Master's thesis

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

Karoline Lillevestre Langli

Sentiment Analysis of Customer Emails Using BERT

Master's thesis in Computer Science
Supervisor: Jon Atle Gulla
Co-supervisor: Benjamin Kille
June 2023

NTNU
Norwegian University of
Science and Technology

Karoline Lillevestre Langli

Sentiment Analysis of Customer Emails Using BERT

Master's thesis in Computer Science
Supervisor: Jon Atle Gulla
Co-supervisor: Benjamin Kille
June 2023

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

In recent years, language models have become popular, and they are currently used to solve various natural language processing tasks. Many companies have stored large quantities of unstructured text that is still not being automatically processed. Therefore, this thesis examines if language models can automatically process customer emails sent to Sparebank 1 SMN.

This thesis utilizes four BERT models, namely NB-BERT, NorBERT, mBERT, and DistilmBERT, to perform sentiment analysis on Norwegian text. The BERT models were also compared to baseline models using TF-IDF combined with SVM, logistic regression, or K-nearest neighbor. Two datasets were used: the publicly available NoReC dataset containing reviews and a dataset of customer emails provided by Sparebank 1 SMN. Active learning was performed on the emails to create sentiment labels. This worked to some extent, but there is still room for improvement.

The models were evaluated using the F1-score and precision of the negative class and by examining the calculated confusion matrices. NB-BERT achieved the highest score on the NoReC dataset, and NorBERT did best on the emails. The baseline outperformed mBERT and DistilmBERT on the NoReC dataset and mBERT, DistilmBERT, and NB-BERT on the email dataset. Additionally, it was shown that the predictions were notably slower with the BERT models compared to the baseline.

The code written during this thesis is available on Github[1].

[1] https://github.com/Karolill/master_thesis/tree/main

Sammendrag

I løpet av de siste årene har språkmodeller blitt veldig populære, og de brukes for øyeblikket til å løse ulike oppgaver innen naturlig språkprosessering (NLP). Mange selskap har store mengder ustrukturert tekst lagret som fortsatt ikke blir prosessert automatisk. Derfor undersøker denne oppgaven om språkmodeller kan brukes for å automatisk prosessere eposter fra kunder sendt til Sparebank 1 SMN.

Oppgaven bruker de fire BERT-modellene NB-BERT, NorBERT, mBERT og DistilmBERT for å utføre sentimentanalyse av norsk tekst. BERT modellene ble også sammenliknet med mindre modeller som brukte TF-IDF etterfulgt av SVM, logistisk regresjon eller KNN. To datasett ble brukt: Det offentlig tilgjengelige datasettet NoReC bestående av anmeldelser, og et datasett med eposter sendt fra kunder til Sparebank 1 SMN. Aktiv læring ble gjennomført for å klassifisere epostene. Dette fungerte til en viss grad, men det er fortsatt rom for forbedring.

Modellene ble evaluert ved bruk av F1-scoren og presisjonen oppnådd på den negative klassen, og ved å se på de genererte forvirringsmatrisene. NB-BERT var den beste modellen på NoReC datasettet, mens NorBERT gjorde det best på epostene. Modellen som brukte TF-IDF oppnådde høyere score enn mBERT og DistilmBERT på NoReC datasettet, og mBERT, DistilmBERT og NB-BERT på epostene. I tillegg til dette ble det vist at prediksjon gjennomføres mye raskere av modellen som brukte TF-IDF enn ved bruk av BERT.

Koden skrevet i denne masteroppgaven er tilgjengelig på Github[2].

[2] https://github.com/Karolill/master_thesis/tree/main

Preface

This master's thesis concludes the work performed during the 2023 spring semester in the course TDT4900 - Computer Science, Master's Thesis. The work has been carried out at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology. This thesis has been written in collaboration with the Norwegian Research Center for AI Innovation (NorwAI) and Sparebank 1 SMN.

The thesis builds on a project from the previous semester in the course TDT4501 - Computer Science, Specialization Project. Continuing in the same field meant less time had to be spent on the literature review, and more effort could be put into the experiments. Specifically, Sections 2.8 to 2.10 and 2.12 to 2.14 are based on the report written in TDT4501.

First, I would like to thank my supervisor Benjamin Kille who has greatly helped with this thesis by answering questions, participating in weekly meetings, and providing feedback on this paper. I am also grateful for the help from Stian Fagerli Arntsen at Sparebank 1 SMN, who also attended the weekly meetings, came with valuable insight into the business side of the project, and created and provided access to the necessary data. Also, thanks to my main supervisor Jon Atle Gulla.

Finally, I want to thank my friends and family for supporting and helping me throughout the five years of my studies. This period of my life would not have been the same without you.

Table of Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**

1.1 Motivation 1

1.2 Research Questions 2

1.3 Approach 2

1.4 Limitations 2

1.5 Outline 3

2 Background Theory **5**

2.1 Classification vs. Regression 5

2.2 Logistic Regression 6

2.3 K-Nearest Neighbor 6

2.4 Neural Networks 7

2.5 Support Vector Machine 8

2.6 Train, Validation, and Test Datasets 9

2.7 Hyper-Parameter Optimization 10

2.8 Evaluation 10

2.9 Sentiment Analysis 12

2.10 Text Representations 13

2.10.1 Text Preprocessing 13

2.10.2 Bag-of-Words - Frequency Vector 13

2.10.3 TF-IDF 14

2.10.4 WordPiece Model 15

2.11 Language Models 15

2.12 Recurrent Neural Networks . . . 16
2.13 Transformers . . . 17
2.13.1 Attention . . . 17
2.13.2 The Transformer Architecture . . . 19
2.13.3 BERT . . . 20
2.13.4 Multilingual BERT . . . 22
2.13.5 NB-BERT . . . 23
2.13.6 NorBERT . . . 23
2.13.7 DistilBERT . . . 23
2.13.8 Strengths and Weaknesses of Transformer-Based Language Models 24
2.14 Methods for Handling Missing Labels . . . 24
2.14.1 Transfer Learning . . . 24
2.14.2 Active Learning . . . 25

3 Related Work **27**
3.1 TF-IDF for Sentiment Analysis . . . 27
3.2 BERT for Sentiment Analysis . . . 28
3.2.1 BERT for Sentiment Analysis in Norwegian . . . 28
3.2.2 BERT for Sentiment Analysis in Finance . . . 29
3.2.3 Distilled Models . . . 30
3.3 Label Generation . . . 30
3.3.1 Transfer Learning . . . 30
3.3.2 Active Learning . . . 31

4 Data **33**
4.1 The NoReC Dataset . . . 33
4.2 The SMN Dataset . . . 35

5 Method **39**
5.1 Tools and Libraries . . . 39
5.2 Preprocessing . . . 40
5.2.1 The NoReC Dataset . . . 41
5.2.2 The SMN Dataset . . . 41
5.3 Research Question 2 - Active Learning . . . 42
5.4 Research Question 3 - Language Models . . . 43

5.4.1 Baseline Models . . . 43
5.4.2 BERT Models . . . 46
5.4.3 Qualitative Evaluation of the Model's Utility . . . 49

6 Results **51**
6.1 Research Question 2 - Active Learning . . . 51
6.2 Research Question 3 - Language Models . . . 51
6.2.1 NoReC Dataset . . . 51
6.2.2 SMN Dataset . . . 57
6.2.3 Qualitative Evaluation of the Model's Utility . . . 60

7 Discussion **61**
7.1 State-of-the-Art Techniques for Sentiment Analysis in Norwegian . . . 61
7.2 Active Learning for Customer Emails . . . 62
7.3 Comparison of Models for Sentiment Analysis . . . 63
7.3.1 Comparison of the Transformer-Based Models . . . 63
7.3.2 Transformer-Based Models vs. Previously Used Methods . . . 65
7.3.3 What Model is the Best? . . . 65
7.4 Challenges . . . 66

8 Conclusion and Future Work **69**
8.1 Conclusion . . . 69
8.2 Future Work . . . 70
8.2.1 Improving the Active Learning Process . . . 70
8.2.2 Testing More Model Versions . . . 71
8.2.3 Reducing Errors from Preprocessing . . . 71
8.2.4 Creating a Larger Dataset . . . 71
8.2.5 Sentiment Analysis from a Business Perspective . . . 72

Bibliography **73**

A Example Description **81**

B $\text{NoReC}_{\text{train\_full}}$ vs. $\text{NoReC}_{\text{train\_no\_neutral}}$ **83**

C Results on $\text{NoReC}_{\text{val}}$ by the BERT Models **85**
C.1 NB-BERT . . . 86

C.2 NorBERT 88
C.3 mBERT 90
C.4 DistilmBERT 92

List of Figures

2.1 Logistic regression. 6

2.2 An illustration of how a KNN model classifies examples. 7

2.3 A neural network architecture. 8

2.4 An illustration of how an SVM solves classification tasks. 9

2.5 Illustration of an example confusion matrix. 12

2.6 An example of BoW. 14

2.7 An example of TF-IDF. 15

2.8 Bigram model. 16

2.9 Illustration of an RNN. 17

2.10 An example of self-attention. 18

2.11 The transformer architecture. 19

2.12 The BERT architecture as explained in [17]. 21

2.13 An example of the input embeddings sent as input to the BERT model. 21

2.14 An illustration of the AL workflow. 25

4.1 The distribution of scores in the original NoReC datasets. 34

4.2 The distribution of original scores in the balanced NoReC training datasets. 35

4.3 Word cloud of the customer emails in the SMN dataset. 36

4.4 A visualization of the label distribution in the three SMN datasets. . . 37

5.1 The preprocessing steps performed on the SMN dataset. 41

5.2 The general architecture of the baseline models. 44

5.3 The normalization process for the baseline models, including an example. 44
5.4 The workflow to create and find the best BERT model for SA. 46
5.5 Architecture of the BERT model for one input text. 46

6.1 Baselines' confusion matrices on $\text{NoReC}_{\text{test}}$. 53
6.2 Results from hyper-parameter tuning of the BERT models. 54
6.3 The BERT models' confusion matrices from predictions on $\text{NoReC}_{\text{test}}$. . 56
6.4 The baseline models' confusion matrices after making predictions on SMN_{test}. 57
6.5 The score of all four BERT models on SMN_{val} after each epoch. 58
6.6 The BERT models' confusion matrices from predictions on SMN_{test}. . . 60

C.1 Results on the validation dataset from the fine-tuning of NB-BERT. . . 87
C.2 Results on the validation dataset from the fine-tuning of NorBERT. . . 89
C.3 Results on the validation dataset from the fine-tuning of mBERT. . . . 91
C.4 Results on the validation dataset from the fine-tuning of mBERT. . . . 93

List of Tables

3.1 Results found by Avinash and Sivasankar [41] on five different datasets. 28

4.1 Overview of the NoReC datasets used for this task. 34

4.2 An overview of the columns from the Sparebank 1 SMN dataset used during this project. 35

4.3 Overview of the SMN datasets used for this task. 37

5.1 The Python libraries used for the experiments. 40

5.2 All ML models trained and the hyper-parameters tuned during the grid search for the best baseline models. 45

5.3 The best model parameters found through a grid search for the best baseline models. 45

5.4 Model specifications. 47

5.5 Overview of which BERT models and parameter values were tested during the grid search. 48

6.1 Results by the six possible baseline models tested on $\text{NoReC}_{\text{test}}$. 52

6.2 Results by the BERT models tested on $\text{NoReC}_{\text{test}}$. 55

6.3 Time spent on prediction of the $\text{NoReC}_{\text{test}}$ dataset. 55

6.4 Results by the three possible baseline models tested on SMN_{test}. 58

6.5 Results by the BERT models tested on SMN_{test}. 59

6.6 Time spent on prediction of the SMN_{test} dataset. 59

B.1 Results when training two similar NorBERT models on $\text{NoReC}_{\text{train\_full}}$ or $\text{NoReC}_{\text{train\_no\_neutral}}$. 83

Abbreviations

AI	Artificial Intelligence
AL	Active Learning
BERT	Bidirectional Encoder Representations from Transformers
BoW	Bag-of-Words
CNN	Convolutional Neural Network
KNN	K-Nearest Neighbor
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Network
RNN	Recurrent Neural Network
SA	Sentiment Analysis
SOTA	State-of-the-Art
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
TL	Transfer Learning

Chapter 1

Introduction

This section starts by discussing the motivation behind this thesis. Then, three research questions are defined, and the approach utilized to answer them will be briefly mentioned. Afterward, limitations that influenced the experiments that were conducted to answer the research questions are stated. Finally, an outline of this thesis will be provided.

1.1 Motivation

In the wake of the recent artificial intelligence (AI) boom, which follows the release of ChatGPT, it is compelling to investigate some of these recently created language models' use cases in the industry. Many companies have large amounts of unstructured textual data stored on their servers, which is not used for automatic processing [62]. Creating AI programs capable of structuring this data and extracting critical information can ease the workload for employees and improve customer relations. Since the creation of the transformer architecture, language models have improved significantly, and they can be used for a variety of natural language processing (NLP) tasks. One such task which has been thoroughly explored and improved throughout the years is sentiment analysis (SA), where the goal is to determine the sentiment on a scale from negative to positive [43]. Therefore, it is interesting to examine if this technology can improve customer relationships.

This thesis was written in collaboration with the Norwegian bank Sparebank 1 SMN. The bank daily receives numerous emails and other written messages from customers. However, there is currently limited automatic processing of these texts, which presents an opportunity for AI to enhance the efficiency of handling these emails. With this in mind, this thesis focused on SA as a step toward automating email processing. Additionally, since the emails were originally unlabeled, meaning they were not marked as positive or negative, examining whether AI methods can generate labels is also a rewarding experiment.

1.2 Research Questions

This thesis aims to gain insight into the use of SA on customer emails sent to Sparebank 1 SMN. To achieve this goal, three research questions guided the experiments performed. They are defined as:

RQ1 What are the state-of-the-art techniques for sentiment analysis in Norwegian?

RQ2 Can active learning be used to create sentiment labels on customer emails?

RQ3 How do different transformer-based language models compare to each other when used to solve sentiment analysis tasks? And how do transformer-based language models compare to previously used methods for sentiment analysis?

This thesis contributes to the research field by labeling a new dataset using active learning (AL) and performing SA on the labeled dataset. Due to privacy reasons, the dataset itself can not be published, but the results will be presented and discussed.

1.3 Approach

To answer the research questions, previous work in the field was examined, and experiments were performed. **RQ1** was answered through a literature study, which looked into recent developments in the field of NLP, and took special care to examine findings related to SA of Norwegian texts. Some of the literature was read last semester as a part of the course TDT4501. **RQ2** and **RQ3** were answered by conducting experiments. **RQ2** was answered by performing AL and receiving feedback from bank employees evaluating the labels' accuracy. The AL process was completed before conducting the experiment related to **RQ3**, as it required the emails to be labeled. To answer **RQ3** two datasets, the dataset of emails provided by Sparebank 1 SMN and a publicly available dataset named NoReC, were used. It was decided to use the NoReC dataset to create reproducible results, as the emails are not publicly available and, therefore, can not be used to reproduce the experiments. Four transformer-based models and a baseline were compared on the two datasets to create the final results, which were later discussed.

1.4 Limitations

The experiments were limited by the time available for the thesis and the access to computing resources. The email dataset provided by the bank was only accessible through a virtual machine, which did not have a GPU. Therefore, computations on that dataset were quite time-consuming. **RQ1** was limited by the available time and will therefore only focus on four transformer-based language models, namely the

Norwegian models NB-BERT and NorBERT and the multilingual models mBERT and DistilmBERT. Other multilingual models that could perform well on the datasets also exist but will not be discussed. **RQ2** was answered by performing a complete round of AL. However, further improvements beyond the first results could not be researched due to time limitations. **RQ3** is limited to testing the same four transformer-based models mentioned in **RQ1**, in addition to TF-IDF combined with support vector machines, logistic regression, and K-nearest neighbor as baselines. Recurrent neural networks for SA will also be mentioned, but it was not used for the experiments. As the experiments conducted to answer **RQ3** required a lot of computing power, they were limited by the lack of access to a GPU as well as the available time.

1.5 Outline

This thesis is divided into eight chapters as follows:

Section 1 - Introduction presents the thesis. It states the motivation for the project and defines three research questions that guided the experiments conducted.

Section 2 - Background Theory introduces theoretical concepts necessary to understand the remainder of the thesis. It presents general machine learning (ML) topics before explaining task-specific theory.

Section 3 - Related Work summarizes previous work in the field relevant to the task. Previous findings are helpful when interpreting the results and trying to understand what caused unexpected results.

Section 4 - Data gives an overview of the two datasets used for the experiments: the NoReC dataset and the email dataset provided by Sarebank 1 SMN.

Section 5 - Method describes the experimental method used to answer **RQ2** and **RQ3**. It explains technical aspects such as the Python libraries and objects used and the steps to perform the experiments.

Section 6 - Results presents the experimental results achieved and highlights the most important findings.

Section 7 - Discussion will discuss the results presented in Section 6. It examines if the expected results were achieved and tries to understand why that was not the case for all results.

Section 8 - Conclusion and Future Work summarizes the findings related to each research question and states future research that helps to improve the performance and understand the problem.

Chapter 2

Background Theory

This section will introduce the theory needed to understand the experiments conducted for this thesis. First Sections 2.1 to 2.8 will present some basic concepts, methods, and models used in many ML applications today before the remaining part of this section will dive into the task-specific theory related to this thesis. Specifically, SA will be introduced, and much focus will be put on text vectorization techniques and language models, as these are necessary to perform SA.

2.1 Classification vs. Regression

Supervised ML tasks can be divided into two categories, namely classification and regression. Kuhn and Johnson [33] discuss both in more detail, but a quick overview of the differences is given here. Classification is an ML task aiming to create a model that predicts one of a limited set of possible answers or classes. For example, a task could be to look at an image of either oranges or apples and attempt to classify it as the correct fruit. Another example, using text, would be looking at a text and predicting whether it is positive or negative. These examples are binary classification problems, as there are only two classes. More classes could also exist, for instance, if an image of a handwritten digit is provided, and the task is to predict what digit the image shows. Then there are ten possible solutions. In most cases, classification tasks will give you one or more continuous answers, representing the probability of a given solution being correct. The class with the highest probability is then the final answer. The answer to what class the example belongs to is also called a label, so a labeled example already has a class, while it is currently unknown what class an unlabeled example belongs to.

Regression problems are tasks where the final solution is any number in a continuous range. For example, if the task is to predict house prices, the model could take in information about the house size, location, and so on, and the answer is any positive number. Although the following sections will focus on ML models for classification, many of the same techniques can also be adapted to solve regression tasks.

2.2 Logistic Regression

Logistic regression is a technique for classifying data points. An illustration of this can be seen in Figure 2.1. You start with example data points from the two classes, where you want one class to have the value $y = 0$ and the other to have the value $y = 1$. Then it is attempted to fit an S-shaped logistic function f to these data points to maximize the likelihood of the data given f. After f is found, the class of a new data point x can be predicted by looking at its y-value $f(x)$, which states the probability of the data point belonging to class 1. If this value is lower than 0.5, the data point probably belongs to class 0, otherwise to class 1. In some instances, adjusting the threshold for choosing a given class might be desirable. For example, the threshold could be set so that the model would have to be 80% certain that a data point belonged to class 1 to classify it as such. The information in this section is based on [54, p. 725-727], where more detailed information about the topic can also be found.

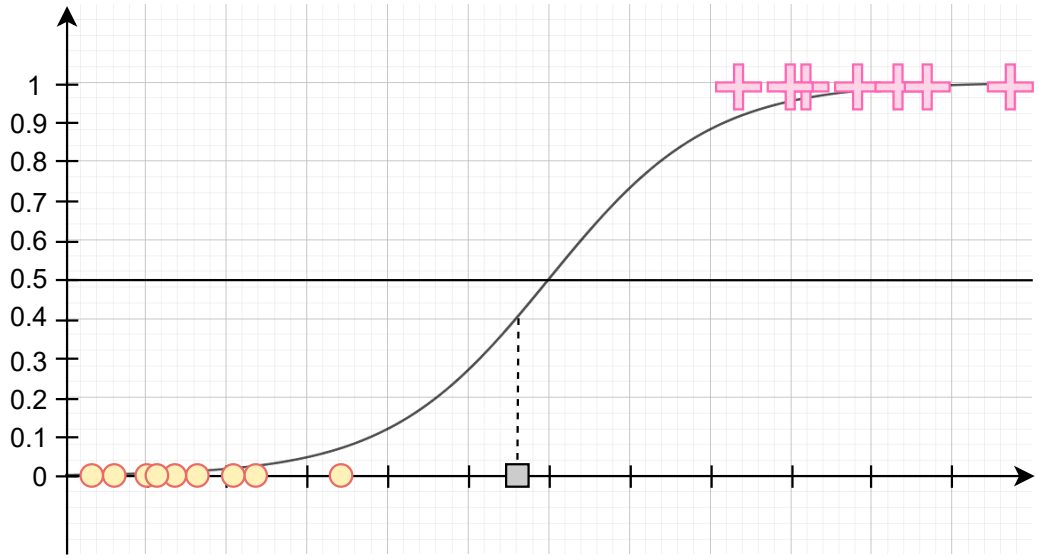


Figure 2.1: Illustration of logistic regression for a binary classification problem. The horizontal line at $y = 0.5$ is the threshold between classes, and the gray square is a new data point with a 40% chance of belonging to class 1 and a 60% chance of belonging to class 0. Class 1 is the pink crosses, and class 0 is the yellow circles.

2.3 K-Nearest Neighbor

As explained by Russell and Norvig [54, p. 738] K-nearest neighbor (KNN) is a method where the k nearest training data points are used to classify new examples. For example, in Figure 2.2, the five closest training examples are used for classification since $k = 5$. In that case, the new example would be classified as a yellow circle which is the majority class of the neighbors. Different distance measures can be used to find the closest training data points, such as the Euclidean or Manhattan distance.

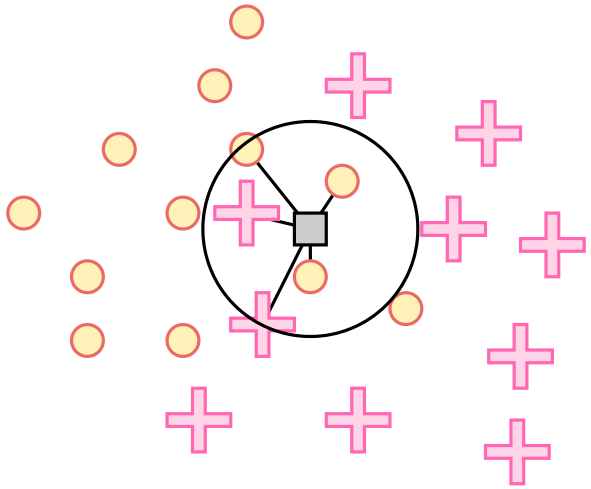


Figure 2.2: An illustration of how a KNN model with $k = 5$ classifies examples in a binary classification problem. The gray square is the example that should be classified based on its five neighbors, and in this case, it would be classified as a yellow circle. The large circle shows the square's "neighborhood".

2.4 Neural Networks

A neural network (NN) is an architecture often used in ML. Figure 2.3 illustrates what a NN can look like. It has three main parts, an input layer, one or more hidden layers, and an output layer. All layers consist of multiple nodes (illustrated as circles in the image), also called neurons. These nodes are connected through edges, sometimes called connections (the lines in the image). Each edge has a weight that decides how important the information being sent across that connection is, and the weight is illustrated through different thickness of the edges in the figure.

The input layer consists of continuous inputs, representing different things in different tasks. In image tasks, the inputs are numbers representing the pixels in the image. If you try to predict the electricity consumption in the next hour, the input might be numbers representing the consumption in the past few hours. Or, if one is working on texts, the input could be numbers representing the text. These inputs are sent to the hidden layer.

When input n, noted as i_n, is sent to node m in the hidden layer, it is multiplied with a weight $w_{n,m}$. The input to the hidden layer can then be written as $i_n \cdot w_{n,m}$. Inside the hidden nodes, all inputs are summed and passed through an activation function, such as ReLU$= \max(0, x)$. This means that the output from a node in the hidden layer is ReLU$\left(\sum_n i_n \cdot w_{n,m}\right)$. If the output from all nodes in the previous layer, here the input layer, is sent to all nodes in the current layer, this is called a fully connected layer. This is the case for the hidden layer in Figure 2.3. It is also possible to have multiple hidden layers, in which case the output from the first hidden layer is sent to the nodes in the next hidden layer. This is called a deep NN.

The outputs from the final hidden layer are sent to the output layer, which typically has as many nodes as there are classes. Also here, each node sums the inputs and passes that through an activation function, for example a sigmoid function, that gives an answer between 0 and 1. Afterward, a softmax transformation is performed on the outputs to make the numbers add up to one. At this point, the output indicates the probability of a given class being correct, and the most probable class is chosen as the answer. The information presented in this section is found in [33, p. 333-337] and [54, p. 727-737], and interested readers can find more in-depth information there as well.

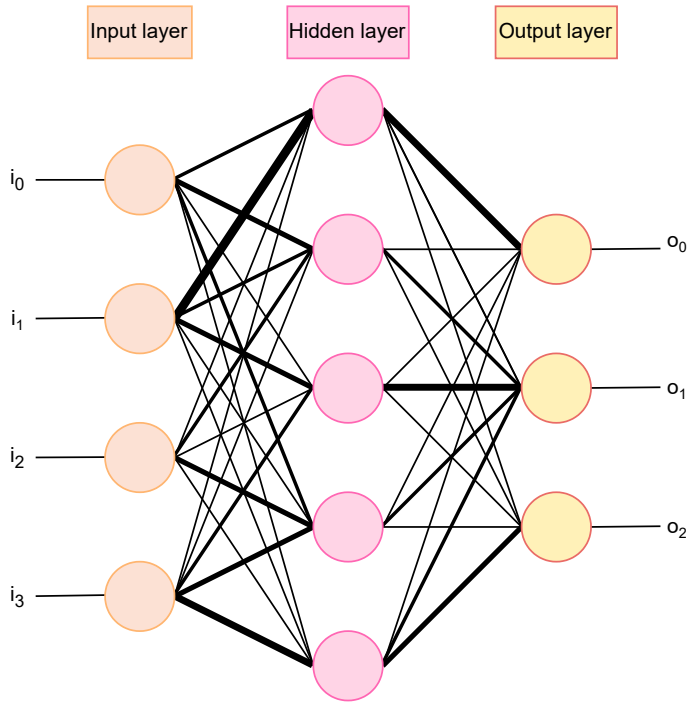


Figure 2.3: A neural network architecture. There are three layer types: one input layer, one or more hidden layers (here: one), and one output layer. There are connections between the nodes in each layer, whose weight is shown through the line's thickness.

2.5 Support Vector Machine

Support vector machine (SVM) is an ML technique that tries to separate the classes using few data points, making it an effective method. Suykens et al. [61] thoroughly explains the subject, but this section will merely outline the basics. An illustration of a simple binary classification task can be seen in Figure 2.4. A line called a decision boundary, has been drawn between the two classes. Two dashed lines have been drawn equally far from and parallel to the decision boundary. The distance between these lines is called the margin, and it is decided by how far away from the decision boundary you can set the dashed lines without crossing a data point. The optimization problem solved by an SVM is where to place the decision boundary to maximize the margin. The data points that affect the decision boundary are called support vectors, which are the points closest to the boundary. These points also decide the model's runtime [22].

When the optimal decision boundary is found, new data points can be classified by checking on what side of the decision boundary the data point lies. Decision boundaries do not have to be linear, they can allow for misclassification, and SVMs can increase the dimensionality of the data points to make the decision boundaries easier to set. However, these topics will not be described further in this section.

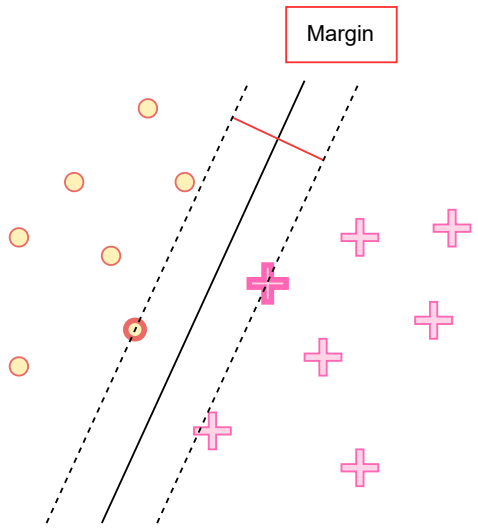


Figure 2.4: An illustration of how an SVM solves a binary classification task. The two points with a thicker edge are support vectors that limit the size of the margin.

2.6 Train, Validation, and Test Datasets

A training dataset is needed to create good ML models such as the ones mentioned above, as the models learn by looking at examples of input-output pairs. For instance, NNs learn by using examples in the training dataset to update the weights [54, p. 733], while SVMs learn by using the training examples to set the decision boundary [33, p. 344]. It is common to use 80% of the labeled examples for training, in which case the remaining 20% are used as a test dataset. After training is finished, the models make predictions on the test dataset, and it is then possible to estimate the performance of the models. The training dataset can not be used for this purpose, as models will typically overfit to it while training, meaning that they learn how to perform almost perfectly on the training dataset, but what the models learn does not generalize well to new unseen data [33, p. 62].

In addition to the two datasets mentioned, it can be good to have a validation dataset. As Russel and Norvig [54, p. 708-709] explain: If the test dataset is used to tune hyper-parameters to create the best model, the developers are peeking at the test dataset during development and will not receive an objective evaluation of which model performs best on unseen data. To get an objective evaluation, the model must be tested on a dataset that has not, in any way, affected the model training. If three datasets are used, the training dataset is used for training, the validation dataset for hyper-parameter optimization, and the test dataset to get a final objective evaluation.

