_2 regularisation, or weight decay, a similar approach was adopted. The selected values to test were based on the previous implementations of Deep SAD, with a few additional ones as supplementation. When running their experiments on Deep SAD, Ruff et al. (2020) employed $\lambda = 1e{-}6$. Jensen (2020) used $\lambda \in \{0.5e{-}4, 0.5e{-}5, 0.5e{-}6\}$, emphasizing that testing only these three values does not result in a sufficiently comprehensive exploration. Based on this, $\lambda \in \{1, 1e{-}1, 1e{-}2, 1e{-}3, 1e{-}4, 1e{-}5, 1e{-}6, 1e{-}7\}$ were all tested. An overview of the tests for λ can be seen in Table 5.4.

Batch size

Furthermore, different values for batch size b were tested for potential enhancements in run times and performance. For this configuration, relatively little exploratory work seems to have taken place with previous implementations. Jensen (2020) uses $b = 64$ in all presented experiments. In their work, they mention that tests were ran using $b = 300$ to match the dimensions of the applied word embeddings. Though this decreased performance, further exploration of different batch sizes is encouraged. Conversely, Ruff et al. (2020) does not explicitly state the batch size

used in their experiments. However, the default value for b is 128, provided in the code for the model described in their work. Subsequently, $b \in \{16, 32, 64, 128\}$ were chosen for testing. An overview of the tests for b can be seen in Table 5.4.

Representation dimension

When testing representation dimension of the output space d, the relatively broad set of values $d \in \{16, 32, 64, 128, 256, 512, 1024\}$ were considered. The difference in increments was chosen to fully map the relatively little explored lower values in the previous implementations of Deep SAD. Jensen (2020) uses $d = 32$ in all experiments, while Ruff et al. (2020) tests $d \in \{2^4, ..., 2^9\}$. Interestingly, the results of the latter show tendencies toward better performance converging to an upper bound with higher values for d, incentivizing further exploration of even higher values. An overview of the tests for d can be seen in Table 5.4.

Filter set

Different sets of filters to be used in the convolutional layer of the network were also tested. As explained in Section 4.3, each filter set f represents filters of size $n \times s$ where $n \in f$ and s is equal to the word embedding vector dimensionality. Thus, when employing BERT embeddings, $s = 768$. For the experiments, $f = \{1, 2, 3\}$, $f = \{1, 2, 3, 4\}$, $f = \{1, 2, 3, 4, 5\}$, $f = \{1, 2, 3, 4, 5, 6\}$, $f = \{1, 2, 3, 4, 5, 6, 7\}$ and $f = \{1, 2, 3, 4, 5, 6, 7, 8\}$ were used. For comparison, Jensen (2020) uses $f = \{1, 2, 3, 4, 5\}$ in all experiments, encouraging extended testing in future implementations. Ruff et al. (2020) uses different approaches to filters depending on which network is chosen. For the networks that make use of convolution, varying numbers of 5×5 filters are employed. However, these values are comparatively less relevant for this thesis as Deep SAD is not NLP-focused, and the networks are made with other data formats in mind, such as images. Furthermore, BSSAD implements a modified version of the ADAHS network presented by Jensen (2020). An overview of the tests for f can be seen in Table 5.4.

5.2.6 Phase 3 - Semi-Supervised Setting

In the final phase of testing, a set of different scenarios were examined with different semi-supervised settings. This phase entailed measuring the model's performance when providing different amounts of labeled normal and anomalous data and combinations of the two in the training phase. It was essential to be able to compare the performance of this model to previous implementations. As such, the setup of the semi-supervised tests was similar to the one presented by Jensen (2020), with a few additional values added for exploration. The tests in this third and final phase were performed using the optimal settings discovered in previous phases.

As described in Section 5.2.1, the tests consider two different scenarios with re-

Table 5.4: Hyperparameter setup

Hyperparameters	η	λ	b	d	f
Values	0.1	1	16	16	$\{1, 2, 3\}$
	1	1e−1	32	32	$\{1, 2, 3, 4\}$
	5	1e−2	64	64	$\{1, 2, 3, 4, 5\}$
	10	1e−3	128	128	$\{1, 2, 3, 4, 5, 6\}$
	20	1e−4		256	$\{1, 2, 3, 4, 5, 6, 7\}$
	30	1e−5		512	$\{1, 2, 3, 4, 5, 6, 7, 8\}$
	40	1e−6		1024	
	50	1e−7			
	60				
	70				
	80				
	90				
	100				
	250				
	500				
	1000				

gards to defining the normal and anomaly classes. In one case, $AC = \{4, 5\}$, while in the other, $AC = \{3, 4, 5\}$. In both cases, the desired ratio of labeled anomalies γ_a is drawn randomly from these categories. Likewise, the desired ratio of the labeled normal samples γ_n are drawn randomly from the remaining categories $\{1, 2\}$ and $\{1, 2, 3\}$, respectively. The values for (γ_n, γ_a) used during the tests are $(\gamma_n, \gamma_a) \in \{(0.00, 0.00), (0.10, 0.00), (0.20, 0.00), (0.00, 0.01), (0.00, 0.02), (0.00, 0.04), (0.00, 0.06)\}$. Additionally, $\gamma_l \in \{0.01, 0.02, 0.04, 0.06, 0.10\}$ were also employed during testing. Recall that whenever a value is set for γ_l, it follows that $\gamma_n = \gamma_a = \gamma_l$, as explained in Section 4.2. In the cases where $(\gamma_n, \gamma_a) \in \{(0.00, 0.00), (0.10, 0.00), (0.20, 0.00)\}$, no labeled anomalies are provided. Moreover, when $(\gamma_n, \gamma_a) = \{(0.00, 0.00)\}$, hereafter named the base scenario, no labeled data is provided to the model whatsoever. An overview of the test configurations are provided in Table 5.5.

While γ_a denotes the desirable ratio of labeled anomalies, the actual ratio achievable is specific to each scenario due to the different number of anomalies available. The total amount of data samples in categories $\{4, 5\}$ make up less than 2% of the dataset. For the scenario where $AC = \{4, 5\}$, the actual value of labeled anomalies, therefore, has an upper bound of 1.84%. Having $\gamma_a > 0.0184$ will result in all anomalies fed to the model during training being labeled. Similarly, when $AC = \{3, 4, 5\}$, an upper bound of γ_a exists at 5.65% for the same reason.

Theoretically, the same applies to the labeled normal samples. However, since the normal data by definition constitutes most of the dataset, the issue has no practical importance when including smaller portions of labeled data.

Table 5.5: Semi-supervised setting setup

Anomaly Classes	γ_n	γ_a
–	–	–
	10%	–
	20%	–
$\{4, 5\}$	–	1%
	–	2%
	1%	1%
	2%	2%
	4%	4%
	6%	6%
	10%	10%
$\{3, 4, 5\}$	–	1%
	–	2%
	–	4%
	–	6%
	1%	1%
	2%	2%
	4%	4%
	6%	6%
	10%	10%

5.2.7 Evaluation Metrics

For each configuration of the model, it is first trained using a portion of the dataset and then tested using another, smaller portion, where metrics are gathered during both processes. Validation loss, training loss, and ROC/AUC scores are calculated for each epoch and stored for analysis during the training. These scores are later plotted in graphs for visual representations that can be analyzed to rate the model's change in performance throughout the training process.

It is advantageous to use a standard metric to evaluate as objectively as possible. The standard metric in the field is AUC, and so this is the main metric used to evaluate the model's performance in this experiment. As such, the AUC score is considered for all experiments. However, some hyperparameters seek to increase performance that is not necessarily reflected in the AUC score. For instance, the purpose of increasing b is to reduce the run time of the experiment. Because of this, the run times for the experiments should also be evaluated when testing b. Similarly, for lr and wd, it is also necessary to consider the loss curves during training.

5.3 Results

The results of the experiments outlined in section Section 5.2 are presented in this section. Each test presents noteworthy values and observable trends. For the hyperparameter tuning phase, the best performing or most suitable value is outlined. The outlined values are the ones used as the configuration for the semi-supervised setting tests. The semi-supervised setting phase shows how the model performs using different amounts of added labeled data.

5.3.1 Phase 1 - BERT Embeddings

The first test of the experiment is to test the BERT embedding implementation using the base configurations as defined in Section 5.2.2. The supervised setting for these three tests was using 10% normal and 10% anomalies. However, as discussed, there is not enough anomaly data when using classes $\{4, 5\}$ and $\{3, 4, 5\}$ as anomalies. For AC = $\{4, 5\}$ the percentage of known anomalies is 1.84%, for AC = $\{3, 4, 5\}$ it is 5.65% and 10% for AC = $\{2, 3, 4, 5\}$.

The goal of this test was to verify the initial viability of using BERT. The results in Table 5.6 show that all three configurations achieved an AUC score above the best AUC score achieved using ADAHS on the Norwegian dataset. The best AUC achieved using ADAHS was 75.3% with $\{4, 5\}$ as anomalies and 77.3% with $\{3, 4, 5\}$ as anomalies (Jensen, 2020). However, the performance when using BSSAD improves with anomalies selected from the more severe categories. The AC = $\{4, 5\}$ test has 2.34% higher AUC than AC = $\{3, 4, 5\}$ which in turn has 5.5% higher AUC than AC = $\{2, 3, 4, 5\}$. As AC = $\{4, 5\}$ has the best result, it was chosen to be used during the parameter tuning in the next phase.

Table 5.6: BERT Embedding results (AUC score in %)

Anomaly Classes	AUC
$\{2, 3, 4, 5\}$	81.23
$\{3, 4, 5\}$	86.73
$\{4, 5\}$	**89.07**

5.3.2 Phase 2 - Hyperparameter Tuning

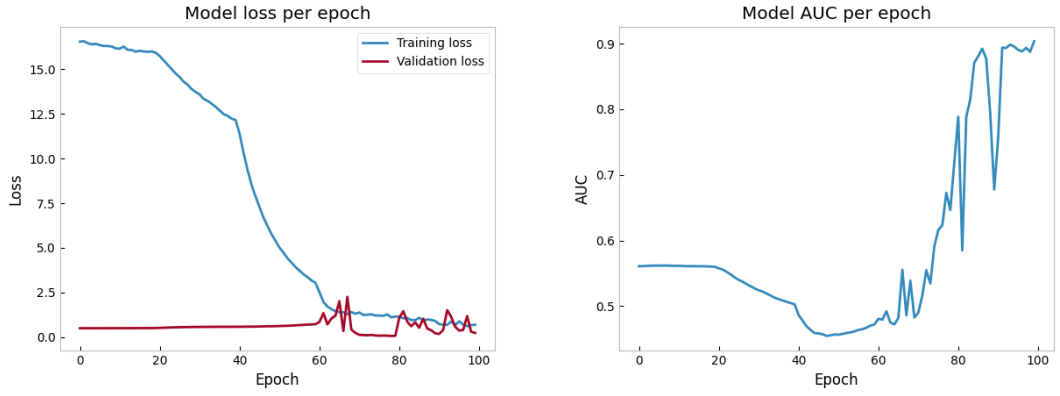
This test consists of tuning the parameters: learning rate lr, eta η, weight decay λ, batch size b, representational dimension of the output space d and filter sets f. As described in Section 5.2.5, the parameters are tested independently using the base configurations.

Learning rate

Figure 5.1a and Figure 5.1b display the model loss and AUC per epoch for the learning rate (lr) annealing test. Looking at the training loss in Figure 5.1a the model is learning slowly during the first 40 epochs before it drops substantially between epoch 40 and 60, where the learning rate is 1e−6. After epoch 60, the training loss drops again before beginning to stabilize at a low number. In Figure 5.1b the AUC starts climbing between epoch 40 and 60. However, the main increase happens between epoch 60 and 80, where the learning rate is 1e−5. This test shows that a potentially optimal learning rate might be around 1e−5 and 1e−6.

