Simen Tvete Aabol & Marcus Klomsten Dragsten

Active Learning in Norwegian Natural Language Processing

Master's thesis in Informatics, Artificial Intelligence
Supervisor: Ole Christian Eidheim
June 2023

Master's thesis

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

NTNU
Norwegian University of
Science and Technology

Simen Tvete Aabol & Marcus Klomsten Dragsten

Active Learning in Norwegian Natural Language Processing

Master's thesis in Informatics, Artificial Intelligence
Supervisor: Ole Christian Eidheim
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

This thesis examines the use of active learning in Norwegian natural language processing models to address the challenge of expensive data labeling. With the growing demand for larger datasets in artificial intelligence, fine-tuning language models has become crucial for various applications. However, creating labeled data for fine-tuning can be time-consuming and costly. Active learning offers a solution by iteratively selecting the most valuable data points for human labeling, thereby reducing the amount of labeled data required. This paper focuses on comparing different fine-tuning methods for Norwegian natural language processing models in terms of accuracy and training time while incorporating active learning. Three well-known active learning sampling methods are evaluated and compared to traditional random sampling to determine their effectiveness in selecting informative samples for labeling for a Norwegian natural language processing model. The model will be evaluated while doing a sentiment classification task. The research questions address the differences in accuracy metrics, how many samples one can save using active learning, the performance of active learning methods, and their impact on training time and training loss. The findings demonstrate the benefits of active learning in the context of natural language processing and provide insights for researchers, businesses, and organizations seeking to train machine learning models more efficiently and cost-effectively. The paper concludes with a discussion of the results, answering the research questions, and suggesting further research avenues.

Sammendrag

Denne avhandlingen undersøker bruken av aktiv læring i norske naturlig språkbehandlingsmodeller for å adressere utfordringen med kostbar merking av data. Med økende etterspørsel etter større datasett innen kunstig intelligens, har finjustering av språkmodeller blitt avgjørende for ulike applikasjoner. Imidlertid kan generering av merket data for finjustering være tidkrevende og kostbart. Aktiv læring tilbyr en løsning ved å iterativt velge de mest verdifulle datapunktene for menneskelig merking, og reduserer dermed mengden merket data som kreves. Denne studien fokuserer på sammenligning av ulike finjusteringsmetoder for norske naturlig språkbehandlingsmodeller med hensyn til nøyaktighet og treningstid, samtidig som aktiv læring inkorporeres. Tre velkjente aktiv læring-utvalgsmetoder evalueres og sammenlignes med tradisjonelt tilfeldig utvalg for å fastslå deres effektivitet i valg av informative datapunkter for merking i en norsk naturlig språkbehandlingsmodell. Språkmodellen vil evalueres ved en sentiment klassifiseringsoppgave. Forskningsspørsmålene tar for seg forskjellene i nøyaktighetsmålinger, hvor mange datapunkter man slipper å merke, ytelsen til aktiv læring-metodene og deres innvirkning på treningstid og treningstap. Resultatene viser fordelene ved aktiv læring i norske naturlig språkbehandlingsmodeller og gir innsikt for forskere, bedrifter og organisasjoner som ønsker å trene maskinlæringsmodeller mer effektivt og kostnadseffektivt. Avhandlingen avsluttes med en diskusjon av resultatene, besvarer forskningsspørsmålene og foreslår videre forskningsmuligheter.

Preface

Machine learning is a rapidly evolving field with significant implications for various domains. In pursuit of exploring the intricacies of this fascinating field through our master thesis, we sought collaboration with Kantega AS, a renowned corporation known for its expertise in software development and machine learning.

Fortunately, Kantega AS was open to our proposal and expressed their willingness to collaborate on our master thesis, together with our advisor from NTNU, Ole Christian Eidheim. We were fortunate to have Lars Nuvin from Kantega join us in this process, whose guidance and support were instrumental in shaping our thesis. Recognizing the significance of an appropriate research topic, Lars called upon Magnus Oplenskedal, a knowledgeable expert within the organization, to provide valuable insights.

Magnus, drawing upon his previous research and expertise, suggested that we delve into the concept of active learning. This recommendation resonated perfectly with our ambitions and motivations, as we were eager to explore the realm of machine learning. Furthermore, the topic enabled us to engage methodically with new datasets and delve into the exciting domain of natural language processing, which has garnered considerable attention in recent times.

The research process started with relative ease, as the data collection and processing procedures were quickly implemented. However, the implementation of the active learning framework and the fine-tuning methods presented unique challenges that required thorough attention and problem-solving. Through determination and collaboration, we overcame these obstacles, contributing to a comprehensive exploration of the subject at hand.

As we near the end of our thesis, we take pride in the knowledge that our research efforts have yielded promising results. The journey has been both mentally engaging and rewarding, allowing us to expand our knowledge, skills, and understanding of the vast landscape of machine learning. We are excited and honored to present this master thesis, hopeful that our contributions will contribute to the ever-growing body of knowledge in the field of machine learning.

Acknowledgments

We would like to take this opportunity to express our deepest gratitude to the individuals who have contributed to the successful completion of our master thesis. Your guidance, support, and insights have been invaluable throughout this journey.

First and foremost, we extend our heartfelt appreciation to our advisors, Ole Christian Eidheim from NTNU and Lars Nuvin from Kantega AS. Your unwavering commitment, expertise, and encouragement have been instrumental in shaping our project and achieving outstanding results. We are grateful for your guidance, patience, and willingness to share your knowledge and experience with us.

We would also like to acknowledge Magnus Oplenskedal from Kantega, whose passion for active learning inspired us to explore this fascinating area. Your enthusiasm and ideas sparked our interest and made the process of researching and writing about active learning both enjoyable and rewarding.

We are grateful to Ole Christian, Lars, and Magnus for their active involvement in our project. Despite their busy schedules, they gladly joined us for weekly meetings on Microsoft Teams, providing valuable feedback, insightful suggestions, and constructive criticism.

Additionally, we express our gratitude to all those who have supported us along the way. To our friends and family, thank you for your unwavering belief in us and for providing a constant source of encouragement and motivation.

To everyone who has played a role in this adventure, directly or indirectly, we sincerely thank you for your contributions. Without your support, this thesis would not have been possible.

SIMEN TVETE AABOL
MARCUS KLOMSTEN DRAGSTEN

Contents

Figures

Tables

Acronyms

AL active learning. 2, 3, 5–7, 10, 15, 17–19, 26, 28–30, 33, 40–42, 45, 46, 49–53, 55–57, 59, 62

API application programming interface. 22, 25, 26, 29, 30, 43, 62

BADGE Batch Active Learning by Diverse Gradient Embeddings. 17

BERT Bidirectional Encoder Representations from Transformers. 11, 14, 15, 17–19, 49, 59, 60, 62

CPU central processing unit. 13

ELMo Embeddings from Language Model. 18

EOSC-Nordic European Open Science Cloud. 18

FP32 single-precision floating-point. 22

GLUE General Language Understanding Evaluation. 11

GPT generative pre-trained transformers. 1, 13, 15, 16

GPU graphics processing unit. 13, 14, 21, 22, 31, 61

LTG Language Technology Group. 18

ML machine learning. 1, 2, 10, 11, 17, 59, 60

NLP natural language processing. 2, 3, 5, 10, 11, 14, 18, 19, 22, 23, 25, 54, 55, 59

NoReC Norwegian Review Corpus. 18, 22–24, 59

NTNU Norwegian University of Science and Technology. 21

RNN recurrent neural networks. 10

SANT Sentiment Analysis for Norwegian Text. 18

Sklearn Scikit-learn. 10, 29, 30, 33, 62

TF32 TensorFloat-32. 21, 22, 31

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, the field of artificial intelligence has been moving towards the creation of deeper and deeper neural networks. This trend has led to an increasing demand for larger amounts of data. The growth of data volumes in machine learning (ML) is accelerating at a staggering pace. For instance, generative pre-trained transformers (GPT)-3.5 (released on the 12th of March 2022) has 175 billion parameters and was trained on around 570GB of datasets, including web pages, books, and other sources [1]. The next iteration, GPT-4 (released on the 14th of March 2023), is reportedly six times bigger and has one trillion parameters [2]. These are massive amounts of data that aid in the development of excellent generative language models. When such models become available to the general public, it may be necessary for companies to capitalize on their potential in order not to lag behind competitors who adopt the technology. One method is to fine-tune the models for various purposes rather than training them from scratch.