In some cases, these datasets are imbalanced, meaning that one class naturally occurs significantly more often than the others [21, p. 19]. The most common class is called the majority class, and the other is the minority class. If the data is imbalanced, you also want the validation and test datasets to be equally imbalanced, as that would show the model's actual performance on new data. If the available dataset is large, randomly splitting the data into a train, validation, and test dataset will typically achieve this. On the other hand, as illustrated in Fernández et al. [21, p. 20-22], feeding the model an imbalanced training dataset will make it less capable of classifying the minority class. Since creating a model that performs equally well on all classes is usually desirable, this issue should be dealt with. Two possible ways to mitigate the problem are undersampling, where some examples from the majority class are removed from the training dataset, and oversampling, where some examples from the minority class are replicated or new examples are made from the original ones [21, p. 80].

2.7 Hyper-Parameter Optimization

ML models have one or more hyper-parameters that can be tuned. Unlike the weights in a NN or the decision boundary in an SVM, the hyper-parameters are not altered by training. These values are set before training starts and affect how the model learns. Examples of such hyper-parameters are the number of neurons in a NN, which will affect how complicated functions the model can learn, and the number of neighbors used to make decisions in a KNN model. Since these hyper-parameters affect the final results, they should be tuned to achieve as high scores as possible.

Bergstra and Bengio [5] explain two methods for hyper-parameter optimization, namely grid search and random search. During a grid search, a set of values for each hyper-parameter is defined, and all possible combinations of these parameters are used to train the model before testing it on a validation dataset. During random search, the hyper-parameters to try are randomly chosen from a given range of continuous values, or a list of discrete values if there is only a limited set of options for a parameter value. The randomly chosen combinations are used to train a model before it is tested on the validation dataset. Finally, the score achieved can be used to choose the best model.

2.8 Evaluation

After performing a classification task, evaluating the performance to compare different methods is important. Christen [14, p. 166-168] describes the much-used evaluation metrics accuracy, precision, F1-score, and recall, which will be explained here.

Before defining the evaluation metrics, some other concepts must first be explained. In binary classification, one of the classes can be called the positive class and the other the negative class. If you have more than two classes, the currently interesting class can be the positive one, and all the other classes combined make up the negative one. The following numbers can be found using these two classes:

- True positive (TP): The number of correctly classified positive examples.
- True negative (TN): The number of correctly classified negative examples.
- False positive (FP): The number of examples classified as positive, despite being negative.
- False negative (FN): The number of examples classified as negative, despite being positive.

The simplest evaluation metric is accuracy, which can be calculated using the following equation:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of total predictions}} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.1}$$

This means that accuracy is the fraction of correctly classified examples. However, accuracy is not a good metric if the test dataset is imbalanced. For example, an accuracy of 0.95 can be achieved by a model that always predicts class A if 95% of the examples belong to class A and 5% belong to class B. Despite its high accuracy, it is not a good model since it never predicts class B. Because of this, other metrics are often preferred when the test dataset is imbalanced, such as precision, recall, and F1-score.

Precision measures how many of the positive predictions were correctly said to be so and is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.2}$$

On the other hand, recall calculates what fraction of the actual positive class was correctly classified, and the equation for this score is:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.3}$$

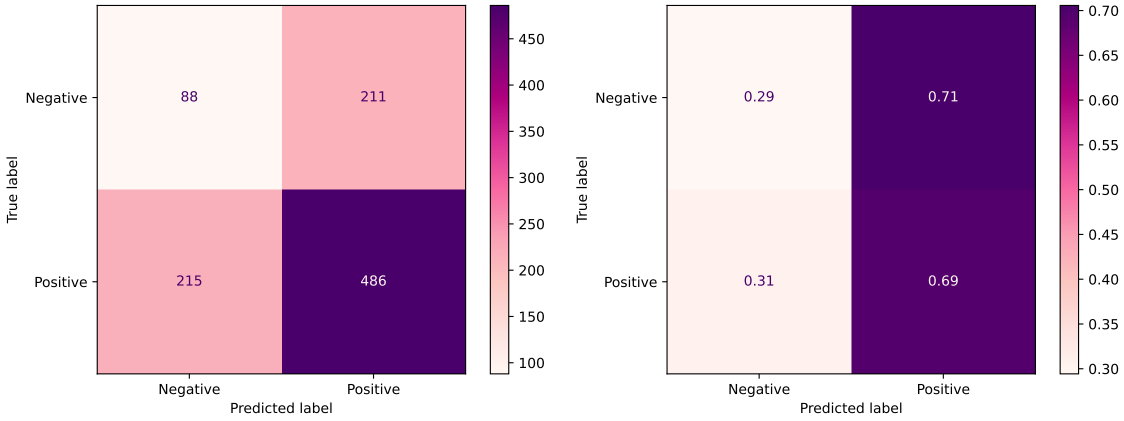
F1-score is the harmonic mean of the precision and recall, which is given by:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.4}$$

These metrics will output a number in the range $[0, 1]$ where 0 is the worst possible score and 1 is the best. They can also be provided as a number between 0 and 100 instead. Each of these metrics can be computed for only one class, meaning that you compute the F1-score of the negative class and the F1-score of the positive class separately, or an average can be found. Two different averages often used are micro and macro average. Macro average will first calculate each class's metric (for example recall) and

average the results. In contrast, micro average calculates the total number of TP, FN, and FP for all classes before using those numbers to calculate the metric [59].

Sometimes it can be interesting to visualize the results from classification tasks for easier comparison. One possible tool is a confusion matrix. This is illustrated in Figure 2.5a for a binary classification problem. Each row represents an actual class, and each column a predicted class [23, p. 91]. The top left corner shows the TN, the top right corner the FP, the bottom left corner the FN, and the bottom right corner the TP. The classification is better the higher the numbers are along the diagonal, which is illustrated by a darker color in the figure. If the matrix is normalized along the rows, as seen in Figure 2.5b, you can read the fraction of correctly and incorrectly classified negative examples in the top row and the misclassified and correctly classified positive examples in the bottom row. This can be helpful if one is interested in the predictive power of a given class, especially if the dataset is imbalanced.



(a) Example of a confusion matrix.

(b) A normalized confusion matrix.

Figure 2.5: Illustration of an example confusion matrix.

2.9 Sentiment Analysis

As explained by Tsytsarau and Palpanas [63], SA is an NLP task where the goal is to predict a document's sentiment. A document is defined as a piece of text in natural language with one or more topics. A topic can, for example, be an event or a person in the document. A sentiment can then be said to be an opinion or attitude directed at the topic. Often the document's sentiment is decided by scoring the sentiment polarity, which corresponds to evaluating how positive or negative the text is. SA can be considered a binary classification problem where the two possible outputs are positive or negative. In some cases, it is also possible to include additional classes, such as neutral. Sentiment predictions can also be made on a numerical scale, in which case the regression problem can be transformed into a classification problem by mapping certain intervals to distinct labels.

SA can be done on different levels, and Medhat et al. [42] explain document-level, sentence-level, and aspect-level SA. Document-level SA aims to predict the overall sentiment polarity of the document. This can be challenging as one text might contain multiple opposing sentiments, making it hard to classify as positive or negative. Sentence-level SA is similar to document-level SA. It finds the sentiment of a sentence, which can be treated as a small document. Sentences can also contain conflicting sentiments, although not as often as longer documents. When performing aspect-level SA, sometimes called aspect-based SA, a document's aspects/topics are first identified before the sentiment directed at each aspect is predicted. This task is more complex since all aspects must first be correctly identified, and the sentiment polarity must then be found for each aspect [26].

2.10 Text Representations

When solving NLP tasks, which is any task involving text written in natural language, ML models face one challenge: They take numbers or vectors as input, not words. Therefore documents must be vectorized before being passed into an ML model, meaning that each word in the document must somehow be turned into a number in a vector or possibly a full vector. This section will first present how the text should be transformed before being vectorized. Afterward, three different methods for vectorization will be explained.

2.10.1 Text Preprocessing

It is often desirable to normalize a text before vectorization [4, p. 72]. During normalization, one might remove punctuation and make all letters lowercase. Other techniques used for normalization include stop word removal, stemming, and lemmatization. Stop word removal removes words such as "the", "or", and "an", as they occur in many texts but have little importance. Stemming uses rules to turn a string into a smaller substring, for example, by removing the "ly"-ending of adverbs. Lemmatization changes words in a similar manner but uses a dictionary instead of rules. An advantage of this preprocessing is that the use of vectorization techniques improves the sparsity. The dimension of the vectors will be smaller because their size depends on the number of unique words in the corpus (a corpus is the collection of all documents).

2.10.2 Bag-of-Words - Frequency Vector

As explained in Bengfort et al. [4, p. 55-58], the bag-of-words (BoW) technique is a set of methods that turn documents into vectors in a way that does not maintain the word order. They start by creating a vector of the same size as the number of unique words in the corpus, so each index in the vector corresponds to a word. How the number at a given index is determined varies, but making a vector representing the

frequency of each word in the document is a simple possibility. Then, the vector for one document is created by counting the number of occurrences of each word in the document and saving that in the corresponding index. An example of this for a corpus of two documents can be seen in Figure 2.6.

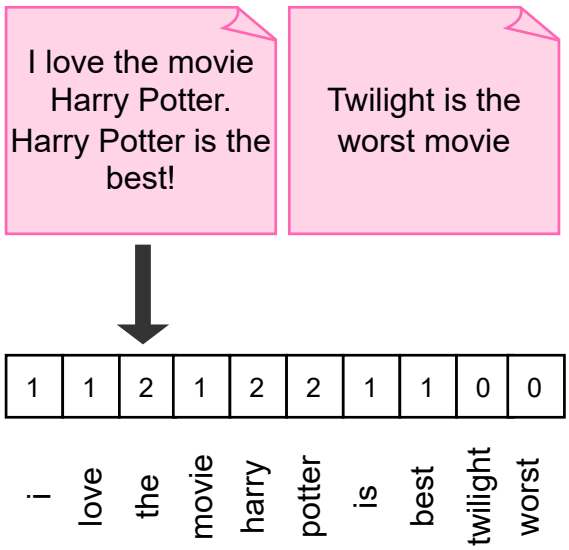


Figure 2.6: An example of BoW, where the vector is calculated for the left document.

2.10.3 TF-IDF

Term frequency-inverse document frequency (TF-IDF) is another type of BoW. Bengfort et al. [4, p. 62-23] explain that TF-IDF will give a higher value to a word frequently occurring in the document. However, the value will decrease if many other documents in the corpus also contain that word. This way, words that are not special for the document are given less importance when the vector is created. This technique will be better than the previously explained frequency vector in many situations since it considers the corpus' context. For example, suppose book reviews were to be classified as either positive or negative. Then, words like "author", "plot", etc., which will occur in many reviews regardless of their sentiment, are given a lower value, despite appearing in the review itself. Figure 2.7 shows an example of using TF-IDF on one document in a corpus containing two documents. TF-IDF can be defined as:

$$TF - IDF(t, d) = TF(t, d) \cdot IDF(t) = (1 + \log f_{t,d}) \cdot \log(1 + \frac{N}{n_t}) \tag{2.5}$$

where t is a term/word, d is the document for which the vector is calculated, $f_{t,d}$ is the term frequency defined as $f_{t,d} = \frac{\text{number of occurrences of } t \text{ in } d}{\text{number of words in } d}$, N is the number of documents, and n_t is the number of occurrences of t in all documents.

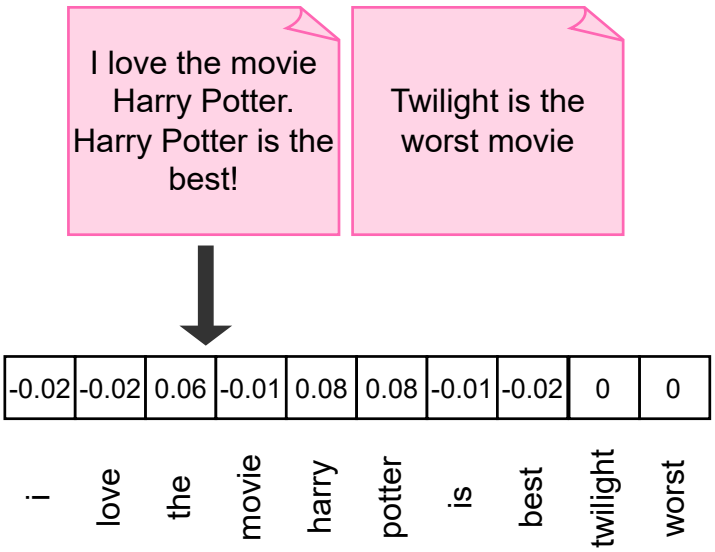


Figure 2.7: An example of TF-IDF, where the vector is calculated for the left document.

2.10.4 WordPiece Model

The wordpiece model is another way to transform documents into a vector. The idea is that each word in the text is split into wordpieces existing in a limited vocabulary [68]. This way, any word can be turned into numbers, as unknown words are split into smaller pieces. For example, "Today is a lovely day" could be turned into the wordpieces "To ##day is a love ##ly day", where ## represents that the wordpiece is not the beginning of the original word. Here, the word Today did not exist in the vocabulary and was therefore split into two smaller existing wordpieces. Each wordpiece has a corresponding value in the dictionary, so an example of the final vector could look like this: [9438, 24696, 105, 1002, 3390, 1024, 372].

2.11 Language Models

A language model aims to estimate the distribution of words in natural language. This means that they predict the probability that a sequence of words, such as a document, is valid [53]. This can be calculated using the following equation:

$$p(x) = \prod_{i=1}^{n} p(w_n | w_1, ..., w_{n-1}). \tag{2.6}$$

Here, w_i is word i in a sequence x that consists of n words. Radford et al. [53] also state that language models should be able to adapt to various tasks, meaning that they must compute $p(output|input, task)$. Language models can also be used to predict the next word in a sequence by considering possible sequences and choosing the one with the highest probability.

A simple language model is the n-gram model. Bengfort et al. [4, p. 132-143] describes how this model works. The value n can be any positive integer, and the simplest models, named unigram models, will use $n = 1$. In this case, the probability of the next word is independent of the previous words, so when generating text, the next word in the sequence is found by choosing w_i such that $p(w_i)$ is maximized. If $n = 2$, it is called a bigram model, and w_i is instead chosen to maximize $p(w_i|w_{i-1})$. In simpler terms, the next word depends on the previous. An illustration of how a sliding window is used to identify all bigrams in a text is shown in Figure 2.8. This process counts all bigrams and finds the frequency at which w_i follows w_{i-1}. These frequencies can then be used to calculate the probabilities. It is also possible to use trigrams, where $n = 3$, or even higher values of n. However, the models become more complex as n increases. One problem with n-gram models is their short memory, meaning that when they perform language modeling tasks, they can only rely on n words at a given time, which typically does not correspond to the whole text. In recent years more complex models such as RNNs and transformer-based language models have been more successful in language modeling tasks, and one of the reasons for this is a longer memory. The following sections will provide the necessary information about these models.

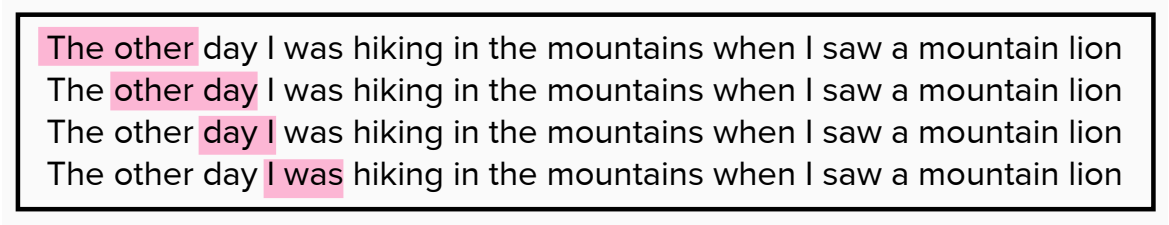


Figure 2.8: Illustration of how a bigram model identifies pairs of words.

2.12 Recurrent Neural Networks

A NN where the output at time step t is sent as input to the same network at time step $t + 1$ is called a recurrent NN (RNN) [54, p. 729]. Figure 2.9 illustrates what this process would look like over time. In text analysis, a time step corresponds to passing one word as input to the network. To perform classification, the output from the hidden layers is sent to a final output layer which provides the probability of a text belonging to a given class.

Because NNs use numerical representations, the text must first be vectorized [31]. A separate vector is created for each word. These vectors are then sent as inputs to the RNN, and one word is passed as input at each time step. Due to this, all words in the text can be passed to the network in the original order, no matter the text's size. This means that an RNN has a longer memory than n-gram models. Also, the fact that RNNs take word ordering into account is an improvement over BoW models. For example, the phrase "not angry" could appear negative if the order was lost, showing that it is beneficial to keep it. Despite these improvements, RNNs still have limited long-term memory because they suffer from the vanishing/exploding gradient

problem during training [15]. In recent years, other models building on RNNs, such as LSTM [27] and GRU [13], have been proposed to mitigate this problem. Another issue with RNNs is that they can not work in parallel, which limits the model's speed.

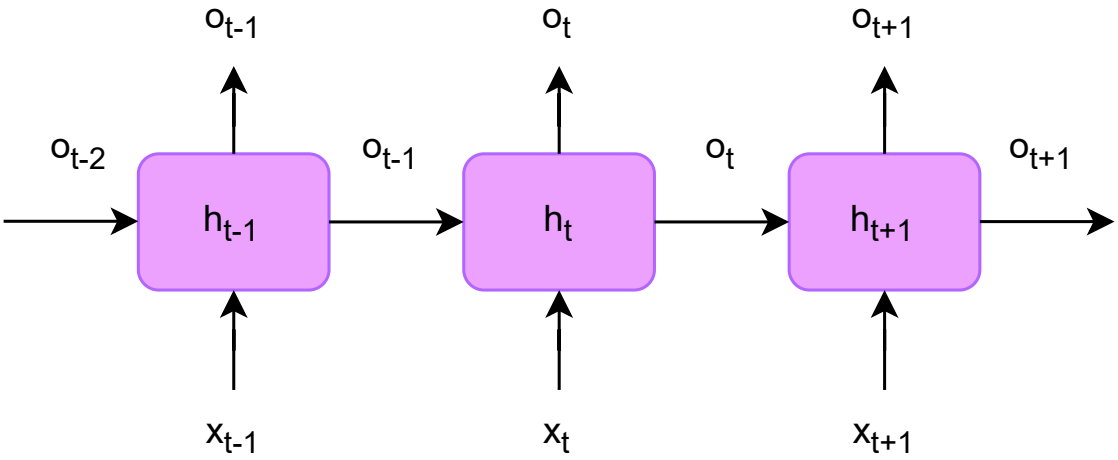


Figure 2.9: Illustration of an RNN where each block corresponds to a time step. The input, output, and the hidden state at time t is given by x_t, o_t, and h_t respectively.

2.13 Transformers

Vaswani et al. [64] introduced the transformer architecture in 2017. It provided a new way to solve language modeling tasks previously solved by more complex RNNs or convolutional NNs (CNNs). Transformers rely only on the attention mechanism, and by using parallelization, they achieved state-of-the-art (SOTA) results on text translation with a lower training cost than previously used models. This section starts by explaining the attention mechanism before describing the transformer architecture. Afterward, the BERT model and a few variations of it will be detailed.

2.13.1 Attention

Bahdanau et al. [1] proposed the attention mechanism in 2014. Intuitively, attention lets the model focus more on important words in the input when processing text. Self-attention was first used in 2016 by Cheng et al. [12]. It is different from attention because attention learns alignments between decoding states and encoded memories, but self-attention has memory and attention *within* an encoder or decoder (more details will be provided about encoders and decoders in Section 2.13.2). Figure 2.10 shows as example of self-attention when a model is reading "I would like to eat the ice cream before it melts". When the word "it" is being read, much attention is put on the words "ice" and "cream" because they are strongly related to "it". The word "to" is less related to "it", and is therefore not given as much attention.

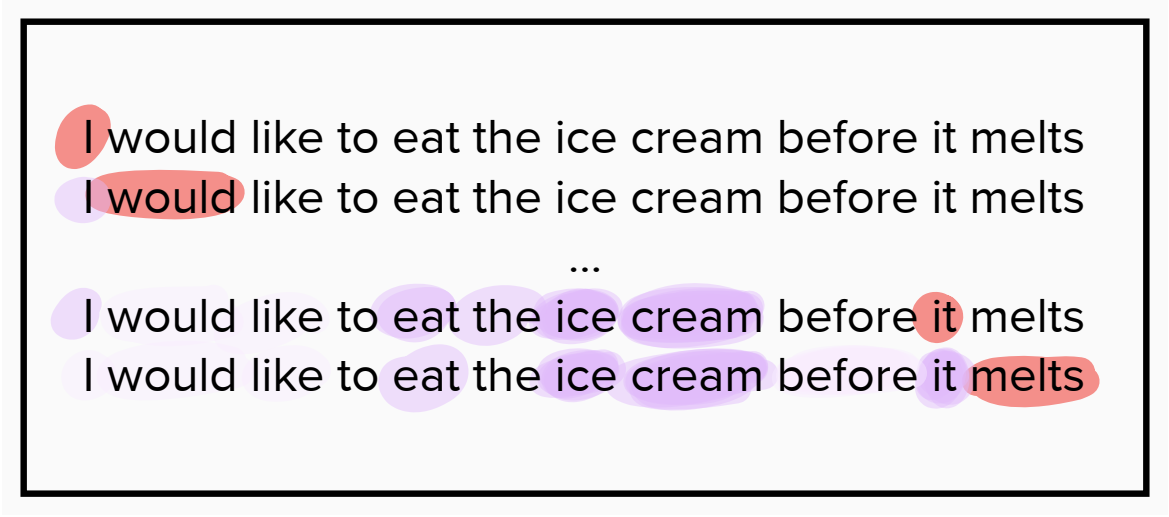


Figure 2.10: An example of self-attention. The red word is the one currently focused on, while the purple ones show how strong the memory activation of each previous word is.

Vaswani et al. [64] introduced the transformer using self-attention. Specifically, scaled dot-product attention was used, which is calculated using the following equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.7}$$

where Q, K and V are queries, keys and values respectively. The matrices Q and K are of dimension d_k, and V is a matrix of dimension d_v. Figure 2.11e visualizes this equation. The output from the softmax function can be thought of as a weight multiplied by V to decide what part of V is the most important.

The paper also proposed using multi-head attention, a technique where h layers, called heads, of scaled dot-product attention are computed in parallel. Figure 2.11d illustrates this. In this architecture, Q, K, and V are linearly projected using different linear projections for each of the h layers. The original dimension of Q, K, and V are kept, so scaled dot-product attention can be performed on the output of each of the h linear projections. This results in h attention matrices, meaning they must be concatenated to one matrix that is afterward projected to give the final results. The following equation summarizes this:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O, \tag{2.8}$$

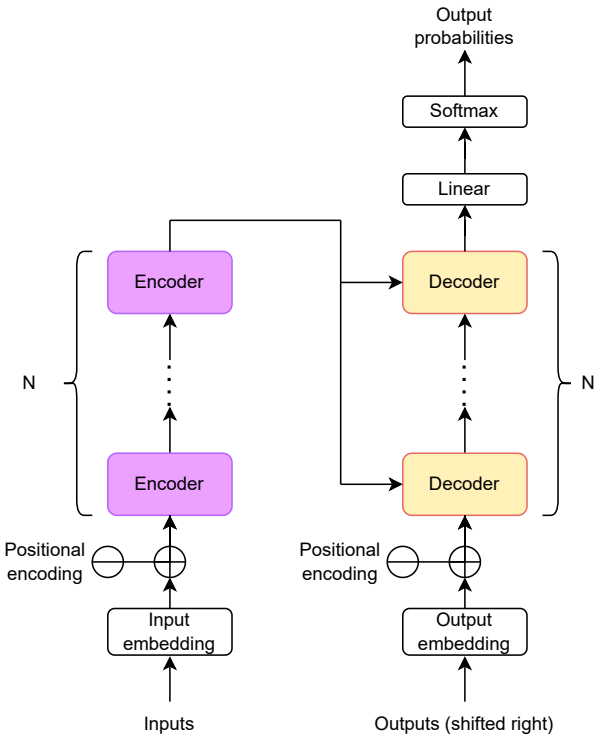
where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

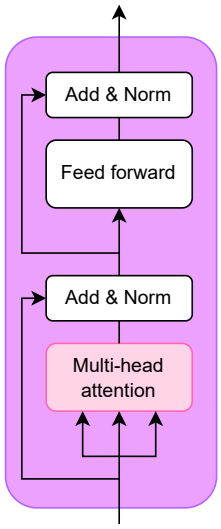
W_i^Q, W_i^K, W_i^V and W^O are the linear projection matrices. The advantage of multi-head attention is that the model can use the information found in different representation subspaces at different positions.

2.13.2 The Transformer Architecture

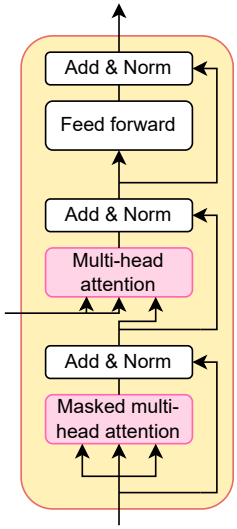
Figure 2.11a shows the transformer architecture created by Vaswani et al. [64]. Encoding is executed on the right side of the figure, and decoding is on the left. An input embedding, which is a vectorized version of the input, is created and added to a positional encoding that says something about the position of the words. This ensures that the final embedding will not be the same if the same word appears in different places in a sentence.



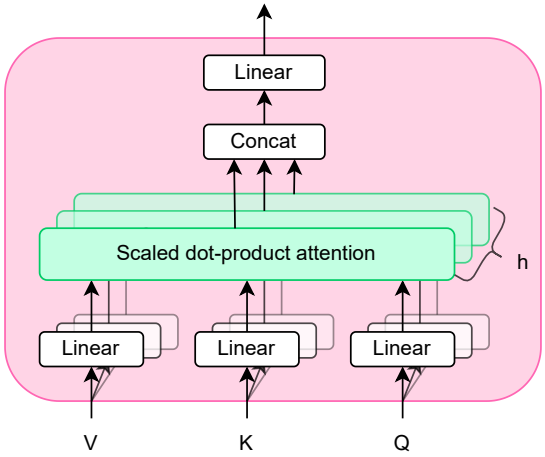
(a) The transformer architecture.



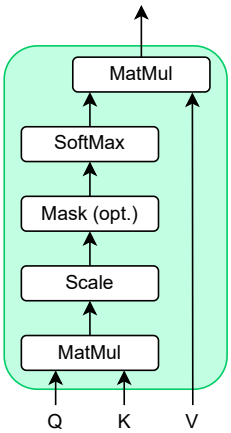
(b) The encoder.



(c) The decoder.



(d) A multi-head attention block.



(e) Scaled dot-product attention.

Figure 2.11: The transformer architecture and its parts as presented in [64].

In Figure 2.11b, the encoder architecture can be seen. Multi-head attention is performed on the encoder's input x_e as explained in Section 2.13.1. The result is added to x_e and normalized, meaning that the first "Add & Norm"-block's output is $y_e = \text{LayerNorm}(x_e + \text{MultiHead}(x_e))$. This is passed to a fully connected feed-forward network before the network's output is added to its input and normalized. In other words, an encoder's output is given by $\text{LayerNorm}(y_e + \text{FeedForward}(y_e))$. A transformer consists of N encoders where one encoder's output is sent as an input to the next. The final encoder's output is passed as input to all decoders.

The decoder architecture can be seen in Figure 2.11c. The first decoder receives a sum of the previous output values (as an embedding) and a positional encoding. From now on, this input is called x_d. First, masked multi-head attention, which means that the future words in the sequence are hidden from the decoder, is performed on x_d before addition and normalization. $y_d = \text{LayerNorm}(x_d + \text{MultiHead}(x_d))$ is then the output from the first "Add & Norm"-block. Afterward, y_d is passed as Q to the next multi-head attention layer, while the final encoder provides V and K. The input to the decoder's final layer, the feed-forward network, is then given by $z_d = \text{LayerNorm}(y_d + \text{MultiHead}(y_d, K, V))$. Finally, the output from one decoder, which is passed as input to the next decoder, can be denoted $\text{LayerNorm}(z_d + \text{FeedForward}(z_d))$.

After the final decoder, the output is passed through a linear layer before a softmax layer is used to output a final vector. This vector's dimension corresponds to the number of possible classes, and each index contains the probability of the corresponding class being the correct answer. The final answer will be the class with the highest probability of being correct.

2.13.3 BERT

Devlin et al. [17] proposed a language model named BERT (Bidirectional Encoder Representations from Transformers) in 2018. This model is illustrated in Figure 2.12. It consists of encoders and can be used for various NLP tasks. Two models of different sizes were created: $\text{BERT}_{\text{base}}$ had 12 encoders and 110 million parameters, while $\text{BERT}_{\text{large}}$ consisted of 24 encoders and 340 million parameters. The model takes one or two texts as input, with a maximum of 512 tokens. When BERT is being discussed, this paper will refer to one of these input texts as a sentence. A sequence refers to the entire input, which is either one or two sentences. For example, when performing question answering, sentence one could be a question, and sentence two a context containing the answer.

Before a sequence is passed to BERT, special tokens are added to it, and the sequence is tokenized and embedded. Two special tokens are used. The first is a classification token noted [CLS], which is added to the beginning of the sequence, and the second token [SEP] is added to the end of each sentence. Afterward, wordpiece tokenization is performed, which results in a vector of numbers as explained in Section 2.10.4. This vector is added to a position embedding, similar to that described in Section 2.13.2, and a segment embedding, a vector added to distinguish the two sentences. An example of BERT's input embedding is shown in Figure 2.13.