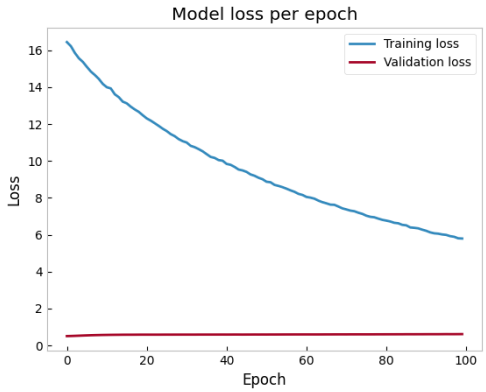
The results of the static learning rate tests are displayed in Table 5.7 and Figure 5.2. The highest AUC score is 89.07%, and is achieved using lr = 1e−5. The lr = 1e−7 and lr = 1e−6 configurations perform poorly, a fact that is also reflected in the loss curves, where the train loss does not drop rapidly enough. The lr = 1e−4 and lr = 1e−3 configurations almost perform comparably to lr = 1e−5, however the training loss curves are non-optimal with a large amount of noise. The lr = 1e−5 configuration is the clear best performing configuration.

The last two entries in Table 5.7 display the results of the two-phase learning rate tests. The test where lr begins with lr = 1e−5 and ends with lr = 1e−6 performed the worst of all learning rate tests. The test that starts with lr = 1e−4 performed better; however still lower than three of the static learning rate tests. Thus, the static configuration of lr = 1e−5 was employed for the semi-supervised setting tests in Section 5.3.3.

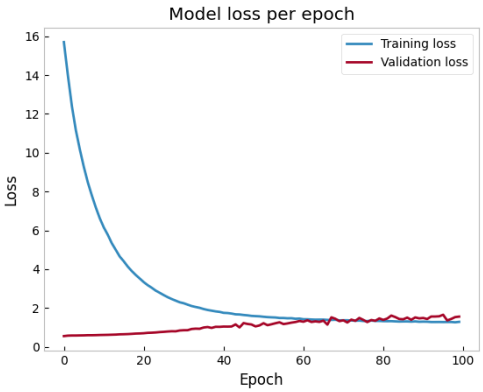


(a) Loss curve (b) Validation AUC curve

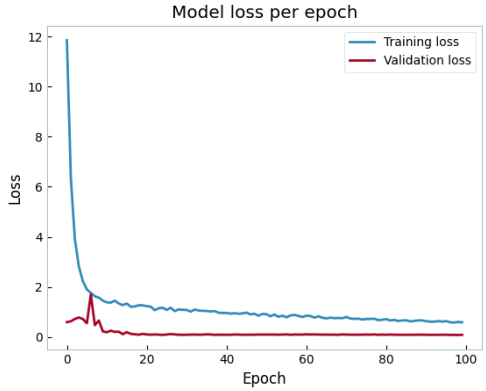
Figure 5.1: Loss curve and AUC curve per epoch for lr annealing test, with starting learning rate = 1e−8, Γ = 10 and Step = 20



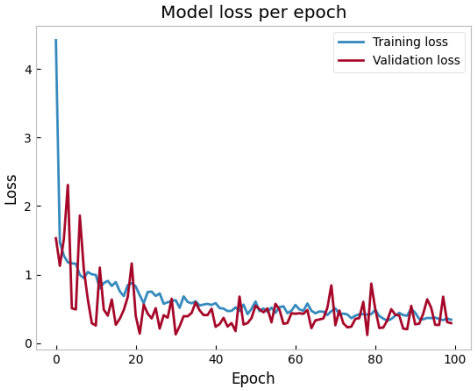
(a) Loss curve for $lr = 1e{-}7$



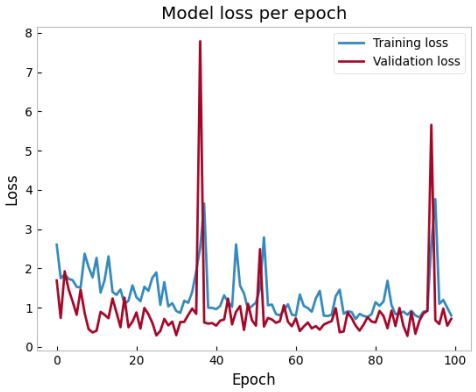
(b) Loss curve for $lr = 1e{-}6$



(c) Loss curve for $lr = 1e{-}5$



(d) Loss curve for $lr = 1e{-}4$



(e) Loss curve for $lr = 1e{-}3$

Figure 5.2: Loss curves for static learning rate tests

Table 5.7: Learning rate results (AUC score in %)

	Starting rate	Γ	Step	Ending rate	AUC
Annealing	1e−8	10	20	1e−4	–
Static *lr*	1e−7	–	–	1e−7	52.14
	1e−6	–	–	1e−6	45.55
	1e−5	–	–	1e−5	**89.07**
	1e−4	–	–	1e−4	86.12
	1e−3	–	–	1e−3	85.57
Two-phase *lr* schedule	1e−4	0.1	50	1e−5	80.34
	1e−5	0.1	50	1e−6	33.18

Eta

Table 5.8 displays the result of testing different values for eta (η). Recall from Section 4.4 that η adjusts the weighing of labeled normal and anomalous data during training. The general trend of the results show that low values for η perform poorly up to $\eta = 10$. The tests run with $\eta \geq 10$ all score close to AUC = 90%, and there does not seem to be any sort of trend. The best performing configuration is $\eta = 500$ with AUC = 91.05%. The second best performing configuration is $\eta = 50$ with AUC = 90.91%. As η is a hyperparameter introduced by Ruff et al. (2020) using values $\eta \in \{10^{-2}, ..., 10^{2}\}$, it may be beneficial to use a value for η within that range for future tests. Therefore, $\eta = 50$ is selected as the value to be used in the semi-supervised setting tests in Section 5.3.3.

Table 5.8: Hyperparameter η results (AUC score in %)

η	AUC
0.1	48.69
1	22.64
5	61.06
10	89.07
20	90.45
30	90.46
40	89.90
50	**90.91**
60	90.56
70	90.71
80	90.22
90	90.58
100	89.83
250	90, 61
500	**91.05**
1000	90.50

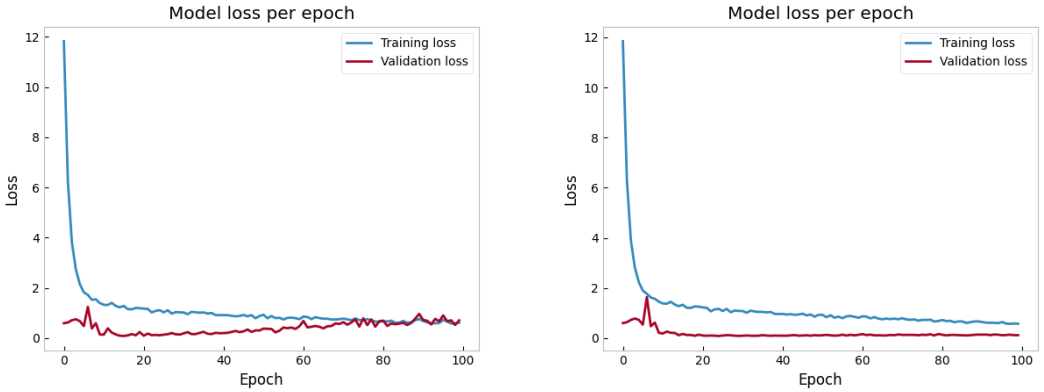
Weight decay

Table 5.9 presents results from the weight decay tests tuning the hyperparameter λ. The best result is achieved using $\lambda = 1e{-}1$ or 0.1, with AUC = 91.77%. All configurations except $\lambda = 1e{-}1$ and $\lambda = 1e{-}2$ score within 0.5% of AUC = 89%, while $\lambda = 1e{-}1$ and $\lambda = 1e{-}2$ perform slightly better.

The aim of tuning λ is to prevent model overfitting. Because of this, it is also important to consider the model loss graph when choosing the better value for lambda. The two best performing values were $\lambda = 1e{-}1$ and $\lambda = 1e{-}2$. While $\lambda = 1e{-}1$ achieved a slightly higher AUC score, its model loss graph presented in Figure 5.3a indicates overfitting. The model loss graph for $\lambda = 1e{-}2$ presented in Figure 5.3b shows a more promising trend, where both training and validation loss decreases in synchronization. Thus, $\lambda = 1e{-}2$ was chosen as value for the semi-supervised setting tests in Section 5.3.3.

Table 5.9: Hyperparameter λ results (AUC score in %)

λ	AUC
1	89.17
1e−1	**91.77**
1e−2	**91.39**
1e−3	88.84
1e−4	89.31
1e−5	89.03
1e−6	89.07
1e−7	88.26



(a) $\lambda = 1e{-}1$ loss curve (b) $\lambda = 1e{-}2$ loss curve

Figure 5.3: Loss curve of λ test, with $\lambda = 1e{-}1$ and $\lambda = 1e{-}2$

Batch size

The four different values tested for batch size (b) are displayed in Table 5.10. The best performing configuration is using $b = 32$ with AUC = 91.70%. The performance drops slightly when decreasing to $b = 16$ or increasing to $b = 64$. Using $b = 128$, comparatively performs extremely poorly. The purpose of increasing the value of b is to utilize more of the computer's capability by having more comments pass through the model at a time. Larger values for b are therefore desirable as they reduce the run time of the experiments. This is reflected in the run times for the different values for b shown in Table 5.10. However, when the value for b becomes too large, the model may be negatively affected. Because the achieved AUC score decreases when increasing b above 32, $b = 32$ was the value chosen for the semi-supervised setting tests in Section 5.3.3.

Table 5.10: Batch size b results (AUC score in % and Run Time in seconds)

b	AUC	Run Time
16	90.88	17 052
32	**91.70**	15 387
64	89.07	14 887
128	37.44	14 552

Representation dimension

The result of testing representation dimension output space (d) is presented in Table 5.11. The best results are achieved using $d \leq 32$. The best performing overall is $d = 16$ with AUC = 90.16%. The tests using $d > 32$ perform very poorly. Thus, $d = 16$ was chosen as value for the semi-supervised setting tests in Section 5.3.3.

Table 5.11: Representation dimension d output space results (AUC score in %)

d	AUC
16	**90.16**
32	89.07
64	39.71
128	25.37
256	56.53
512	52.68
1024	53.50

Filter set

Table 5.12 displays the results when using different filter sets (f). The best result is achieved using $f = \{1, 2, 3, 4\}$ with AUC = 89.41%. There is a slight decrease in performance when increasing to $f = \{1, 2, 3, 4, 5\}$ and another decrease when using $f = \{1, 2, 3, 4, 5, 6, 7, 8\}$. The other configurations have a more substantial decrease in performance. Thus, $f = \{1, 2, 3, 4\}$ was chosen as value for the semi-supervised setting tests in Section 5.3.3.

Table 5.12: Filter sets f results (AUC score in %)

f	AUC
$\{1, 2, 3\}$	80.14
$\{1, 2, 3, 4\}$	**89.41**
$\{1, 2, 3, 4, 5\}$	89.07
$\{1, 2, 3, 4, 5, 6\}$	83.92
$\{1, 2, 3, 4, 5, 6, 7\}$	82.52
$\{1, 2, 3, 4, 5, 6, 7, 8\}$	88.42

5.3.3 Phase 3 - Semi-Supervised Setting

As described in Section 5.2.6, this third and final phase of tests is performed using the optimal configurations from the results of the preceeding phases. These optimal configurations are: $AC = \{4, 5\}$, $lr = 1e{-}5$, $\eta = 50$, $\lambda = 1e{-}2$, $b = 32$, $d = 16$,$f = \{1, 2, 3, 4\}$.