Fine-tuning language models is a process that requires less data than general model training. However, it necessitates data on the relevant task to train the model on. This data can often be unavailable. As a result, one must label this oneself or hire someone to do it for them. This can quickly become time-consuming and expensive. For instance, a stock analyst might want to fine-tune a model to analyze all financial news found on the internet in order to automatically buy or sell if the model discovers something interesting. In this case, one would have to train a model with financial news/articles and a label for whether one should invest in or sell out of this stock/company. Normally, this procedure would have required one to manually go through all financial news and create a label for it. One may have to label tens of thousands of pieces of financial news in order to find a connection to the development of the share price. The work with this classification necessitates a domain expert in finance, which usually comes with a very high salary and can be very costly for the project.

Fortunately, the issue of labeling data can be dealt with by implementing an active learning (AL) approach. AL is a popular method for reducing the amount of labeled data required for training a ML model. When compared to the traditional sampling approach, AL sampling algorithms can achieve high accuracy with less training data by iteratively selecting the most valuable data points for humans to label. Businesses can use this method to allow their employees to focus on labeling only the most informative data points for the ML model. The person labeling data will save a lot of time by not having to manually label every data point in a large dataset by using this method. This process not only saves time but also helps the model improve at an earlier stage of the training process. As a result, using AL results in a more efficient and cost-effective data labeling process.

The motivation for this paper is to demonstrate that AL can be used to address the issue of expensive data labeling. Additionally, the paper aims to make language models available in a problem where labeled data is scarce for traditional fine-tuning of language models, such as sentiment classification. The findings of this paper can help to advance the research of AL in the context of ML. It will also assist researchers, businesses, and organizations seeking to train ML models more efficiently and cost-effectively. It is hoped that by demonstrating the benefits of AL in natural language processing (NLP), further research in the field will be inspired.

1.2 Scope and Research Questions

The scope of this paper is narrowed down to the use of AL in Norwegian NLP models. Specifically, the paper will compare different ways of fine-tuning these NLP models, in terms of accuracy metrics and training time, while at the same time using AL. Additionally, three different methods of AL sampling methods will be compared to the traditional random sampling method, in terms of effectiveness. The AL sampling methods used are three out of four well-known methods in the field of AL [3]. The paper will examine how this approach can be used to select the most valuable samples for labeling in Norwegian NLP models, in a sentiment classification problem. The scope of the paper leads to these research questions:

RQ1 *What are the differences in the accuracy metrics of Norwegian natural language processing models for classification problems using different fine-tuning methods while incorporating active learning?*

RQ2 *Which of the well-known active learning methods score highest in accuracy metrics when selecting informative samples for labeling in Norwegian natural language processing models, compared to the traditional random sampling method? Additionally, how many samples can be potentially saved by employing active learning?*

RQ3 *How do different active learning methods, combined with different fine-tuning methods, affect the performance, in terms of training time and training loss, when training Norwegian natural language processing models, compared to traditional data sampling methods?*

1.3 Report Outline

This report is organized into 8 chapters. After the introduction, Chapter 2 presents the theoretical background for the reader to understand the concepts and methods used in the paper. In Chapter 3, the related work is presented to describe the state-of-the-art in the field of AL and NLP. The methods used in this paper will be described in Chapter 4. This chapter will describe the process for the project, in addition to allowing the reader to reproduce the results achieved. Chapter 5 describes the results that complement Chapter 4. The discussion of these results can be found in Chapter 6. Chapters 7 and 8 will conclude the paper by answering the research questions and describing further work for the master thesis.

Chapter 2

Theory

This chapter will contain a theoretical background that may be useful to understand the research group's methods and procedures.

2.1 Active Learning

When working with training natural language processing (NLP) models with supervised learning, active learning (AL) is a technique that may be used to speed up the process and possibly improve the model's accuracy [4] [5]. In the absence of employing active learning strategies, the process of sampling data can resemble the depiction illustrated in Figure 2.1, wherein a human selects data in a random manner.

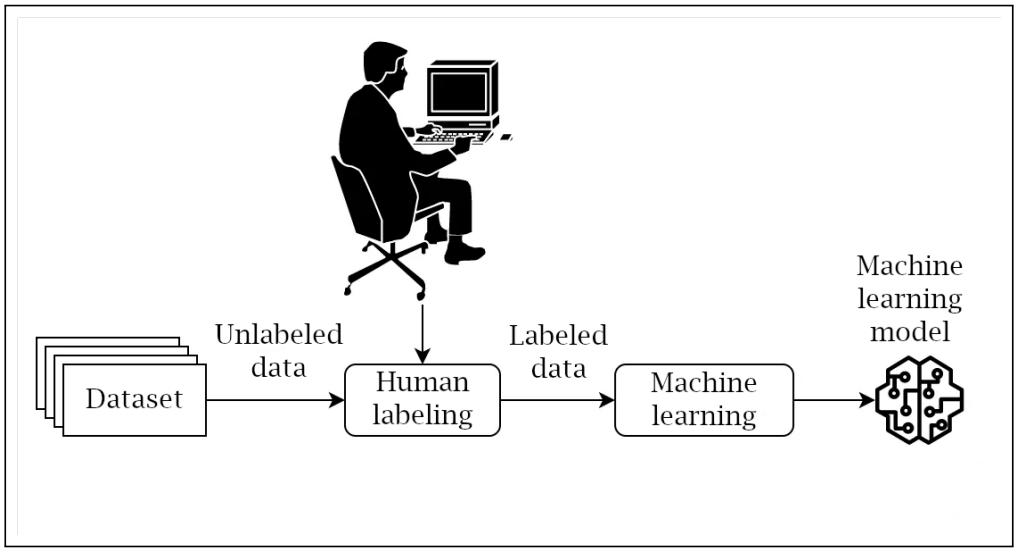


Figure 2.1: Illustration of traditional sampling process [6].

It is customary in classic supervised learning to label all the data before one begins training on it. Because there is often such an enormous amount of data, labeling can often take a long time and be unsustainable. This is due to the fact that labeling has typically been done manually by people, and in some cases requires experts within the domain [5] [4] [7].

One can avoid having to label the entire dataset by using AL. AL's core element is to only label the data that a model will learn the most from. In other words, the idea is that one should label the entities that are difficult to classify, and not the more trivial ones [4] [8].