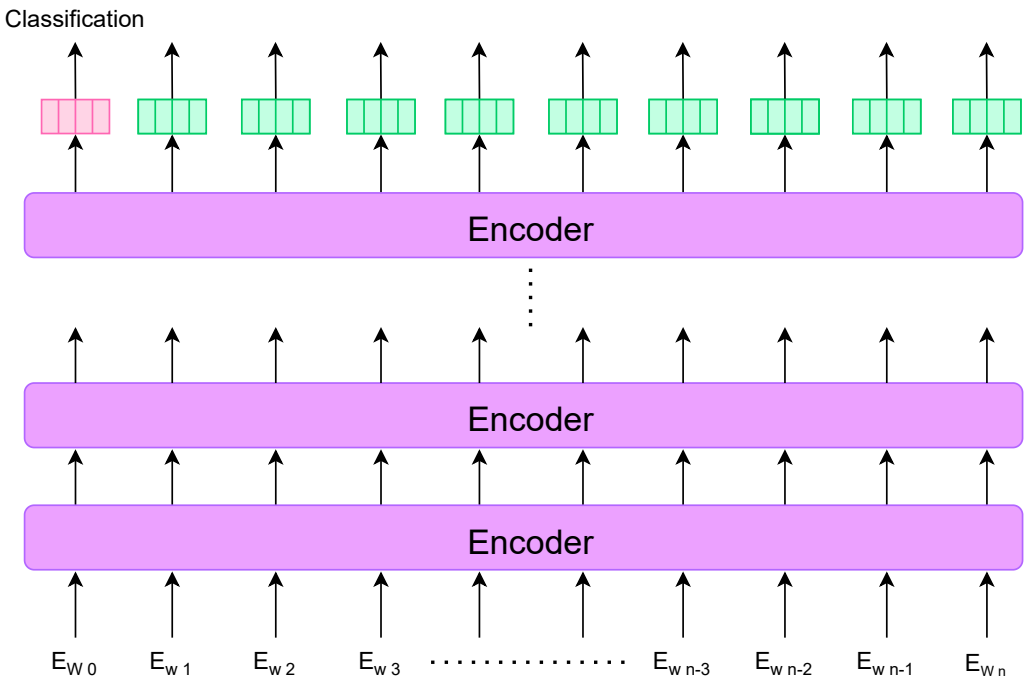


Figure 2.12: The BERT architecture as explained in [17]. The input is embeddings as shown in Figure 2.13. The main part of the BERT model is the 12 or 24 encoder layers that result in a set of output vectors. The first vector (pink) corresponds to the input [CLS] token and is used for classification tasks, while the other (turquoise) vectors are used for token-level tasks.

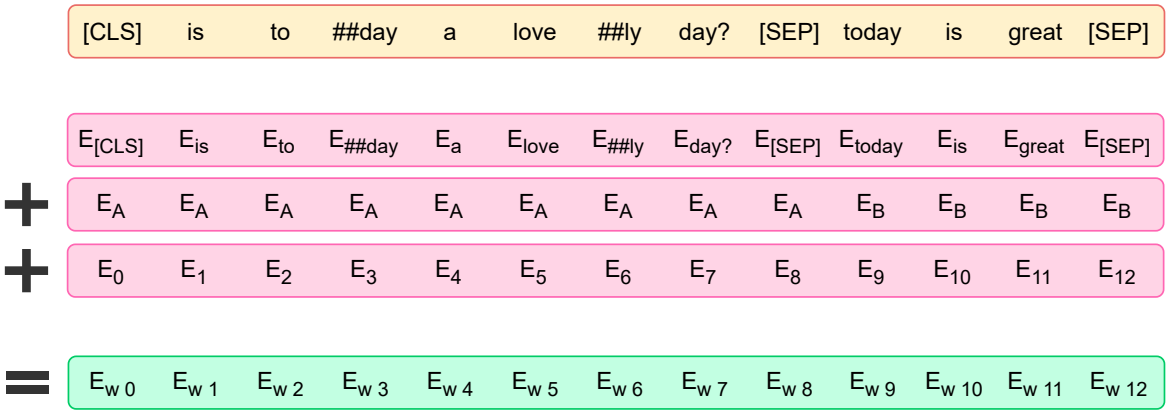


Figure 2.13: An example of the input embeddings sent as input to the BERT model. On top is the individual tokens before they are embedded. The top pink rectangle is the word embeddings, the middle pink is the sentence embeddings, and the bottom pink rectangle is the position embeddings. The green rectangle is the final embedding used by BERT.

BERT performs training in two stages. First, pre-training is done on a large corpus of texts, giving the model a good general understanding of language. During this stage, two techniques are used. The first is masked learning, where some of the words in the input are hidden before being passed into the model, which then attempts to predict them. This makes the model bidirectional because it uses words both to the right and left of the masked word to predict it. The second technique for pre-training is next sentence prediction. The input sequence is given two sentences, and the model attempts to predict if the second sentence follows the first one. The data used by Devlin et al. for pre-training BERT was the BooksCorpus [73] and English Wikipedia.

After pre-training, labeled data is used to fine-tune the model for a specific task, called a downstream task. This process is relatively quick compared to the pre-training. Input-output pairs are handed to the model, and the model parameters are fine-tuned end-to-end by comparing the output generated by the model to the actual values. The input and output format depends on the task, but at most two sentences can be provided as input. For a question answering task, where the first sentence is the question and the second is a text containing the answer, an example sequence can look like this:

> [CLS] How many Harry Potter stories exist? [SEP] Throughout the years, J.K. Rowling has written many books about the Harry Potter universe, and the main series consists of seven books. [SEP]

Document-level SA is another task that would have a different input. Then, the first sentence would be the document to classify, and the second sentence would be empty. An example looks like this:

> [CLS] Although I really like paint-and-sip, last week's event was just not that joyful. [SEP]

After these sequences are embedded, they are passed through the BERT model.

BERT outputs one vector for each input token. These can be used to make the final predictions. How predictions are performed depends on the task. A token-level task, such as question answering, is solved by sending the necessary BERT outputs into separate task-specific output layers, but the exact architecture is task-dependent. Classification tasks like document level SA are solved by sending the output corresponding to the [CLS] token into a final output layer. This layer performs classification. All of BERT's output vectors are illustrated in Figure 2.12. After predictions, fine-tuning is executed by comparing the model's answers to the correct ones. Based on this, all parameters in the model are updated.

2.13.4 Multilingual BERT

Devlin et al. [17] also created mBERT, which is a multilingual $\text{BERT}_{\text{base}}$ model. It was pre-trained on texts in 104 languages, two of which were Norwegian Bokmål and Nynorsk [6]. Wikipedia articles in all of the languages were used to pre-train the model. Besides the training dataset, mBERT is the same as the original $\text{BERT}_{\text{base}}$ model detailed above.

2.13.5 NB-BERT

After the introduction of BERT, it has been shown that BERT models pre-trained on specific languages or topics achieve higher scores than general BERT models [34], [35], [37], [60]. One model created for Norwegian is NB-BERT, created by Kummervold et al. [34] at the National Library of Norway in 2021. They first created a dataset for pre-training, which they named the Colossal Norwegian Corpus. This dataset contained texts from books, newspapers, Wikipedia, and more, most of which were in Norwegian Bokmål and Nynorsk. Kummervold et al. initialized the model with mBERT's weights and continued the pre-training using the created corpus. Pre-training was necessary to adapt the model to Norwegian texts.

2.13.6 NorBERT

Kutuzov et al. [35] also proposed a Norwegian BERT model in 2021 called NorBERT. Norwegian news and Wikipedia articles in both Bokmål and Nynorsk were used for pre-training. NorBERT has a larger vocabulary than NB-BERT, meaning that tokens used by NorBERT better resemble the actual Norwegian words. An example provided by Kutuzov et al. shows that NB-BERT's tokenization splits "kvinnefotballen" into "k ##vinne ##fo ##t ##ball ##en", and NorBERT splits it into "kvinne ##fotball ##en". The same model size as $\text{BERT}_{\text{base}}$ was used.

2.13.7 DistilBERT

Although the BERT models perform well on various NLP tasks, one drawback is their large size, which causes predictions to be relatively time-consuming, and computations require a substantial memory capacity. To reduce these issues, Sanh et al. [55] created a distilled BERT called DistilBERT in 2020. Knowledge distillation is a process where a smaller student model, in this case DistilBERT, learns to replicate a teacher model's behavior, which in this paper was $\text{BERT}_{\text{base}}$. So instead of simply learning the correct class label, it tries to give the same probability to example data as the teacher would. To reduce the model size, DistilBERT removed half of the layers, decreasing the size from 12 to 6, and the parameter size from 110 million to 66 million. The distillation process was performed as part of the pre-training stage of the model, meaning that DistilBERT is a general language model that can be further fine-tuned for specific tasks. Sanh et al. also created a multilingual DistilBERT model, which for the remainder of this paper will be called DistilmBERT.

2.13.8 Strengths and Weaknesses of Transformer-Based Language Models

One of the benefits of transformer-based language models like BERT is their adaptability to various NLP tasks. BERT only requires fine-tuning to learn specific tasks, which is faster than the pre-training phase [17]. This is advantageous in two ways. First, learning a new task could be less time-consuming than training a complex RNN or CNN from scratch, plus parallelization can further reduce the training time. Second, transformer-based models require fewer labeled training examples, which is beneficial as real-world data is often unlabeled. Additionally, these models achieve SOTA scores on multiple tasks, outperforming the previously used methods. Despite these benefits, it should be noted that pre-training these models is quite energy- and time-consuming, making it less ideal to pre-train new models from scratch. Furthermore, due to the millions or billions of parameters, making predictions with these models takes longer than previously used methods like TF-IDF.

2.14 Methods for Handling Missing Labels

Real-world data is often unlabeled. However, the ML models discussed above require labeled data to learn the tasks to perform. If the dataset is large, labeling all examples manually is time-consuming, and other methods might be preferred. Two such methods are transfer learning (TL) and active learning (AL), which are explained below.

2.14.1 Transfer Learning

If the data is unlabeled, one possible method of labeling is TL. Before discussing TL, the term domain must be explained. According to Pan and Yang [50], a domain $\mathcal{D}$ is characterized by a feature space $\mathcal{X}$ and a probability distribution $P(X)$, meaning that it can be represented as $\mathcal{D} = \{\mathcal{X}, P(X)\}$. Two domains can be distinct if the texts are written in different languages or if the content concerns different topics or themes. When solving an ML task, the common approach is to use labeled data from the target domain $\mathcal{D}_\mathcal{T}$ to learn how to solve a task $\mathcal{T}$. Tasks can be sentiment analysis, question answering, or any other ML task. The solution is a function $f_\mathcal{T}$, which approximates the actual distribution of answers. However, if there is no labeled data available from $\mathcal{D}_\mathcal{T}$ to learn $f_\mathcal{T}$, one could instead attempt to use a source domain $\mathcal{D}_\mathcal{S}$ to approximate $f_\mathcal{T}$. If the source and target domains are similar, it is still possible to achieve satisfactory results without any labeled data from the target domain. TL refers to the use of knowledge from one domain to make predictions in another. For instance, if there is no labeled dataset for sentiment analysis of movie reviews, but a labeled dataset exists for hotel reviews, TL can be used. Since both domains may share some words or phrases used to express sentiment, f_{movie} can be approximated by training on labeled data from $\mathcal{D}_{\text{hotel}}$.

2.14.2 Active Learning

AL is a technique used in ML to save time when creating labeled data, which is explained by Settles [56]. Instead of labeling the entire dataset manually, this is only done to a fraction of the examples, and the remaining are labeled by an ML model. The idea is to train the model on the "best" examples to minimize the number of training examples needed to achieve a good performance. Figure 2.14 illustrates the AL process. Initially, a small dataset is labeled and used to train the model. Then, the trained model automatically predicts the labels of the remaining examples. A few automatically labeled examples are selected for manual labeling. One way to choose these samples is through uncertainty sampling, where the examples whose labels the model was least certain of are selected. In a binary classification task, this means selecting the examples where the probability of the label is closest to 0.5. These manually labeled examples can then be added to the set of previously labeled examples, and the model is trained again. This loop continues until you decide to stop. At this point, a final prediction will be performed on the data that has not yet been manually labeled. The method explained here, where the model makes predictions on all unlabeled data before deciding which examples to label manually, is named pool-based AL. AL results in a fully labeled dataset, although not all labels will be correct since many were created using an ML model, not through manual labeling.

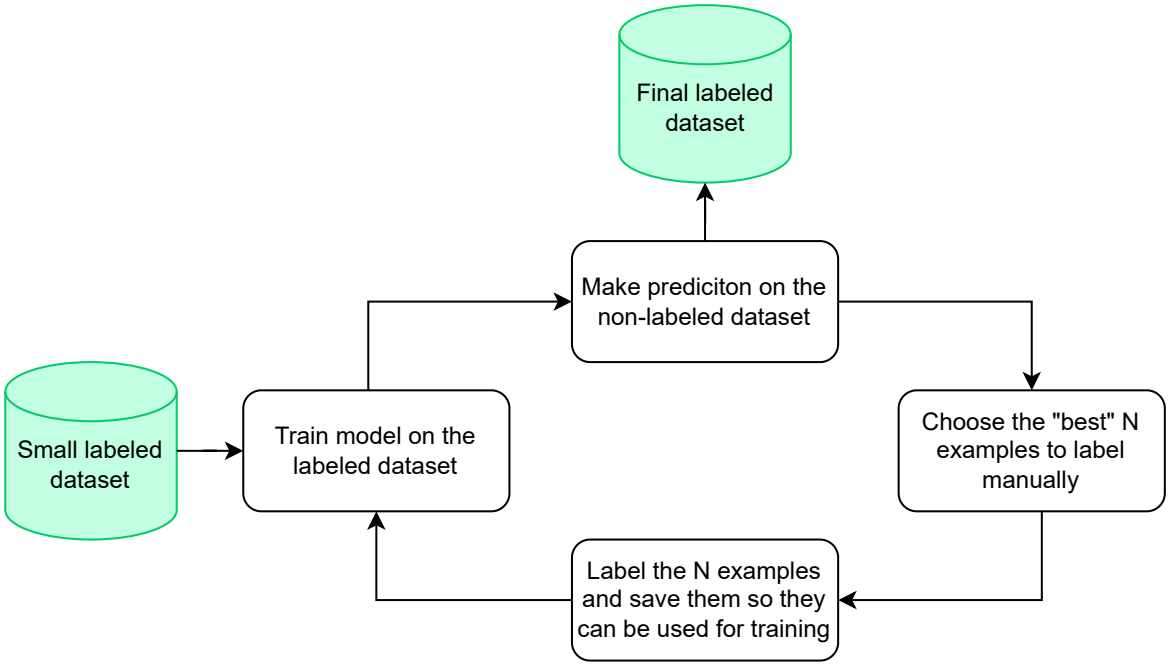


Figure 2.14: An illustration of the AL workflow.

Chapter 3

Related Work

When deciding on which methods to try for the experiments and discussing the results of those experiments, it will be beneficial to have insight into previous work in the field to get an idea of possible solutions and challenges. This section will start by examining previous work on TF-IDF and BERT, before a few papers focusing on TL and AL will be mentioned.

3.1 TF-IDF for Sentiment Analysis

Before transformers were used for SA, TF-IDF in combination with ML models was frequently used. In 2019 Avinash and Sivasankar [41] compared TF-IDF and another vectorization method named Doc2Vec in combination with six different ML models: logistic regression, SVM using RBF or a linear kernel, KNN, decision tree, and Naive Bayes. As this thesis only uses TF-IDF, the results from Doc2Vec will not be mentioned. The models were trained and tested on five datasets of reviews for SA. Table 3.1 summarizes the datasets and the best scores achieved. SVM with an RBF kernel achieved the highest accuracy on three datasets, and logistic regression did best on the remaining two.

Experiments have also been conducted on Norwegian SA using TF-IDF. In 2019 Liu et al. [39] performed SA on NoReC. They tried TF-IDF and two other features, but the other features will not be mentioned here as they are irrelevant to the thesis. TF-IDF was combined with Naive Bayes, logistic regression, SVM, and a NN for predictions. SVM performed the best and achieved the highest area under the ROC curve with a score of 0.8310. This metric gives a score between 0 and 1, and a higher score is better. The next best model was logistic regression, which performed slightly worse, and the NN and Naive Bayes achieved the lowest scores.

Table 3.1: Results found by Avinash and Sivasankar [41] on five different datasets. LR means logistic regression, and the parenthesis behind SVM indicates which kernel was used.

Dataset	**Description**	**Best model**	**Best model's accuracy**
Small movie review dataset [40]	1500 labeled movie reviews from IMDb.	LR	83.428
Sentiment labeled sentences data set [32]	2000 reviews were used from a dataset of reviews from imdb.com, yelp.com, and amazon.com.	SVM (RBF)	82.350
Polarity dataset v2.0 [51]	200 movie reviews were extracted from the full dataset.	SVM (RBF)	98.887
Sentence polarity dataset v1.0 [52]	10662 labeled sentences from rottentomatoes.com.	SVM (RBF)	82.820
Large movie review dataset [40]	50000 labeled movie reviews from IMDb.	LR	88.460

3.2 BERT for Sentiment Analysis

Since the creation of BERT, many experiments have been conducted using various model versions. This section will show some of the results and essential findings from the experiments. Even though document-level SA is the most relevant task for this thesis, some examples of more complex types of SA will also be presented, as much of the research in recent years after the release of BERT has been focused on this, and one part of those tasks is also sentiment analysis.

3.2.1 BERT for Sentiment Analysis in Norwegian

When Kutuzov et al. [35] created NorBERT, they tested the model on various NLP tasks, including SA. Both fine-grained SA (which tries to model the fact that sentiment is directed at entities [48]) and document-level SA were performed. The dataset used for both tasks was $\text{NoReC}_{\text{fine}}$ [48]. However, it had to be altered before document-level SA could be performed. Kutuzov et al. [35] created $\text{NoReC}_{\text{sentence}}$, an aggregated version of $\text{NoReC}_{\text{fine}}$ where each document consisted of one sentence with only one polarity. On this dataset, NorBERT achieved a macro F1-score of 77.1. They also tested NB-BERT on the same dataset and found that using this model resulted in a higher F1-score of 83.9. In fine-grained SA, the F1-score was calculated for identifying the sentiment holder, target, etc., and it was found that NorBERT outperformed NB-BERT on all scores.

Kummervold et al. [34] also tested NB-BERT and NorBERT on the $\text{NoReC}_{\text{sentence}}$ dataset and got the macro F1-scores 86.4 and 81.7 respectively. There was some difference in the results, but they both suggest that NB-BERT is a better model for document-level SA. The differences between [34] and [35]'s results could be due to having selected different examples for training and testing.

Both Kutuzov et al. [35] and Kummervold et al. [34] tested the performance of mBERT on $\text{NoReC}_{\text{sentence}}$, and it achieved an F1-score of 67.7 and 69.7 respectively. In both experiments, mBERT performed significantly worse than NB-BERT and NorBERT. Nonetheless, the scores indicate that the task has to some extent been learned. Kutuzov et al. also tested mBERT on fine-grained SA, where NB-BERT and NorBERT outperformed mBERT on all F1-scores calculated.

3.2.2 BERT for Sentiment Analysis in Finance

BERT has also been used for SA in finance. Sousa et al. [60] used $\text{BERT}_{\text{base}}$ for SA of stock news gathered from various sources such as the New York Times and CNBC. Manual data labeling was performed before fine-tuning of BERT could be done. BERT was compared to multiple ML models, specifically SVM, Naive Bayes, and a CNN. They tried both BoW and TF-IDF as vectorization methods for Naive Bayes and SVM. The input to the CNN was word embeddings obtained from fastText [10]. BERT achieved an F1-score of 0.725, outperforming the second-best model, the SVM combined with BoW, by 0.124 points.

Since research has shown that language models pre-trained on a specific domain perform better than general language models [3], [37], Yang et al. [69] created a $\text{BERT}_{\text{base}}$ model pre-trained on financial data named FinBERT. Analyst reports, corporation reports, and transcripts from conference calls were used for training. Another step towards creating a more task-specific model was to make a new vocabulary named FinVocab. They created FinBERT with the original BERT vocabulary, cased and uncased, and FinBERT with FinVocab, cased and uncased. These four models and the original BERT model were tested on three financial datasets for SA, and it was found that all FinBERT models outperformed the original BERT, and the best model was the uncased FinBERT using FinVocab.

In 2020 Mishev et al. [43] compared multiple methods for SA in finance. Among the methods tested were TF-IDF and pre-trained transformer-based models. The models were evaluated on a dataset containing 1093 positive and 1093 negative sentences. TF-IDF was followed by SVM, a dense layer, or an XGB classifier, and SVM achieved the highest score with an F1-score of 0.836. Among the transformer-based models, BERT, FinBERT, $\text{BART}_{\text{large}}$ [38], and more were tried. All provided better results than TF-IDF, in the range [0.862, 0.947], and $\text{BART}_{\text{large}}$ was the best. Some other scores worth noting are $\text{BERT}_{\text{base}}$ cased which scored 0.890, $\text{BERT}_{\text{large}}$ cased with 0.928, and FinBERT with an F1-score of 0.893. The paper also pointed out that financial data can have a different vocabulary than other data, so models adapted to this domain will, in many cases, be the best solution.

3.2.3 Distilled Models

In recent years distilled language models have been created to save time and computing resources when performing NLP tasks. This section will present the results from two papers comparing distilled language models to their larger counterparts. One paper compared DistilBERT to $\text{BERT}_{\text{base}}$, while the other compared DistilmBERT to mBERT and a couple of other multilingual models. Although they were tested on multiple NLP tasks, this section will focus on results on SA tasks, as they are most relevant for this paper.

When Sanh et al. [55] created DistilBERT, they tested the model on various tasks and compared it to $\text{BERT}_{\text{base}}$. They first performed a comparison on the GLUE benchmark [66], which is a collection of various NLP tasks, one of which is SA. The SA task uses the SST-2 corpus [58] of movie reviews for evaluation. On this task, DistilBERT achieved a score of 91.3%, 1.4% below the larger $\text{BERT}_{\text{base}}$. DistilBERT was also tested on other downstream tasks, one of which was SA of the IMDb dataset [40]. On that dataset, DistilBERT got an accuracy of 92.82%, 0.64% lower than $\text{BERT}_{\text{base}}$. Sanh et al. also looked at the inference time on the GLUE task STS-B and found that DistilBERT was almost 40% faster than $\text{BERT}_{\text{base}}$.

In 2020 Kittask et al. [30] examined the performance of four multilingual language models on Estonian NLP tasks. The models used were mBERT, DistilmBERT, XLM-100 [36] and XLM-RoBERTa [16]. The Estonian Valence corpus [49], which contains 4088 labeled paragraphs from the newspaper Postimees Daily, was used for SA. Two different sequence lengths of 128 and 512 tokens were tried. When the sequence length was 128, DistilmBERT had an accuracy of 65.95%, and mBERT had an accuracy of 70.23%. When the sequence length was increased to 512, they scored 66.95% and 69.52%, respectively. This means that mBERT outperformed the distilled model in both cases. Kittask et al. also found that both XLM models achieved higher accuracies than mBERT.

3.3 Label Generation

This section will examine some previous work for generating labels. Since the experiments performed during this thesis utilized both TL and AL, work involving both of these techniques will be mentioned.

3.3.1 Transfer Learning

In 2019 Myagmar et al. [45] looked into the performance of two transformer-based models for SA on the Amazon review dataset [9], one of which was BERT. The dataset contains reviews in five categories, and the models were trained on one category and performed prediction on all the others. This resulted in 20 cross-domain SA tasks.

BERT was compared to several baseline models that were previously shown to give good results, which were based on CNNs, adversarial networks, or attention networks. With an average accuracy of 92.61%, BERT outperformed the baseline models on all tasks. However, it is worth noting that the other transformer-based model tested here, a model trained on English text named XLNet [70], outperformed BERT on all tasks, showing that there have been further advancements in the field of language models after the creation of BERT. In 2020 Du et al. [19] also used BERT for cross-domain SA on the Amazon review dataset. They tested the original BERT and compared it to four other BERT models whose cross-domain capabilities they had tried to improve using new techniques. They found that the changes did increase the score when the models were tested, showing that simply training BERT on one domain and testing it in another will not provide optimal results.

Although TL could provide satisfactory results, certain issues might hinder performance. Al-Moslmi et al. [44] performed a literature review focused on previous work in the field of cross-domain SA. It pointed to some critical challenges one might face when performing TL with insufficient training in the target domain. Specifically, four problems were mentioned. The first one, sparsity, occurs when essential words or phrases often occur in the target domain but rarely in the source domain. Secondly, polysemy is the issue where the meaning of a word might change from one domain to another. Feature divergence is the third problem, where the features a model learns for a source domain do not perform well on the target domain. Finally, polarity divergence is the term for words with one sentiment in the source domain and another in the target domain.

3.3.2 Active Learning

AL has been used successfully in the past. Hoi et al. [28] proposed a method for pool-based AL where multiple examples were chosen for manual labeling at every iteration. They pointed out that by selecting more than one example at each iteration, several similar and possibly identical examples might be chosen, causing some of the training to be redundant. To mitigate this problem they used the Fisher information matrix, which represents the uncertainty of the classification model. Their goal was to choose examples that minimized the Fisher information, and this method was combined with a logistic regression model. The method was compared to two other models, logistic regression and SVM, that chose the next examples to label based on the uncertainty of the current label. They were tested on three text classification datasets with a total of 27 classes, and the proposed model outperformed the others on 23 classes.

The issue pointed out by Hoi et al. [28], that many emails chosen for manual labeling might be similar, has also been examined by others. Farquhar et al. [20] mentions that when AL is performed, only some of the examples are labeled manually, and these examples might not match the distribution of the original dataset. As explained in the paper, this leads to a sampling bias which causes an error when optimizing the model during training, as the training objective is wrong. This is also related to the issue of overfitting. Since only a fraction of the data is used to train the model before

labeling, the model might not be able to generalize what it learns to the rest of the dataset because it might look too different. Chen et al. [11] and Wang et al. [67] both considered the issue of overfitting in their work. Chen et al. found that some examples hurt the performance of the AL process and suggested that those examples caused the model to overfit more than others. Wang et al. pointed to overfitting being an even more significant problem if there are initially very few labeled samples, and they looked into using a pseudo-validation dataset to mitigate this problem.

Another possible issue with AL is the cold-start problem. When performing AL, we often start with no labeled data and no appropriate model to help decide which examples to label first. Zhang et al. [72] summarized some possible methods previously used. Random sampling is the most common, and given a large dataset, it should keep the original label distribution. A second way to get a better starting point is by using TL and starting by using a model trained on another dataset for labeling.

Yuan et al. [71] focuses on AL with transformer-based models. They specify that with today's transformers, training on thousands of examples is not desirable because it requires too much computational resources. They highlight the findings of Guo et al. [25] that confidence scores (the output probabilities) in today's NNs are not particularly precise. Therefore, instead of using the normal uncertainty from a model, Yuan et al. utilized the masked language modeling loss from a pre-trained BERT model to mitigate the cold-start problem. That loss is used to help choose the examples to label first. Yuan et al. also take into consideration that the examples chosen should be representative of the whole dataset, and uses clustering to avoid labeling too similar example texts.

Chapter 4

Data

For the experiments conducted here, two datasets were used. The first dataset is the NoReC dataset created by Velldal et al. [65] in 2018, whereas the second dataset consists of emails from customers sent to Sparebank 1 SMN. This section will provide an overview of the original datasets and the final datasets after preprocessing was performed. The preprocessing will be further outlined in Section 5.

4.1 The NoReC Dataset

The NoReC dataset [65] consists of Norwegian reviews, each with a score from 1–6. It is publicly available and can be downloaded from Github[1]. [65] and [46] explain that two versions of the NoReC dataset have been made, the first containing 35,000 reviews and the second containing an additional 8425 reviews. Therefore, the current dataset contains more than 43,000 reviews, each belonging to one of the following categories: screen, music, literature, products, games, restaurants, stage, sports, or miscellaneous. When the dataset was downloaded, it was already split into three datasets: one training, validation, and test dataset. The distribution of scores in these three datasets can be seen in Figure 4.1.

To match the purpose of this thesis, the scores were converted into 0 for negative and 1 for positive or neutral reviews. For the rest of this paper, positive and neutral examples belong to the same class, called the positive class. Four NoReC datasets were created: Two training datasets, one validation dataset, and a test dataset, all of which have been summarized in Table 4.1. Since Sparebank 1 SMN was primarily interested in finding negative texts, and the purpose of the NoReC dataset was to get insight into a similar problem with an openly available dataset, reviews labeled as 3 or 4 were set as 1 for positive. These reviews will probably contain positive and negative sentiments and can therefore be considered neutral. As the NoReC dataset contains relatively few reviews with a score of 1 or 2, the datasets were originally highly imbalanced. Since training on an imbalanced dataset could make the models bad at detecting the minority

[1] https://github.com/ltgoslo/norec

class, in this case the negative class, it was decided that the training dataset should be balanced. Therefore 2000 negative and 2000 positive examples were extracted from the training dataset and combined into the final training dataset $\text{NoReC}_{\text{train\_full}}$. An additional balanced training dataset named $\text{NoReC}_{\text{train\_no\_neutral}}$, where the neutral reviews were dropped, was also created. The distribution of scores in the balanced training datasets can be seen in Figure 4.2. The validation and test datasets remained unchanged and imbalanced to reflect the real-world data better.