Table 5.13 displays the result of the semi-supervised setting test. The base scenario, providing no labeled data to the model during training, performed with AUC = 42.79%, which is not a viable result. Adding known normal in $\gamma_n = 0.10$ and $\gamma_n = 0.20$ did not significantly improve performance, and all cases performed comparably to random guess. The γ_a tests show that the AUC scores increase when increasing the number of known anomalies. This is the case for both $\{4, 5\}$ and $\{3, 4, 5\}$, where the AUC is strictly increasing from the lowest γ_a to the highest. The same trend is not apparent from increasing the amount of known normal data. This can not be viewed in the γ_n tests as they have poor results, but rather in the γ_l tests. When considering AC = $\{4, 5\}$, only 1.84% anomaly data is available. Because of this, the tests $\gamma_l \in \{0.02, 0.04, 0.06, 0.10\}$ all use the same amount of known anomaly data but an increasing amount of known normal data. These tests perform similarly, and all fall within a range of about 0.5% variation, with no definitive increasing trend. Going from $\gamma_l = 0.06$ to $\gamma_l = 0.10$ when using AC = $\{3, 4, 5\}$ also only increases the amount of known normal data and achieves comparable results.

The best result for both $\{4, 5\}$ and $\{3, 4, 5\}$ are found when only adding known anomalies and no known normal data. However, the performance difference between the γ_a tests and the γ_l test is minuscule. The highest AUC score is achieved using AC=$\{4, 5\}$ and $\gamma_a = 0.02$, which is using all available known anomalies and no

known normal with AUC = 92.94%. The best-performing score using AC=$\{3, 4, 5\}$ is also achieved using all available known anomalies and no known normal with AUC = 88.36%.

Table 5.13: Semi-supervised setting results using AUC (in %)

	B	γ_n		γ_a				γ_l				
AC	0.00	0.10	0.20	0.01	0.02	0.04	0.06	0.01	0.02	0.04	0.06	0.10
$\{3, 4, 5\}$				82.97	86.96	87.89	**88.36**	83.40	85.54	87.02	87.91	87.86
$\{4, 5\}$				90.77	**92.94**			90.68	92.66	92.75	92.24	92.23
None	42.79	56.78	55.40									

5.4 Summary

In this chapter, the experiment plan and setup were presented, along with the results of all the tests. Below is a summary of the experiment, providing an overview of the main results of the testing.

Initially, in phase 1, the model was tested using BERT embeddings with different configurations for the anomaly class. The best performing configuration was having $AC = \{4, 5\}$, achieving an AUC score of 89.07%. In phase 2, a set of hyperparameters were tuned and tested in parallel.

For learning rate (lr), the best performing configuration used a static lr with the value $lr = 1e-5$. This configuration achieved an AUC score of 89.07% and showed no sign of overfitting on the loss curve.

When testing the hyperparameter eta (η), the configurations $\eta = 50$ and $\eta = 500$ achieved the highesd AUC scores of 90.91% and 91.05%, respectively. $\eta = 50$ was consideres the preferred configuration as it falls within the range of tested values in Ruff et al. (2020), in which η was first introduced.

Weight decay was also adjusted and tested by tuning the hyperparameter λ. The two best configurations when considering AUC scores were λ = 1e−1 and λ = 1e−2, yielding 91.77% and 91.39%, respectively. However, when inspecting the loss graphs, λ = 1e−1 showed signs of overfitting, making λ = 1e−2 the preferrable configuration.

When testing different batch sizes (b), the best performing configuration in terms of AUC score was $b = 32$, achieving a score of 91.70%. Increasing b above 32 resulted in lower run times but revealed tendencies of decreasing AUC scores. Thus, $b = 32$ was chosen as the best configuration.

The best performing configuration of representation dimension output space (d) was $d = 16$, achieving an AUC score of 90.16%. When d increased above 64, the model performed very poorly.

The best performing filter set (f) was $f = \{1, 2, 3, 4\}$, achieving an AUC score of 89%. A slight decrease in AUC scores was detected when adding more numbers to the set.

In phase 3, different semi-supervised settings were tested. This final phase of testing was performed using the optimal configurations discovered in previous phases. The best performing configurations when $AC = \{3, 4, 5\}$ was achieved when $(\gamma_n, \gamma_a) = (0.00, 0.06)$. For these configurations, the resulting AUC score was 88.36%. The best performing configurations when $AC = \{4, 5\}$ was achieved when $(\gamma_n, \gamma_a) = (0.00, 0.02)$. This was the best performing configuration out of all the tests, achieving an AUC score of 92.94%.

The main findings of the experiment are the substantial outperformance compared to ADAHS, the optimized configuration found in phase 2, and the high impact of adding known anomaly data versus the insignificant impact of adding known normal data in phase 3.

CHAPTER 6

Evaluation and Discussion

This chapter consists of two sections, the first directly evaluating the results presented in Section 5.3. The second section consists of the proposed solution's general discussion, including its advantages, challenges, and potential improvements, before revisiting the research questions.

6.1 Evaluation

In this section, the results of each phase of the experiments presented in Section 5.3 are discussed. First, the BERT embedding phase is evaluated and compared to the results of Jensen (2020). Then the experiments on tuning the hyperparameters are discussed by analyzing the AUC scores and their trends and other interesting factors such as the training/validation loss and computation time. The semi-supervised setting tests are then evaluated by describing the discovered trends and explaining their significance. Lastly, an overall evaluation is made considering all of the tests together.

6.1.1 Phase 1 - BERT Embeddings

The first test was conducted to try out the BERT embedding implementation using the default hyperparameters from ADAHS. The goal of the test was to find out the immediate viability and briefly compare it to the results of ADAHS using the Norwegian dataset. All three splits of anomaly classes for BSSAD immediately outperformed the best result of ADAHS. BSSAD outperforming ADAHS was an expected result as BERT is a more sophisticated word embedding than fastText, used in ADAHS. The most significant difference is that BERT interprets the context within texts and handles OOV (Out Of Vocabulary) words and terms. It was not entirely expected that the BSSAD model would perform at this level using the

hyperparameters from ADAHS. The word embeddings of BSSAD are much larger and more complex than ADAHS. This could imply that the model requiring more finely tuned hyperparameters to reach its full potential. It is also important to note that the hyperparameters chosen in ADAHS were not thoroughly explored but rather chosen as an acceptable level for the tests employed by Jensen (2020). As opposed to ADAHS, BSSAD performed better using $\{4, 5\}$ as known anomaly classes rather than $\{3, 4, 5\}$. For BSSAD every test using AC $= \{4, 5\}$ outperformed the ones using AC $= \{3, 4, 5\}$, while the opposite is true for ADAHS. This could be explained by fastText not being as sophisticated as BERT and needing more anomalous samples to perform well.

An explanation for AC $= \{4, 5\}$ being better is that classes 4 and 5 are the actual hateful classes in the dataset, while class 3 is not. In addition, class 3 outnumbers class 4 and 5, which causes the model to be trained more towards offensive texts than actual hateful texts.

In addition to the two sets of known anomaly classes used by ADAHS, this test included an experiment having AC $= \{2, 3, 4, 5\}$. This configuration performed worse than using AC $= \{3, 4, 5\}$ and AC $= \{4, 5\}$, but still better than ADAHS. Running the experiment with AC $= \{2, 3, 4, 5\}$ was not expected to provide the best performance but was rather conducted to check how it compares to the other two. Category 2 is likely too similar to the normal class, which in this case only consists of category 1, whereas anomalies by nature are supposed to be deviating. On the other hand, using only category 5 as anomaly class is ineffective as the number of comments in category 5 is insufficient.

6.1.2 Phase 2 - Hyperparameter Tuning

Learning rate

The learning rate (lr) test was conducted in three separate parts with results displayed in Table 5.7, Figure 5.1 and Figure 5.2. The first part involved performing an annealing test to find a rough estimate for an optimal learning rate value. The initial observation was that the training loss dropped the most when using $lr = 1e{-}6$ and $lr = 1e{-}5$, where the latter was the initial value used for the previous test. The next part of the learning rate test was to use static values for lr, which immediately showed that using $lr = 1e{-}6$ was not a desirable configuration. The reason for this can be observed in Figure 5.2b, where the graph for training loss is not steep enough, meaning the model is learning too slowly and therefore achieves a poor AUC score.

Observing Figure 5.2, a clear trend can be seen from $lr = 1e{-}7$ to $lr = 1e{-}3$. $lr = 1e{-}7$ shows a slowly decreasing training loss curve which does not start to converge toward the end. $lr = 1e{-}6$ displays a more promising curve, which is then even further improved when using $lr = 1e{-}5$. When increasing beyond to $lr = 1e{-}4$, training loss has a much lower initial value and introduces more noise.

This means that the model may be overfitted, meaning it specializes excessively on the given training data. Using $lr = 1e{-}3$, the resulting training loss curve is even noisier and barely shows a decrease over the range of epochs.

Table 5.7 displays the AUC score for the static learning rate tests. In addition to having the best training loss curve, using $lr = 1e{-}5$ also has the best AUC score. Furthermore, the results show that increasing the lr causes small performance decreases compared to decreasing the lr, where the AUC score plummets. From this test, there is further potential for investigating lr values between $1e{-}6$ and $1e{-}4$.

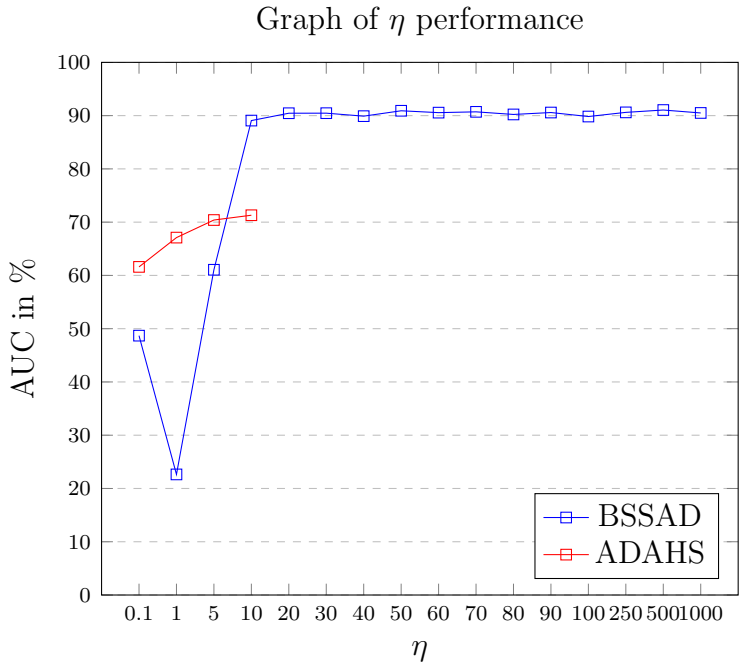
The third part of the learning rate test was to use one learning rate for the first 50 epochs and then decreasing it for the last 50 epochs. This was only explored in two configurations, where one yielded an unusable result and the other performed worse than the two learning rates did statically. However, this approach may show potential with more appropriately selected starting and ending learning rate values.

Eta

The hyperparameter eta (η) was tested using the values presented in Section 5.2.5. This parameter is different from many of the others as it is created by Ruff et al. (2020) and its range and growth are not clearly defined. The largest differences in performance can be observed in the range from $\eta = 0.1$ to $\eta = 10$ in Table 5.8. Using $\eta = 0.1$ results in AUC = 48.69% which is slightly worse than random guess. When η is less than 1, added known normal training data is weighed more than the added known anomaly data. When the model weighs the normal and anomaly data equally at $\eta = 1$, the model performs with AUC = 22.64%. It is unclear what caused the AUC score to be so low, but it still shows that the configuration is unsatisfactory. At $\eta = 5$, the result is AUC = 61.06%, which begins to show some potential. When using $\eta = 10$, the performance increases substantially to AUC = 89.07% and seems to start stabilizing around this performance as η increases. This trend can be observed in Figure 6.1. As mentioned in Section 5.3.2, the configuration chosen as the best result was $\eta = 50$ even though $\eta = 500$ scored slightly higher. This can be further justified by observing the graph as there is no clear trend for one configuration to be the undisputed best.