The four primary possibilities for AL, according to Burr and Settles [4], are as follows: (1) *Membership query synthesis*, where the learner creates new artificial instances to be labeled. (2) *Pool-based*, where the learner has access to the closed set of unlabeled instances, known as the pool. (3) *Stream-based*, where the learner receives one instance at a time and has the option of keeping or discarding it. (4) *Batch-mode* AL is used when the pool-based scenario runs on a group of instances rather than a single instance [9]. In a text classification environment, the dataset is usually a closed set. The batch-wise procedure reduces training operations, causing waiting periods for the user. Therefore, a pool-based batch-mode is the most suitable for this work [9].

In AL, the data is labeled iteratively during training, based on how uncertain a classification model is in classifying the data that is in the training dataset [5] [4] [8]. The data from the training dataset that the model is least certain about is taken out, labelled, and placed in a training pool. The procedure can then be repeated when the model has been trained once again using the newly sampled data [4]. This process is illustrated in Figure 2.2.

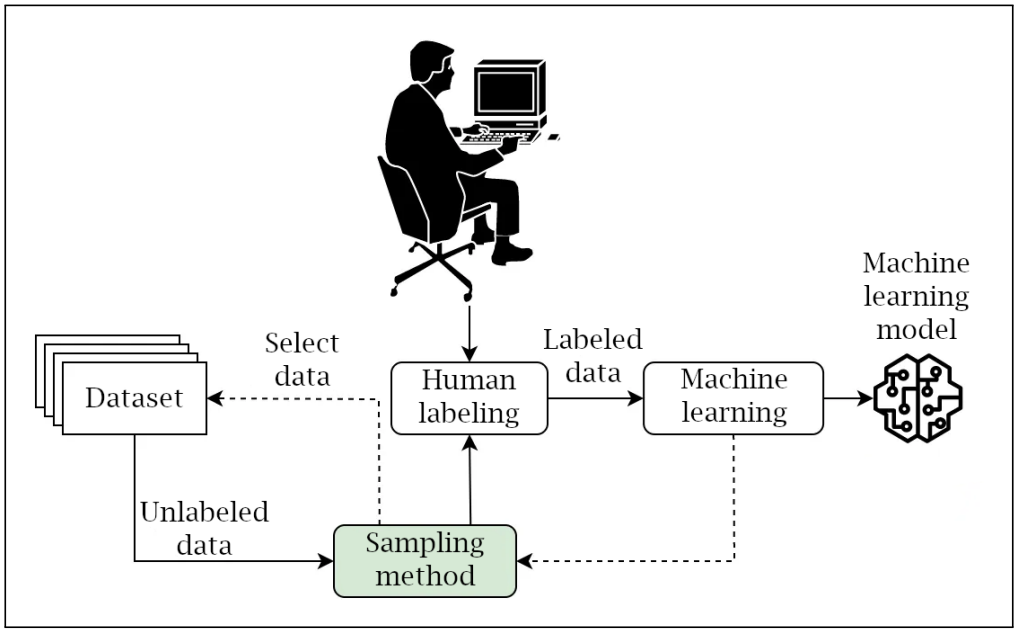


Figure 2.2: Illustration of active learning sampling process [6]

The concept is that by identifying the data that a model is most unsure of; the model should be able to categorize the easy entities without having to label as many of them. Numerous approaches exist for calculating uncertainty, some of which will be discussed later. The most common ones are *Least Confident*, *Smallest Margin*, *Largest Margin*, and *Entropy Reduction* [4]. There are also other ways of measuring the uncertainty, which will not be discussed here, but which the interested reader can familiarize themselves with [5].

A practical example will be conducted using three distinct methods of assessing uncertainty in order to provide a brief and straightforward explanation of how AL works. The data in Table 2.1 is an arbitrary output from a classification model, which classifies a text excerpt into the categories "Sports", "Politics", "Economy", "Entertainment", and "Technology". The further description of the methods for calculating uncertainty is based on this output. Be aware that this data has been normalized. The language models that will be covered later in this paper produce *logits* as output. In the context of neural networks, logits typically refers to the vector of raw (non-normalized) predictions that a classification model generates, which is ordinarily then passed to a normalization function. If the model is solving a multi-class classification problem, logits typically become an input to the softmax function. The softmax function then generates a vector of (normalized) probabilities with one value for each possible class.

Table 2.1: Arbitrarily output data from a classification model.

	Sport	Politics	Economy	Entertainment	Technology
Text 1	0.23	0.09	0.12	0.56	0.12
Text 2	0.19	0.17	0.19	0.21	0.24
Text 3	0.71	0.01	0.05	0.20	0.03
Text 4	0.20	0.20	0.20	0.20	0.20
Text 5	0.01	0.48	0.49	0.01	0.01

2.1.1 Least Confidence

The *Least Confidence* method takes all the maximum values of each output and selects the one with the lowest maximum value of these [5]. In this case, this gives the maximum values: Text 1: 0.56, Text 2: 0.24, Text 3: 0.71, Text 4: 0.2, Text 5: 0.49.

This means that the method selects Text 4, with the value 0.2. The method therefore considers Text 4 to be the most uncertain. It will therefore be this text that will be marked as part of the AL process. If one more entity was to be included, this would be Text 2.

The most informative instance, or the best query, according to some query selection algorithm A is denoted from this point on by the notation:

$$x^*_{\mathrm{A}} \tag{2.1}$$

The mathematical formula for Least Confidence is:

$$x^*_{\mathrm{LeastConfidence}} = \arg\min_x \ P(y^*|x) \tag{2.2}$$

where

$$y^* = \arg\max_y \ P(y|x) \tag{2.3}$$

In the accompanying Table 2.2, the text with the highest level of uncertainty is highlighted in red, while the text with the second-highest level of uncertainty is highlighted in orange.

Table 2.2: Example for Least Confidence method. The algorithm's most uncertain entity is indicated by the color red, while the next is shown by orange.

	Sport	Politics	Economy	Entertainment	Technology
Text 1	0.23	0.09	0.12	0.56	0.12
Text 2	0.19	0.17	0.19	0.21	0.24
Text 3	0.71	0.01	0.05	0.20	0.03
Text 4	0.20	0.20	0.20	0.20	0.20
Text 5	0.01	0.48	0.49	0.01	0.01

2.1.2 Smallest Margin

The Smallest Margin method is based on all maximum values and the second highest for each output. The difference between these two values is calculated on each row. The row with the lowest difference is the output one is most uncertain about in this method [5]. This gives: Text 1: 0.33 (0.56-0.23), Text 2: 0.03 (0.24-0.21), Text 3: 0.51 (0.71-0.20), Text 4: 0 (0.20-0.20), Text 5: 0.01 (0.49-0.48).

In this case, it is Text 4 that has the greatest uncertainty, and therefore the one that will be labelled manually. If one more row were to be included, this would have been Text 5.

The mathematical formula for Smallest Margin is:

$$x^*_{\mathrm{SmallestMargin}} = \arg\min_x \ (P(y^*_{\mathrm{max}}|x) - P(y^*_{\mathrm{max}-1}|x)) \tag{2.4}$$

where

$$y^* = \arg\max_y \ P(y|x) \tag{2.5}$$

In the accompanying Table 2.3, the text with the highest level of uncertainty is highlighted in red, while the text with the second-highest level of uncertainty is highlighted in orange.

Table 2.3: Example for Smallest Margin method. The algorithm's most uncertain entity is indicated by the color red, while the next is shown by orange.