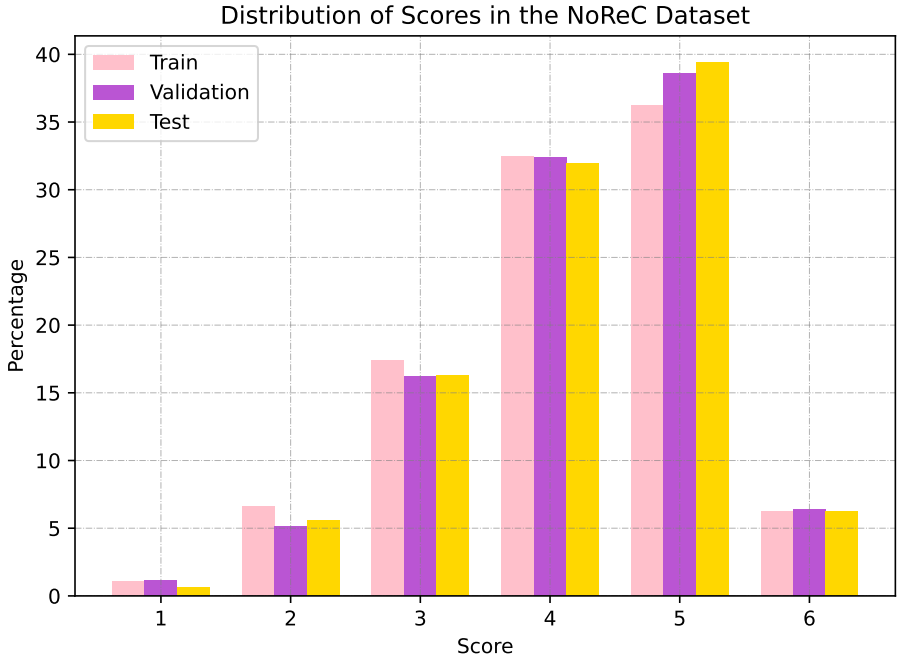
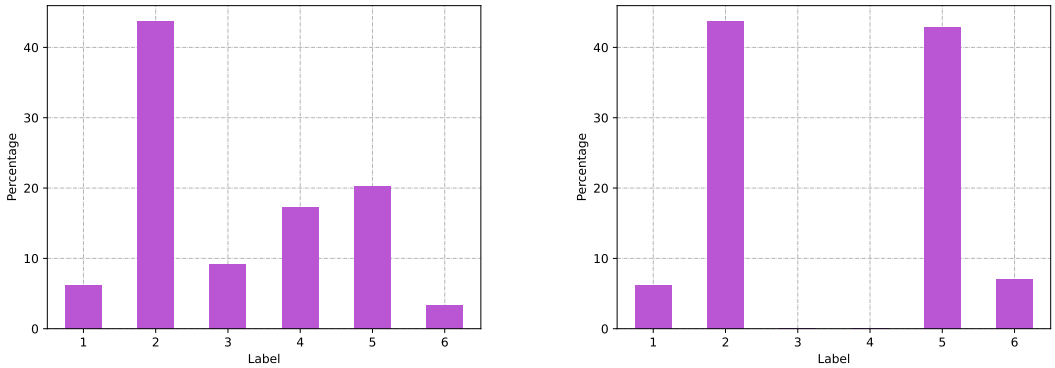


Figure 4.1: Bar chart illustrating the distribution of scores in the three NoReC datasets before they were converted to 0 or 1.

Table 4.1: Overview of the NoReC datasets used for this task. The columns named #1 and #0 show the number of positive and negative examples in the datasets.

Dataset name	**#1**	**#0**	**Description**
$\text{NoReC}_{\text{train\_full}}$	2000	2000	Label 0 corresponds to 1 or 2 in the original dataset, and 1 indicates a score of 3–6. This dataset is for training language models.
$\text{NoReC}_{\text{train\_no\_neutral}}$	2000	2000	Label 0 corresponds to 1 or 2 in the original dataset, while 1 indicates a score of 5 or 6. The "neutral" reviews were dropped. This dataset is for training language models.
$\text{NoReC}_{\text{val}}$	4084	276	This dataset is used for hyper-parameter tuning of the BERT models.
$\text{NoReC}_{\text{test}}$	4081	270	This dataset is used for testing the model after training is performed and will be used to decide which model performs best.



(a) $\text{NoReC}_{\text{train\_full}}$ (b) $\text{NoReC}_{\text{train\_no\_neutral}}$

Figure 4.2: Bar charts illustrating the distribution of scores before turning all scores to 0 or 1 in the two balanced NoReC training datasets.

4.2 The SMN Dataset

A dataset of emails sent from customers to Sparebank 1 SMN was provided for this thesis. Due to privacy reasons, the dataset is not publicly available. However, this section will explain the dataset's features relevant to the assignment, show example data, and highlight some important statistics regarding the final dataset.

Here, only the columns relevant to this task will be explained. Table 4.2 gives a quick summary of the three columns used. Two columns were used to filter irrelevant emails from the dataset during preprocessing, namely sender_domain and subject. sender_domain is one of a limited list of common email domains used by customers (such as @gmail.com and @hotmail.com). When the dataset was provided, all emails from other domains had already been removed, ensuring that, for example, emails from other employees at the bank were not added to the dataset. How these two columns were utilized for further preprocessing will be explained in more detail in Section 5.2.2.

Table 4.2: An overview of the columns from the Sparebank 1 SMN dataset used during this project.

Column	**Description**
description	Contains the raw HTML email sent to Sparebank 1 SMN.
sender_domain	States the sender's email domain, such as @gmail.com. It is used to separate customer emails from forms being filled out by customers.
subject	The subject of the email. If a form was filled out, the customer could choose between a set of predefined subjects, such as "kort og betaling" (card and payment). Used during preprocessing.

The most interesting column was the description, which contains the text to classify. The descriptions were initially formatted as HTML, and an example description can be seen in Appendix A. As HTML is not useful for the model, the emails themselves had to be extracted from the description. One description could also contain multiple emails going back and forth between the customer and an employee at the bank. However, neither the HTML format nor the extracted texts offered a consistent way to separate the emails. Nonetheless, the first email from each description had to be extracted since it was from a customer, and how this problem was solved is explained in Section 5.2.2. Some additional preprocessing is also detailed, resulting in a dataset consisting of single emails sent from customers to the bank. For the remainder of this thesis, an email is a single email of pure text extracted from the HTML, while a description is the raw HTML. Since the emails are not publicly available, example emails can not be provided. However, a word cloud consisting of the most common words in the emails, after stop word removal and stemming, is shown in Figure 4.3 to give an idea of common topics. It should also be mentioned that some emails were written in English.

Figure 4.3: Word cloud of the customer emails in the SMN dataset.

Two other things worth noting about the description column are that the content is all uppercase letters and that, due to privacy concerns, some information was removed before the data was provided. Specifically, names were replaced with [PERSON], email addresses with [EPOST], and all numbers with more than three digits were replaced with [SIFFER]. [PERSON] has, in some cases, also replaced words other than names, such as HILSEN (BEST REGARDS), and which words have been replaced is unpredictable. Section 7.4 will discuss how this might have affected the results.

To perform SA on the dataset, labels are needed for training and evaluation. These labels were not present in the dataset, and had to be created. The process used to label the dataset for SA is detailed in Section 5.3, but some statistics about the final dataset after labeling, from now on called the SMN dataset, is presented here. The dataset contains 12490 emails, of which 400 were used for label generation. 2191 of the

remaining 12090 examples were labeled as negative, and the other 9899 examples were labeled positive. The SMN dataset was then split into a training, validation, and test dataset from now on named $\text{SMN}_{\text{train}}$, SMN_{val}, and SMN_{test}, respectively. 80% of the emails were extracted for training, and many positive emails in the training dataset were removed to balance it. The remaining 20% of emails was again split in two, and 80% was used for SMN_{test} and the rest for SMN_{val}. These datasets were stratified, meaning the proportion of positive and negative emails is the same in both datasets. All datasets are summarized in Table 4.3 and Figure 4.4.

Table 4.3: Overview of the SMN datasets used for this task. The columns named #1 and #0 show the number of positive and negative examples in the datasets.

Dataset name	**#1**	**#0**	**Description**
$\text{SMN}_{\text{train}}$	1753	1753	This dataset is for training language models, and because the dataset originally contained more positive than negative emails, it has been balanced.
SMN_{val}	396	88	Dataset used to evaluate at which epoch the BERT models perform best.
SMN_{test}	1584	350	Dataset used for the final evaluation of which model is the best.

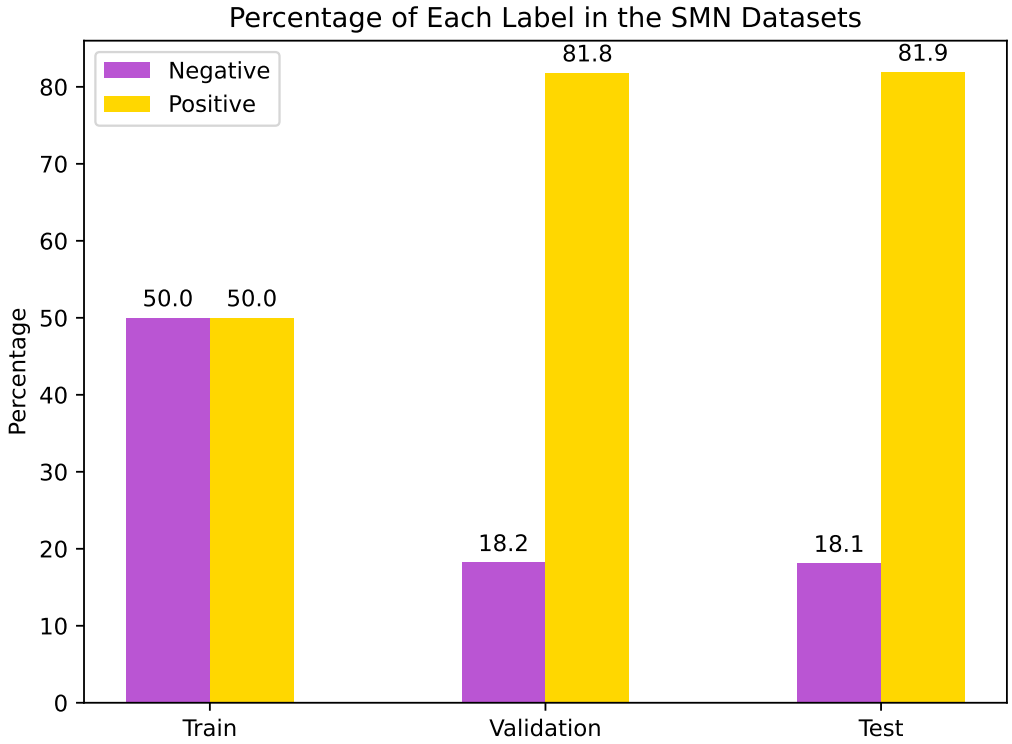


Figure 4.4: A visualization of how many percent of the emails in $\text{SMN}_{\text{train}}$, SMN_{val}, and SMN_{test} belongs to the positive and the negative class.

Chapter 5

Method

In Section 1.2, the research questions were defined as follows:

RQ1 What are the state-of-the-art techniques for sentiment analysis in Norwegian?

RQ2 Can active learning be used to create sentiment labels on customer emails?

RQ3 How do different transformer-based language models compare to each other when used to solve sentiment analysis tasks? And how do transformer-based language models compare to previously used methods for sentiment analysis?

This section will explain the method and experiments used to answer **RQ2** and **RQ3**. **RQ1** will not be mentioned since it was researched through a literature study, not experimentation. First, the tools and libraries used for this project will be presented. Afterward, the preprocessing of the datasets will be explained, followed by detailed descriptions of how the experiments related to **RQ2** and **RQ3** were executed. The code written for this thesis can be found on Github[1].

5.1 Tools and Libraries

Python was used for these experiments, as it provides many packages for working with ML. Hugging Face [29] is a resource that was also used much. It provides datasets and language models for NLP tasks, including the BERT models used during this project. The website also shows code examples for how to work with the libraries and resources they provide. An overview of all libraries used during this project is given in Table 5.1.

[1] https://github.com/Karolill/master_thesis/tree/main
[2] https://beautiful-soup-4.readthedocs.io/en/latest/
[3] https://huggingface.co/docs/datasets/index
[4] https://huggingface.co/docs/evaluate/index
[5] https://matplotlib.org/
[6] https://www.nltk.org/

Table 5.1: The Python libraries used for the experiments. General information about the library and specific use cases for this project are provided.

Library	Information
`BeautifulSoup`[2]	For extracting text from HTML. Here as part of the preprocessing.
`datasets`[3]	Hugging Face library used for loading datasets from the Hugging Face hub or local files, and creating a format that can be used when training BERT models.
`evaluate`[4]	Hugging Face library that provides multiple evaluation metrics such as F1-score, precision and more.
`matplotlib`[5]	This is for visualizing data and was used to plot results.
`nltk`[6]	Natural language toolkit is a library that has methods for working with natural language and supports Norwegian. It was used for preprocessing of text.
`numpy`[7]	For working with arrays and computation.
`pandas`[8]	A python library that often uses DataFrames for storing and working with datasets.
`re`[9]	Library for using regular expressions. Here used for preprocessing of data.
`scikit-learn`[10]	A library for ML tasks. It is used here to create various baseline models, parameter tuning, and calculate test scores. It is also used to create confusion matrices. It is often called `sklearn`.
`torch`[11]	Pytorch is necessary for the `transformers` library to work and to run code on GPU.
`transformers`[12]	Hugging Face library for loading and training pre-trained models such as BERT. It is also used for tokenizing data.

5.2 Preprocessing

Before the datasets could be used for SA, preprocessing had to be executed. Since the datasets were provided in different formats, the process differed for them. This section will first explain the preprocessing performed on the NoReC dataset before the same is done for the SMN dataset.

[7] https://numpy.org/
[8] https://pandas.pydata.org/
[9] https://docs.python.org/3/library/re.html
[10] https://scikit-learn.org/stable/
[11] https://pytorch.org/
[12] https://huggingface.co/docs/transformers/index

5.2.1 The NoReC Dataset

As mentioned in Section 4.1, the NoReC dataset was fetched from Github. The texts did not require any preprocessing, but they had to be matched to a score saved in a separate JSON file containing metadata. When this was done, the scores were converted to either positive (1) or negative (0), as explained in Section 4.1. The texts and corresponding sentiment could then be converted to a `pandas` DataFrame, which was saved in a CSV file to make it easily accessible for other parts of the code.

5.2.2 The SMN Dataset

The description column in the SMN dataset that would be used for SA contained emails in an HTML format which could not be passed to the language models. Therefore, preprocessing was performed, and Figure 5.1 summarizes the steps.

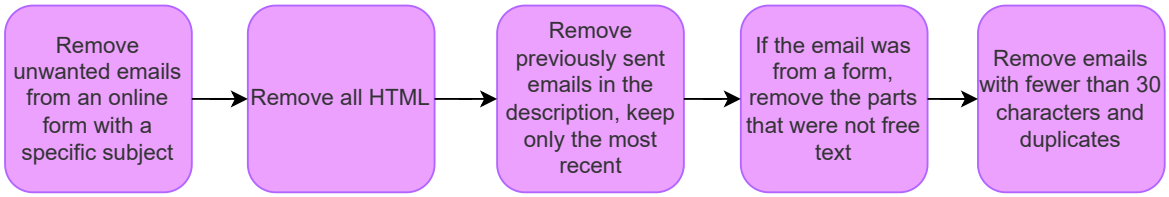


Figure 5.1: The preprocessing steps performed on the SMN dataset.

The first step was to remove possibly unwanted rows. After looking at the data, it was found that if the email was created through an online form, which could be seen by the sender_domain column, and contained a specific subject, the description mainly consisted of answers to yes/no or multiple-choice questions, and little to no free text. Therefore these rows were dropped from the dataset.

The next step was to extract the emails from the HTML in the descriptions. This was quickly done using `BeautifulSoup`. As mentioned in Section 4.2, one description might contain multiple emails in a conversation, so they had to be split. Since neither the HTML nor the text within it seemed to have a consistent way to separate the emails, a few phrases in the text were chosen to split on. These were typical phrases found between separate emails, such as “FROM:”, “SUBJECT:”, “SENT FROM MY IPHONE”, or similar phrases in both Norwegian and English. Only the first email in each description was kept since it was the new email, and older ones could exist in a different row in the dataset.

Still, some emails contained unnecessary text, which was removed. First phrases such as “MED VENNLIG HILSEN” (best regards) were removed as they say little about the sentiment and are ofte written due to habit. Second, if the text “SKJEMATITTEL” was present in the email, it was a form containing some but not only free text. In this case, the free text messages had to be extracted, which was done by identifying which strings surrounded them, and afterward removing the unnecessary text based on this. There seemed to be three possible surrounding strings based on which form was being filled out, which made it straightforward to write the code for the extraction.

Ultimately, emails with fewer than 30 characters and duplicate emails were deleted. Short emails were deleted as they were partially the results of an imperfect preprocessing and could, for example, consist of only a date or some other phrase indicating that there was no interesting text in the email. Duplicates were removed since the language models must see a variety of texts during training, not the same text repeatedly. That could cause overfitting, which is unwanted. This concluded the preprocessing, and AL could then be performed on the emails.

5.3 Research Question 2 - Active Learning

After the emails in the SMN dataset were correctly formatted, labels had to be set. For this purpose, AL was performed because it would be very time-consuming to label all texts manually.

The AL process resembled the method detailed in Section 2.14.2, but the start differed. First, an NB-BERT model already trained on the NoReC dataset was used as a starting point, meaning that TL was used in an attempt to achieve better results faster. Afterward, AL was performed the following way: First, predictions were made using the NB-BERT model mentioned above. Then the 50 most uncertain examples were corrected manually and added to the training dataset. This dataset was used for training the model for three epochs, so it was ready for another round of predictions. This was repeated until 300 examples had been labeled and used for training. At this point, it was decided to label 100 examples in each round for another two rounds. Manually labeled texts were always added to the dataset of previously manually labeled emails. In total, 500 examples were manually labeled. However, after some inspection, it was found that the predictions made by the model trained on 400 examples seemed better than those created by the model trained on 500 examples. Therefore, the predictions made by the model trained on 400 examples were used as the final predictions.

To be consistent during labeling, it was attempted to follow some general guidelines for what label to set. Emails that were negative for the bank would be labeled as negative. This could, for example, be if the customer wanted to end part of or the whole customer relationship. Emails indicating poor customer service were also marked as negative. This included emails where it seemed like the customer had been waiting longer than expected for an answer, indications of too little information being provided to the customer, or clear expressions of unsatisfactory behavior from the bank. Otherwise, if emails seemed neutral or positive, just asked a question, or if the customer was not pleased with something unrelated to the bank, it was marked as positive.

When the labels were decided, the 50 most certain and uncertain positive emails and the 50 most certain and uncertain negative emails were sent to employees at Sparebank 1 SMN to get feedback on the quality of the AL process. When the employees corrected the labels, they only looked at the sentiment, not the email's effect on the bank. So a neutral email where a customer wanted to end their insurance would have been labeled negative during the AL process, but it was said to be positive by the SMN employees. There were 11 cases where examples would have been labeled as negative during the

AL process, but the employees meant they should be positive. It was decided to override the employees' decision on these examples, as it would be beneficial to know when an email negatively impacts the bank and because it was necessary to evaluate the AL process properly.

It was attempted to add the 200 corrected examples to the 400 previously used training examples created during the AL process and do one more round of training. However, it was observed that this caused almost no emails to be labeled as negative, meaning that the labeled dataset was worse than the one trained on 400 examples and corrected by the Sparebank 1 SMN employees. Due to this observation, it was decided to use the dataset trained on 400 examples as the final SMN dataset. This is the dataset detailed in Section 4.2.

5.4 Research Question 3 - Language Models

To answer **RQ3**, a set of baseline models and four BERT models were trained and evaluated on the NoReC and SMN datasets. Formally, the problem this experiment attempts to solve can be defined as:

$$f(x|\theta) = y \in [0, 1] \tag{5.1}$$

where x is an input text, θ is a set of parameter values, and f is either a BERT model or a baseline model. The closer y is to 0, the more negative x is, and the closer it is to 1, the more positive it is.

Section 5.4.1 will explain how multiple baseline models were created and tested before a final one was chosen to compare to the BERT models. Afterward, Section 5.4.2 will detail the creation of the BERT models. Finally, the utility of incorporating these models at Sparebank 1 SMN was qualitatively evaluated, and how this was done will be explained in Section 5.4.3.

5.4.1 Baseline Models

To get an idea of how well the BERT models should be able to perform, baseline models were created for comparison. The overall architecture of the baseline models can be seen in Figure 5.2. The text is normalized, TD-IDF is performed, and classification is executed using an ML model. The method to create baseline models was the same on the SMN and NoReC datasets, and for the NoReC dataset, both $\text{NoReC}_{\text{train\_full}}$ and $\text{NoReC}_{\text{train\_no\_neutral}}$ were used (separately) to train the baseline models to examine what dataset resulted in the best performance.

The process of creating a baseline model starts with a dataset of texts in natural language. The dataset is first normalized, as this is necessary to do before passing the text to TF-IDF to achieve better performance. The normalization process starts by making all letters lowercase before removing non-letter characters. This means only

the lowercase letters a–å and spaces were left in the texts. Multiple spaces in a row were turned into a single space, as multiple spaces after each other do not express sentiment. To remove stop words, `nltk.corpus.stopwords.words('norwegian')` was used, which provided a list of Norwegian stop words to remove from the texts. Finally, stemming was performed using the `nltk NorwegianStemmer` object. The resulting output was the normalized text which could be sent to TF-IDF. The preprocessing steps are also listed and illustrated with an example in Figure 5.3.

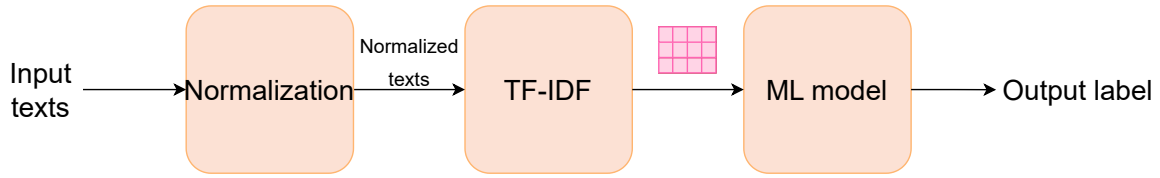


Figure 5.2: The general architecture of the baseline models. Input texts is a list of strings in natural language (the raw text). Each row in the pink matrix is a TF-IDF vector that can be passed to an ML model for classification.

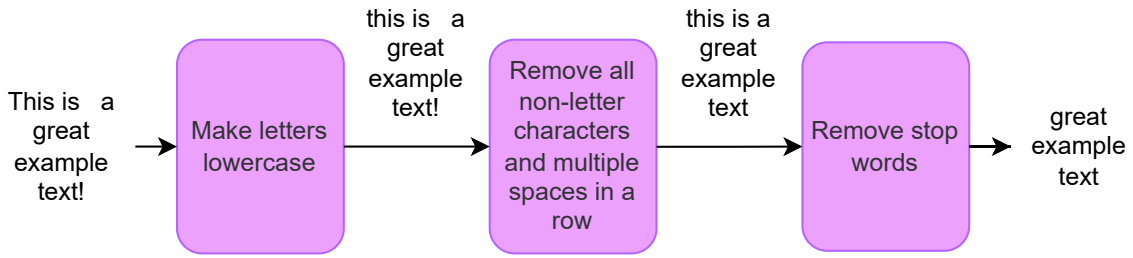


Figure 5.3: The normalization process for the baseline models, including an example.

As ML models can not take text as input for a classification task, feature extraction was performed. Since, as explained in Section 2.10.3, TF-IDF assigns more weight to words characteristic of a document than a frequency vector does, this method was utilized for the baseline model. The resulting vector was passed into the ML models. The models used were SVM, logistic regression, and KNN. They were imported from `scikit-learn` through the objects `SVC`, `LogisticRegression`, and `KNeighborsClassifier` respectively. For each model, a grid search was performed for hyper-parameter tuning, using the `sklearn GridSearchCV` object to find the best parameters. Which parameters were tuned and which parameter values were tried for each model can be seen in Table 5.2. Finally, each model could be trained with the optimal parameters found by the grid search, which can be seen in Table 5.3, and the final models were used for predictions.

After predictions were made, the `sklearn classification_report()` function calculated each model's precision, recall, and F1-score, making them easy to compare. In this project, several metrics are relevant. Therefore, using a function that returns all of them is practical. However, the most important feature, that will be given the most weight when evaluating the results, is the F1-score of the negative class. It is the most important score because it indicates the model's ability to predict the negative class

correctly, and it is more important for the bank to discover negative emails than positive ones. The precision of the negative class will also impact what model is considered the best, as it states how many positive examples are misclassified as negative, and for the bank, having many emails marked as negative when they are positive is also not desirable. In addition to the metrics, confusion matrices were created using `sklearn` to visualize the results. In all confusion matrices made during this project, the rows were normalized due to the datasets' class imbalance, and normalization made it easier to compare the performance on the two classes. In the end, the best-performing model on $\text{NoReC}_{\text{test}}$ and the best model tested on SMN_{test} were chosen as the baselines to compare to the BERT models trained on their respective datasets.

Table 5.2: A list of all ML models trained and the hyper-parameters tuned during the grid search for the best baseline models. LR is logistic regression.

Model	**Parameter**	**Values tried**	**Parameter description**
SVM	`C`	0.01, 0.1, 1, 10, 100, 1000	Decides how much error is accepted and affects the margin size.
	`gamma`	0.01, 0.1, 1, 10, 100, 1000	Decides the reach of the data points and affects the decision boundary.
LR	`C`	0.001, 0.01, 0.1, 1, 10, 100	Decides how much error is accepted.
	`penalty`	'l2', None	Penalization strategy for models with many parameters.
	`solver`	'lbfgs', 'newton-cg', 'sag'	Algorithm to use in the optimization problem.
KNN	`leaf_size`	5, 10, 30, 50	Minimum number of data points in a leaf. Its impact depends on another automatically chosen parameter named algorithm.
	`n_neighbors`	5, 10, 20	The number of neighbors to use when making decisions.
	`weights`	'uniform', 'distance'	Weight function used during prediction.

Table 5.3: The best model parameters found through a grid search for the best baseline models. LR is logistic regression, F is $\text{NoReC}_{\text{train\_full}}$, and N is $\text{NoReC}_{\text{train\_no\_neutral}}$.

Model	**Parameter**	**F**	**N**	$\mathbf{SMN_{train}}$
SVM	`C`	10	1000	1
	`gamma`	0.1	0.01	1
LR	`C`	0.001	100	1
	`penalty`	None	None	'l2'
	`solver`	'lbfgs'	'sag'	'lbfgs'
KNN	`leaf_size`	5	5	5
	`n_neighbors`	20	20	20
	`weights`	'uniform'	'distance'	'distance'

5.4.2 BERT Models

For this task, four different BERT models were fine-tuned and used for classification. To achieve as high scores as possible, parameter tuning was first performed on all models using the training dataset for training and the validation dataset for validation. Afterward, each of the four models with the highest score on the validation dataset (one NB-BERT model, one NorBERT model, one mBERT model, and one DistilmBERT model) was tested on the test dataset. These results were used to compare the four BERT models. Figure 5.4 illustrates this process, and a simple overview of the BERT models' architecture is shown in Figure 5.5. This section will mainly focus on the parameter tuning process and the specific libraries, classes, functions, and models used for the BERT models, many of which come from Hugging Face.

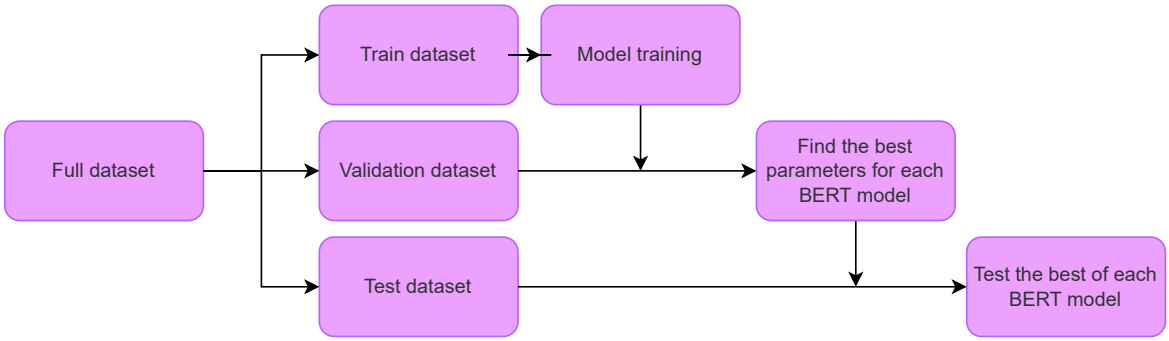


Figure 5.4: The workflow to create and find the best BERT model for SA.

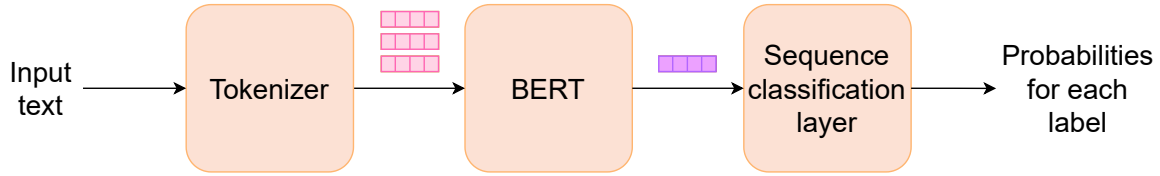


Figure 5.5: Architecture of the BERT model for one input text. The input is a text in natural language, the pink vectors are `input_ids`, `attention_mask` and `token_type_ids`, and the purple vector is used for classification.

Since two NoReC datasets, $\text{NoReC}_{\text{train\_full}}$ and $\text{NoReC}_{\text{train\_no\_neutral}}$, were created, the first decision that had to be made when training the models on this dataset was which training dataset to use. By comparing two NorBERT models that were trained with the same parameters, where the only difference was the training dataset used, it was found that training on $\text{NoReC}_{\text{train\_no\_neutral}}$ resulted in higher F1-scores, indicating a higher predictive ability. These results are available in Appendix B. Also, since the bank is primarily interested in negative cases, disregarding the neutral ones for training might provide better insight into the problem. Due to this, the BERT models were trained on $\text{NoReC}_{\text{train\_no\_neutral}}$. In the case of the SMN dataset, only one training dataset was created, so naturally, $\text{SMN}_{\text{train}}$ had to be used for training.

After the dataset was chosen, the texts had to be tokenized. As mentioned in Section 2.13.3, BERT uses wordpiece encoding for this. Tokenization is easily performed using the `AutoTokenizer.from_pretrained()` function from the `transformers` library. This function loads the correct tokenizer from the model being used, which results in three vectors for each input text, namely `input_ids`, `attention_mask` and `token_type_ids` [24]. `input_ids` is the vector created by wordpiece tokenization explained in Section 2.13.3, `attention_mask` is a vector of 0s or 1s, where 0 indicates that a token should be ignored, and `token_type_ids` correspond to the segment embedding. Since BERT only takes one input sentence for classification, all numbers will be 0 in `token_type_ids` for this task. Since more than one text is processed at a given time, the tokenizer will output three matrices instead of vectors, where one row in a matrix corresponds to one input text. These matrices can then be sent to a BERT model.