From this test, it is made clear that adding known anomaly data is significantly more impactful towards improving performance than adding known normal data. Jensen (2020) also conducted a test with the η parameter with the values $\eta \in \{0.1, 1, 5, 10\}$. Their results are quite different when looking at the same configuration for BSSAD and can be viewed alongside each other in Figure 6.1. ADAHS has a clear trend of increasing AUC score when increasing η where all of the scores are significantly larger than 50, while BSSAD does not score above AUC = 50% before η is increased to 5. This indicates that ADAHS makes better use of added normal data than BSSAD even though it does not perform as well as BSSAD when using higher values for η. The effects of adding known anomaly data versus adding known normal data are further discussed in Section 6.1.3.

Figure 6.1: Graph of different configurations of η by AUC score (in %) for ADAHS by Jensen (2020) and BSSAD



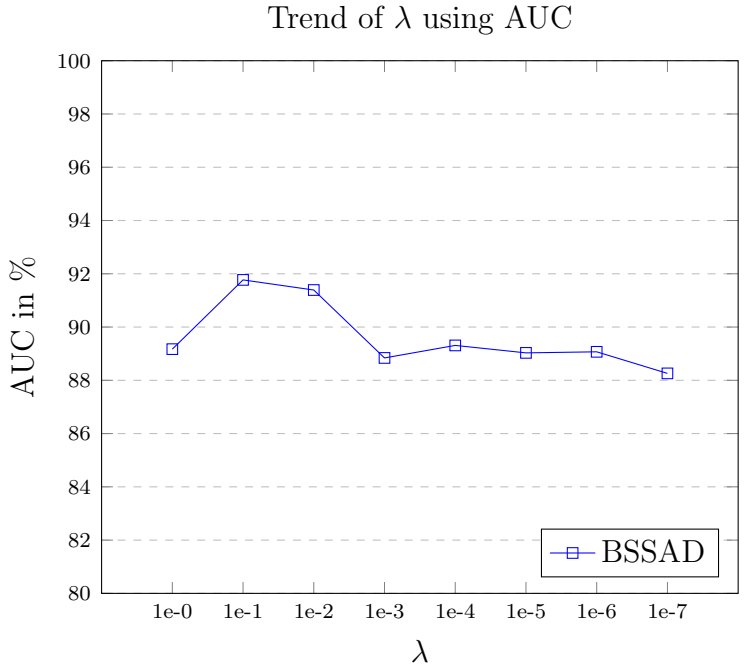
Weight decay

Weight decay was tested using different values for λ found in Table 5.4. As explained in Section 4.4, weight decay is important to optimize in order to prevent overfitting. Keeping his in mind, the AUC score alone is not a good metric to evaluate the performance when employing different values for λ. Instead, the training and validation loss of the model should be included as an additional part of the evaluation process.

When considering AUC scores only, displayed in Figure 6.2, all the tests ran performed reasonably well, with most variations achieving AUC between 88.17% and 89.17%. A slight trend can be observed in that the two best scoring configurations lie within proximity of each other, namely $\lambda = 1e{-}1$ and $\lambda = 1e{-}2$. These configurations achieved AUC scores of 91.77% and 91.39%, respectively, indicating that $\lambda = 1e{-}1$ is the best performing option. However, when inspecting the model loss graphs for the configurations, as seen in Figure 5.3a and Figure 5.3b, it becomes apparent that $\lambda = 1e{-}1$ shows signs of overfitting. This can be recognized by the fact that while training loss decreases, validation loss increases, suggesting the model is specializing too much on the training data.

In contrast, $\lambda = 1e{-}2$ produces no such indications. It was for this reason that $\lambda = 1e{-}2$ was the chosen value for phase 3 of the tests. Nevertheless, inspecting Figure 5.3b, the argument could be made that neither training loss nor validation loss decreases sufficiently for continued training to create a meaningful impact on performance in later stages of the experiment. Indeed, while training loss undergoes

Figure 6.2: Graph of different configurations of λ by AUC score (in %)



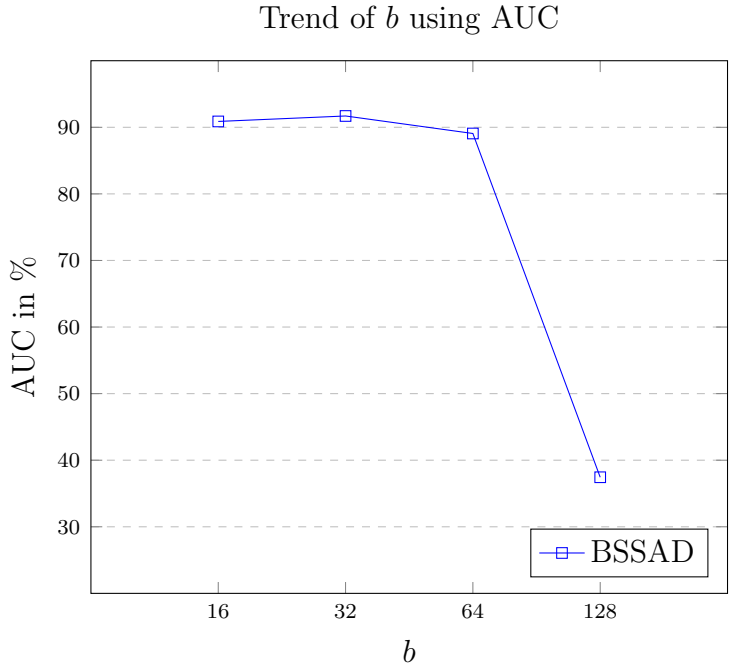
a slight, consecutive decrease towards the end, validation loss seems to remain fairly constant after 20 epochs. Therefore, early stopping could be considered to avoid unnecessarily long training, possibly with little to no impact on the model's performance.

Batch size

The value for batch size (b) dictates how many comments are processed by the network in a single pass-through. As described in Section 4.4, b can therefore be used to shorten the run time of the experiments. However, setting b too high might negatively affect the model's performance. Because of this, it was essential to consider both AUC scores and run times for the experiments when evaluating the b alternatives.

When considering run times, a clear tendency appears from the results seen in Table 5.10. As batch size increases, the time it takes for running through all 100 epochs decreases consistently. This is expected as larger batch sizes allow for higher utilization of hardware. However, the decrease in run time is relatively small, saving on average 625 seconds for each doubling of b. Moreover, the impact decreases for higher values of b. Doubling b from 16 to 32 shortens run time by 1665 seconds, which translates to 27 minutes and 45 seconds, whereas doubling b from 64 to 128 results in a difference of 335 seconds, or 5 minutes and 35 seconds. More importantly, none of the improvements bear an overwhelming impact compared to the average total run time, which for the 4 runs is 15 469.5 seconds, or 4 hours, 17 minutes, and 49.5 seconds.

Figure 6.3: Graph of different configurations of b by AUC score (in %)

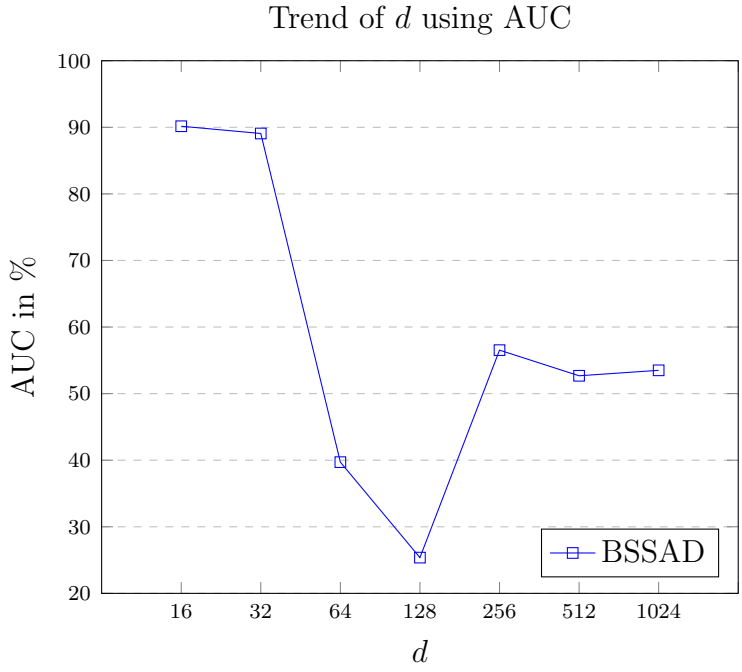


The AUC scores in Figure 6.3 show less of a trend when compared to the run times. When $b \in \{16, 23, 64\}$, the model performs similarly, achieving and AUC scores of 90.88%, 91.70% and 89.07%, respectively. However, when $b = 128$, the model's performance decreases drastically, yielding an AUC score of 37.44%. This suggests that a limit exists for the value of bath size between 64 and 128, where the model's performance rapidly decreases. When choosing which value to use in the third phase of testing, it was decided that $b = 32$ was the best option as it provided an appropriate balance between model performance and run time. While $b = 64$ has improved run time, its difference of 500 seconds was considered less important than the AUC score decrease of 2.63% compared to the chosen value.

Representation dimension

The representation dimension (d) dictates the size of the output vector of the model. An optimal value for d makes the output vector complex enough to contain important information while still being simple enough to avoid obscurities and noise. Moreover, a large output vector dimension will result in a larger memory load, and a higher computational cost (Goodfellow et al., 2016). However, the difference in computational cost for the different values of d does not seem consequential as the computation time per epoch is seemingly unchanged for all values. The results of the tuning of d are represented in Figure 6.4 where $d = 16$ and $d = 32$ are the clear best results. The AUC score of $d = 16$ is slightly higher. However, they are very similar to the rest of the scores. There seems to be some threshold between $d = 32$ and $d = 64$ where the model can no longer learn anything of value. Jensen (2020) tested $d = 32$ and $d = 300$, where $d = 300$ was able to achieve comparable

Figure 6.4: Graph of different configurations of d by AUC score (in %)



results to $d = 32$ after many more epochs. The same effect does not happen with values above $d = 32$ for BSSAD, as none of the graphs portraying validation AUC per epoch have an upward trend towards the end of the 100 epochs, as shown in Appendix A. There is still much room for experimenting with different values for d around the values 16 and 32 to gain potential performance increases.

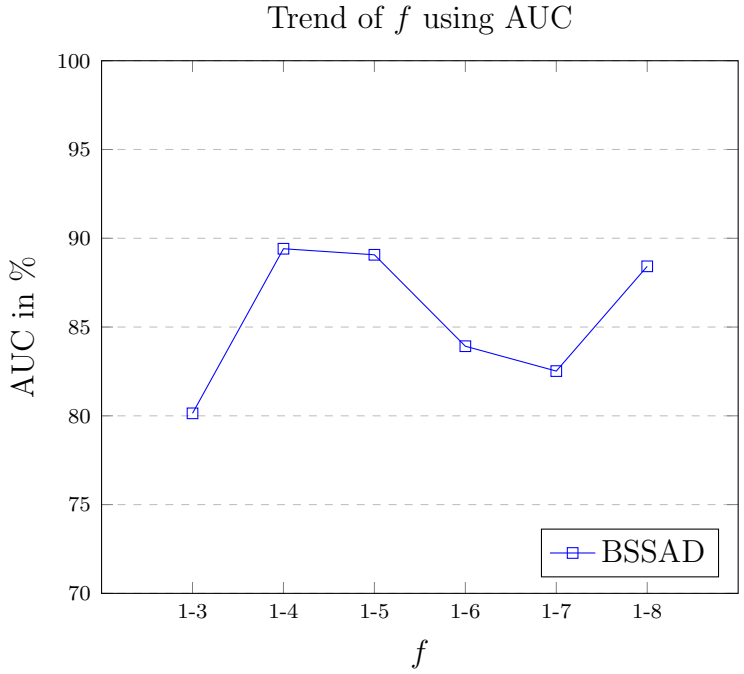
Filter set

As explained in Section 4.3, the filter set (f) represents the number of consecutive words, or n-grams, that are being processed by the filters in the convolutional layer of the model. Increasing f produces filters that can capture greater context, but could also lead to the detection of obscure patterns and has a higher computational cost. Conversely, too small n-grams might lose contextual meaning as they only contain shallow information.