	Sport	Politics	Economy	Entertainment	Technology
Text 1	0.23	0.09	0.12	0.56	0.12
Text 2	0.19	0.17	0.19	0.21	0.24
Text 3	0.71	0.01	0.05	0.20	0.03
Text 4	0.20	0.20	0.20	0.20	0.20
Text 5	0.01	0.48	0.49	0.01	0.01

2.1.3 Shannon Entropy

When entropy is applied to a probability distribution, each likelihood is multiplied by its own log, and the negative total is calculated [4] [10]. In theory, entropy is a measure of the amount of uncertainty or randomness in the data. The methodology for calculating Shannon Entropy differs slightly from the first two that have been described. Shannon Entropy generates a value by using the entire row. Higher levels signify greater uncertainty. Therefore, a model will be most unsure about and need to choose the greatest values.

The mathematical formula for Shannon Entropy is:

$$x^*_{\text{Entropy}} = \arg\max_x \left(-\sum_i P(y_i^*|x) \log P(y_i^*|x) \right) \tag{2.6}$$

where

$$y^* = \arg\max_y \ P(y|x) \tag{2.7}$$

Here it is worth noting that the base number of the logarithm is 2, as described by Shannons [10]. The computation for "Text 1" is shown below. The other entities adhere to the same process.

(2.8)

$$\begin{aligned}(Text1) = \\ &(-0.23 * \log_2(0.23)) + (-0.09 * \log_2(0.09)) + \\ &(-0.12 * \log_2(0.12)) + (-0.56 * \log_2(0.56)) + \\ &(-0.12 * \log_2(0.12))\end{aligned}$$

(2.9)

$$\begin{aligned}(Text1) = \\ &(-0.23 * (-2.16)) + (-0.09 * (-3.17)) + \\ &(-0.12 * (-2.73)) + (-0.56 * (-0.85)) + \\ &(-0.12 * (-2.73)) = 1.91\end{aligned}$$

In the accompanying Table 2.4, the text with the highest level of uncertainty is highlighted in red, while the text with the second-highest level of uncertainty is highlighted in orange. The values shown in the extra column "Entropy" in the table 2.4 are the outcome of calculating for each row.

Table 2.4: Example for Shannon Entropy method. The algorithm's most uncertain entity is indicated by the color red, while the next is shown by orange.

	Sport	Politics	Economy	Entertainment	Technology	Entropy
Text 1	0.23	0.09	0.12	0.56	0.12	1.91
Text 2	0.19	0.17	0.19	0.21	0.24	2.31
Text 3	0.71	0.01	0.05	0.20	0.03	1.25
Text 4	0.20	0.20	0.20	0.20	0.20	2.32
Text 5	0.01	0.48	0.49	0.01	0.01	1.21

2.1.4 Active Learning Library

modAL is an AL library for Python that is built on top of Scikit-learn (Sklearn) [11]. It allows you to rapidly create AL workflows. The techniques covered in the chapter on AL in Chapter 2.1 are available in this library. Because of this, using these methods is simple and requires minimal effort.

2.2 Transformers

The *transformer* is a type of neural network architecture that was introduced in 2017 by Vaswani et al. in the paper "Attention is All You Need". Since then, it has gained popularity as a method for machine learning (ML) applications such as NLP and others. Self-attention is a important aspect in a transformer architecture. The key idea behind self-attention is that the network can learn to attend to different parts of the input sequence simultaneously when making predictions or generating outputs. This is in contrast to traditional recurrent neural networks (RNN), which process the input sequence one element at a time in a fixed order [12].

The transformer architecture consists of an encoder and a decoder. The encoder processes the input sequence and generates a sequence of hidden states, while the decoder uses these hidden states to generate the output sequence. The key innovation in the transformer architecture is the use of multi-head self-attention, which allows the network to attend to different parts of the input sequence simultaneously [12].

Self-attention is a mechanism that allows the network to weigh different parts of the input sequence when generating each output element. The weight assigned to each input element is learned during training, based on the context of the input sequence and the current state of the network. The mechanism can be described as a function that takes as input a sequence of vectors and returns a sequence of vectors of the same length [12]. The self-attention mechanism is used multiple

times in parallel, each time with different learned parameters. This is called multi-head self-attention, and it allows the network to attend to different parts of the input sequence with different "heads" of attention. The output of each head is concatenated and passed through a linear layer to generate the final output [12].

2.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a powerful pre-trained language model developed by Google in 2018. BERT's capacity to comprehend the context and meaning of words in a phrase, which results in superior performance on a range of NLP tasks, is widely recognized as a key NLP milestone [13].

BERT uses a transformer architecture, as described in Chapter 2.2. It is created to predict the following word in a sentence given the context of the preceding words and is trained on a huge corpus of text, such as the whole Wikipedia corpus [13].

The capability of BERT to manage bidirectional context is one of its important characteristics. Traditional language models process text in a unidirectional manner, meaning they can only look at the words that come before the target word. However, BERT can process text in both directions, allowing it to consider the entire context of a sentence when predicting the next word [13].

BERT has produced cutting-edge outcomes for a variety of NLP tasks, including sentence categorization, question answering, and language inference. It has even outperformed human performance on some benchmarks, such as the General Language Understanding Evaluation (GLUE) benchmark for natural language understanding [14] [15].

In conclusion, the pre-trained BERT model stands as an exemplary language model, making significant strides in the field of NLP. As a result of its ability to understand the context and meaning of words in a phrase, it has shown better performance on a range of NLP tests. BERT has thus gained popularity among both NLP scholars and practitioners [13] [14] [15].

2.4 Evaluation Metrics

There are a number of metrics that can be used to evaluate natural language models within ML and data analysis. These metrics are widely used to evaluate how well a model does at correctly classifying or forecasting events and to compare the performance of multiple models. This chapter will examine six commonly used metrics: *accuracy score*, *balanced accuracy score*, *precision score*, *recall score*, *F1 score*, and *training loss*.

2.4.1 Accuracy Score

The accuracy score measures the proportion of accurate predictions made by a model over the total number of forecasts. This metric may not always be the optimal one despite being well-liked and straightforward. When assessing scores on a skewed dataset, it is important to consider the potential for misleading outcomes. Skewed datasets are characterized by an unequal distribution of values, where a substantial proportion of the data is concentrated towards one end of the range. This can result in distorted interpretations of the data and may lead to inaccurate conclusions. This is because a model can predict the class that is overrepresented and thus achieve good accuracy [16].

2.4.2 Recall Score

The recall score measures the proportion of true positives among all the instances that are truly positive. This metric is useful in cases where false negatives are costly, such as identifying fraudulent transactions [17].

2.4.3 Precision Score

The precision score measures the proportion of true positives among all the instances that the model classified as positive. This metric is useful in cases where false positives are costly, such as in medical diagnosis [18].

2.4.4 F1 Score

The F1 score is a weighted average of the precision- and recall score, with equal weight given to both. This metric provides a balance between precision- and recall score, and is useful in cases where both false positives and false negatives are important [19].

2.4.5 Balanced Accuracy Score

The balanced accuracy score is a modification of the accuracy score that takes into account the imbalance between the classes. To guarantee that the score is not skewed toward the majority class, it is computed as the average of the recall scores for each class. This metric is particularly useful in cases where classes are imbalanced [20].

2.4.6 Training Loss

Training loss is a metric used during the training phase of the model to measure the error between the predicted values and the actual values. It is used to optimize

the model parameters to minimize the loss [21].

2.5 Hardware to Choose

Language models are taught to anticipate the following word or series of words in a given text. To create a proficient language model, it is often necessary to utilize extensive computational power. Such substantial capacity may either stem from a central processing unit (CPU) or a graphics processing unit (GPU). The advantages and disadvantages of training language models on CPU and GPU will be discussed in this chapter.