During these experiments, four BERT models were used for the classification tasks, namely NB-BERT[13], NorBERT[14], mBERT[15] and DistilmBERT[16]. Table 5.4 summarizes the models' specifications. The parameter sizes vary depending on the implementation, meaning they might differ from those listed in Section 2.13. The parameter size of NorBERT and NB-BERT is similar to that of $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$ mentioned in Section 2.13.3. However, as stated in DistilmBERT's model card on Hugging Face [18], the multilingual models utilized here are larger than the English models discussed in Sections 2.13.3 and 2.13.7. All BERT models were easily loaded from Hugging Face using the `transformers` library's `AutoModelForSequenceClassification.from_pretrained()` function. Both during training and testing, the BERT models output a set of vectors, but as mentioned in Section 2.13.3, only the first vector is passed to the classification layer for prediction.

Table 5.4: Overview of the specifications of the four BERT models used during the experiments. The size is the number of parameters.

Model	Base/Large	Cased/Uncased	Size	Layers	Sources
NB-BERT	Large	Uncased	340M	24	[34], [47]
NorBERT	Base	Cased	110M	12	[35]
mBERT	Base	Cased	177M	12	[7], [18]
DistilmBERT	Base	Cased	134M	6	[18]

The linear classification layer creates a matrix consisting of one row per input text[8]. A row contains logits, which could look like the following: [-1.5607, 1.6123] [2]. Each index in the row corresponds to one of the possible output labels, which is 0 or 1 in this task. The logits are normalized so that the numbers represent probabilities of a given class being correct, such as: [0.0402, 0.9598]. In this example, the second number indicates a 95.98% probability that the input text was positive. Finally, the model outputs the most likely label and the probability of that label being correct.

[13] https://huggingface.co/NbAiLab/nb-bert-large
[14] https://huggingface.co/ltg/norbert2
[15] https://huggingface.co/bert-base-multilingual-cased
[16] https://huggingface.co/distilbert-base-multilingual-cased

When training a model, the input texts are passed through the model as explained above, and the model is updated based on the results. Model training utilizes the `Trainer` object from the `transformers` library. It takes multiple parameters as input, including what model, training and validation dataset, and evaluation metric to use. Additionally, it takes as input a `TrainingArguments` object, which specifies the training parameters. Some parameters were kept the same for all models. `evaluation_strategy` was set to 'epoch' because it was interesting to know at which epoch the models performed the best. This ensures that predictions on the validation dataset are performed after each epoch and that the scores are saved. `load_best_model_at_end` had to be set to True to ensure that the final model saved was the one from the best epoch. Finally, `metric_for_best_model` was chosen, and it was set to the F1-score of the negative class, as this project focuses on detecting negative texts.

`TrainingArguments` also had a set of input parameters that were tuned to achieve as good results as possible. On the NoReC dataset, it was decided to tune the number of training epochs, the learning rate, the warmup ratio, and the optimizer. If the optimizer was 'adamw_hf', an additional parameter specifying the weight decay was also tuned. All parameters tried can be seen in Table 5.5. Which parameters to tune and which values to try were partially decided based on previous work on similar tasks [17], [34]. A grid search was performed using these parameters. In total this corresponds to 48 models being trained on $\text{NoReC}_{\text{train\_no\_neutral}}$ and evaluated on $\text{NoReC}_{\text{val}}$ for 5 epochs. The script for the grid search also created and saved all scores and plots necessary to compare the models so this data could be easily accessed later. All models trained during the grid search were temporarily saved, although irrelevant models eventually had to be deleted due to limited memory. To ensure that the best of each BERT model would be easy to load and reuse, these four models trained on the NoReC dataset were saved to Hugging Face[17] [18] [19] [20]. Since performing the grid search and testing on a CPU would be very time-consuming, NTNUs GPU cluster IDUN [57] was used. The runtime for a grid search on one BERT model was between 5 hours and 17 minutes, and 1 day, 6 hours, and 12 minutes, depending on the model size.

Table 5.5: Overview of which BERT models and parameter values were tested during the grid search. Each model tried all combinations of parameters. `weight_decay` was only tuned for the optimizer 'adamw_hf'.

Model	Parameters	Parameter values
NB-BERT NorBERT mBERT DistilmBERT	`num_train_epochs`	1, 2, 3, 4, 5
	`learning_rate`	2e-5, 3e-5, 4e-5, 5e-5
	`warmup_ratio`	0, 0.01, 0.1
	`optim`	'adamw_hf', 'adafactor'
	`weight_decay`	0, 0.01, 0.1

[17] https://huggingface.co/karolill/norbert_LR5e-05_WR0_OPTIMadamw_hf_WD0.01
[18] https://huggingface.co/karolill/nb-bert_LR5e-05_WR0.1_OPTIMadamw_hf_WD0
[19] https://huggingface.co/karolill/mbert_LR3e-05_WR0.1_OPTIMadamw_hf_WD0.1
[20] https://huggingface.co/karolill/distilmbert_LR3e-05_WR0.1_OPTIMadamw_hf_WD0.01

The hyper-parameter tuning on the SMN dataset differed from that explained above for the NoReC dataset because the data could not be transferred to IDUN. Due to this, parameter tuning had to be performed on a CPU. Since the extensive grid search would be too time-consuming to run on a CPU, only the number of epochs was tuned. For each of the four BERT models, the hyper-parameters that lead to the best results on $\text{NoReC}_{\text{val}}$ were used, and the model was trained for five epochs on $\text{SMN}_{\text{train}}$. After each epoch, the model was tested on SMN_{val}, and in the end, the model from the best epoch was chosen for further testing.

Each of the four BERT models that received the highest F1-score on $\text{NoReC}_{\text{val}}$ and SMN_{val} were then tested on $\text{NoReC}_{\text{test}}$ and SMN_{test} respectively. This was done since, as mentioned in Section 2.6, having a separate test set helps gain objective results. The same architecture explained above was used to make predictions on the test dataset. The `transformers pipeline()` function was used to tokenize the data and pass it through the model to receive the predicted labels. Afterward, `sklearn.metrics.classification_report()` was used to see the results of the evaluation metrics, and confusion matrices were plotted.

5.4.3 Qualitative Evaluation of the Model's Utility

As a final step in the evaluation process, the employees at Sparebank 1 SMN would evaluate the utility of incorporating a language model that predicts sentiment in their workplace. Seventy-four new and recently received emails were provided from the bank in the same format as the original dataset. These emails were preprocessed as explained in Section 5.2.2, and predictions were performed using the model that achieved the best results on SMN_{test}. Afterward, the emails were sent to employees at Sparebank 1 SMN, who looked at them and assessed if the sentiment of an email and how confident the model is about its predictions could help prioritize cases. Written feedback about their findings was provided when they were done. Another employee checked how many of the examples had been labeled correctly.

Chapter 6

Results

This section will present the results achieved by the various models. First, Section 6.1 will show the results achieved from the experiments related to **RQ2** before Section 6.2 presents the results related to **RQ3**.

6.1 Research Question 2 - Active Learning

As mentioned in Section 5.3, employees at Sparebank 1 SMN corrected 200 of the labels created by the AL process. Of the 100 emails labeled positive, 94% were correct. Also, 61% of the 100 negatively labeled emails were initially correct, but after some sentiment labels were updated, 72% were correct.

6.2 Research Question 3 - Language Models

Once the baseline and BERT models completed their predictions, various metrics and confusion matrices were computed to show their capabilities. This section will display these findings, starting with the results on the NoReC dataset, followed by the scores attained on the SMN dataset.

6.2.1 NoReC Dataset

As mentioned in Section 5.4.1, a few different baseline models were attempted to ensure that the final baseline model that would be compared to BERT was of decent quality. First, the results from the six baseline models will be presented, and a final baseline model will be chosen. Afterward, the results from the BERT models are shown and compared to each other and the final baseline model.

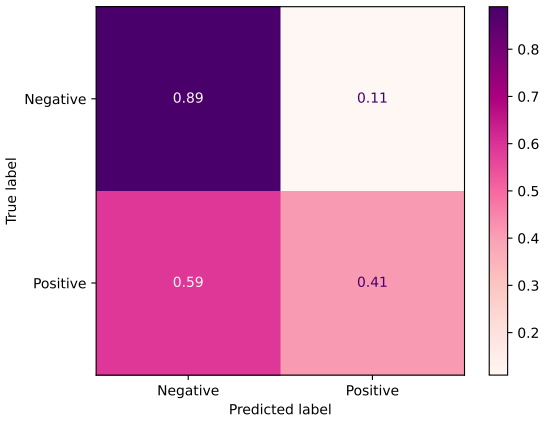
Baseline

The result of the baseline models when tested on $\text{NoReC}_{\text{test}}$ can be seen in Table 6.1 and Figure 6.1. Table 6.1 shows the results from the `classification_report()` function for each model. Figure 6.1 shows the normalized confusion matrices for the predictions made by each model.

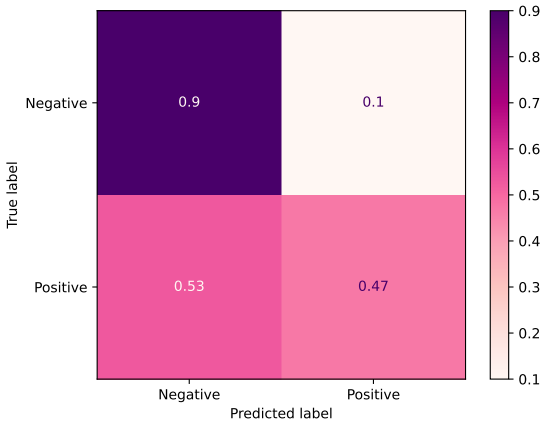
As stated in Section 5.4.1, the most important metric for this project is the F1-score for the negative class, as the final goal for the SMN dataset is to find negative texts, and it is desirable to treat the NoReC dataset the same way. The second most important metric is the precision of the negative class. Based on these metrics, the best baseline model that will later be compared to the BERT models is the SVM model trained on $\text{NoReC}_{\text{train\_no\_neutral}}$, which had the highest F1-score and precision on the negative class with scores of 0.21 and 0.12 respectively. Also, it can be seen from the confusion matrices that of all models that correctly classified 90% of the negative class, the SVM trained on $\text{NoReC}_{\text{train\_no\_neutral}}$ performed best on the positive class. From now on, this model will be referred to as $\text{Baseline}_{\text{NoReC}}$.

Table 6.1: The results from the `classification_report()` function on the six possible baseline models tested on $\text{NoReC}_{\text{test}}$. LR is logistic regression, F is $\text{NoReC}_{\text{train\_full}}$, N is $\text{NoReC}_{\text{train\_no\_neutral}}$, A is accuracy, P is precision, and R is recall. Precision, recall, and F1-score are provided for the negative (0) and positive (1) classes. The best score in each column is marked in bold.

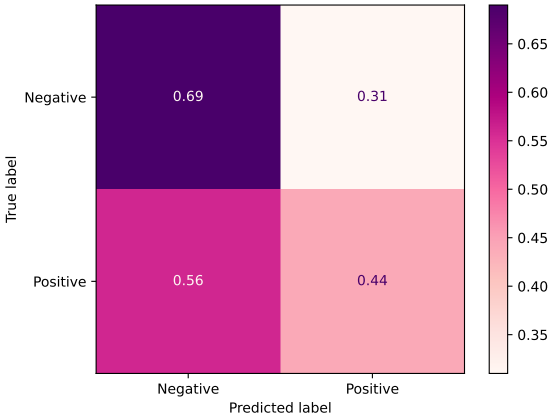
Model	**Dataset**	**A**	**P 0**	**P 1**	**R 0**	**R 1**	**F1 0**	**F1 1**
SVM	F	0.44	0.09	0.98	0.89	0.41	0.17	0.58
LR	F	0.50	0.10	**0.99**	**0.90**	0.47	0.18	0.64
KNN	F	0.46	0.08	0.96	0.69	0.44	0.14	0.61
SVM	N	0.57	**0.12**	**0.99**	**0.90**	0.55	**0.21**	0.70
LR	N	0.56	0.11	**0.99**	**0.90**	0.54	0.20	0.70
KNN	N	**0.64**	0.10	0.96	0.57	**0.65**	0.16	**0.77**



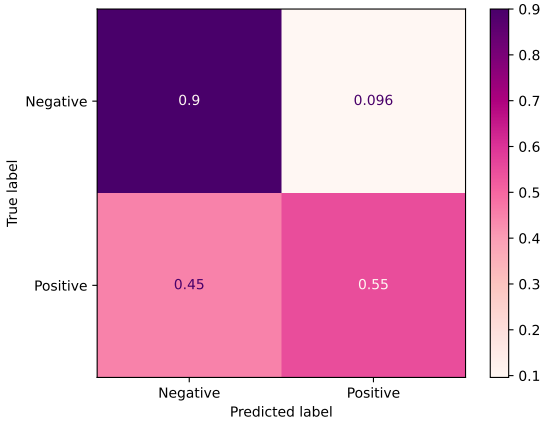
(a) Confusion matrix for the best SVM model trained on $\text{NoReC}_{\text{train\_full}}$.



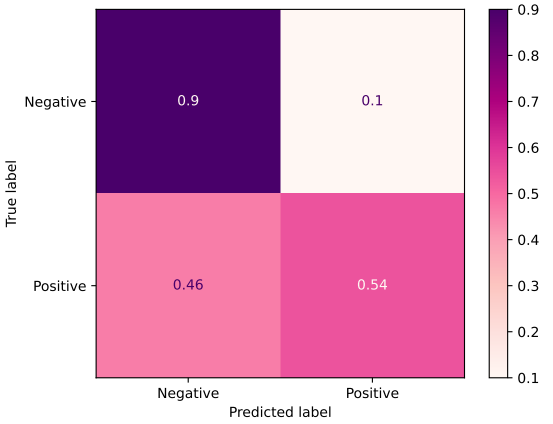
(b) Confusion matrix for the best logistic regression model trained on $\text{NoReC}_{\text{train\_full}}$.



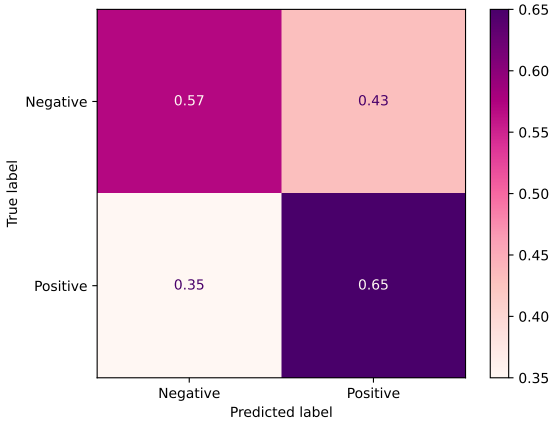
(c) Confusion matrix for the best KNN model trained on $\text{NoReC}_{\text{train\_full}}$.



(d) Confusion matrix for the best SVM model trained on $\text{NoReC}_{\text{train\_no\_neutral}}$.



(e) Confusion matrix for the best logistic regression model trained on $\text{NoReC}_{\text{train\_no\_neutral}}$.

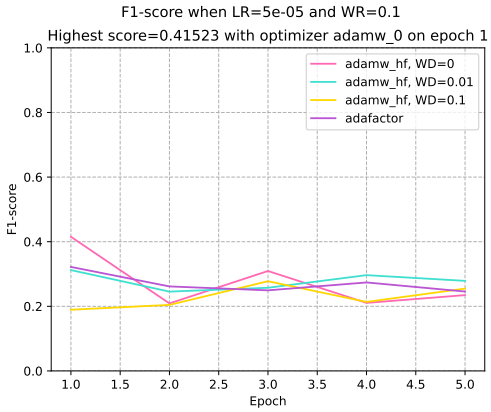


(f) Confusion matrix for the best KNN model trained on $\text{NoReC}_{\text{train\_no\_neutral}}$.

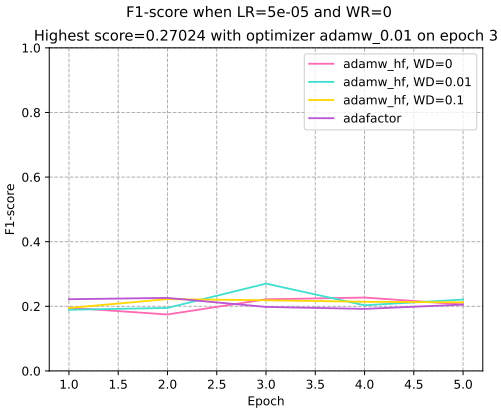
Figure 6.1: Confusion matrices with normalized rows generated from predictions on $\text{NoReC}_{\text{test}}$. Different ML models were used in combination with TF-IDF.

BERT

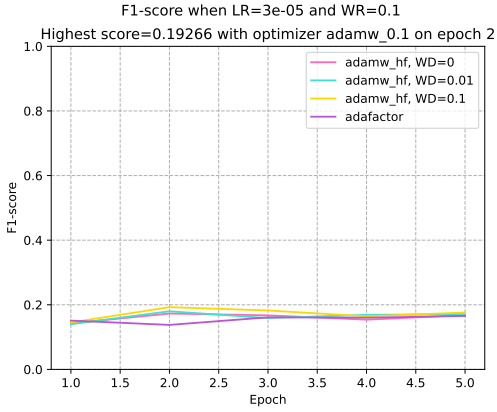
As mentioned in Section 5.4.2, a grid search was first performed to tune the hyper-parameters of the four BERT models. The resulting models, from each combination of parameters, were evaluated on $\text{NoReC}_{\text{val}}$. Figure 6.2 shows the four BERT models' highest F1-score and which parameter values were used to achieve the score. All results from the grid search can be seen in Appendix C. The highest scores achieved by any fine-tuned NB-BERT, NorBERT, mBERT, and DistilmBERT model on the validation dataset were 0.41523, 0.27024, 0.19266, and 0.16384, respectively. Another interesting observation is that the hyper-parameter tuning does not appear to have affected the results much on mBERT and DistilmBERT, considering all graphs (including those in Appendix C) are relatively similar and flat. However, NB-BERT and NorBERT's performance seems to depend more on the hyper-parameter tuning.



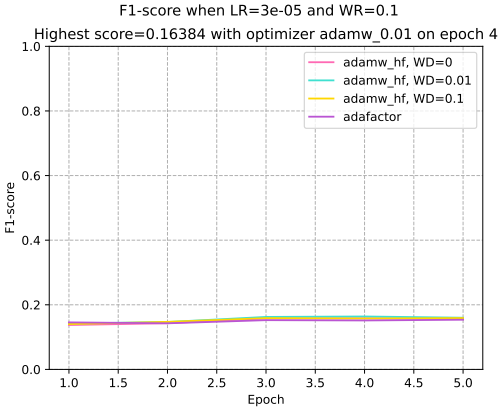
(a) The best result achieved when fine-tuning NB-BERT was 0.41523.



(b) The best result achieved when fine-tuning NorBERT was 0.27024.



(c) The best result achieved when fine-tuning mBERT was 0.19266.



(d) The best result achieved when fine-tuning DistilmBERT was 0.16384.

Figure 6.2: These plots show the F1-scores achieved on the negative class during the grid search. The highest score in each plot is the highest F1-score achieved by any fine-tuned version of the respective BERT model on $\text{NoReC}_{\text{val}}$. The text above each plot shows what parameters were used, where the notation "optimizer adamw_0.01" or similar states that the optimizer was adamw_hf with a weight decay of 0.01.

The fine-tuned BERT model of each type that achieved the best scores on the validation dataset was tested on $\text{NoReC}_{\text{test}}$. The results from these predictions can be seen in Table 6.2 and Figure 6.3. As mentioned earlier, the F1-score and precision of the negative class are the most important metrics for this task. Based on these metrics and the confusion matrices, it seems reasonable to say that NB-BERT is the best model for the NoReC dataset, as it achieved the highest F1-score and precision on the negative class. Even though NorBERT identifies more negative texts, it can not be said to be the best model because of how many positive texts are misclassified. $\text{Baseline}_{\text{NoReC}}$ was also better at finding negative texts than NB-BERT was, but due to how many positive ones $\text{Baseline}_{\text{NoReC}}$ predicted to be negative, NB-BERT still seems to be the best. Another noteworthy result is that $\text{Baseline}_{\text{NoReC}}$ outperformed both multilingual BERT models.

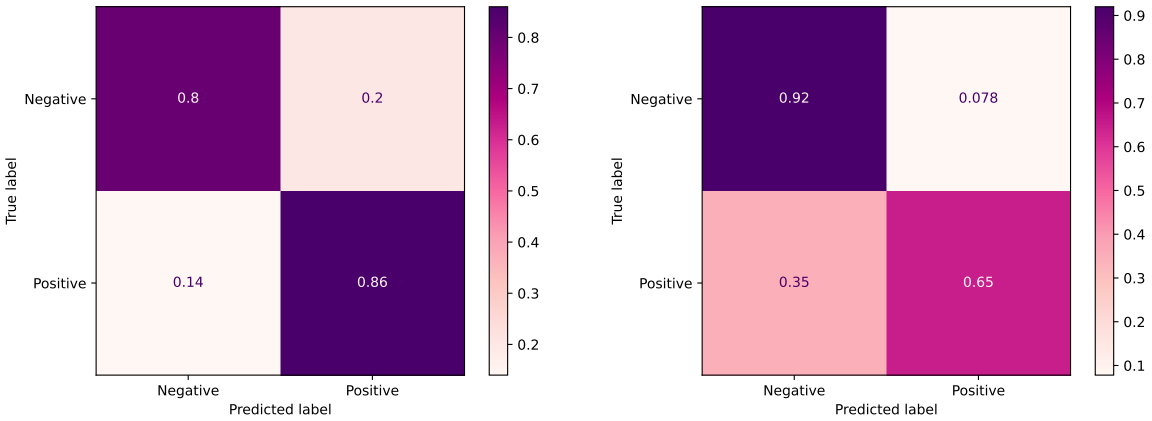
Table 6.2: The results from the `classification_report()` function on the predictions made by the BERT models on $\text{NoReC}_{\text{test}}$. The best score of each column is marked in bold. A is accuracy, P is precision, and R is recall. Precision, recall, and F1-score are provided for the negative (0) and positive (1) classes.

Model	**A**	**P 0**	**P 1**	**R 0**	**R 1**	**F1 0**	**F1 1**
NB-BERT	**0.86**	**0.27**	**0.99**	0.80	**0.86**	**0.41**	**0.92**
NorBERT	0.67	0.15	**0.99**	**0.92**	0.65	0.26	0.79
mBERT	0.54	0.10	0.98	0.83	0.52	0.18	0.68
DistilmBERT	0.39	0.09	0.98	0.90	0.36	0.16	0.53

Since it can also be interesting to know which models perform predictions faster, the predictions on $\text{NoReC}_{\text{test}}$ were timed. These results can be seen in Table 6.3. Since the predictions using BERT models were run on a GPU on IDUN, these times can not be compared to the time spent by $\text{Baseline}_{\text{NoReC}}$, as that was run on a CPU. DistilmBERT was the fastest of the BERT models, and NB-BERT was the slowest. $\text{Baseline}_{\text{NoReC}}$ was faster than all BERT models, despite running on a CPU. When comparing the runtimes to the model sizes, one can see that the $\text{BERT}_{\text{large}}$ model (NB-BERT) is slower than all $\text{BERT}_{\text{base}}$ models (the other BERT models). Also, mBERT is larger and slower than DistilmBERT, and NorBERT is smaller and slower than both of them. This indicates that the time somewhat correlates to the size, although the NorBERT model is an exception when compared to mBERT and DistilmBERT.

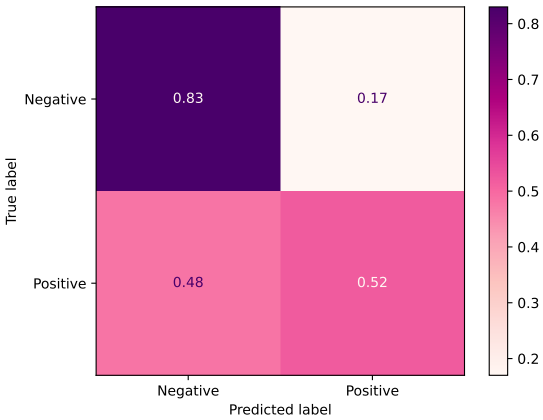
Table 6.3: The rounded time in seconds spent on prediction of the $\text{NoReC}_{\text{test}}$ dataset. The BERT models were run on GPU, while $\text{Baseline}_{\text{NoReC}}$ was run on CPU. The models' sizes are also listed. For BERT models, it is the number of parameters, and for the baseline it is the number of support vectors, as the runtime is dependent on that number.

Model	$\text{Baseline}_{\text{NoReC}}$	NB-BERT	NorBERT	mBERT	DistilmBERT
Time[s]	25	137	61	56	55
Size	2825	340 million	110 million	177 million	134 million

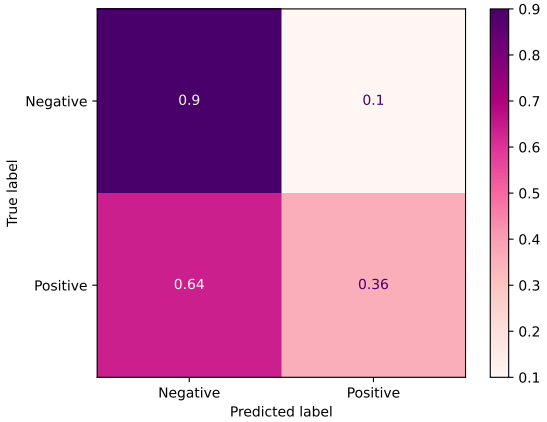


(a) NB-BERT

(b) NorBERT



(c) mBERT



(d) DistilmBERT

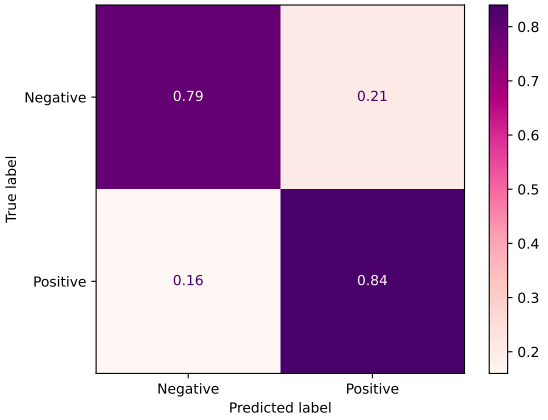
Figure 6.3: These confusion matrices visualize the results on $\text{NoReC}_{\text{test}}$ by the different BERT models. The rows of the matrices are normalized.

6.2.2 SMN Dataset

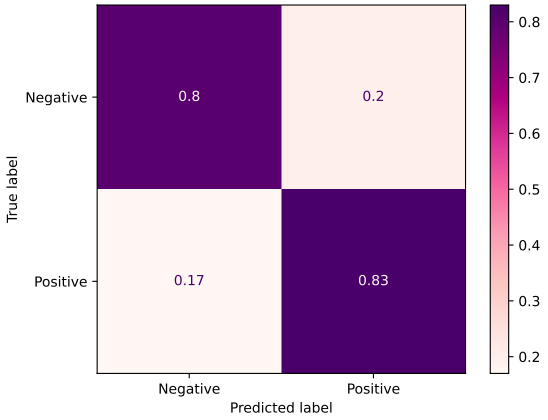
This section will start by presenting the scores achieved by the baseline and BERT models on the SMN dataset. Then the feedback from the Sparebank 1 SMN employees regarding the utility of incorporating SA in the business will be stated.

Baseline

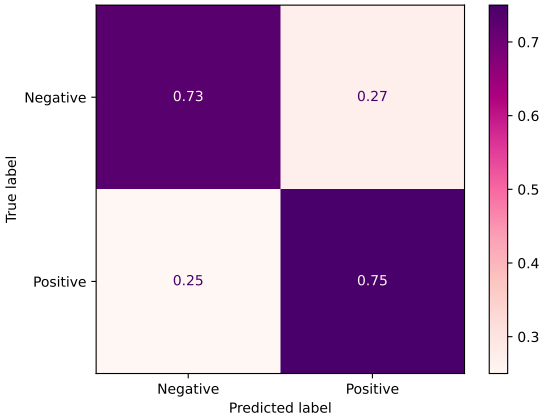
Table 6.4 and Figure 6.4 show the results of the three baseline models. As mentioned earlier, the most important metrics are the F1-score and precision of the negative class. Based on the models' results on these metrics, the best baseline model is the SVM, which was slightly better than logistic regression. From now on, This SVM model will be called $\text{Baseline}_{\text{SMN}}$, and it will be compared to the BERT models. It should be noted that logistic regression had very similar scores to the SVM and identified slightly more negative emails than the SVM, which can be seen from the value in the top left corner of the confusion matrix where it scored higher.



(a) Confusion matrix for the best SVM model trained on $\text{SMN}_{\text{train}}$.



(b) Confusion matrix for the best logistic regression model trained on $\text{SMN}_{\text{train}}$.



(c) Confusion matrix for the best KNN model trained on $\text{SMN}_{\text{train}}$.

Figure 6.4: Confusion matrices with normalized rows generated from predictions on SMN_{test}. Different ML models were used in combination with TF-IDF.

Table 6.4: The results from the `classification_report()` function on the three possible baseline models tested on SMN_{test}. LR is logistic regression, A is accuracy, P is precision, and R is recall. Precision, recall, and F1-score are provided for the positive (1) and negative (0) classes. The best score in each column is marked in bold.