The computational cost of increasing the size of the filter set has a noticeable increase in the computation time of training the model. In Table 6.1 the average computation time of all epochs are displayed for all configurations of f. There is an average increase of 18.17 seconds per epoch from $f = \{1, 2, 3\}$ to $f = \{1, 2, 3, 4, 5, 6, 7, 8\}$. This is expected behavior, as adding higher values to f means that new, larger filters are employed in the convolution process in addition to the current ones.

The resulting AUC scores of the f tests are displayed in Figure 6.5. The best score is achieved using $f = \{1, 2, 3, 4\}$, and is very slightly worse at $f = \{1, 2, 3, 4, 5\}$.

Figure 6.5: Graph of different configurations of f by AUC score (in %)



Decreasing to $f = \{1, 2, 3\}$ results in the poorest performance, which likely means that there is not enough complexity to this filter set, and it loses some of the contexts of the texts. The models performance decreases from $f = \{1, 2, 3, 4\}$ to $f = \{1, 2, 3, 4, 5, 6, 7\}$, and then unexpectedly improves significantly at $f = \{1, 2, 3, 4, 5, 6, 7, 8\}$. One explanation for this may be that some unique pattern in the dataset matches the 8-gram created by the filter set, which is significant enough to affect the performance. The performance spike at $f = \{1, 2, 3, 4, 5, 6, 7, 8\}$ may indicate that even larger filter sets could be investigated. However, as the computation time increases with the size of f, it is beneficial to select a smaller f, which in this case, there is no compromise to be made as $f = \{1, 2, 3, 4\}$ is the best performing configuration while also being the second smallest one.

Table 6.1: Average time per epoch in seconds for tested values of f

f	Average seconds per epoch	Time for 100 epochs
$\{1, 2, 3\}$	142.98	3:58:18
$\{1, 2, 3, 4\}$	144.33	4:00:33
$\{1, 2, 3, 4, 5\}$	147.59	4:05:59
$\{1, 2, 3, 4, 5, 6\}$	149.90	4:09:50
$\{1, 2, 3, 4, 5, 6, 7\}$	155.46	4:19:06
$\{1, 2, 3, 4, 5, 6, 7, 8\}$	161.15	4:28:35

6.1.3 Phase 3 - Semi-Supervised Setting

The semi-supervised setting test results shown in Table 5.13 show that the inclusion of labeled data in the training phase can have a drastic effect on the model's performance. The lowest scoring configuration was the base scenario, where no labeled data was provided to the model. This resulted in an AUC score of 42.79%, which is comparable to random guess. Interestingly, adding known normal data yielded a slight performance increase; however, this increase was not large enough to be considered impactful. Even when adding as much as 20% normal data, the resulting AUC score of 55.40% is still comparable to random guess. However, adding only 1% of labeled anomalies and no labeled normal data, the performance increased drastically, achieving AUC scores of 82.97% and 92.94% when $AC = \{3, 4, 5\}$ and $AC = \{4, 5\}$, respectively. Moreover, the performance strictly increases when increasing γ_a both for $AC = \{3, 4, 5\}$ and $AC = \{4, 5\}$.

The trend persists for the γ_l tests, where both labeled anomalous and normal data is added. When considering AC $= \{4, 5\}$, the AUC score increases from $\gamma_l = 1\%$ to $\gamma_l = 2\%$. In this step, the number of anomalies increases from 1% to 1.84%. However, because only 1.84% anomalies are available, no further anomalies are added from $\gamma_l = 2\%$ to $\gamma_l = 10\%$. In these steps, only normal samples are added to the labeled data. These tests perform similarly, and all fall within a range of about 0.5% variation, with no definitive increasing trend. The scenario repeats when considering AC $= \{3, 4, 5\}$. In this case, 5.65% anomalies are available, and an increase in AUC scores can be seen from $\gamma_l = 1\%$ to $\gamma_l = 6\%$. However, increasing $\gamma_l = 6\%$ to $\gamma_l = 10\%$, only increasing the amount of labeled normal labeled data, does not result in any significant change in performance.

One reason for the drastic impact of the labeled anomalies could be the value of η. When $\eta = 1$, the model weighs the anomalous and normal data equally. As η increases, the model puts more emphasis on the anomalous data. From the previous testing phase, the value chosen for η is 50, meaning that the model finds the anomalous data to be exceedingly more important than the normal data. This could explain why adding large amounts of normal data has little impact on performance, while adding comparatively small amounts of anomalous data results in significant differences. This is somewhat concerning because the model might miss out on valuable insights regarding normal data that might negatively impact its performance. If time had allowed for it, it would have been prudent to explore the same tests with lower values for η to uncover what impacts could have been made by weighing the labeled normal data to a higher degree.

Overall Evaluation

Overall the tests show a substantial performance increase when compared to the results of ADAHS. The highest AUC score achieved using AC $= \{4, 5\}$ was 75.3% for ADAHS and 92.94% for BSSAD, which translates to an increase of 17.64%. Using AC $= \{3, 4, 5\}$, ADAHS achieved 77.3% and BSSAD achieved 88.36%, which

is an increase of 11.06%.

The hyperparameter tuning phase of the experiment yielded a noticeable increase in performance. All of the tested parameters except for lr resulted in a new value with a higher AUC score. The AUC differences from the original value using base configurations with the new configuration are displayed in Table 6.2. The largest difference was found when increasing b by 2.63%. However, this came with the cost of a longer overall run time. On the opposite end, using the new value for f only yielded an increase of 0.34% and reduced the run time. Before the hyperparameter tuning phase, the model scored an AUC of 89.07% using AC = $\{4, 5\}$. The test run only changing the values of the hyperparameter tuning phase scored an AUC of 92.23% (AC = $\{4, 5\}$ and $\gamma_l = 0.10$ in Table 5.13), which is an overall increase of 3.16%. One possible disadvantage to independently testing all of the hyperparameters was that the new values would be less compatible with each other, resulting in an unsatisfactory performance increase when combined. However, this does not seem to be the case as the increase is greater than the highest individual increase.

It is vital to underline that random factors may cause some slight differences in the AUC score, and it may not always be beneficial to select the configuration with the highest score. Identifying trends in the data was crucial when selecting the best possible values and separating actual performance differences from anomalous values.

Table 6.2: Performance increase of parameter tuning phase

Hyperparameters	Original value	New value	AUC increase
lr	1e−5	1e−5	−
η	10	50	1.84%
λ	1e−6	1e−2	2.32%
b	64	32	2.63%
d	32	16	1.09%
f	$\{1, 2, 3, 4, 5\}$	$\{1, 2, 3, 4\}$	0.34%

6.2 Discussion

This section discusses the results of the experiments concerning the research questions. Firstly by discussing the advantages of the proposed solution and where it was successful, then about the challenges and disadvantages faced, followed by suggestions for overcoming the challenges and other thoughts on how to improve the solution. Finally, the last section directly discusses the research questions and how successful the solution has been in fulfilling them.

6.2.1 Advantages

It becomes apparent from the results of the experiments that the BSSAD solution performs better than the preceding ADAHS solution when applied to the same Norwegian dataset. The best performing configurations, as presented in Chapter 5, achieved an AUC score of 92.94%. This is a significant improvement of the results presented by Jensen (2020), achieving an AUC score of 75.3%.

The BSSAD solution is an iteration of ADAHS. The main difference between the two is the inclusion of BERT embeddings. Therefore, it would seem like having more sophisticated embeddings that can capture context and handle OOV words has a considerable impact on hate speech detection. This is not surprising, as hate speech detection is a complex task, even for humans. As presented in Section 2.1, hate speech is often context dependant and might hide in what at first glance appears as a well constructed, grammatically correct sentence. In these situations, it is helpful not to produce word embeddings by having predefined values for each token but instead produce them on a per-occurrence basis, taking into account the context of the surrounding text. While this can be implemented by employing n-grams, BERT improves upon this by producing context-specific word embeddings that might differ for two identical tokens depending on the sentence.

Moreover, a common technique to avoid automatic detection of hateful content is to create slang or replacement terms that resemble or symbolize offensive ones, which simple filterings like blacklists could easily flag. In previous works, these OOV words are usually handled by employing simple solutions. One such solution is to ignore OOV words, as employed by Bojanowski et al. (2017) and Mitra et al. (2016). Jensen (2020) handle OOV words by initializing them as 0-vectors with the same dimensionality as the other word embeddings. Another approach still is randomizing the word vectors or creating ones as a function of the rest of the vectors, for instance, using the mean values. A drawback of such simple solutions is the loss of important information. Ignoring the words and utilizing 0-vectors involves losing potentially meaningful word embeddings. Furthermore, randomizing and averaging the vector might make the model believe that the word conveys some false meaning. If the generated word vector is similar to other word vectors in the vocabulary, the model might falsely believe that the two words carry similar meanings. Comparatively, it is advantageous to assign a word embedding to this OOV word based on the surrounding text, as is the technique employed by

BERT.

The BSSAD solution is so far tested only using the Norwegian dataset presented by Jensen (2020) and Andreassen Svanes and Gunstad (2020). The dataset contains 41 139 labeled comments, where only 759 of these are in categories 4 and 5, which are the moderately hateful and hateful categories. Compared to available datasets used for hate speech detection in English, the Norwegian dataset contains drastically fewer comments. For instance, the English *Jigsaw* dataset from a *Toxic Comment Classification Challenge* available from Kaggle [1] as used by Jensen (2020) contains more than 220 000 data points. When training a model, an abundance of data is desirable, and having too little available is a challenge that can prevent good performance even for well-designed solutions. Indeed, both Jensen (2020), and Andreassen Svanes and Gunstad (2020) mention that ideally, the Norwegian dataset would have more samples and that the limited amount might have negatively affected the performance of the proposed solutions. Jensen (2020) goes one step further, showing that the ADAHS solution experiences a substantial increase in performance of about 20% in AUC score when using the *Jigsaw* dataset. Therefore, it is a considerable advantage of the BSSAD solution that it achieves AUC scores in the 90% range using the Norwegian dataset, highlighting the model's ability to perform well even when trained using smaller datasets. This ability suggests that the BSSAD solution is applicable to hate speech detection in other less-used languages in addition to Norwegian. A common issue for hate speech detection in less-used languages such as Norwegian is the lack of large, available datasets appropriate for hate speech detection. Because of this, solutions such as BSSAD that can learn well from smaller datasets have a large advantage and are sought out for such types of problems. However, the BSSAD solution relies on pre-trained BERT models being available for the language in question. While creating such models requires large sets of corpora, the models can be created on less domain-specific data. Thus, it presumably requires less effort to provide datasets that can facilitate the creation of pre-trained BERT models than to create large, high-quality datasets for hate speech detection.

In the experiments presented in this thesis, a set of hyperparameters and configurations were adjusted to improve performance. Looking at the results presented in Chapter 5, it becomes apparent that tuning the chosen set of parameters had a significant impact on the model. Moreover, the results of the experiments reveal valuable trends and provide insights that can be used for further exploration. For instance, the results show that values for lr show potential around 1e$-$5, and that further testing in the proximity of this value might provide even better results. Furthermore, it has been discovered that the model performs best when providing only labeled anomalous data, with a strict increase in performance whenever more labeled anomalies are provided. Following this trend, the implication seems to be that, when this solution is employed with the current best-performing configurations, the performance could benefit even further by adding more labeled anomalous data. In such a scenario, a binary labeling technique could be used to label either hateful or non-hateful comments, removing the need for finely categorized datasets

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

that could be more time-consuming to produce.

6.2.2 Challenges

Using BERT word embeddings for the model instead of the previously implemented word embeddings caused the model's training to increase significantly in computation time. Using simpler word embeddings, such as fastText used by Jensen (2020) allows for fast fetching of the word embeddings using a stored dictionary. When using BERT, each token does not have a corresponding word embedding, which requires the word embeddings to be created from the given dataset. In addition to this, the BERT embedding vectors are also considerably larger than the fastText embedding vectors. Due to these factors, the embedding of the dataset could not be done previous to the testing as it would overflow the amount of available memory. Instead, the word embeddings were created iteratively for each batch. As a consequence of the increased computation time, the number of hyperparameter values to be tested was constricted.