One advantage of using a CPU to train language models is the availability of low-cost hardware. Since desktop and laptop computers have CPUs, many academics and business people view them as a useful and affordable option. Furthermore, since it doesn't call for the usage of specialist software or hardware, training language models on CPUs is frequently less difficult than on GPUs.

The biggest drawback of employing CPUs is their comparatively poor processing speed. This can drastically lengthen the training process for language models, making them less practical for models with bigger datasets or more intricate architectures. Additionally, the size of the models that may be trained could be constrained by CPUs' relatively low memory capacities. Contrarily, because GPUs process information far more quickly than CPUs do, researchers can train larger and more complicated language models in a shorter period of time. GPUs also have larger memory capacity than CPUs, allowing for larger models to be trained. These advantages make GPUs a popular choice for training state-of-the-art language models, such as generative pre-trained transformers (GPT)-4 [22].

The increased cost of GPUs compared to CPUs is one key disadvantage. Since GPUs may be significantly more expensive than CPUs, academics and professionals on a limited budget may find it difficult to use them. Furthermore, GPUs could in some situations need specific software and hardware, which might complicate their use and maintenance.

The unique requirements of the project, the size of the dataset, including the complexity of the model architecture, and the availability or lack of funding, affect whether to use a CPU or a GPU for training language modeling . While CPUs offer low-cost and convenient hardware, they may not be suitable for large or complex models. Although GPUs are more complex and costly, they also offer quicker processing rates and frequently have more memory. It is important to remember that it requires far less computing power to fine-tune a data model compared to training one from scratch.

It is also possible to use several GPUs when training language models. Here it has been noted that it is not always better to have more in all cases. When one uses several GPUs at the same time, this brings a lot of overhead and administration.

Since the work needs to be distributed, moving from a single GPU to numerous ones requires some kind of parallelism. To produce parallelism, there are a number

of methods, including pipeline, data, and tensor parallelism. The ideal settings depend on the hardware one is using because there isn't a single answer that works for everyone [23]. In those cases where one works with extremely large datasets and or with extremely large models, it may not be possible to run everything on a single GPU. In such situations, it is necessary to use several GPUs.

2.6 Hugging Face

Hugging Face is an open-source platform for NLP that provides a number of tools and resources for developing and utilizing NLP models. The platform was introduced in 2016 and has since gained popularity among NLP academics and programmers [24].

Hugging Face's notable feature is its emphasis on open-source development and community-driven innovation. The platform is based on the frameworks TensorFlow and PyTorch, which are well-known and widely used within the NLP community. All source code is also on GitHub, so everyone can review and contribute. This approach has created an active ecosystem of developers and researchers who share models, datasets, code, and best practices [25]. Hugging Face has become an essential player in the NLP landscape, providing a flexible and robust platform for constructing and deploying NLP models. Hugging Face also provides a variety of tools for learning about NLP and staying up to date with current industry advancements, including blog entries, tutorials, and a community forum for knowledge exchange and question-and-answer sessions.

2.7 Fine-Tuning Pretrained Models

Fine-tuning a BERT model is a common technique to improve the performance of pre-trained language models on specific downstream tasks. This process involves training a pre-trained BERT model on a smaller dataset that is specific to the task of interest, using supervised learning.

There are many pretrained BERT models one can find, e.g. on Hugging Face, so it might be good to do some research on which one best suits the task. BERT models are typically pre-trained on large, general-domain corpora and are available in various sizes and configurations. The pre-trained model that is chosen depends on the particular job at hand as well as the size and complexity of the dataset.

Once one have found a suitable model, one is ready to adapt it to the task it will perform. This is done by replacing the final layer(s) of the BERT model with task-specific layers, which are then trained on a labeled dataset using supervised learning. The labeled dataset should be representative of the task and should contain sufficient examples to train the model effectively [13].

It is important to consider a variety of factors, one of these is the choice of the hyperparameters, which might affect how well the model performs. The learning

rate, batch size, and training epoch count are some of these hyperparameters. To obtain best performance, it is crucial to choose suitable values for these hyperparameters.

Evaluation of the trained model should also be taken into account. A validation set distinct from the training set should be used to assess the performance of the model. Through subsequent iterations of the fine-tuning process, this evaluation might assist in detecting possible overfitting or underfitting concerns.

When utilizing AL to fine-tune models, one is faced with a choice, reusing the same model, or initiating a new one each time. The most common method is to initiate new models in each training cycle. The two approaches are described below.

2.7.1 Iterative Use of Models

Fine-tuning the same BERT model through the entire training process involves reusing the same model for each iteration of AL. This approach can be beneficial if the labeled training set is large enough to prevent overfitting. However, reusing the same model can also lead to overfitting if the remaining set is too small.

2.7.2 Initialize New Models

Initializing a new model at each training session involves starting with the same fresh BERT model and fine-tuning it on the labeled training set. This could be necessary because fine-tuning a pre-trained BERT model on a new task can result in overfitting, where the model becomes too specialized to the training set and performs poorly on new data. Starting with a new model lowers the possibility of overfitting and improves the model's ability to generalize to new data.

To address overfitting in either approach; *regularization*, *dropout*, and *early stopping* can be employed. Regularization involves adding a penalty term to the loss function that encourages the model to learn simpler patterns. Early stopping involves monitoring the validation loss during training and stopping the training process when the validation loss starts to increase. Dropout randomly drops out neurons during training, preventing the model from relying too heavily on any one feature and forcing it to learn more robust representations [26].

2.8 Environmental Impact in Language Model Training

It is well-known that the use of computational resources has an environmental impact. The power and resources required to train GPT-4 are immense. However, it is important to recognize that sustainability and the green shift require contributions from all parties, not just the major players. The expression "Many small streams make one big river" resonates in this context. Although GPT-4 is relatively new, accurate estimates of its energy consumption during training are limited.

Nonetheless, a study titled "Making AI Less Thirsty," released on April 6th, 2023, investigated the energy and water consumption of ChatGPT-3. The study revealed that a single interaction with ChatGPT-3 is equivalent to wasting half a liter of water. Furthermore, in terms of energy consumption, the researchers reported that the volume of fresh water used to cool the server farms running ChatGPT-3 could fill a nuclear reactor's cooling tower. Training a software like ChatGPT-3 requires energy comparable to building batteries for 320 Tesla cars or 370 BMW cars [27].

Chapter 3

Related Work

3.1 Active Learning and Fine-Tuning Models

Active learning (AL) is a well-known concept in machine learning (ML) that has been described in papers as early as 1998 and 2002 [28] [29]. Despite this, AL has not been widely adopted as the norm in ML, especially deep models [30]. To further develop AL within the scope of Norwegian text analysis, it is interesting to look at related work in the field and understand what has been done and what can be done.

Schröder (2020) [30] discusses the concept of employing AL in deep neural networks. He addresses the issue of training a deep neural network with AL. As mentioned earlier, small datasets are not adequate for deep neural network training. On the other hand, Schröder highlights certain AL applications for deep neural networks that are appropriate. One of these tasks is the process of fine-tuning a trained model. This is highlighted because the amount of data required for this is far less than the actual training of the model. Another obstacle that is highlighted is how to make good query strategies on the output from a neural network. Among other things, the more conventional techniques that Settles introduced will not always work for a neural network since they lack an inherent indicator of uncertainty [4]. Schröder refers to other authors who have considered alternative solutions to this. Ash et al.'s article from 2019 presents Batch Active Learning by Diverse Gradient Embeddings (BADGE) [31]. This is a query strategy adapted to deep neural networks. The final layer's gradients are seeded with K-means++ [32]. By doing this, one obtains a query based on uncertainty and diversity.