Model	A	P 0	P 1	R 0	R 1	F1 0	F1 1
SVM	**0.83**	**0.52**	**0.95**	0.79	**0.84**	**0.63**	**0.89**
LR	0.82	0.51	**0.95**	**0.80**	0.83	0.62	**0.89**
KNN	0.74	0.39	0.93	0.73	0.75	0.51	0.83

BERT

As mentioned in Section 5.4.2, the only hyper-parameter tuned on the SMN dataset was the number of epochs for each model. Figure 6.5 shows the four models' performance when they were tested on SMN_{val} after each epoch. NB-BERT performed best after epoch two, NorBERT after epoch five, mBERT after five, and DistilmBERT after three epochs. These models from the best epochs were then used for predictions on SMN_{test}. An interesting observation in the plot is NB-BERT's learning curve, which drops to zero on epoch three. Since the plot shows the F1-score on the negative class, this indicates that all (or at least almost all) negative examples have been labeled as positive. Also, if one calculates the F1-score of the negative class if every email is labeled as negative, that would be just below 0.31, which is approximately the value NB-BERT scored on epoch two, meaning that NB-BERT might simply be classifying everything as negative at that epoch. Based on these observations, the model training of NB-BERT failed.

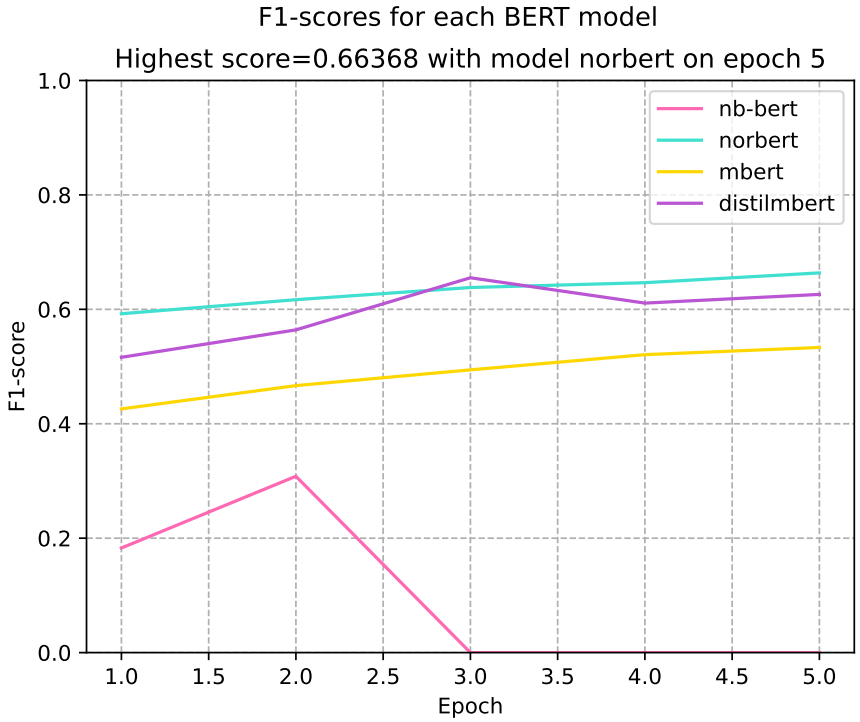


Figure 6.5: The score of all four BERT models on SMN_{val} after each epoch.

The resulting scores from the BERT models tested on SMN_{test} can be seen in Table 6.5 and Figure 6.6. NorBERT scored best when it was tested, which can be seen by the confusion matrices, where it had the highest number of true positive labels, and it was only beaten by NB-BERT on the true negatives. Apart from the precision of the positive class and the recall of the negative class NorBERT also scored highest on all metrics calculated. NB-BERT was best at predicting negative examples. However, since it classified all except for two examples as negative, which was seen by looking at the labeled examples, it can not be said to have any predictive ability. Therefore, this model is the worst tested on SMN_{test}. When comparing the BERT models to $Baseline_{SMN}$, NorBERT achieved higher scores on the most important metrics (the F1-score and precision of the negative class), while the other BERT models did not.

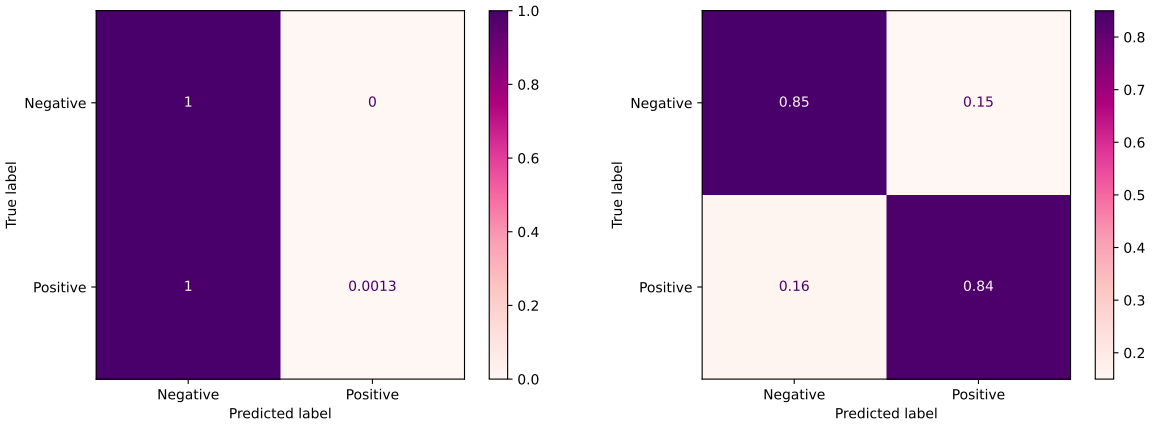
Table 6.5: The results from the `classification_report()` function when the BERT models were tested on SMN_{test}. A is accuracy, P is precision, and R is recall. Precision, recall, and F1-score are provided for the positive (1) and negative (0) classes. The best score in each column is marked in bold.

Model	**A**	**P 0**	**P 1**	**R 0**	**R 1**	**F1 0**	**F1 1**
NB-BERT	0.18	0.18	**1.00**	**1.00**	0.00	0.31	0.00
NorBERT	**0.84**	**0.54**	0.96	0.85	**0.84**	**0.66**	**0.90**
mBERT	0.80	0.47	0.95	0.79	0.81	0.59	0.87
DistilmBERT	0.81	0.48	0.96	0.84	0.80	0.61	0.87

The time was also checked when making predictions on SMN_{test}. The runtimes are presented in Table 6.6. These times are not comparable to the ones found for predictions on $NoReC_{test}$ as that code ran on GPU while the predictions of SMN_{test} were performed on CPU. Predictions on SMN_{test} would run faster than predictions on $NoReC_{test}$ if they were run on the same computer since the average email was shorter than the reviews in the NoReC dataset. Since both predictions with $Baseline_{SMN}$ and the BERT models were performed on a CPU, these times can be compared. $Baseline_{SMN}$ is much faster than all BERT models, and among the BERT models, DistilmBERT was the fastest and NB-BERT the slowest. Like the NoReC dataset results, the SMN dataset also shows a correlation between prediction time and model size. Unlike the run times on $NoReC_{test}$, NorBERT was faster than mBERT when performing predictions on SMN_{test}.

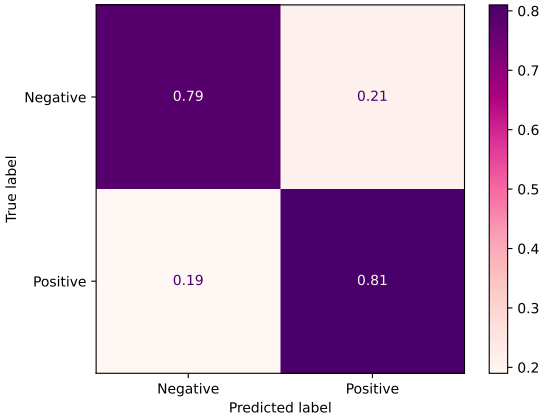
Table 6.6: Time in seconds spent on prediction on SMN_{test} when using CPU. The models' sizes are also listed. For BERT models, it is the number of parameters, and for the baseline it is the number of support vectors, as the runtime is dependent on that number.

Model	$Baseline_{SMN}$	NB-BERT	NorBERT	mBERT	DistilmBERT
Time[s]	0.95	317.11	107.82	189.49	81.06
Size	3119	340 million	110 million	177 million	134 million

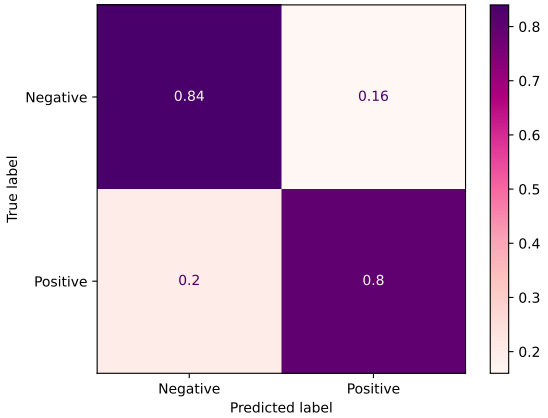


(a) NB-BERT

(b) NorBERT



(c) mBERT



(d) DistilmBERT

Figure 6.6: These confusion matrices visualize the results on SMN_{test} by the different BERT models. The rows of the matrices are normalized.

6.2.3 Qualitative Evaluation of the Model's Utility

Finally, the results on the small dataset of recently received emails will be presented. When the bank employee checked the labels, it was found that 87% of the emails predicted to be negative were true negative, and 87% of the positively labeled examples were true positive. When it comes to prioritizing emails, the employees found that a system that helps prioritize cases could be helpful. However, negative emails are not the only ones that should be prioritized. It could also be beneficial to quickly answer positive emails that provide a sales opportunity for the bank.

Chapter 7

Discussion

This section will discuss the findings from the results and examine if they were as expected based on the related work listed in Section 3. The first three sections will discuss the research questions defined in Section 1.2 before the final section will consider other challenges.

7.1 State-of-the-Art Techniques for Sentiment Analysis in Norwegian

As stated in Section 1.2, the first research question is *"What are the state-of-the-art techniques for sentiment analysis in Norwegian?"*. This was explored through a literature study, resulting in two Norwegian BERT models, namely NB-BERT and NorBERT, and two multilingual models later being used for SA to answer the other research questions. As shown in Section 3, previously performed experiments show that these models typically outperform older methods used for SA, such as TF-IDF, RNNs, and CNNs. Although newer models have achieved higher scores than BERT on various tasks, including SA, these models are not trained specifically on Norwegian text. For example, XLNet and XLM-RoBERTa have achieved higher scores than BERT on SA tasks, but they are English and multilingual, respectively. Since, as seen in Section 3, domain-specific models tend to be better, NB-BERT and NorBERT will often still be the best models for SA of Norwegian text, since they are trained on the Norwegian domain. The findings related to **RQ3**, which will be further discussed later, also support this theory. It was found that NorBERT outperformed the multilingual models on both classification tasks performed, while NB-BERT outperformed them on one.

7.2 Active Learning for Customer Emails

The second research question was *"Can active learning be used to create sentiment labels on customer emails?"*. AL was performed on the SMN dataset to examine this, and SMN employees checked the labels. This section will first discuss if the AL process gave the expected results and how the resulting labels might have affected the final score from the models tested on SMN_{test}. Then, possible weaknesses or issues with the process will be discussed, including the use of TL as a starting point for AL. Finally, the difference between what emails were labeled as negative during the AL process and the correction by the SMN employees will be considered.

As stated in Section 6.1, 94% of the positive and 72% of the negative labels were correct. The result on the positively labeled examples is good, and although there is still room for improvement, this is sufficient for further testing on SA. However, quite a lot of the examples labeled as negative were positive. This could negatively affect the other experiments performed on the dataset, as it is harder to evaluate the models' performance on an SA task when the labels in SMN_{test} are inaccurate. Specifically, the score achieved does not state the models' ability to predict an email's sentiment, as the labels themselves do not accurately state the sentiment.

It is worth discussing possible reasons why the AL process did not improve beyond the abovementioned results. As mentioned in Section 5.3, further training in the AL process decreased the model's ability to predict the sentiment correctly, instead of improving it which would have been the expected outcome. Section 3.3.2 listed some possible reasons for this issue, namely overfitting, the cold-start problem (starting with no labeled data), and the fact that the output probabilities might not accurately state the probability of a label being correct. It was attempted to mitigate the cold-start problem through TL, and even if it did not have the desired effect, it would not cause the model to perform worse with time; it would just give a poor starting point. Looking at the negatively labeled emails after training on 600 examples, overfitting seems to be the most likely cause of the problem. Of 164 negatively labeled examples, 130 were related to insurance or interests. This could indicate that the model overfitted to the belief that only emails about these subjects could be negative. Another possibility is that inconsistent labeling in the manual labeling process caused the issues. Despite the difficulty in labeling certain emails, from the observations mentioned earlier, this does not appear to be the determining factor in the model's decreased performance. Based on these findings, a promising way to improve the AL process would be to choose examples that are not too similar to manually label at each round. This would cause the training examples to better represent the whole dataset.

TL was used to mitigate the cold-start problem so that even the first labeled examples were helpful to the model. A few possible issues regarding this approach should be mentioned. First, as seen in Section 6.2.1, the NB-BERT model trained on $\text{NoReC}_{\text{train}}$ only predicted 80% of the negative texts correctly, meaning that even in the same domain, it would not be a perfect starting point. However, as this is a TL task and the model was used in a different domain, with a different style of writing and other topics, the expected starting point would be even worse. One issue seen when using TL that

would not have occurred if the first examples to label had been chosen randomly is that the first labeled examples did not represent the distribution. Randomly selecting a set of examples will typically lead to a good representation of the underlying distribution, meaning that the labeled texts represent all emails well. However, throughout the AL process, it was experienced that most emails in one round of labeling were similar in either topic or structure. Hence, the distribution of emails was not well represented. This means that TL could lead to a worse starting point. A glance at the emails labeled in the first round revealed that they were relatively short and similar in structure but with different topics, indicating that the emails did not properly represent the whole corpus. As with overfitting in general, incorporating methods to ensure that also the first examples to label represent the underlying distribution of emails would likely improve the results.

Finally, some will be said about the disagreement regarding what emails were considered negative when the AL process was performed and when SMN employees corrected the dataset. One could argue that changing the results provided by the SMN employees is not optimal, but in this case, it was necessary to evaluate the AL process. As mentioned in Section 5.3, emails written in a positive or neutral tone that negatively impacted the bank were labeled negative during the AL process and positive when corrected by the SMN employees. To properly evaluate the AL process, the final dataset had to be assessed using the same criteria used during labeling. That said, it was still valuable to get insight from SMN employees, as some emails were difficult to label. In those cases hearing from a professional is helpful. Whether or not emails written in a positive tone with a negative impact on the bank should be labeled as negative can be further discussed in the future. Nonetheless, it is still possible to evaluate the results from the AL process, as done in Section 6.1.

7.3 Comparison of Models for Sentiment Analysis

"How do different transformer-based language models compare to each other when used to solve sentiment analysis tasks? And how do transformer-based language models compare to previously used methods for sentiment analysis?" was the final research question. This section will start by discussing the first part of the research question before the second part will be examined. Since two datasets were used for experimentation, the following paragraphs will mention both. Finally, this section will try to answer what model should be used for the tasks performed.

7.3.1 Comparison of the Transformer-Based Models

First, the BERT models' performance on the NoReC dataset will be discussed. NB-BERT performed the best, as it was the only model which could decently predict the positive reviews. NorBERT was the second-best model, followed by mBERT and DistilmBERT, which achieved the lowest score. This order matched what could be expected based on previous work in the field, mentioned in Sections 3.2.1 and 3.2.3,

which had similar findings. Nonetheless, it was surprising that their F1-scores were so low and that the three worst models struggled as much as they did with the prediction of positive texts. When Kutuzov et al. [35] and Kummervold et al. [34] performed their experiments, the Norwegian models achieved macro F1-scores in the range 0.771–0.864, but the highest macro F1-score on $\text{NoReC}_{\text{test}}$ was 0.665 by NB-BERT. There are a few possible reasons for these differences. First, their experiments used $\text{NoReC}_{\text{sentence}}$ instead of the original dataset. Unlike $\text{NoReC}_{\text{sentence}}$, the NoReC dataset used here consists of longer texts, some of which contain conflicting sentiments. Some reviews also contain objective information that indicates a sentiment, although it does not affect the label, such as the plot of a movie. Second, reviews considered neutral with a score of 3 or 4 were labeled as positive. However, in many cases, these contain a mix of positive and negative sentiments, not only neutral sentiment, making them harder to classify. Lastly, it was observed that some texts consisted of two or more reviews with different scores. It is unknown how frequently this issue appeared, but it makes sense that the models would struggle to make predictions in the cases where it happened.

Next, the models' performance on the SMN dataset will be considered. As expected, NorBERT outperformed the multilingual models. DistilmBERT achieved a higher score than mBERT, which might be surprising since this is typically not seen in previous work. However, the biggest surprise is the poor performance of NB-BERT since previous work in the field suggested that it should have achieved the highest score. The model classified all except for two examples as negative. This matches the observations noted in Section 6.2.2, which stated that the model's F1-score on the validation dataset could have been a result of labeling nearly all examples as negative. One possible reason this model performed so poorly is the use of the F1-score of the negative class for evaluation when choosing the best model. One could try to change the evaluation metric, but it does not seem ideal since it worked well for the other models. Another possibility for mitigating this issue is hyper-parameter optimization, which can affect the results by providing a better starting point. Since NB-BERT performs worse after epoch one than the other models tested on SMN_{val}, it is plausible that the starting point was unfortunate, so finding a better starting point would impact the results. It was also seen that NB-BERT's performance on $\text{NoReC}_{\text{val}}$ to a large degree depended on the hyper-parameters. Therefore, it is likely that hyper-parameter tuning would impact the results on the SMN dataset as well. However, further experimentation would have to be performed to be sure.

When the BERT models' prediction times were measured, the time was expected to correlate with the number of parameters. This was observed when using NB-BERT since it was both the largest and slowest model. However, it was surprising that NorBERT was not the fastest BERT model on either dataset, considering it was the smallest. Other architectural differences could be why the $\text{BERT}_{\text{base}}$ models' prediction times did not correlate with the number of parameters. For example, as mentioned in Section 5.4.2, DistilmBERT has half as many layers as NorBERT and mBERT, which is a possible reason why DistilmBERT is faster than NorBERT, despite having more parameters. As for why mBERT is larger than NorBERT and makes predictions faster than NorBERT on the NoReC dataset, it is difficult to say without more information.

7.3.2 Transformer-Based Models vs. Previously Used Methods

To answer how transformer-based models compare to previously used methods, it is beneficial first to discuss the performance of all baseline models tried here. As seen in Section 6.2.1 and 6.2.2, using TF-IDF in combination with SVM resulted in the highest F1-score and precision of the negative class, outperforming TD-IDF combined with KNN or logistic regression. This matched the expectations based on previous work explained in Section 3.1 by Avinash and Sivasankar [41] and Liu et al. [39]. It is also worth noting that the SOTA models before the creation of the transformer architecture were not TF-IDF combined with ML models but rather models based on RNNs or CNNs taking other word or sentence vectors as input. Knowing this, one can conclude that there might be models not based on transformers that could have outperformed the baseline models.

On the NoReC dataset, NB-BERT and NorBERT scored higher than $\text{Baseline}_{\text{NoReC}}$, while DistilmBERT and mBERT did not. As noted in Section 3.2.2, Mishev et al. [43] showed that TF-IDF combined with SVM can achieve scores close to that of $\text{BERT}_{\text{base}}$, as using TF-IDF resulted in an F1-score of 0.836, only 0.054 points below BERT. Also, as stated in Section 3.2.1, Kummervold et al. [34] and Kutuzov et al. [35] showed that multilingual BERT models tend to do worse than monolingual models. Based on these observations, it is unsurprising that a good TF-IDF model can achieve a higher score than a multilingual BERT model. The fact that NB-BERT and NorBERT outperform $\text{Baseline}_{\text{NoReC}}$ also matches previous work in the field. Similar results were seen on the SMN dataset, and the scores achieved by NorBERT, mBERT, and DistilmBERT match the expectations just described. However, NB-BERT performed worse than all other models, but its poor performance was discussed earlier and will not be repeated here.

7.3.3 What Model is the Best?

Finally, when discussing the performance of various models for SA in Norwegian, a few words should be said about what model is the best and should be used for future work. The answer depends on the task to solve. If correct predictions are critical, using NorBERT or NB-BERT will be the best option. Despite NB-BERT's low score on the SMN dataset, hyper-parameter tuning would likely improve the results, possibly making it the best model for achieving a high F1-score. If a somewhat worse accuracy or F1-score is acceptable, or if time is limited, using TF-IDF in combination with SVM is a better solution, as the results are not much worse, but the computations require less time and resources. In the experiments performed here, good performance is more important than speed, and NB-BERT or NorBERT is therefore preferred.

7.4 Challenges

Throughout the work of this thesis, other challenges not directly connected to the research questions were also faced. They might still be interesting to discuss so that possible future work related to this project can improve the process or solve the issues. This section will start by explaining problems with the preprocessing of the SMN dataset and what impact that could have had on later work before discussing how various limitations of the experiments affected the results.

When preprocessing the SMN dataset, a few challenges had to be overcome. First, it was difficult to properly split the emails when the description contained multiple emails going back and forth. As explained in Section 5.2.2, this was done by splitting on common phrases typically found at the end of emails or between two emails in the description. Although observations indicate that this approach was successful most of the time, there exist cases where the end of emails is removed due to the split or where two emails were not split, causing the final text to consist of two or more emails. If the end of an email is removed, text containing sentiment might be gone, making predictions harder. And if the split was unsuccessful, leaving more than one email in the text, the model might be confused by the sentiment in the second email, which should have been removed. Another issue with the preprocessing was that a limit was set for how long an email had to be before it was added to the dataset. This was done because it was observed that some short emails were empty, only contained a date, or otherwise suggested issues with preprocessing. Due to this, models were not trained or evaluated on short emails such as "Ok" or similar, although they are likely to exist in the original dataset. Despite this, the models might have learned how to label these emails based on longer emails containing the same words. Therefore, if short emails occur in a new dataset, the models might be able to label them correctly. Nonetheless, it would be better if no limit had to be set and all emails were used for training and testing, but only if the shorter emails were formatted correctly.

Another part of the preprocessing performed before access was given to the dataset also affected the results. Section 5.2.2 mentions that names were replaced with [PERSON], emails with [EPOST], and numbers with more than three digits with [SIFFER]. There did not appear to be problems with [EPOST] and [SIFFER]. However, there were some issues with [PERSON], which had replaced words other than people's names. For example, the name of months and places, but also other seemingly random words such as "millioner" (millions) or "mulige" (possible) were replaced. Note that the example words used here could be wrong as the underlying word had to be guessed based on the context. In certain emails, such a significant portion of the content is replaced with [PERSON] that it may even become challenging for humans to comprehend. However, observations suggest that in most cases it is possible for a person to guess the hidden word based on the context. Nonetheless, words containing sentiment might have been hidden, negatively impacting the model training and predictions.

Some limitations also affected the results and should therefore be mentioned. First, the lack of access to a GPU when working on the SMN dataset was an issue. The data could not be moved to IDUN, meaning the experiments had to be run on CPU. This greatly

limited the speed at which the code could run. Combined with a second limitation, the time available for experimentation, it was impossible to tune the BERT models' hyper-parameters on $\text{SMN}_{\text{train}}$. As mentioned earlier, no hyper-parameter tuning is probably why NB-BERT performed so poorly on SMN_{test}, so its score likely could have been improved by hyper-parameter tuning. And because previous work and results on the NoReC dataset have found that NB-BERT tends to work best on SA tasks, the best score on SMN_{test} by any model might have been improved through hyper-parameter tuning. The time limitation also affected the results of the AL process, as attempts to improve the results could have been made if more time had been available.

Chapter 8

Conclusion and Future Work

This thesis has explored the use of transformer-based language models for SA. First, a literature study was performed to find the SOTA models for SA in Norwegian. These models were then used to perform SA on two Norwegian datasets; a dataset of reviews named NoReC and a dataset of emails provided by Sparebank 1 SMN. The datasets were preprocessed as necessary before they were used for model training and prediction. As a part of the preprocessing of the SMN dataset, the thesis also explored AL as a method for labeling to reduce the cost of manual label generation. In the following section, a summarizing answer will be provided to each research question before the final section provides possible ideas for future work.

8.1 Conclusion

At the beginning of this project, three research questions were defined. They specified what the thesis would explore and guided which experiments were performed.

RQ1 - What are the state-of-the-art techniques for sentiment analysis in Norwegian?

After reading previously written articles regarding SA, it became clear that the introduction of transformers in 2017 led to a new model type that outperformed the previously used ML models and deep NNs on many tasks, including SA. One type of transformer-based model that has excelled on SA tasks is BERT, first introduced in 2018. Afterward, many BERT models were created and trained on varying languages and domains. Two Norwegian BERT models exist, namely NB-BERT and NorBERT, but multilingual models such as mBERT and DistilmBERT, which are partially trained on Norwegian text, are also available. However, the monolingual models adapted to a specific language tend to achieve better results.

RQ2 - Can active learning be used to create sentiment labels on customer emails?

AL was performed using an NB-BERT model fine-tuned on the NoReC dataset as a starting point, followed by multiple rounds of manual labeling. In the end, 400 emails were manually labeled and used to fine-tune the BERT model, which labeled the remaining dataset. Employees at Sparebank 1 SMN evaluated parts of the labels and found that 92% of the positive labels were correct, but this was only the case for 72% of the negatively labeled examples. These findings suggest that AL can be used for labeling. However, there is room for improvement. The incorrect labels could have affected the results on **RQ3**, as the related experiments relied on the labeled SMN dataset.

RQ3 - How do different transformer-based language models compare to each other when used to solve sentiment analysis tasks? And how do transformer-based language models compare to previously used methods for sentiment analysis?

These two questions were answered by training and making predictions on the NoReC and SMN datasets. On the NoReC dataset, NB-BERT achieved the highest score, followed by NorBERT, mBERT, and DistilmBERT. This order also corresponded well with previous work in the field. The experiments conducted on the SMN dataset found that NorBERT outperformed all others, while mBERT and DistilmBERT also showed decent results. However, NB-BERT classified almost every example as negative, indicating issues that did not occur when training the others, such as a poor choice of hyper-parameters or overfitting. The previously much used method TF-IDF combined with SVM was used as a baseline model and compared to all BERT models. NB-BERT and NorBERT performed better than TF-IDF on the NoReC dataset, while only NorBERT outperformed TF-IDF on the SMN dataset. Another aspect worth noting is the time spent on predictions. It was found that TF-IDF was much faster than all BERT models, and larger models are generally slower.

8.2 Future Work

Due to the limited time frame of the project, the experiments conducted were constrained in scope. As a result, there is room for further improvements in the experiments, which will be discussed below.

8.2.1 Improving the Active Learning Process

As the results from the AL process were not as good as desired, future work should examine possibilities to improve the process. As mentioned in Section 7.2, the most likely reason for the poor results was overfitting, meaning that methods to mitigate overfitting should be tested. One possible approach is to do as Hoi et al. [28] and use

the Fisher information matrix when determining which examples to label. Another possibility is to use clustering. Then, examples far from each other should be chosen to label so that examples representing the entire dataset are used for training.

8.2.2 Testing More Model Versions

It would be good to tune the hyper-parameters of the BERT models on the SMN dataset, as this can improve the results. To do this, finding a way to run the code on GPU would be beneficial, since this would significantly reduce the time needed for the hyper-parameter tuning. It can also be interesting to examine what hyper-parameters have the most significant effect on the final results. If they are found, more care can be taken to tune them properly, and less time has to be spent on hyper-parameters with little impact on the results.

In recent years, there has been a significant increase in the creation of new models. While not all of them are trained on the Norwegian language, models that achieve good scores on Norwegian tasks are occasionally released. Therefore, it is advantageous to keep an eye open for new models in the future and test them on relevant tasks so that their potential can be determined.

8.2.3 Reducing Errors from Preprocessing

In the future, some effort should be spent on improving the preprocessing for better results. As noted in Section 7.4, splitting multiple emails found in one description was a challenge. Examining if there exists a better and more consistent way to split emails could improve the results, as the current splits sometimes fail. Models can be confused if two emails are in the provided description or if a part of the email containing sentiment is removed because it is split too early. Also, since the string [PERSON] replaced many words that should have been kept, further work to identify people's names should be done to mitigate this problem.

8.2.4 Creating a Larger Dataset

Collecting a larger dataset of emails might also have certain benefits. It could improve model training since the BERT models are expected to learn more when trained on more examples. Showing the models more example data could also decrease the risk of overfitting. Furthermore, having a larger dataset would also result in a larger validation dataset, which can better represent all emails. The validation dataset plays an important role in selecting the final model, and using a dataset that does not represent the actual emails might lead to wrong decisions being made. As a result, a better model could be available, but it might not be discovered during the validation process.

8.2.5 Sentiment Analysis from a Business Perspective

Finally, the business aspect of the applications should also receive some focus in the future. One thing that needs to be examined from a business perspective is how many negative and positive examples can be misclassified. If it is acceptable that many positive emails are misclassified as negative, more negative emails will be detected. However, depending on the use case for the SA, this might cause more work for employees. Also, before a model can be chosen, it should be investigated how the SA will be used and whether runtime is essential for the application. For example, if real-time SA should be performed, choosing a faster model is more important than selecting one with a slightly higher accuracy.

Bibliography

[1] D. Bahdanau, K. Cho and Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, May 2016. DOI: 10.48550/arXiv.1409.0473. [Online]. Available: http://arxiv.org/abs/1409.0473 (visited on 20th Jan. 2023).

[2] *Behind the pipeline*. [Online]. Available: https://huggingface.co/course/chapter2/2?fw=pt#behind-the-pipeline (visited on 22nd Mar. 2023).

[3] I. Beltagy, K. Lo and A. Cohan, "SciBERT: A Pretrained Language Model for Scientific Text", in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3615–3620. DOI: 10.18653/v1/D19-1371. [Online]. Available: https://aclanthology.org/D19-1371 (visited on 20th Jan. 2023).

[4] B. Bengfort, R. Bilbro and T. Ojeda, *Applied Text Analysis with Python*. O'Reilly Media, 2018.