Originally it was desired to use a more sophisticated method of hyperparameter tuning such as grid search. However, the long computation time immediately excluded that as a possibility. Grid search generally suffers from the curse of dimensionality, where the amount of required tests grows exponentially with the number of parameters. Running grid search with the values used in the hyperparameter testing phase in Section 5.2.5 with all possible combinations would require $16 \times 8 \times 4 \times 7 \times 6 = 21\,504$ tests. This is an extreme example, and the parameter values would be selected more appropriately in an actual test. Additionally, parameters that are independent of each other should not be included. Nevertheless, the example reveals the increased cost of using a method such as grid search.

In addition to testing the parameters comparatively, it would be favorable to test each value more thoroughly. As mentioned in Section 6.1.3, the AUC scores of the tests were susceptible to random factors such as the data distribution. This made it difficult to accurately analyze the test results and distinguish what values performed better than others. One common way of accounting for this is to use k-fold cross-validation, where each test is run k times with different data distributions. This method also had to be discarded because of time constraints, as the total computation time would be increased k times.

The results of testing different distributions of added labeled data to the model clearly show that adding labeled anomalies is beneficial. However, there is no evidence of increased performance for including labeled normal data. This was first discovered by testing the η hyperparameter, where low values that weigh the added normal data more importantly performed poorly. This does not coincide with the discoveries of Jensen (2020) and Ruff et al. (2020) which also experimented with the value of η. The former concludes that adding labeled anomalies is more useful than adding labeled normal data; however, the difference is between the two types is not as severe. The latter found that using an even split of labeled normal and anomaly data performed the best. However, this is using several data

types, not just text. The BSSAD model builds upon the Deep SAD model by Ruff et al. (2020) which assumes that "the entropy of the latent distribution for normal data should be lower than the entropy of the anomalous distribution," and that may be a cause for the model not learning as well from the normal data. When considering text as a data type, there is inherently a large amount of entropy, and it can be impossible to create a complete model of the normal data. As the BERT embedding vectors are larger than the ones used by Jensen (2020) and include additional information such as context and out-of-vocabulary words, the entropy is even further increased. Even though Jensen (2020) performed better with low η values, their tests using different distributions of added labeled data showed no improvements by adding normal data. The same result was found in this thesis as discussed in Section 6.1.3. The cause of the models not increasing in performance when adding labeled normal data may have been caused by the high value of η that was set for both models. The high value of η may have caused the model to ignore the contribution of normal data. Running the same test discussed in Section 6.1.3, however, with a lower value for η could provide further insight into whether adding labeled normal data is truly inconsequential to the performance.

6.2.3 Improvements

The BSSAD model has overall had great results, improving in performance compared to previous implementations. The hyperparameter tuning phase led to further performance increases; however, as mentioned above, there were some challenges. The main challenge discussed has to do with the computation time, which constricted the number of tests that could be run. As creating word embeddings from the tokens is the most time-consuming part of the experiments, it would be beneficial to find a way to store and reuse the embeddings between experiments. Other factors may also help reduce the computation time, such as using a smaller filter set, increasing the batch size, or reducing the number of epochs. However, that may come at the cost of the overall performance. Many tests show that the model is adequately trained after approximately 20 epochs. Therefore, either simply reducing the total number of epochs or implementing an early stopping method for when the model is adequately trained could significantly reduce the run time. Reduced computation time would allow for further and more sophisticated tests to be run for the hyperparameters, such as grid search to find a better overall configuration for the model or k-fold cross-validation to validate the model's performance more accurately.

In this thesis, the BSSAD model has only been tested using one dataset and only directly compared to the results from Jensen (2020). One logical next step would be to apply the BSSAD model to other, more widely used datasets in English and possibly other languages. This would allow for further comparison of the results to other state-of-the-art solutions and further anomaly detection as a viable method to perform hate speech detection.

6.2.4 Research Questions

The following section revisits the research questions presented in the problem specification in Section 1.2. The overall research question for this thesis is:

> "How can existing approaches for hate speech detection in languages other than English be improved?"

The research question is split into three sub-questions, and how the thesis has attempted to answer each one is discussed.

RQ1 *How can recently developed techniques in the field, such as BERT, be integrated to provide state-of-the-art results?*

Most previous works rely on classification methods. However, classification has some disadvantages when it comes to hate speech detection. The main one is that of dataset imbalance which is inherent for hate speech. A suggested approach for solving issues related to dataset imbalance is using anomaly detection. Anomaly detection has the advantage of relying on the imbalance between normal data and anomaly data, which in this case is hateful content. Another factor to consider is the evolution of language, where elements such as vocabulary and topics change over time.

Traditional classification methods rely on defining a set of classes in which to categorize data. This classification inherently assumes some form of similarity between the entries within a given class. However, this assumption is not optimal for hate speech detection due to the constant change in abusive language to avoid detection, as stated by Nobata et al. (2016). As language evolves, instances may no longer fit within the boundaries of the defined classes. Anomaly detection assumes no similarity between the anomalous data points, making it possible to identify even evolved forms of hate speech.

Jensen (2020) presents ADAHS as an approach to hate speech detection using anomaly detection. The approach is tested on a Norwegian and an English dataset. The English dataset is substantially larger with 223 549 comments compared to the Norwegian dataset with 41 139 comments. The model performed well on the English dataset and acceptably on the Norwegian dataset. The goal of ADAHS was to test the viability of anomaly detection as an approach to hate speech detection, not to find the best performing configuration. An opportunity was discovered to build upon the ADAHS solution to explore its potential when optimized further. The main focus for optimization was to change the way the model interprets the data. The ADAHS model utilized static word embedding methods. BERT was discovered as a comparatively advanced and superior tool for creating more valuable word embeddings, and recently, pre-trained BERT models were made publicly available for the Norwegian language. Combining the approach of using anomaly

detection for hate speech detection with the state-of-the-art BERT method laid the foundation for the solution proposed in this thesis.

RQ2 *How can the performance be improved for approaches using smaller datasets?*

As laid out previously, non-English hate speech detection does not have adequate datasets available. It was decided to focus the scope of the thesis to Norwegian and use the dataset developed by Jensen (2020) and Andreassen Svanes and Gunstad (2020). While the created dataset represents a decent foundation, it is of inadequate size when compared to English datasets. One possible means for progression for this thesis was to expand upon the dataset. Expanding the dataset was deemed a too time-consuming task and would impede other goals of the thesis. Instead, it was decided to approach the problem differently by looking at ways to improve performance on smaller datasets in general. This approach could lead to discoveries that would benefit languages other than Norwegian, which also face the challenge of small datasets. To improve performance on smaller datasets, the available data had to be used more effectively. This thesis builds upon the previous approach of ADAHS, which uses fastText word embeddings that do not utilize important factors such as context within a sentence and misspellings. BERT embeddings include these factors and are, therefore, able to acquire more valuable information from the same amount of text. The results in phase one of the experiment in Section 5.3 show a substantial increase in performance when incorporating BERT into the model compared to the results of ADAHS, both using the same dataset. To further investigate the hypothesis of BERT increasing the performance on small datasets, it would be beneficial to test the model on other small datasets. However, this was considered beyond the scope of this thesis.

RQ3 *How can we determine what factors affect the results?*

In addition to combining state-of-the-art techniques to improve performance, there are more specific factors to consider to optimize the solution. Using the default configurations of the selected techniques may not be entirely compatible with each other. In addition, the configuration of ADAHS was only optimized to achieve acceptable performance by testing only a few values for a small set of parameters. Therefore, reviewing the factors that may affect the results was essential to attain optimal performance. The main focus of optimization was on the model's hyperparameters and the semi-supervised setting. When determining what hyperparameters to test and the range of values, the most important resource was previous works. A plan was made selecting values from previous work in addition to auxiliary values for further exploration. The experiment discovered how the different hyperparameters affected the results. For all hyperparameters except for one, a better performing configuration was found through testing, and the overall performance increased when combining the best configurations. The goal was not solely to improve the performance but also to discover trends and the general effects of the hyperparameters on the model. Some had significant effects on the model's

performance; others were more important for the sake of computation time. Finally, it was discovered how the distribution of added known normal and anomaly data affected the model.

CHAPTER 7

Conclusion and Future Work

The following chapter provides a conclusion to the thesis by discussing the research and work conducted to improve hate speech detection in non-English languages. Furthermore, the chapter provides possible improvements and future work relevant to continue research on this thesis.

7.1 Conclusion

Hate speech in social media is an important topic and can have a large negative impact on individuals or groups. Machine learning techniques for detecting hate speech have become a rapidly growing field of research in recent years. However, most of the research is conducted with the English language in mind. This thesis has the goal of improving hate speech detection in non-English languages. As a proxy for non-English languages in general, Norwegian was selected to improve hate speech detection methods. The approach chosen for the thesis was to extend upon existing solutions, and the selected solution to improve was one called ADAHS presented by Jensen (2020). To be able to create a substantial improvement, state-of-the-art methods had to be applied. Therefore, the recently developed BERT word embedding technique was implemented. The aforementioned laid the foundation for the novel Bert Semi-Supervised Anomaly Detection (BSSAD) approach laid forth in this thesis.

Using the BSSAD approach, extensive experiments were planned and conducted with multiple goals in mind. The primary goal was to test the overall performance of the new approach and the relative performance increase compared to ADAHS. Another aim was to gain insights regarding the hyperparameters of the model and how to optimize performance. For this, the experiment implemented the tuning of a wide range of hyperparameters. The final goal of the experiments was to explore how providing different amounts of labeled normal or anomaly data during

training would impact performance. For this purpose, the experiments included a phase in which performance was measured for different semi-supervised settings. Throughout the experiments, the Norwegian dataset provided by Jensen (2020) and Andreassen Svanes and Gunstad (2020) was applied.

From the experiments and their results, a number of observations and conclusions can be made. Firstly, the results revealed a substantial increase in performance when compared to the results of ADAHS. This is reflected in the best performing AUC scores for the two approaches, which were 77.3% and 92.9% for ADAHS and BSSAD, respectively. When considering these AUC scores, it is important to note that the same Norwegian dataset was used when achieving both scores. Secondly, the hyperparameter tuning phase successfully improved performance and provided insights regarding how each parameter affects the model's behavior. Moreover, combining the best performing configurations for the hyperparameters yielded a better performing solution overall. Finally, the results show that the proposed BSSAD approach achieves more than satisfactory performance even without extensive datasets.

Ultimately, the contributions of this thesis include a literature review, a novel approach to hate speech detection, and an elaborate experiment with accompanying results and discussions. The literature review provides insight into the field of hate speech detection for both English and non-English languages. The novel approach presented in the thesis, BSSAD, extends upon previous work and includes recently developed state-of-the-art techniques. Finally, the conducted experiment includes a structured plan explaining the goals and presents the results and the discussion of the findings to yield valuable insights regarding the performance of the BSSAD method.

7.2 Future Work

The research on the topics discussed in this thesis is by no means thoroughly explored yet. Hate speech detection is being developed continuously, and there is still room for improvements. The BSSAD model is no exception and shows excellent potential for improvement by further tuning its configurations and applying it to other datasets or other problems. This section will present suggested next steps towards future research using the BSSAD model.

The dataset used in the experiments employed the same cleaning process as ADAHS. However, it might not be advantageous to clean the dataset in the same fashion when using BERT models due to their ability to interpret context. For instance, the cleaning process included the removal of punctuation and concatenation of sentences. This cleaning effectively reduces each text to a single, potentially long sentence with incorrect grammar. Such removal of punctuation is beneficial when using static word embeddings but is potentially unfortunate when employing BERT because the BERT model understands sentence separation. Thus, modifying the preprocessing of the data with BERT in mind could enhance the resulting word

embeddings and, in turn, the performance of the BSSAD model.