3.2 BERT-Models and Active Learning

A very central and relevant study for the scope of the report at hand is the empirical study by Ein-Dor et al. [33]. The study looks at AL for Bidirectional Encoder Representations from Transformers (BERT) models. Ein-Dor et al. also looks at

how the use of AL can boost the performance of BERT, even under a challenging setting, with a small annotation budget and highly skewed data. This reflects many aspects of the thesis at hand, including AL and already trained BERT-models. Ein-Dor et al.'s report sets some guidelines for the use of AL in this project.

The study highlights the challenges presented by real-world scenarios for text classification, where labels are usually expensive and data is often characterized by class imbalance. AL is a widely adopted approach that addresses the challenge of data scarcity. It is a popular paradigm used to overcome the limitations posed by limited data availability. Recently, pre-trained NLP models such as BERT have received massive attention due to their outstanding performance in various NLP tasks. However, the use of AL with deep pre-trained models has so far received little consideration. This study provides valuable insights into the use of active learning techniques with BERT-based classification and demonstrates their potential to improve performance in practical scenarios

3.3 Norwegian Natural Language Processing

Touileb et al. [34] wrote a report about sentiment analysis for book reviews in Norwegian using the Norwegian Review Corpus (NoReC) [35], but only the subset containing book reviews. This report is relevant for the project at hand because of the dataset used, in addition to the use of sentiment analysis with binary sentiment classification. Touileb et al. also provides guidelines for dataset splitting of dice ratings, where ratings 1, 2, and 3 are negative while rating 6 is positive. To balance the dataset, Touileb et al. randomly sampled ratings of 5 as positive.

There are several trained deep neural networks that can be used for text analysis for the Norwegian language. The report "Large-Scale Contextualized Language Modeling for Norwegian" highlights these [36]. The models in this report are based on two different frameworks, BERT and Embeddings from Language Model (ELMo). Some of the models discussed in this report have been created specifically for Norwegian text, while others are multilingual. An interesting factor the report mentions is that a model can perform better with a clean and small dataset compared to a model that has been trained on a larger dataset with more noise [36].

Of the models compared, the NorBERT model comes out best in *fine-grained sentiment analysis* evaluated on a sentiment graph and a non-polar sentiment graph [37]. In the tests on sentence-level binary sentiment classification, the NorBERT model comes out second best, only beaten by the model NB-BERT. Here it is interesting to see that the multilingual model mBERT, developed by Google, produces noticeably worse results by a large margin. Keep in mind that Kutuzov's report used NorBERT in the comparison, and not NorBERT 2. It can therefore be speculated that NorBERT 2 would achieve similar, if not better, results than its predecessor [36].

A combined effort of the projects European Open Science Cloud (EOSC-Nordic) and Sentiment Analysis for Norwegian Text (SANT), managed by the Language

Technology Group (LTG) at the University of Oslo, resulted in a publication of NorBERT 2 on February 7th, 2022 [38]. The reports by Kutuzov et al. from 2021 were used as a basis for their effort to construct this model. There has been a significant increase in vocabulary from NorBERT to NorBERT 2. NorBERT features a custom 30,000 WordPiece vocabulary, while NorBERT 2 features a custom 50,000 WordPiece vocabulary. This means that NorBERT 2 has a 40% larger vocabulary than its predecessor [38]. NorBERT 2 was evaluated on several of the same benchmarks that Kutuzov et al. used in their report. This investigation demonstrates that NorBERT 2 produced the best results for binary sentiment analysis. The outcomes from all the models have been included in Table 3.1. Here it can be observed that NorBERT 2 outperforms NB-BERT by a small margin.

Table 3.1: Binary sentiment analysis task comparison.

Model	**F1-score**
mBERT	67.7
XLM-R	71.8
NorBERT	77.1
NorBERT 2	**84.2**
NB-BERT-Base	83.9

3.4 Research Gap

Surveying the usage of AL within text analysis, with an emphasis on Norwegian text, has been the key research gap in this report. Since this is unexplored ground, this study will be valuable and serve as a resource for future work in the field. The content of this chapter illustrates that there is great potential within AL in NLP, with Norwegian NLP using AL in particular. This is because no one, to the best of current knowledge, has worked on this before with the Norwegian language. In addition, the combination of AL and NLP with the English language has produced promising results. One of those who has illustrated the possibilities for AL within NLP is Ein-Dor et al., who have worked on this on an English-trained BERT model [33]. These findings lay the foundation for this study, where the aim is to produce results for Norwegian NLP using AL on a binary sentiment classification problem.

Chapter 4

Method

4.1 Hardware and Tools

As mentioned in the Chapter 2, Theory, training language models can require a lot of machine resources. Therefore, it was established early on that the project group would receive available resources from Kantega AS and the Norwegian University of Science and Technology (NTNU). From Kantega AS, the group was provided with its own environment in Azure Databricks. This was flexible and the project group could configure the environment according to their own wishes, but with limitations on computing power and the availability of a graphics processing unit (GPU). NTNU offered access to their own cluster system, Idun. Idun uses a system called Slurm Workload Manager, where one can make requests for a job, and after a certain waiting time one can be granted this [39]. This is a large GPU cluster that offers extremely high computing power. The cluster consists, among other things, of the GPUs NVIDIA V100, NVIDIA P100, and NVIDIA A100 [40]. The downside of Idun was that this is a shared cluster with other students and researchers at NTNU. This could sometimes result in up to 50 hours of waiting time to run a Python file. Despite this, the project group chose to use Idun instead of Azure Databricks. The biggest driving force for this choice was the availability of costly resources. The project group also had access to Idun earlier in the startup phase and had everything set up on Idun. It would therefore have been a transaction cost for the group to switch over to Azure Databricks.

Since Idun could have up to 50 hours of waiting time, much of the testing was done on local machines, while the actual training was carried out on the Idun machines. Based on NVIDIA's own report, the project group chose to use the NVIDIA A100 GPU [41]. This one has a compute capability of 8.0, which enables the use of the TensorFloat-32 (TF32) file format [41] [42]. Tensor cores are specialized processing units that can perform mixed-precision matrix operations, which are commonly used in deep learning models. As described in a blog post by NVIDIA's developer website, *Accelerating AI Training with TF32 Tensor Cores*, TF32 is a mixed-precision data format that uses 10 bits for the exponent and 22 bits for the mantissa. This format allows for increased precision in the accumulation of

small gradients in the model, while also leveraging the Tensor Cores for faster computation. According to NVIDIA, the TF32 format can achieve up to 20% faster training times compared to the previous single-precision floating-point (FP32) format, with negligible impact on model accuracy. This improvement in training speed can be particularly beneficial in large-scale deep learning applications, where training times can often be a bottleneck. If one wants to reproduce and run the source code for this project and do not have access to at least a NVIDIA Ampere hardware with a compute capability of 8.0, one will need to change the format.

The project team had six A100 GPUs available, but only used one during the training. The waiting time to run code on Idun increased dramatically when requesting more resources, and this was simply not feasible within the time frame. Two of the GPUs had 40GB of memory and 48 cores, while the remaining four had 80GB of memory and 64 cores. The project team does not know which of these two was used, and it is possible that both have been used interchangeably. All training was run multiple times in the starting phase without significant differences in runtime and results. Training was run with 16GB of memory to reduce the wait time on Idun. During the project, different numbers of cores were tested, but it was found that it was feasible to use only one. It was faster with multiple nodes, but the wait time on Idun also became considerably longer. In addition, the project team considers it good practice towards other fellow students to not use more resources than necessary in order to make the resources available to more people. The project group's decision on the duration of each training run was influenced by this factor. Specifically, the number of epochs used in each iteration of training was affected. Further details on this topic can be found in Chapter 4.3.2.