[5] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization", en, *Journal of machine learning research*, vol. 13, no. 2, 2012.

[6] *BERT*, original-date: 2018-10-25T22:57:34Z, 2019. [Online]. Available: https://github.com/google-research/bert/blob/eedf5716ce1268e56f0a50264a88cafad334ac61/multilingual.md (visited on 20th Jan. 2023).

[7] *Bert-base-multilingual-cased · Hugging Face*, Nov. 2022. [Online]. Available: https://huggingface.co/bert-base-multilingual-cased (visited on 20th Mar. 2023).

[8] *BertForSequenceClassification*. [Online]. Available: https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertForSequenceClassification (visited on 22nd Mar. 2023).

[9] J. Blitzer, M. Dredze and F. Pereira, "Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification", in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic: Association for Computational Linguistics, Jun. 2007, pp. 440–447. [Online]. Available: https://aclanthology.org/P07-1056 (visited on 20th Jan. 2023).

[10] P. Bojanowski, E. Grave, A. Joulin and T. Mikolov, "Enriching Word Vectors with Subword Information", *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, Jun. 2017, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051. [Online]. Available: https://doi.org/10.1162/tacl_a_00051 (visited on 24th Apr. 2023).

[11] J. Chen, A. Schein, L. Ungar and M. Palmer, "An Empirical Study of the Behavior of Active Learning for Word Sense Disambiguation", in *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, New York City, USA: Association for Computational Linguistics, Jun. 2006, pp. 120–127. [Online]. Available: https://aclanthology.org/N06-1016 (visited on 27th Apr. 2023).

[12] J. Cheng, L. Dong and M. Lapata, *Long Short-Term Memory-Networks for Machine Reading*, Sep. 2016. DOI: 10.48550/arXiv.1601.06733. [Online]. Available: http://arxiv.org/abs/1601.06733 (visited on 20th Jan. 2023).

[13] K. Cho, B. van Merrienboer, C. Gulcehre *et al.*, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, Sep. 2014. [Online]. Available: http://arxiv.org/abs/1406.1078 (visited on 17th Apr. 2023).

[14] P. Christen, *Data Matching*, en. Springer Berlin, Heidelberg, 2012. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-31164-2 (visited on 20th Jan. 2023).

[15] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, Dec. 2014. DOI: 10.48550/arXiv.1412.3555. [Online]. Available: http://arxiv.org/abs/1412.3555 (visited on 20th Jan. 2023).

[16] A. Conneau, K. Khandelwal, N. Goyal *et al.*, *Unsupervised Cross-lingual Representation Learning at Scale*, Apr. 2020. DOI: 10.48550/arXiv.1911.02116. [Online]. Available: http://arxiv.org/abs/1911.02116 (visited on 24th Apr. 2023).

[17] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, May 2019. DOI: 10.48550/arXiv.1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805 (visited on 19th Jan. 2023).

[18] *Distilbert-base-multilingual-cased · Hugging Face*, Nov. 2022. [Online]. Available: https://huggingface.co/distilbert-base-multilingual-cased (visited on 20th Mar. 2023).

[19] C. Du, H. Sun, J. Wang, Q. Qi and J. Liao, "Adversarial and Domain-Aware BERT for Cross-Domain Sentiment Analysis", in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 4019–4028. DOI: 10.18653/v1/2020.acl-main.370. [Online]. Available: https://aclanthology.org/2020.acl-main.370 (visited on 14th May 2023).

[20] S. Farquhar, Y. Gal and T. Rainforth, *On Statistical Bias In Active Learning: How and When To Fix It*, May 2021. DOI: 10.48550/arXiv.2101.11665. [Online]. Available: http://arxiv.org/abs/2101.11665 (visited on 27th Apr. 2023).

[21] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk and F. Herrera, *Learning from Imbalanced Data Sets*, en. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-98074-4. DOI: 10.1007/978-3-319-98074-4. [Online]. Available: http://link.springer.com/10.1007/978-3-319-98074-4 (visited on 15th Apr. 2023).

[22] D. Geebelen, J. A. K. Suykens and J. Vandewalle, "Reducing the Number of Support Vectors of SVM Classifiers Using the Smoothed Separable Case Approximation", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 4, pp. 682–688, Apr. 2012, Conference Name: IEEE Transactions on Neural Networks and Learning Systems, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2012.2186314.

[23] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019, vol. 2, ISBN: 978-1-4920-3264-9. (visited on 15th Apr. 2023).

[24] *Glossary*. [Online]. Available: https://huggingface.co/docs/transformers/glossary (visited on 2nd May 2023).

[25] C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, *On Calibration of Modern Neural Networks*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1706.04599 (visited on 27th Apr. 2023).

[26] M. Hoang, O. A. Bihorac and J. Rouces, "Aspect-Based Sentiment Analysis using BERT", in *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, Turku, Finland: Linköping University Electronic Press, Sep. 2019, pp. 187–196. [Online]. Available: https://aclanthology.org/W19-6120 (visited on 20th Jan. 2023).

[27] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[28] S. C. H. Hoi, R. Jin and M. R. Lyu, "Large-scale text categorization by batch mode active learning", en, in *Proceedings of the 15th international conference on World Wide Web*, Edinburgh Scotland: ACM, May 2006, pp. 633–642, ISBN: 978-1-59593-323-2. DOI: 10.1145/1135777.1135870. [Online]. Available: https://dl.acm.org/doi/10.1145/1135777.1135870 (visited on 8th May 2023).

[29] *Hugging Face – The AI community building the future.* [Online]. Available: https://huggingface.co/ (visited on 2nd May 2023).

[30] C. Kittask, K. Milintsevich and K. Sirts, *Evaluating Multilingual BERT for Estonian*, en. IOS Press, Sep. 2020, ISBN: 978-1-64368-117-7.

[31] S. Kostadinov, *Recurrent Neural Networks with Python Quick Start Guide : Sequential Learning and Language Modeling with TensorFlow*, English. Birmingham, UK: Packt Publishing, 2018, ISBN: 978-1-78913-233-5. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1950552&site=ehost-live&scope=site (visited on 21st Jan. 2023).

[32] D. Kotzias, M. Denil, N. de Freitas and P. Smyth, "From Group to Individual Labels Using Deep Features", in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, New York, NY, USA: Association for Computing Machinery, Aug. 2015, pp. 597–606, ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783380. [Online]. Available: https://dl.acm.org/doi/10.1145/2783258.2783380 (visited on 15th May 2023).

[33] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, English, 1st ed. 2013, Corr. 2nd printing 2018 edition. New York: Springer, May 2013, ISBN: 978-1-4614-6849-3. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4614-6849-3.

[34] P. E. Kummervold, J. De la Rosa, F. Wetjen and S. A. Brygfjeld, "Operationalizing a National Digital Library: The Case for a Norwegian Transformer Model", in *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, Reykjavik, Iceland (Online): Linköping University Electronic Press, Sweden, May 2021, pp. 20–29. [Online]. Available: https://aclanthology.org/2021.nodalida-main.3 (visited on 20th Jan. 2023).

[35] A. Kutuzov, J. Barnes, E. Velldal, L. Øvrelid and S. Oepen, *Large-Scale Contextualised Language Modelling for Norwegian*, Apr. 2021. DOI: 10.48550/arXiv.2104.06546. [Online]. Available: http://arxiv.org/abs/2104.06546 (visited on 20th Jan. 2023).

[36] G. Lample and A. Conneau, *Cross-lingual Language Model Pretraining*, Jan. 2019. DOI: 10.48550/arXiv.1901.07291. [Online]. Available: http://arxiv.org/abs/1901.07291 (visited on 24th Apr. 2023).

[37] J. Lee, W. Yoon, S. Kim *et al.*, "BioBERT: A pre-trained biomedical language representation model for biomedical text mining", *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, Feb. 2020, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz682. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz682 (visited on 20th Jan. 2023).

[38] M. Lewis, Y. Liu, N. Goyal *et al.*, *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, Oct. 2019. DOI: 10.48550/arXiv.1910.13461. [Online]. Available: http://arxiv.org/abs/1910.13461 (visited on 24th Apr. 2023).

[39] P. Liu, C. Marco and J. A. Gulla, "Semi-supervised Sentiment Analysis for Under-resourced Languages with a Sentiment Lexicon", en, in *INRA@ RecSys*, 2019, pp. 12–17.

[40] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng and C. Potts, "Learning Word Vectors for Sentiment Analysis", in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA: Association for Computational Linguistics, Jun. 2011, pp. 142–150. [Online]. Available: https://aclanthology.org/P11-1015 (visited on 24th Apr. 2023).

[41] A. Madasu and S. E, *A Study of Feature Extraction techniques for Sentiment Analysis*, Jun. 2019. [Online]. Available: http://arxiv.org/abs/1906.01573 (visited on 14th May 2023).

[42] W. Medhat, A. Hassan and H. Korashy, "Sentiment analysis algorithms and applications: A survey", en, *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, Dec. 2014, ISSN: 2090-4479. DOI: 10.1016/j.asej.2014.04.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447914000550 (visited on 20th Jan. 2023).

[43] K. Mishev, A. Gjorgjevikj, I. Vodenska, L. T. Chitkushev and D. Trajanov, "Evaluation of Sentiment Analysis in Finance: From Lexicons to Transformers", *IEEE Access*, vol. 8, pp. 131 662–131 682, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3009626.

[44] T. Al-Moslmi, N. Omar, S. Abdullah and M. Albared, "Approaches to Cross-Domain Sentiment Analysis: A Systematic Literature Review", *IEEE Access*, vol. 5, pp. 16 173–16 192, 2017, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2690342.

[45] B. Myagmar, J. Li and S. Kimura, "Cross-Domain Sentiment Classification With Bidirectional Contextualized Transformer Language Models", *IEEE Access*, vol. 7, pp. 163 219–163 230, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2952360.

[46] *NoReC: The Norwegian Review Corpus*, original-date: 2017-09-25T06:34:21Z, Nov. 2022. [Online]. Available: https://github.com/ltgoslo/norec (visited on 3rd Mar. 2023).

[47] *Norwegian Transformer Model*, original-date: 2020-09-17T08:18:48Z, Mar. 2023. [Online]. Available: https://github.com/NbAiLab/notram (visited on 20th Mar. 2023).

[48] L. Øvrelid, P. Mæhlum, J. Barnes and E. Velldal, "A Fine-grained Sentiment Dataset for Norwegian", English, in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, May 2020, pp. 5025–5033, ISBN: 979-10-95546-34-4. [Online]. Available: https://aclanthology.org/2020.lrec-1.618 (visited on 23rd Apr. 2023).

[49] H. Pajupuu, R. Altrov and J. Pajupuu, "Identifying Polarity in Different Text Types", *Folklore: Electronic Journal of Folklore*, vol. 64, pp. 125–142, Jun. 2016. DOI: 10.7592/FEJF2016.64.polarity.

[50] S. J. Pan and Q. Yang, "A Survey on Transfer Learning", *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, Conference Name: IEEE Transactions on Knowledge and Data Engineering, ISSN: 1558-2191. DOI: 10.1109/TKDE.2009.191.

[51] B. Pang and L. Lee, "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts", in *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, Jul. 2004, pp. 271–278. DOI: 10.3115/1218955.1218990. [Online]. Available: https://aclanthology.org/P04-1035 (visited on 15th May 2023).

[52] B. Pang and L. Lee, "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales", in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 115–124. DOI: 10.3115/1219840.1219855. [Online]. Available: https://aclanthology.org/P05-1015 (visited on 15th May 2023).

[53] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language Models are Unsupervised Multitask Learners", *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[54] S. Russell and P. Norvig, *Artificial Intelligence : A Modern Approach* (Prentice Hall series in artificial intelligence), 3rd, global edition. Pearson, 2016, ISBN: 1-292-15396-2.

[55] V. Sanh, L. Debut, J. Chaumond and T. Wolf, *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter*, Feb. 2020. DOI: 10.48550/arXiv.1910.01108. [Online]. Available: http://arxiv.org/abs/1910.01108 (visited on 24th Jan. 2023).

[56] B. Settles, "Active Learning Literature Survey", en, University of Wisconsin-Madison Department of Computer Sciences, Technical Report, 2009, Accepted: 2012-03-15T17:23:56Z. [Online]. Available: https://minds.wisconsin.edu/handle/1793/60660 (visited on 21st Jan. 2023).

[57] M. Själander, M. Jahre, G. Tufte and N. Reissmann, *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure*, Feb. 2022. DOI: 10.48550/arXiv.1912.05848. [Online]. Available: http://arxiv.org/abs/1912.05848 (visited on 9th May 2023).

[58] R. Socher, A. Perelygin, J. Wu *et al.*, "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank", in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://aclanthology.org/D13-1170 (visited on 24th Apr. 2023).

[59] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", en, *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, Jul. 2009, ISSN: 0306-4573. DOI: 10.1016/j.ipm.2009.03.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457309000259 (visited on 13th Apr. 2023).

[60] M. G. Sousa, K. Sakiyama, L. d. S. Rodrigues, P. H. Moraes, E. R. Fernandes and E. T. Matsubara, "BERT for Stock Market Sentiment Analysis", in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, ISSN: 2375-0197, Nov. 2019, pp. 1597–1601. DOI: 10.1109/ICTAI.2019.00231.

[61] J. A. K. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor and J. Vandewalle, *Least Squares Support Vector Machines*, English. River Edge, NJ: World Scientific Publishing Company, Nov. 2002, ISBN: 981-238-151-1.

[62] Thomas H. Davenport, Jim Guszcza, Tim Smith and Ben Stiller, *Analytics and AI-driven enterprises thrive in the Age of With*, en, Jul. 2019. [Online]. Available: https://www2.deloitte.com/us/en/insights/topics/analytics/insight-driven-organization.html (visited on 1st Jun. 2023).

[63] M. Tsytsarau and T. Palpanas, "Survey on mining subjective data on the web", en, *Data Mining and Knowledge Discovery*, vol. 24, no. 3, pp. 478–514, May 2012, ISSN: 1573-756X. DOI: 10.1007/s10618-011-0238-6. [Online]. Available: https://doi.org/10.1007/s10618-011-0238-6 (visited on 21st Jan. 2023).

[64] A. Vaswani, N. Shazeer, N. Parmar *et al.*, *Attention Is All You Need*, Dec. 2017. DOI: 10.48550/arXiv.1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 21st Jan. 2023).

[65] E. Velldal, L. Øvrelid, E. A. Bergem, C. Stadsnes, S. Touileb and F. Jørgensen, "NoReC: The Norwegian Review Corpus", in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://aclanthology.org/L18-1661 (visited on 3rd Mar. 2023).

[66] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy and S. R. Bowman, *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*, Feb. 2019. DOI: 10.48550/arXiv.1804.07461. [Online]. Available: http://arxiv.org/abs/1804.07461 (visited on 24th Apr. 2023).

[67] Z. Wang, S. Yan and C. Zhang, "Active learning with adaptive regularization", en, *Pattern Recognition*, Semi-Supervised Learning for Visual Content Analysis and Understanding, vol. 44, no. 10, pp. 2375–2383, Oct. 2011, ISSN: 0031-3203. DOI: 10.1016/j.patcog.2011.03.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320311000938 (visited on 27th Apr. 2023).

[68] Y. Wu, M. Schuster, Z. Chen *et al.*, *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, Oct. 2016. DOI: 10.48550/arXiv.1609.08144. [Online]. Available: http://arxiv.org/abs/1609.08144 (visited on 21st Jan. 2023).

[69] Y. Yang, M. C. S. UY and A. Huang, *FinBERT: A Pretrained Language Model for Financial Communications*, Jul. 2020. DOI: 10.48550/arXiv.2006.08097. [Online]. Available: http://arxiv.org/abs/2006.08097 (visited on 21st Jan. 2023).

[70] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov and Q. V. Le, *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, Jan. 2020. DOI: 10.48550/arXiv.1906.08237. [Online]. Available: http://arxiv.org/abs/1906.08237 (visited on 21st Jan. 2023).

[71] M. Yuan, H.-T. Lin and J. Boyd-Graber, *Cold-start Active Learning through Self-supervised Language Modeling*, Oct. 2020. [Online]. Available: http://arxiv.org/abs/2010.09535 (visited on 27th Apr. 2023).

[72] Z. Zhang, E. Strubell and E. Hovy, *A Survey of Active Learning for Natural Language Processing*, Feb. 2023. [Online]. Available: http://arxiv.org/abs/2210.10109 (visited on 27th Apr. 2023).

[73] Y. Zhu, R. Kiros, R. Zemel *et al.*, *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*, Jun. 2015. DOI: 10.48550/arXiv.1506.06724. [Online]. Available: http://arxiv.org/abs/1506.06724 (visited on 21st Jan. 2023).

Appendix A

Example Description

This appendix shows an example of what the HTML in the original SMN dataset's description field might look like. However, it is a made-up example, not an actual received email, and is only added here to give an idea of the format. The format also varies between different emails.

```
<HTML>
    <HEAD>
        \R\N<META HTTP-EQUIV="CONTENT-TYPE" CONTENT="TEXT/HTML;
        ↪  CHARSET=UTF-8">
    </HEAD>
    <BODY DIR="AUTO">
        THIS IS THE BEGINNING OF THE EMAIL SENT FROM A CUSTOMER.
        <DIV><BR></DIV>
        <DIV>
            THEN THERE MIGHT BE ANOTHER PARAGRAPH OR SOMETHING.
            <DIV><BR></DIV>
            <DIV>
                BEFORE THEY SAY BEST REGARDS ME. AND THEN THERE IS
                ↪  SOME MORE HTML.
                <!--\R\N\N\T{FONT-FAMILY:"CAMBRIA MATH"}\R\N\N\T
                ↪  {FONT-FAMILY:CALIBRI}\R\NP.MSONORMAL,LI.MSON
                ↪  ORMAL, DIV.MSONORMAL\R\N\T{MARGIN:0CM;\R\N\T
                ↪  FONT-SIZE:11.0PT;\R\N\TFONT-FAMILY:"CALIBRI"
                ↪  ,SANS-SERIF}\R\NSPAN.EPOSTSTIL17\R\N\T{FONT-
                ↪  FAMILY:"CALIBRI",SANS-SERIF;\R\N\TCOLOR:WIND
                ↪  OWTEXT}\R\N.MSOCHPDEFAULT\R\N\T{FONT-FAMILY:
                ↪  "CALIBRI",SANS-SERIF}\R\NWORDSECTION1\R\N\T{
                ↪  MARGIN:70.85PT 70.85PT 70.85PT 70.85PT}\R\ND
                ↪  IV.WORDSECTION1\R\N\T{}\R\N-->
            \R\N
            </DIV>
            <DIV CLASS="WORDSECTION1">
```

```
                <P CLASS="MSONORMAL">
                    THIS IS THE BEGINNING OF AN EMAIL SENT FROM
                     ↪  SOMEONE IN SPAREBANK 1.
                </P>
                <P CLASS="MSONORMAL"> </P>
                <P CLASS="MSONORMAL"> </P>
                <P CLASS="MSONORMAL">
                    THIS IS MORE OF THE SAME EMAIL. THIS IS ALSO THE
                     ↪  EMAIL THE CUSTOMER IS ANSWERING. AND IT CAN BE
                     ↪  SEVERAL MESSEGES GOING BACK AND FORTH LIKE
                     ↪  THIS, EVEN THOUGH THIS EXAMPLE ONLY INCLUDES
                     ↪  ONE FROM SPAREBANK 1 AND ONE FROM A CUSTOMER.
                    <SPAN LANG="EN-US" STYLE=""><BR> </SPAN>
                    <SPAN LANG="EN-US"></SPAN>
                </P>
            </DIV>
        </DIV>
    </BODY>
</HTML>
```

Appendix B

$\text{NoReC}_{\text{train\_full}}$ vs. $\text{NoReC}_{\text{train\_no\_neutral}}$

The table below shows the scores achieved by training two similar NorBERT models on the two different datasets $\text{NoReC}_{\text{train\_full}}$ and $\text{NoReC}_{\text{train\_no\_neutral}}$. The hyperparameters used were the same for both models. This was done to decide which NoReC dataset should be used for training.

Table B.1: The results from the `classification_report()` function on $\text{NoReC}_{\text{test}}$ when using two different training datasets. F is $\text{NoReC}_{\text{train\_full}}$, N is $\text{NoReC}_{\text{train\_no\_neutral}}$, A is accuracy, P is precision, and R is recall. Precision, recall, and F1-score are provided for the negative (0) and positive (1) classes. The best score in each column is marked in bold.

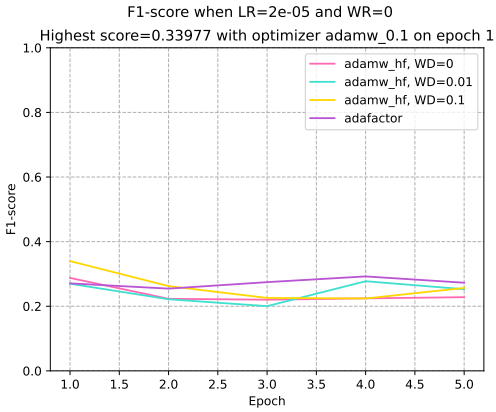
Dataset	**A**	**P 0**	**P 1**	**R 0**	**R 1**	**F1 0**	**F1 1**
F	0.45	0.09	**0.99**	0.91	0.42	0.17	0.59
N	**0.66**	**0.14**	**0.99**	**0.93**	**0.64**	**0.25**	**0.78**

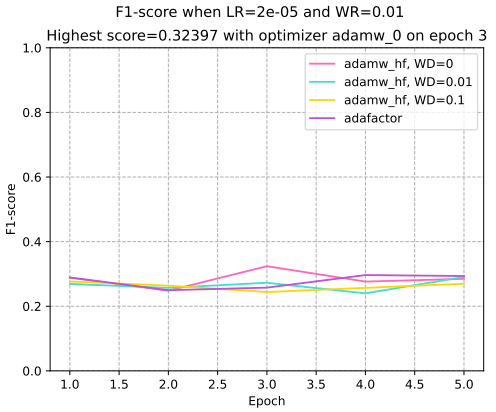
Appendix C

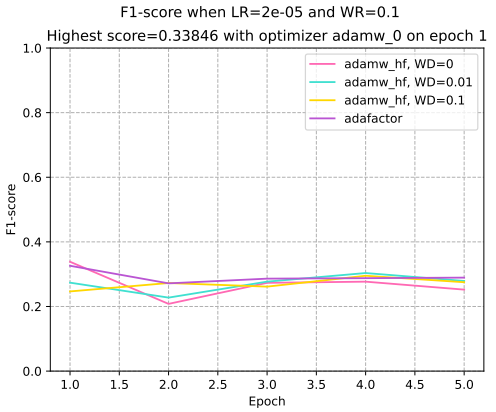
Results on $\text{NoReC}_{\text{val}}$ by the BERT Models

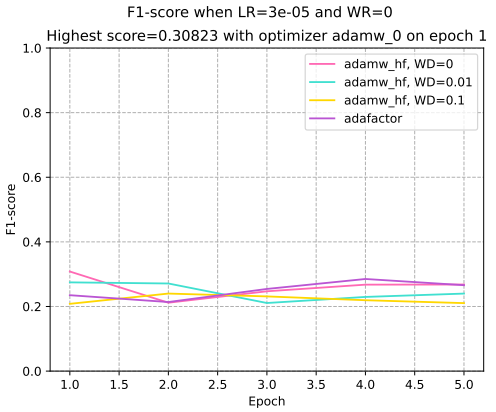
This appendix shows the performance of the various BERT models evaluated on $\text{NoReC}_{\text{val}}$ during parameter tuning. The figures show the F1-score on the negative class when evaluating the models on the dataset. All figures show the score of four different fine-tuned versions of the BERT models, where each version used a different combination of parameters. The highest score in each figure is specified above the graph in the figure.

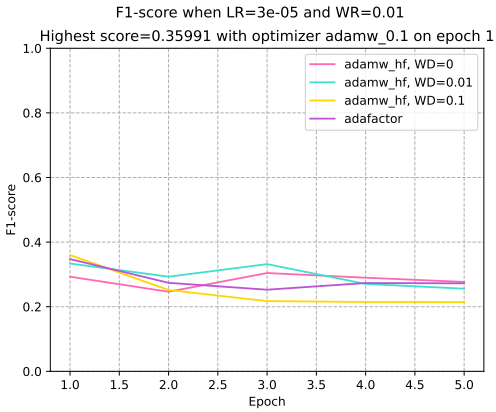
C.1 NB-BERT

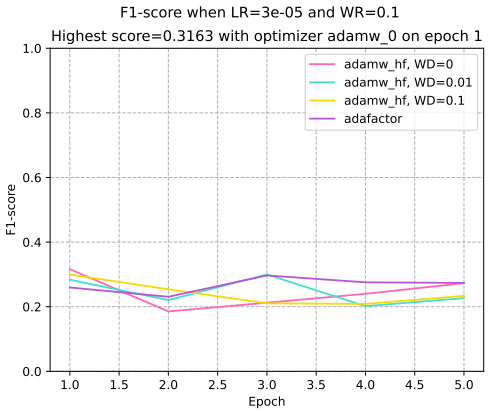

F1-score when LR=2e-05 and WR=0
Highest score=0.33977 with optimizer adamw_0.1 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

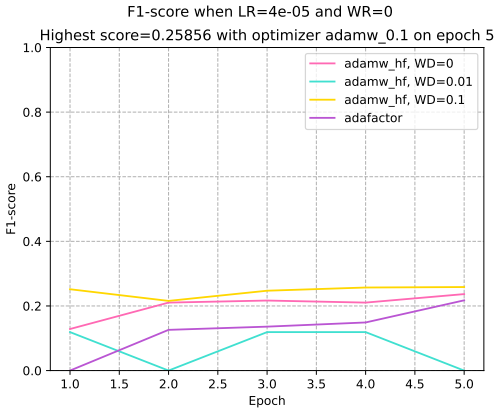

F1-score when LR=2e-05 and WR=0.01
Highest score=0.32397 with optimizer adamw_0 on epoch 3
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

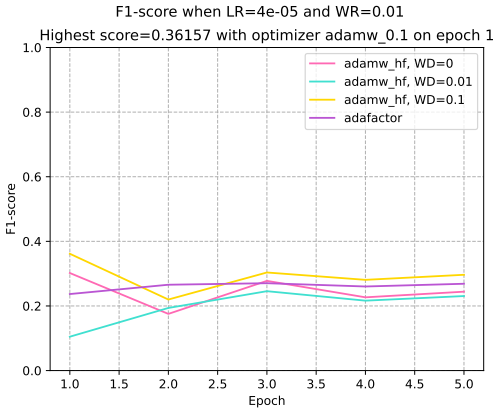

F1-score when LR=2e-05 and WR=0.1
Highest score=0.33846 with optimizer adamw_0 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0
Highest score=0.30823 with optimizer adamw_0 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0.01
Highest score=0.35991 with optimizer adamw_0.1 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0.1
Highest score=0.3163 with optimizer adamw_0 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=4e-05 and WR=0
Highest score=0.25856 with optimizer adamw_0.1 on epoch 5
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=4e-05 and WR=0.01
Highest score=0.36157 with optimizer adamw_0.1 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

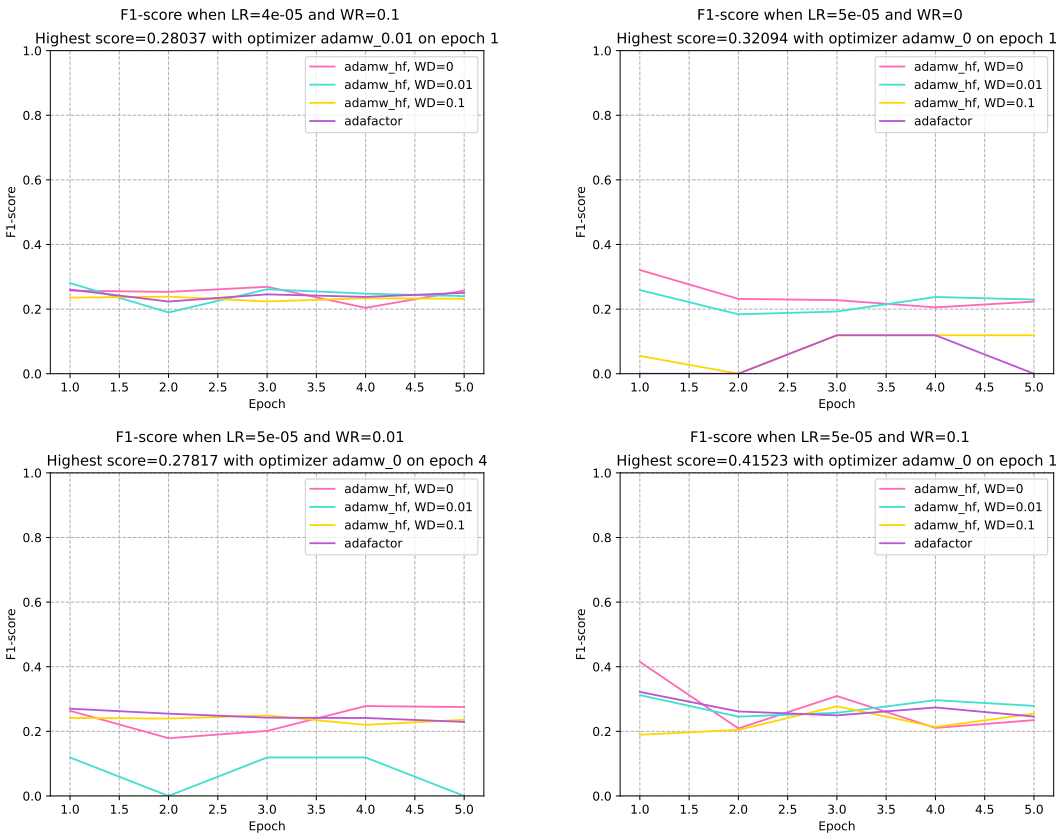
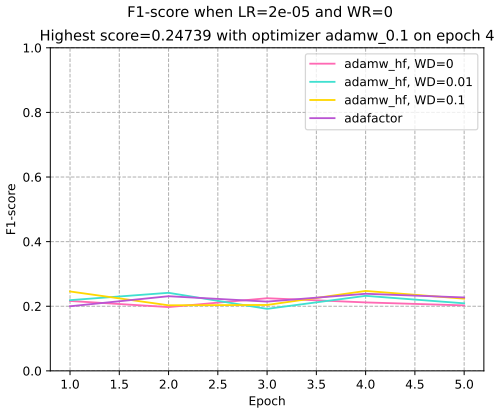
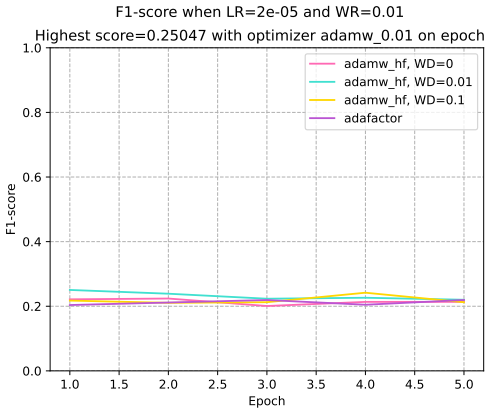
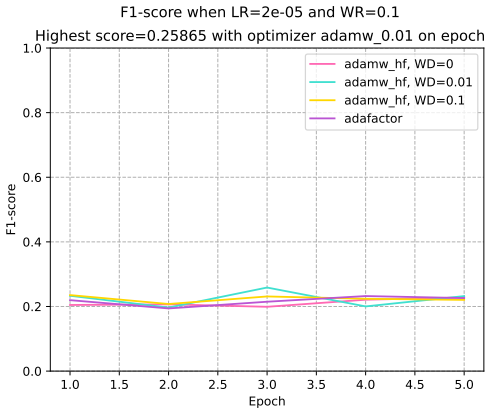


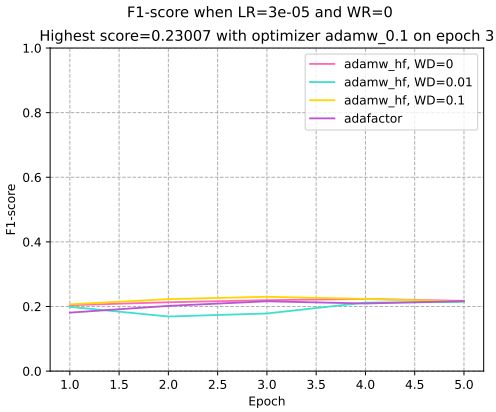
Figure C.1: Results on the validation dataset from the fine-tuning of NB-BERT.