The BSSAD approach proposed in this thesis has only been tested on one dataset and one language. A future improvement would be to apply the method to other datasets and in other languages. Testing on well-known datasets such as the one created by Waseem and Hovy (2016) and the Jigsaw dataset would allow for direct comparison of the model's performance compared with multiple state-of-the-art approaches. Using larger datasets would also allow for testing when using variable amounts of data as input. The BSSAD approach could be tested using various sizes of subsets of the complete dataset to determine how the input dataset's size affects performance. Furthermore, it would be pertinent to test the BSSAD approach on other languages. However, this would require a slight modification to the BSSAD approach. Currently, it only supports the pre-trained model for Norwegian Bokmål.

One of the main challenges of this thesis was the increased computation time when using BERT. For further exploration of the model, it would be beneficial to first attempt to reduce the overall computation time. Each batch of data fed to the model has to be converted into word embeddings, which is a time-consuming process. A suggested approach to solving the issue is to create word embeddings for the entire dataset and storing them appropriately. Then, the word embedding could be reused for each run. Each batch would then fetch the relevant texts as word embeddings into memory. Fetching the word embeddings would still require computation time, however, much less than creating them. Another approach to increasing the efficiency is to apply early stopping to the training of the model. Every experiment run with the BSSAD model was set to run for 100 epochs. However, most of the configurations seemed to be sufficiently trained after approximately 20 epochs. Either simply decreasing the total amount of epochs or implementing early stopping techniques would severely reduce computation time.

Reducing the computation time mentioned above would allow for the application of more sophisticated testing methods. The hyperparameter testing experiment performed in this thesis tested all of the selected hyperparameters individually. This approach was not optimal as many of the hyperparameters are interdependent, meaning that changing one could affect the performance of another. This meant that combining the optimal values for each individual hyperparameter may not have led to an overall optimal configuration. A more sophisticated method is grid search, where all the combinations of selected values for the hyperparameters are tested. Grid search would ultimately be able to find a more optimal configuration and reveal the model's true potential.

Another potent next step would be to perform validation on the results. As mentioned previously in this thesis, the results are affected by some arbitrary factors, and small deviations in the AUC scores are difficult to analyze accurately. Using k-fold cross validation would help by training and testing the model configurations on multiple distributions of the dataset. Having more accurate values or plotting the values with standard deviation would allow for a better analysis of the results. Validation could also be improved by using other metrics such as precision, recall, f-score, and confusion matrices.

Bibliography

Andreassen Svanes, M. & Gunstad, T. S. (2020). *Detecting and grading hateful utterances in the norwegian language* (Master's thesis). NTNU.

Athiwaratkun, B., Wilson, A. G. & Anandkumar, A. (2018). Probabilistic fasttext for multi-sense word embeddings. *arXiv preprint arXiv:1806.02901*.

Baeza-Yates, R. & Ribeiro-Neto, B. (2011). *Modern information retrieval: The concepts and technology behind search* (2nd). Addison-Wesley Publishing Company.

Bayes, T. (1763). Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53), 370–418.

Bengfort, B., Bilbro, R. & Ojeda, T. (2018). *Applied text analysis with python: Enabling language-aware data products with machine learning.* " O'Reilly Media, Inc."

Bengio, Y. (2009). *Learning deep architectures for ai*. Now Publishers Inc.

Bird, S., Klein, E. & Loper, E. (2009). *Natural language processing with python: Analyzing text with the natural language toolkit.* " O'Reilly Media, Inc."

Blei, D. M., Ng, A. Y. & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*(Jan), 993–1022.

Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Brown, D. E. & Huntley, C. L. (1992). A practical application of simulated annealing to clustering. *Pattern recognition*, *25*(4), 401–412.

Burkal, R. & Veledar, A. (2018). Hatefulle ytringer i offentlig debatt på nett.

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 & Internet*, *7*(2), 223–242.

Chandola, V., Banerjee, A. & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 1–58.

Collins, M., Schapire, R. E. & Singer, Y. (2002). Logistic regression, adaboost and bregman distances. *Machine Learning*, *48*(1-3), 253–285.

Davidson, T., Warmsley, D., Macy, M. & Weber, I. (2017). Automated hate speech detection and the problem of offensive language. *arXiv preprint arXiv:1703.04009*.

de Gibert, O., Perez, N., Garcıa-Pablos, A. & Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. *arXiv preprint arXiv:1809.04444*.

Del Vigna, F., Cimino, A., Dell'Orletta, F., Petrocchi, M. & Tesconi, M. (2017). Hate me, hate me not: Hate speech detection on facebook. *Proceedings of the First Italian Conference on Cybersecurity (ITASEC17)*, 86–95.

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*.

Dinakar, K., Jones, B., Havasi, C., Lieberman, H. & Picard, R. (2012). Common sense reasoning for detection, prevention, and mitigation of cyberbullying. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *2*(3), 1–30.

Fagni, T., Nizzoli, L., Petrocchi, M. & Tesconi, M. (2019). Six things i hate about you (in italian) and six classification strategies to more and more effectively find them. *ITASEC*.

Fan, W., Miller, M., Stolfo, S., Lee, W. & Chan, P. (2004). Using artificial anomalies to detect unknown and known network intrusions. *Knowledge and Information Systems*, *6*(5), 507–527.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, *27*(8), 861–874.

Gambäck, B. & Sikdar, U. K. (2017). Using convolutional neural networks to classify hate-speech. *Proceedings of the first workshop on abusive language online*, 85–90.

Gitari, N. D., Zuping, Z., Damien, H. & Long, J. (2015). A lexicon-based approach for hate speech detection. *International Journal of Multimedia and Ubiquitous Engineering*, *10*(4), 215–230.

Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Gröndahl, T., Pajola, L., Juuti, M., Conti, M. & Asokan, N. (2018). All you need is" love" evading hate speech detection. *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, 2–12.

Hendrycks, D., Mazeika, M. & Dietterich, T. (2018). Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*.

Hosseinmardi, H., Mattson, S. A., Rafiq, R. I., Han, R., Lv, Q. & Mishra, S. (2015). Detection of cyberbullying incidents on the instagram social network. *arXiv preprint arXiv:1503.03909*.

Isaksen, V. & Gambäck, B. (2020). Using transfer-based language models to detect hateful and offensive language online. *Proceedings of the Fourth Workshop on Online Abuse and Harms*, 16–27.

Jensen, M. H. (2020). *Detecting hateful utterances using an anomaly detection approach* (Master's thesis). NTNU.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *European conference on machine learning*, 137–142.

Mehdad, Y. & Tetreault, J. (2016). Do characters abuse more than words? *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 299–303.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C. & Joulin, A. (2017). Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, *26*, 3111–3119.

Mitra, B., Nalisnick, E., Craswell, N. & Caruana, R. (2016). A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*.

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press.

Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y. & Chang, Y. (2016). Abusive language detection in online user content. *Proceedings of the 25th international conference on world wide web*, 145–153.

Nockleby, J. T. (2000). Hate speech. *Encyclopedia of the American constitution*, *3*(2), 1277–1279.

Pennington, J., Socher, R. & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

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

Ruff, L., Vandermeulen, R. A., Görnitz, N., Binder, A., Müller, E., Müller, K.-R. & Kloft, M. (2019). Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*.

Ruff, L., Vandermeulen, R. A., Görnitz, N., Binder, A., Müller, E., Müller, K.-R. & Kloft, M. (2020). Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*.

Safavian, S. R. & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, *21*(3), 660–674.

Sanguinetti, M., Poletto, F., Bosco, C., Patti, V. & Stranisci, M. (2018). An italian twitter corpus of hate speech against immigrants. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Schmidt, A. & Wiegand, M. (2017). A survey on hate speech detection using natural language processing. *Proceedings of the Fifth International workshop on natural language processing for social media*, 1–10.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J. & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*, *13*(7), 1443–1471.

Sharma, S., Agrawal, S. & Shrivastava, M. (2018). Degree based classification of harmful speech using twitter data. *arXiv preprint arXiv:1806.04197*.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. *2017 IEEE winter conference on applications of computer vision (WACV)*, 464–472.

Spackman, K. A. (1989). Signal detection theory: Valuable tools for evaluating inductive learning. *Proceedings of the sixth international workshop on Machine learning*, 160–163.

Unsvåg, E. F. (2018). *Investigating the effects of user features in hate speech detection on twitter* (Master's thesis). NTNU.

Van Hee, C., Lefever, E., Verhoeven, B., Mennes, J., Desmet, B., De Pauw, G., Daelemans, W. & Hoste, V. (2015). Automatic detection and prevention

of cyberbullying. *International Conference on Human and Social Analytics (HUSO 2015)*, 13–18.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wahl, M. & Skjåstad, S. G. (2020). *Study on ai techniques for automatically detecting hate speech in norwegian short texts*.

Waseem, Z. & Hovy, D. (2016). *Proceedings of the NAACL student research workshop*, 88–93.

Zhang, Z. & Luo, L. (2019). Hate speech detection: A solved problem? the challenging case of long tail on twitter. *Semantic Web*, *10*(5), 925–945.

Zhong, H., Li, H., Squicciarini, A. C., Rajtmajer, S. M., Griffin, C., Miller, D. J. & Caragea, C. (2016). Content-driven detection of cyberbullying on the instagram social network. *IJCAI*, 3952–3958.

Zhou, Z.-H. (2009). Ensemble learning. *Encyclopedia of biometrics*, *1*, 270–273.

APPENDIX A

Additional graphs for tuning of representation dimension hyperparameter

This appendix presents figures of validation AUC per epoch for different values of representation dimension size (d), which are discussed in Section 6.1.