4.2 Hugging Face Implementation

When working with natural language processing (NLP), many useful resources can be found on Hugging Face, as previously mentioned. In particular, Hugging Face helped the project group to store their altered datasets, as well as their created models. This sped up the experimental process, since both datasets and models could easily can be imported using their application programming interface (API). In this chapter, the project group's usage of datasets and models from Hugging Face will be described.

4.2.1 Datasets

There is a significant lack of good Norwegian datasets after researching both Hugging Face and other various sources. If the keyword "Norwegian" is searched on Hugging Face for datasets, only 12 results are displayed. If the features; "Norwegian", "Norwegian Bokmål, and "Norwegian Nynorsk" are selected, only 10 results are displayed. After further research, many of these datasets are either empty, not Norwegian, or not useful for the projects purpose. From the related work in Chapter 3.3, the Norwegian Review Corpus (NoReC) [35] was deemed very useful

for the projects purpose, as it is a large Norwegian corpus with reviews that can be used for sentiment classification. There is a total of 43,000 reviews with dice ratings in the corpus. The reviews are gathered from various Norwegian news sources, and the reviews dates as far back as 1998. The reviews also originate from different genres of entertainment. These include: audiovisual media, music, miscellaneous items, literature, products, games, restaurants, stage, and sports. After skimming through the reviews in the dataset, the quality seemed fair, and little preprocessing were needed.

The search for datasets on Hugging Face were not in vain, as one dataset seemed useful for the purpose of the project. The "Norwegian_sentiment" dataset, created by user "sepidmnorozy" [43], contains 3,608 rows of reviews. These reviews have labels of 0 or 1, which indicates their sentiment. There is little information or documentation about this dataset, but it was also deemed as useful, since it could be concatenated with the NoReC dataset, to create more variation and a larger data foundation. In addition, both datasets has tags for the train/validation/test split, which indicates that the data is evenly distributed in terms of for example categories in the NoReC dataset. Therefore, for this project both the NoReC dataset and "Norwegian_sentiment" datasets were used for sentiment classification.

4.2.2 Preprocessing

The Norwegian Review Corpus

After determining what datasets to use, a small exploratory data analysis was performed by the project group on both of the datasets to find important features to keep or not. The NoReC dataset was clearly the largest, and contained some intricacies. Firstly, the rating disparity was clear. Figure 4.1 shows this. In particular there are few reviews for the ratings 1, 2, and 6, with 389, 2,346, and, 2,410 reviews respectively. This already suggests that a classification on dice ratings might be challenging with so small sample sizes. Furthermore, it was found that there are 460 reviews that is written in Norwegian Nynorsk. These reviews were kept, since the Norwegian NLP model chosen has also trained on this language. Lastly, it was found that around 5,000 of the reviews had empty excerpts. The text in the reviews are obviously important, so these reviews were deleted.

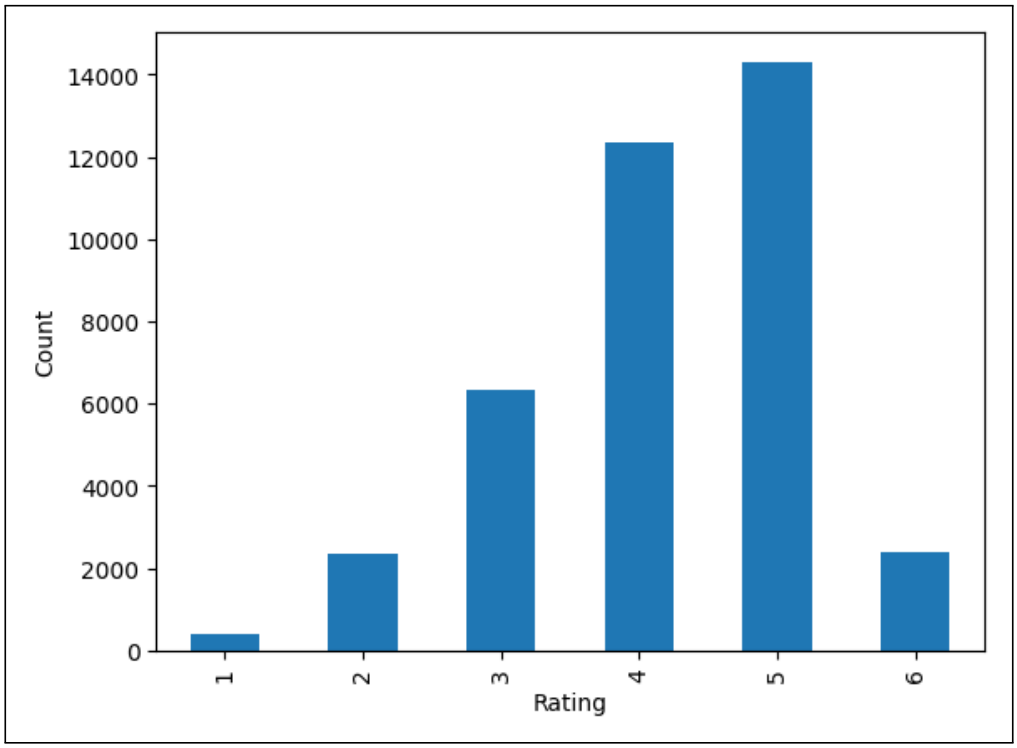


Figure 4.1: Rating disparity for the Norwegian Review Corpus dataset.

Norwegian Sentiment Dataset

This dataset [43] seemed to be cleaned up before uploaded to Hugging Face, as there were no empty excerpts. Additionally, the rating disparity was somewhat tighter than the NoReC dataset. The value counts for the labels are 1,186, and 2,422, for negative and positive sentiment respectively. Since the rating disparity of the NoReC data was significant, the "Norwegian_sentiment" dataset will act as a buffer of data. In particular, for the negative sentiment classification. As mentioned in Chapter 4.2.2 and Chapter 3.3, classification for dice ratings would be challenging with minimal amounts of data for some of the ratings. Therefore, the project group decided to concatenate and split the datasets, to simplify the classification. Figure 4.1 shows that ratings with value 4 is a clear separation between negative reviews and positive reviews. Therefore, ratings of 4 were deleted from the dataset, and the two sides were remapped to negative (1, 2, and 3) and positive (5 and 6). Lastly, the "Norwegian_sentiment" dataset was concatenated with the altered NoReC dataset to act as the dataset the model will train on.

4.2.3 Final Dataset Specifics

After preprocessing both the NoReC dataset and the "Norwegian_sentiment" dataset, and concatenating them, it is now possible to examine some specifics regarding how they were used in the training process. Firstly, the total length of the processed dataset is 29,383 rows. This is further divided into three separate datasets: train, validation, and test. The dataset sizes are 23,354 for the train set, 3,001 for

the validation set, and 3,028 for the test set. All three datasets are evenly split in terms of review categories, where each has reviews representing every category. The label disparity in terms of positive/negative sentiment is also evenly skewed for each dataset. There are approximately double the amount of positive reviews as negative in each dataset. This caused some intricacies in terms of the accuracy score, which is why balanced accuracy score is also represented in the results chapter. The dataset specifics are collected in Table 4.1:

Table 4.1: Distribution of positive and negative reviews for each dataset, in addition to the total size of each dataset.