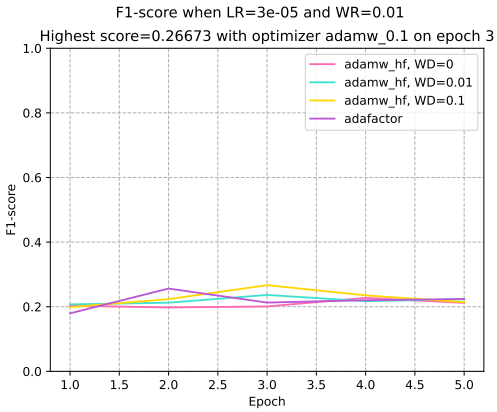
C.2 NorBERT

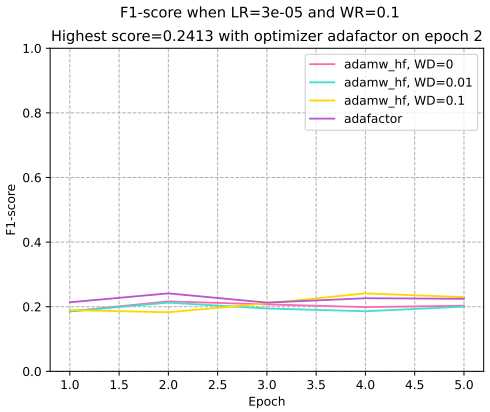

F1-score when LR=2e-05 and WR=0
Highest score=0.24739 with optimizer adamw_0.1 on epoch 4
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

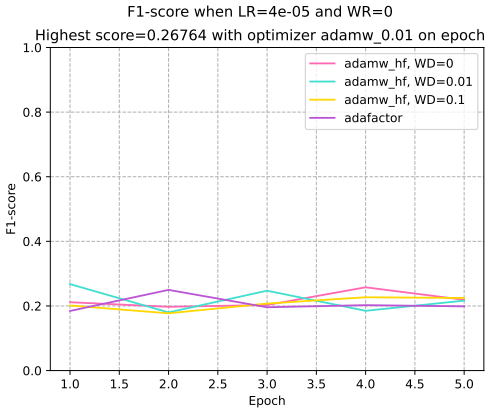

F1-score when LR=2e-05 and WR=0.01
Highest score=0.25047 with optimizer adamw_0.01 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

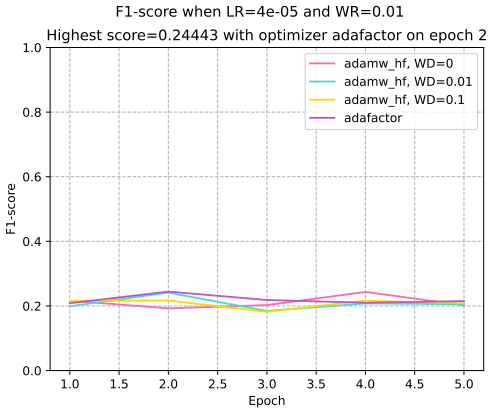

F1-score when LR=2e-05 and WR=0.1
Highest score=0.25865 with optimizer adamw_0.01 on epoch 3
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0
Highest score=0.23007 with optimizer adamw_0.1 on epoch 3
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0.01
Highest score=0.26673 with optimizer adamw_0.1 on epoch 3
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0.1
Highest score=0.2413 with optimizer adafactor on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=4e-05 and WR=0
Highest score=0.26764 with optimizer adamw_0.01 on epoch 1
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=4e-05 and WR=0.01
Highest score=0.24443 with optimizer adafactor on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

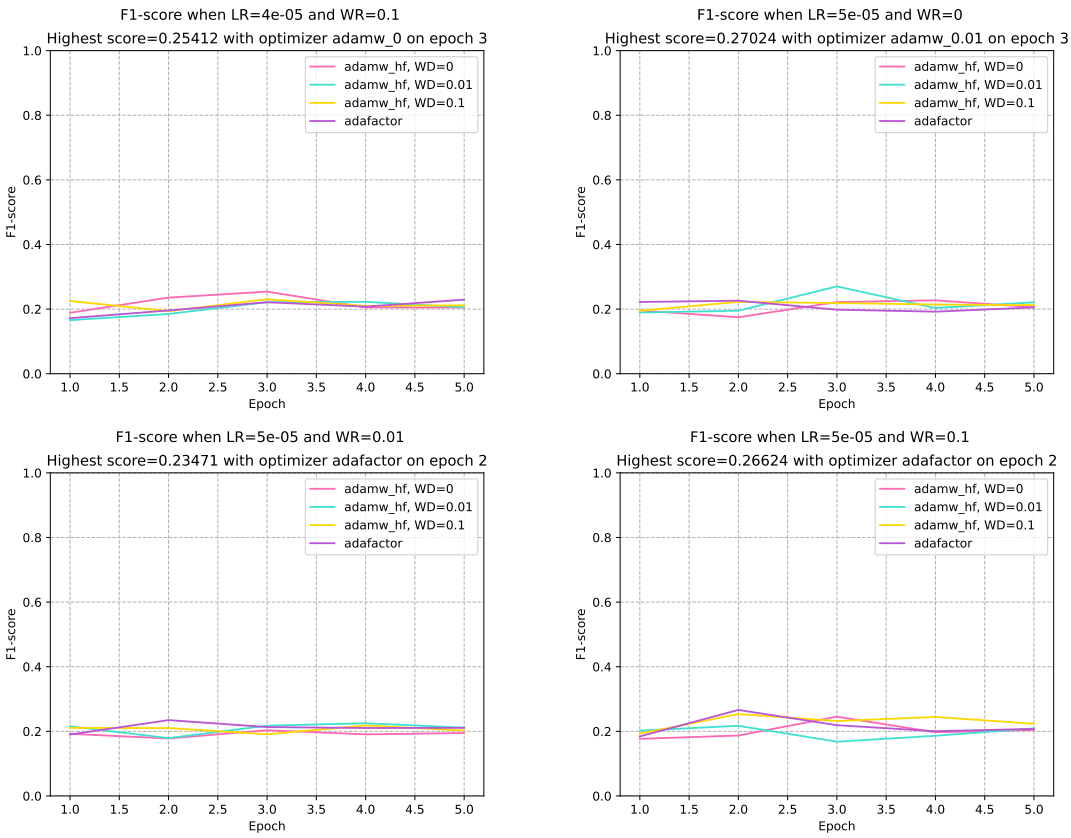
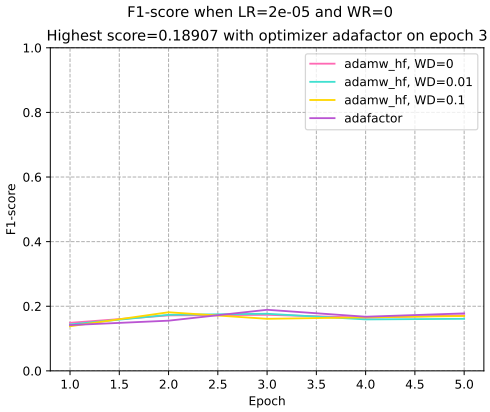
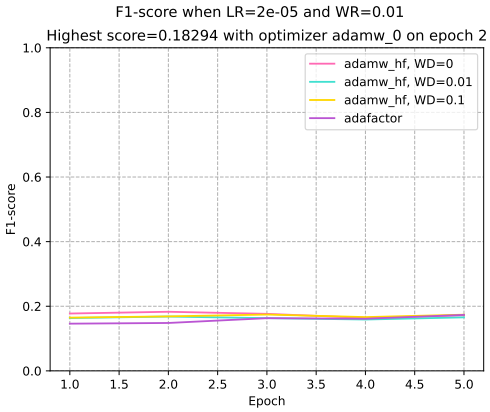
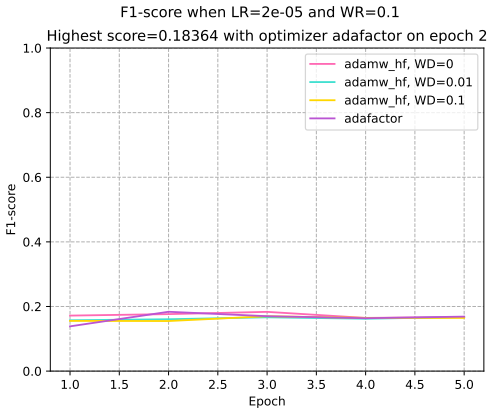


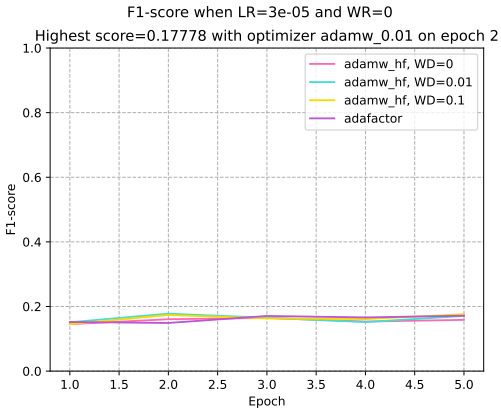
Figure C.2: Results on the validation dataset from the fine-tuning of NorBERT.

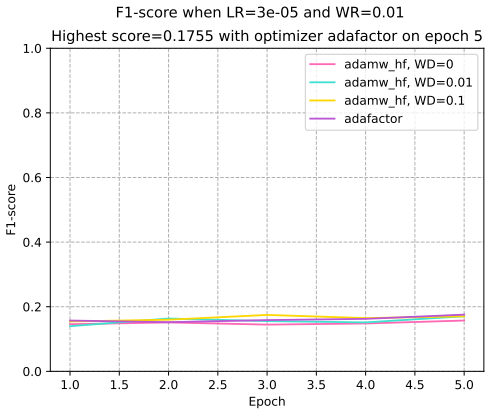
C.3 mBERT

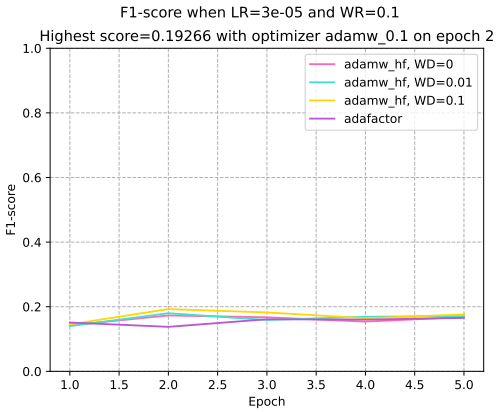

F1-score when LR=2e-05 and WR=0
Highest score=0.18907 with optimizer adafactor on epoch 3
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

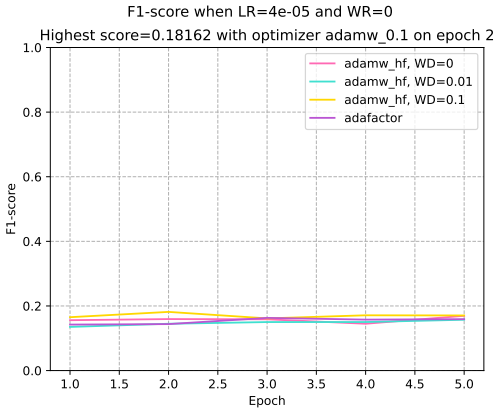

F1-score when LR=2e-05 and WR=0.01
Highest score=0.18294 with optimizer adamw_0 on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

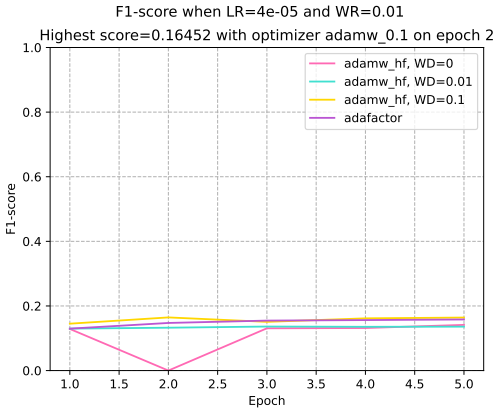

F1-score when LR=2e-05 and WR=0.1
Highest score=0.18364 with optimizer adafactor on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0
Highest score=0.17778 with optimizer adamw_0.01 on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0.01
Highest score=0.1755 with optimizer adafactor on epoch 5
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=3e-05 and WR=0.1
Highest score=0.19266 with optimizer adamw_0.1 on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=4e-05 and WR=0
Highest score=0.18162 with optimizer adamw_0.1 on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch


F1-score when LR=4e-05 and WR=0.01
Highest score=0.16452 with optimizer adamw_0.1 on epoch 2
adamw_hf, WD=0
adamw_hf, WD=0.01
adamw_hf, WD=0.1
adafactor
F1-score
Epoch

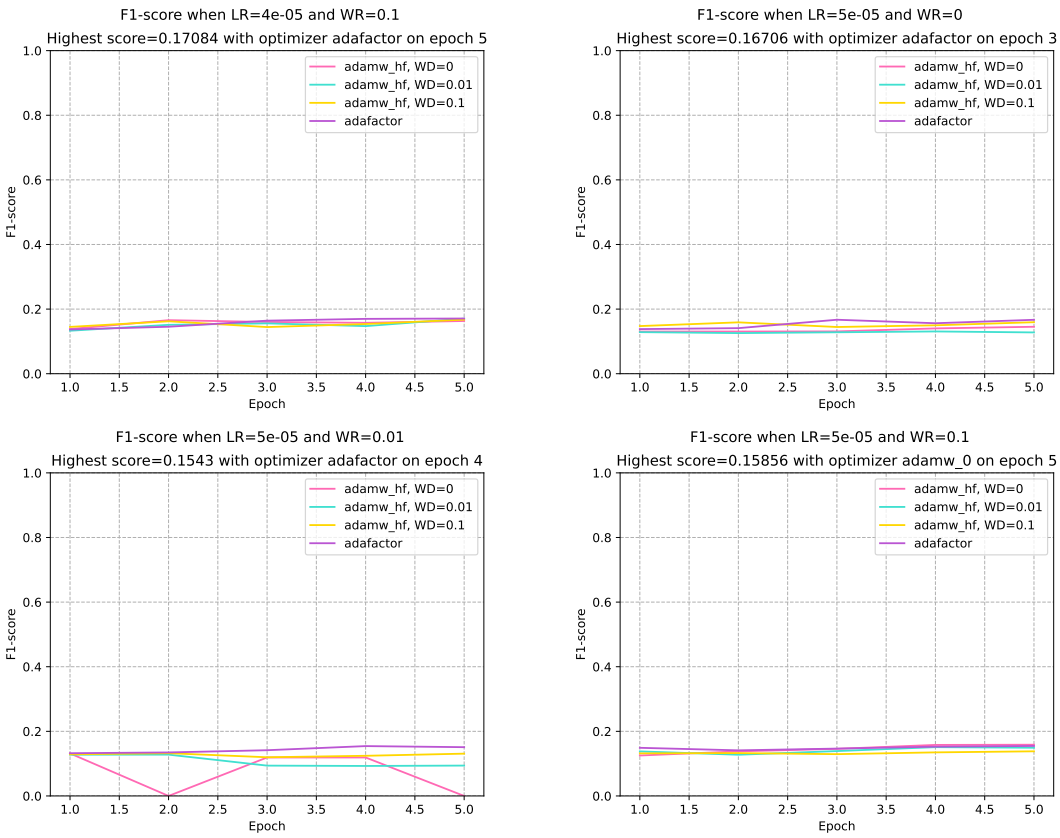
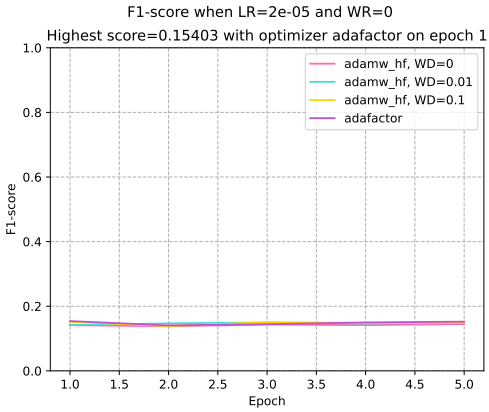
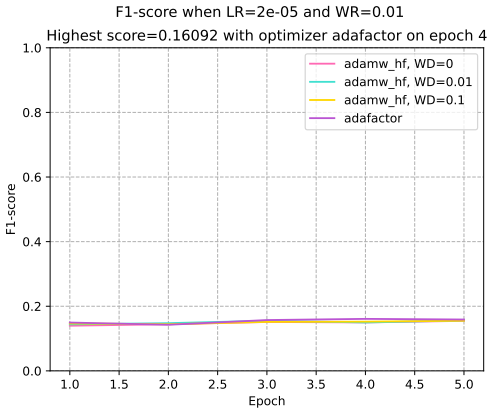
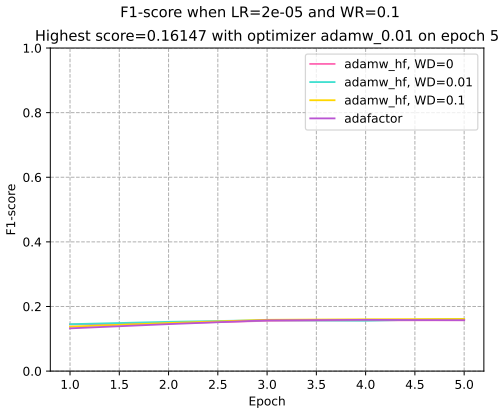


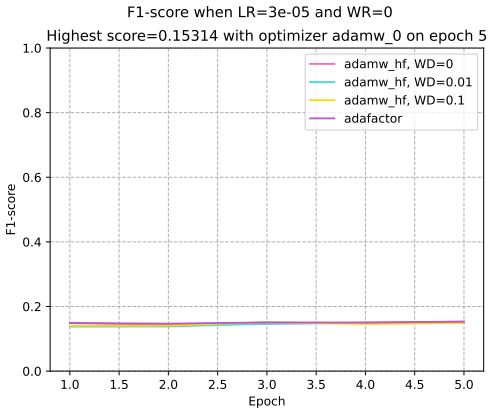
Figure C.3: Results on the validation dataset from the fine-tuning of mBERT.

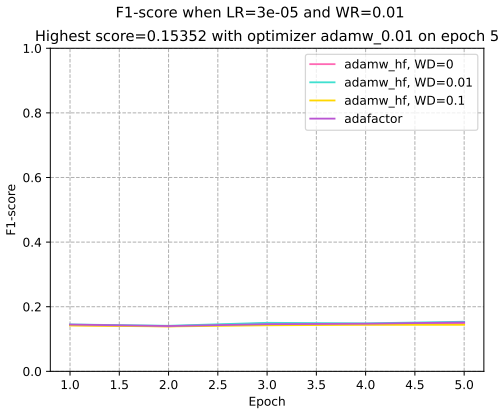
C.4 DistilmBERT

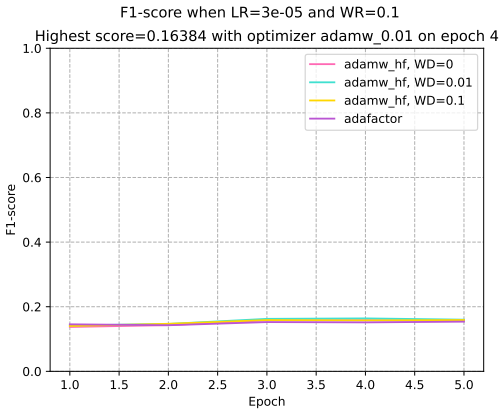


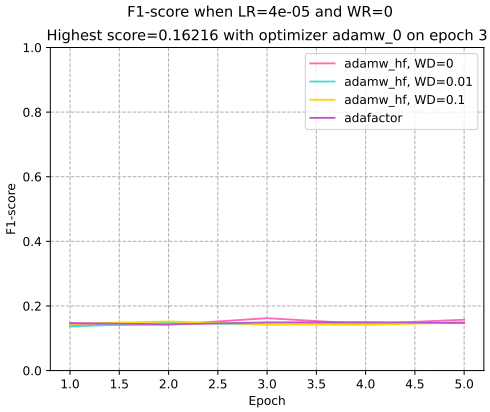


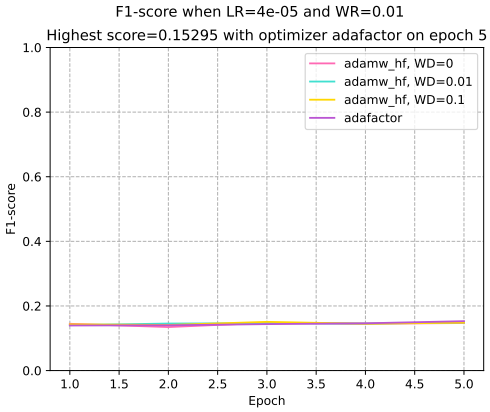












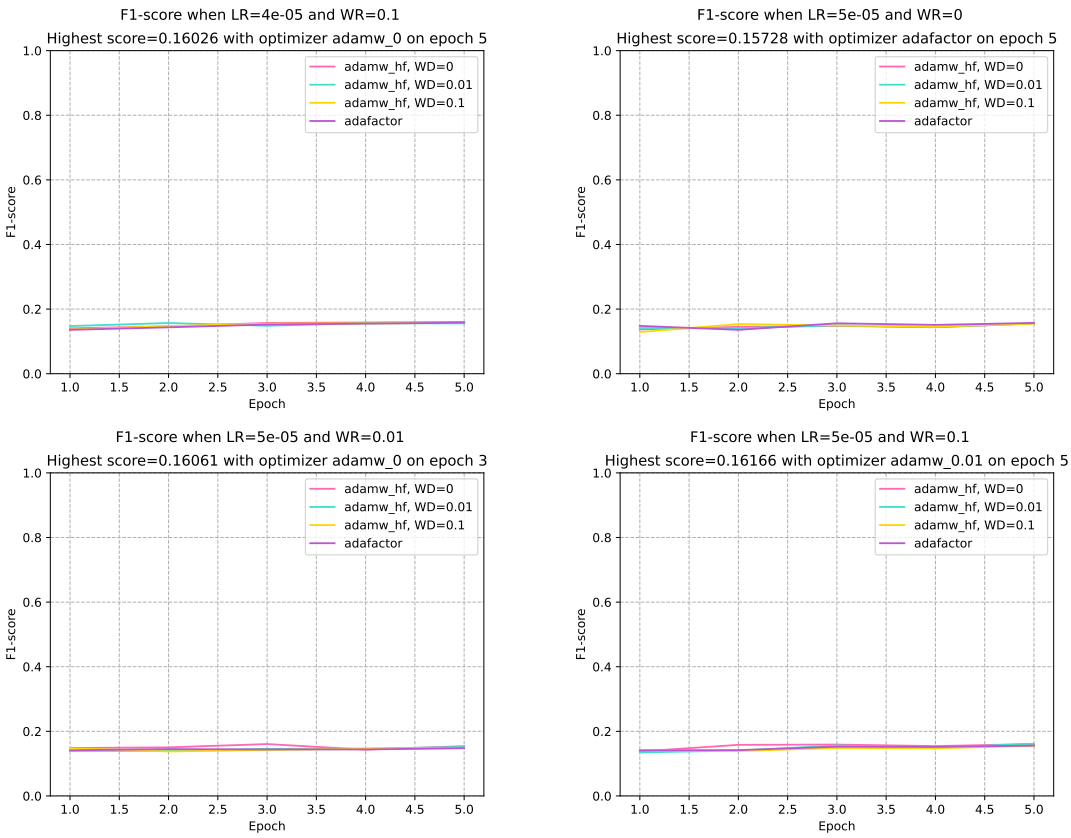
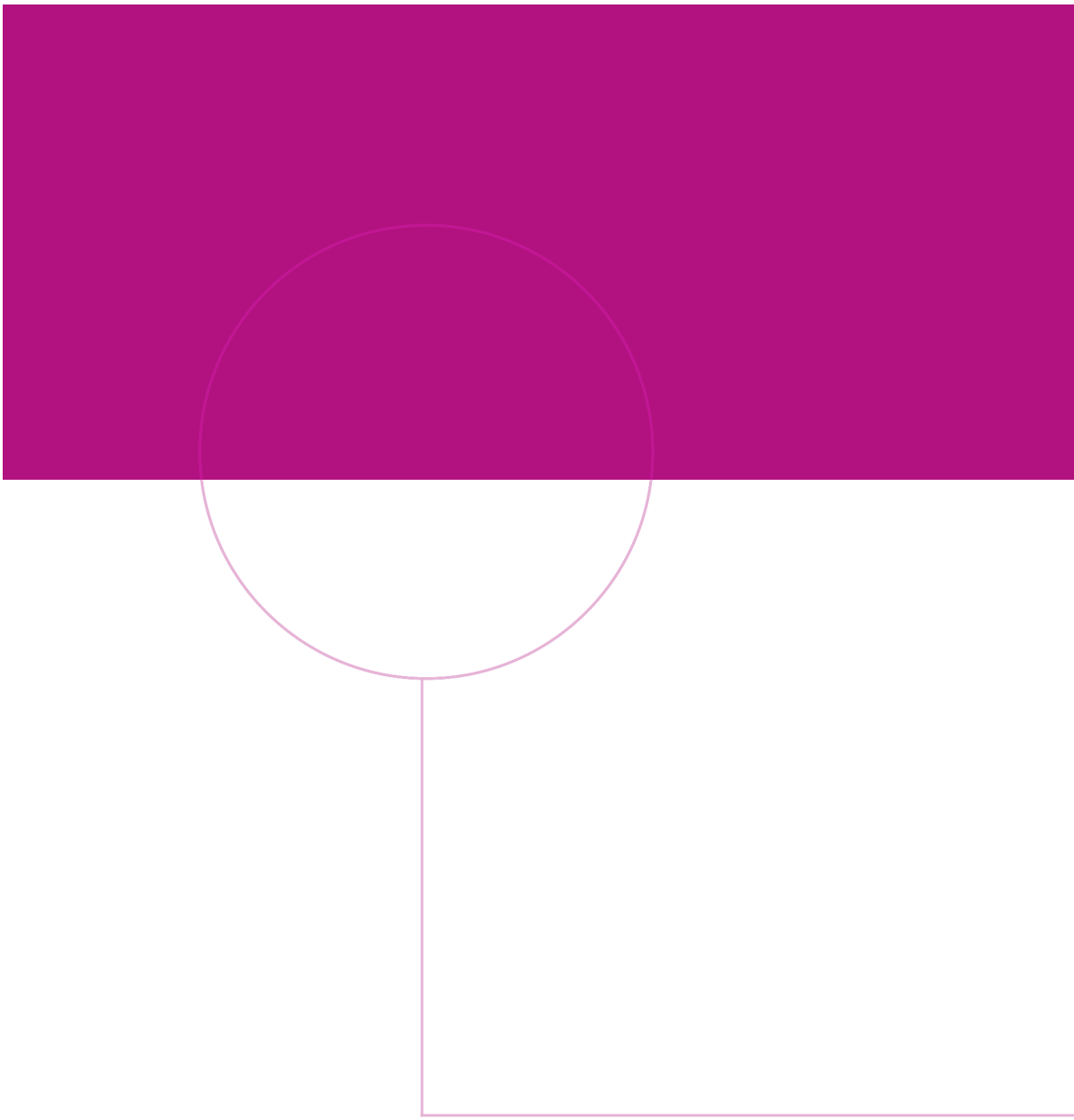


Figure C.4: Results on the validation dataset from the fine-tuning of mBERT.



NTNU

Norwegian University of
Science and Technology