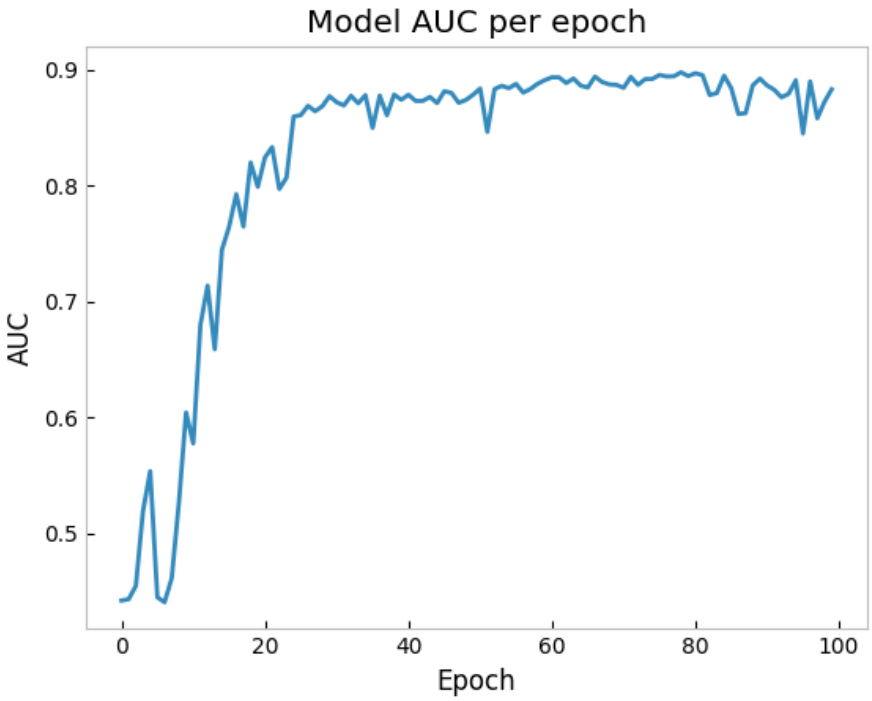


Figure A.1: Validation AUC per epoch curve for $d = 16$

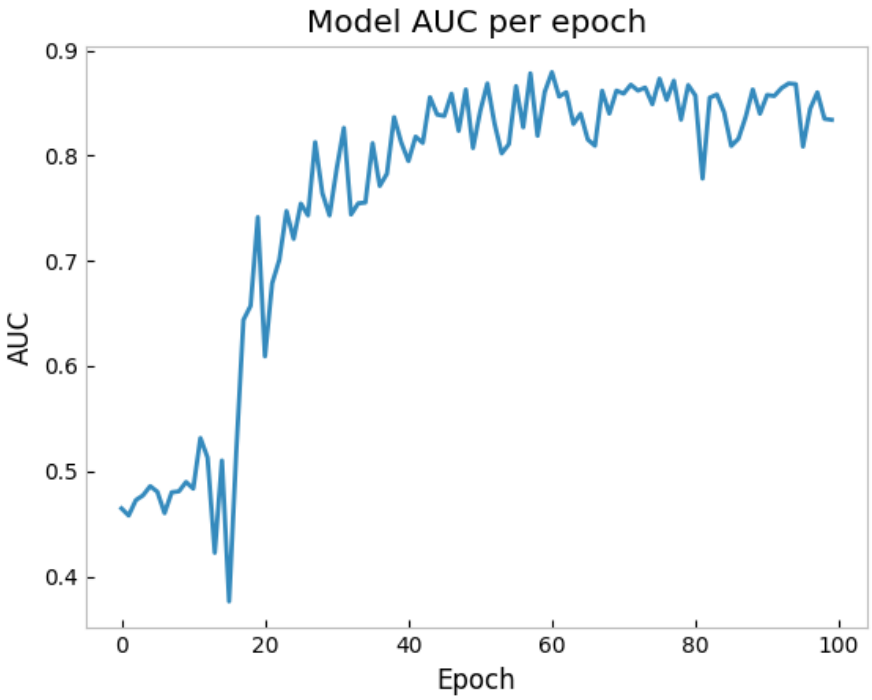


Figure A.2: Validation AUC per epoch curve for $d = 32$

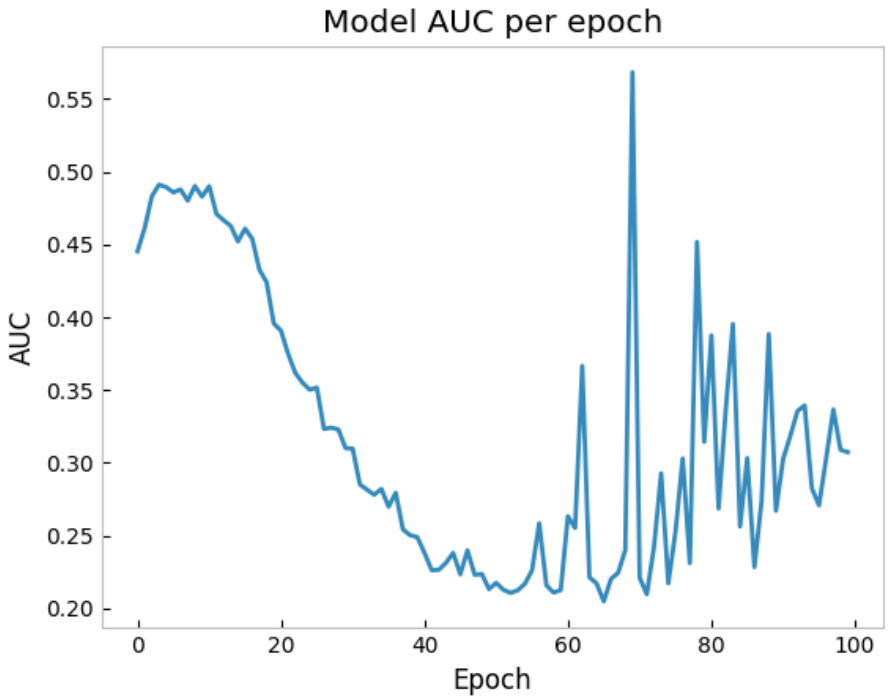


Figure A.3: Validation AUC per epoch curve for $d = 64$

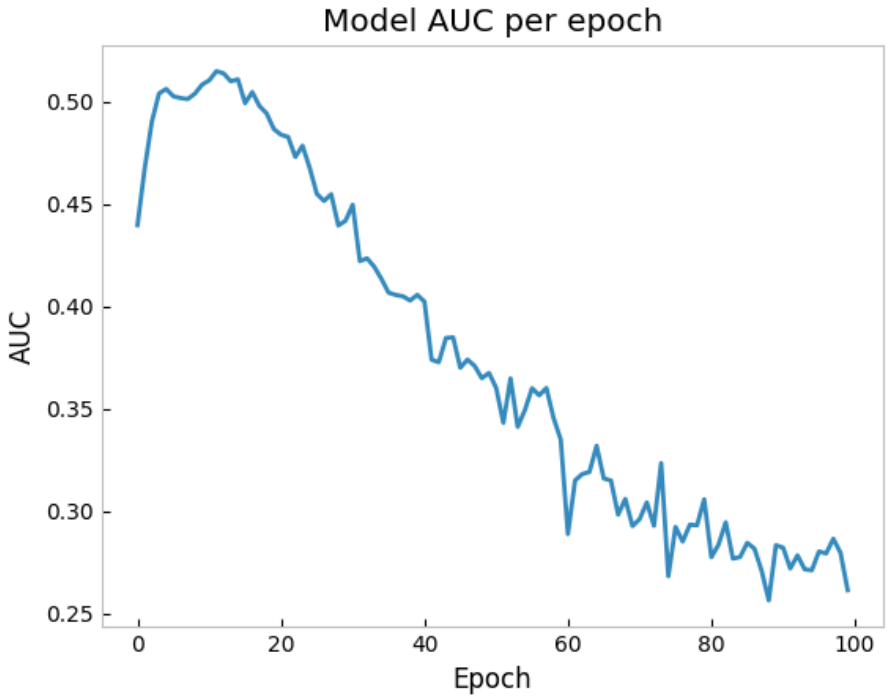


Figure A.4: Validation AUC per epoch curve for $d = 128$

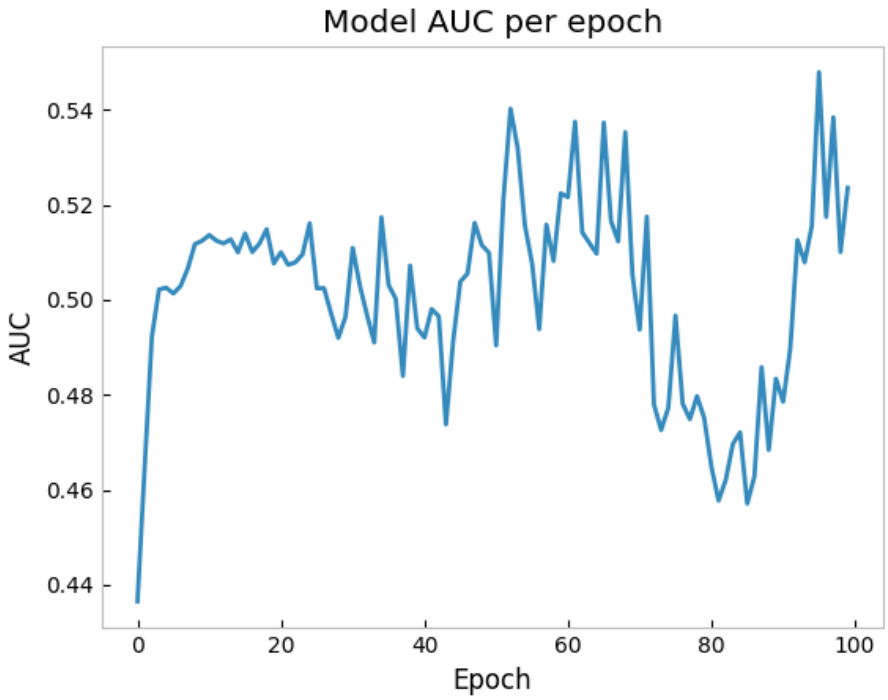


Figure A.5: Validation AUC per epoch curve for $d = 256$

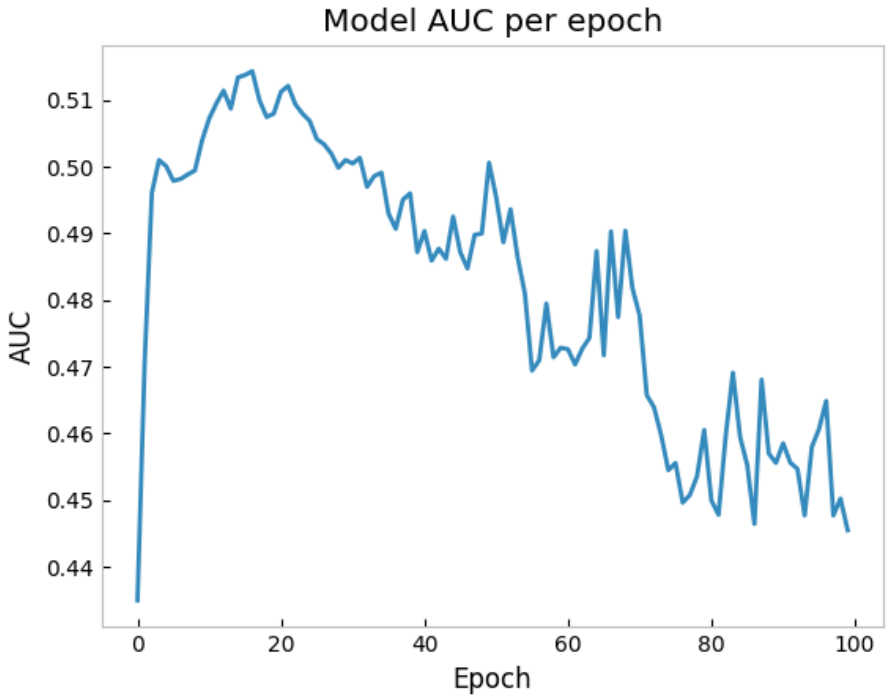


Figure A.6: Validation AUC per epoch curve for $d = 512$

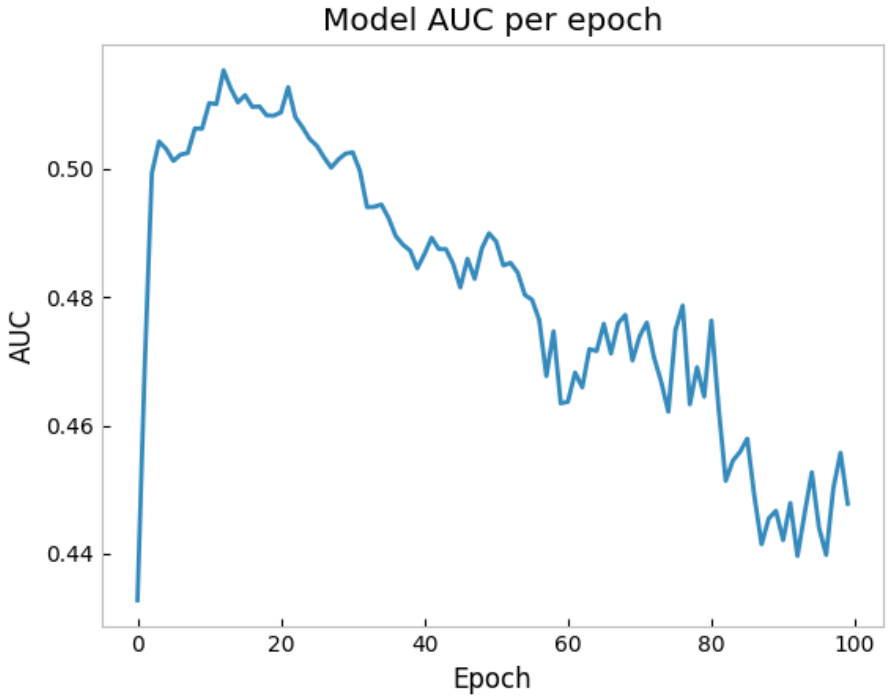


Figure A.7: Validation AUC per epoch curve for $d = 1024$

Mathias Wahl & Sondre Grav Skjåstad

Detecting Hate Speech in Norwegian Texts Using BERT Semi-Supervised Anomaly Detection

NTNU
Norwegian University of
Science and Technology