	Total size	Nr. of positives	Nr. of negatives
Train dataset	23,354	14,958	8,396
Validation dataset	3,028	2,087	941
Test dataset	3,001	2,095	906

4.2.4 Models and Tokenization

As mentioned in Chapter 3.3, NorBERT 2 is the model of choice for this project. This model is available from Hugging Face [44], and can be used for fine-tuning and other purposes. In normal NLP pipelines, the model that is supposed to be fine-tuned, can also supply different tools for the text it will train on. These include the tokenizer and a data collator, in addition to the model itself. Therefore, in the code for the project, a *tokenize dataset* function is defined to tokenize the whole dataset in terms of the excerpt. This is shown in Figure 4.2.

```
1  from transformers import AutoTokenizer
2
3  def tokenize_dataset(dataset, model_checkpoint):
4      tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
5
6      def tokenize_function(examples):
7          return tokenizer(examples["text"], padding="max_length", truncation
               =True)
8
9      train_data = dataset["train"]
10     val_data = dataset["validation"]
11     test_data = dataset["test"]
12
13     train_data = train_data.map(tokenize_function, batched=True)
14     val_data = val_data.map(tokenize_function, batched=True)
15     test_data = test_data.map(tokenize_function, batched=True)
16
17     return train_data, val_data, test_data
```

Figure 4.2: The *tokenize_dataset* method for preparing the dataset to be trained on. It uses the Hugging Face *AutoTokenizer* with *padding='max_length'* and *truncation=True*

The tokenizer is imported from the Hugging Face application programming interface (API) with *AutoTokenizer*, where NorBERT 2 is input in the *from_pretrained()*

method. This tokenizer is used with *padding='max_length'* and *truncation=True*. The padding option pads short sentences to create equal sized tensors up to the models maximum token length. The truncation option shortens long sentences back to the max length which is the same as in the padding option. The code in Figure 4.2 illustrates how the review text in the whole dataset is tokenized.

4.3 Active Learning Implementation

The active learning (AL) training method used in this project has been composed by the project group as an attempt to combine a common AL implementation, with the Hugging Face trainer API. Figure 4.3 shows a semi-pseudo code version of the training loop, to help understand how the project group has trained the models in this project. Note that some of the methods used in this code snippet are altered for the sake of understanding the code. The full source code for the project is available in a repository on GitHub, for those interested in the whole unaltered training loop and other resources used in the project can find it **here** [45].

```python
from transformers import Trainer

def train_active_learning(n, model_checkpoint, active_learning,
    type_sampling, train_new_models):

    # Get model from Hugging Face
    tokenizer, data_collator, model = get_model_util(model_checkpoint)

    # Get custom dataset uploaded to Hugging Face
    train_set, validation_set, test_set = get_dataset()

    # Pick n amount of reviews to start training on
    train_set, new_reviews = pick_n_random_reviews(train_set, n)

    # Create training pool, and add the reviews to it
    train_pool = create_dataset_from_empty_df()
    train_pool = add_samples_to_pool(train_pool, new_reviews)

    # Train until the train set is empty
    while len(train_set) > n:

        # Initialize the trainer
        trainer = Trainer(model=model, args=training_args, train_dataset=
            train_pool, data_collator=data_collator, tokenizer=tokenizer,
            eval_dataset=validation_set, compute_metrics=compute_metrics)

        # Train on the data provided and evaluate on test set
        trainer.train()
        score = trainer.evaluate(test_set)

        if active_learning:
            # Get samples using chosen active learning method
            if type_sampling == "least_confidence":
                new_reviews = least_confidence_sampling(train_set, n,
                    trainer)
            elif type_sampling == "margin":
                new_reviews = margin_sampling(train_set, n, trainer)
            elif type_sampling == "entropy":
                new_reviews = entropy_sampling(train_set, n, trainer)
        else:
            # Get the n indices randomly
            train_set, new_reviews = pick_n_random_reviews(train_set, n)

        if train_new_models:
            # Get a new non-fine-tuned model and expand the training pool
            tokenizer, data_collator, model = get_model_util(
                model_checkpoint)
            train_pool = add_samples_to_pool(train_pool, new_reviews)
        else:
            # Train iteratively by keeping the same model, with new rows
            train_pool = create_dataset_from_empty_df()
            train_pool = add_samples_to_pool(train_pool, new_reviews)
```

Figure 4.3: A semi-pseudo code version of the main training loop for active learning training with the Hugging Face application programming interface. All methods used are either shortened or described elsewhere in the source code. This figure serves a purpose for understanding how the models are trained.

The main takeaway from the snippet in Figure 4.3 is the opportunity to customize the training process. The program has multiple input variables to control this process. This includes:

- How many samples to train on in each iteration.
- What model to download and use for fine-tuning from Hugging Face.
- Whether one wants to train using AL or not.
- What type of AL sampling methods to use - if AL is chosen.
- Whether one wants to train iteratively or use new-models for each iteration of training.

Another key element to the snippet is the fact that there are two ways the training pool is altered. This depends if the *train_new_models* flag is active or not. If one wants to use the new-models training method, the model is initialized a new, and the dataset increases in size with the new samples found for later iterations. If one wants to train iteratively, a new model will not be initialized, but the training pool will only consist of the newly found samples. This separates the two methods used in this project.

In addition, it is noteworthy that the majority of the outcomes presented in Chapter 5 will be derived from the metrics computed on line 26 in Figure 4.3. This line employs a predefined function for calculating metrics that has been specified by the project team. The function is displayed in Figure 4.4. It returns a dictionary of the metrics found. The results obtained are based on evaluation on the test dataset derived.

```python

from sklearn.metrics import (accuracy_score, balanced_accuracy_score,
    precision_score, recall_score, f1_score)

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)

    # Calculate accuracy using sklearn's functions
    acc = accuracy_score(labels, preds)
    balanced_accuracy = balanced_accuracy_score(labels, preds)
    f1 = f1_score(labels, preds)
    recall = recall_score(labels, preds)
    precision = precision_score(labels, preds)

    accuracy_dict = {
        'accuracy': acc,
        'balanced_accuracy': balanced_accuracy,
        'f1_score': f1,
        'recall': recall,
        'precision': precision,
    }

    return accuracy_dict
```

Figure 4.4: Predefined function that derives metrics from the predictions of the model, evaluated on the test dataset.

4.3.1 Scikit-Learn and Hugging Face API

In the final version of the source code for this project, the Hugging Face API accounted for most of the solution. Scikit-learn (Sklearn) was primarily used for measuring different accuracy metrics for the results, as seen in Figure 4.4. This was not always the case, as there were many useful tools offered by Sklearn that the project group wished to use.

For one, there were multiple PyPi packages that offer useful functionality for the project. *modAL* [11] is a package that offer methods for AL sampling. This was recommended to the project group early in the process, but a problem with compatibility with the Hugging Face trainer API occurred. Every sampling method in the package requires a *BaseEstimator*, which is a base class for all estimators in Sklearn. Therefore, the project group started implementing a Sklearn solution to accommodate the estimator problem. This involved using Sklearn's *pipeline* tool [46], to use the transformer language from Hugging Face and feed these weights to a Sklearn classifier (such as *LogisticRegression [47]*), for it to have the BaseEstimator that modAL requires. An example of this is illustrated in an article written on Hugging Face by the Sklearn team [48].

The problem with the Sklearn solution was the training itself. Unlike the Hugging Face trainer API, one cannot train on all of the classifiers iteratively. There are some classifiers that support a method called *partial_fit* [49], which the project

group tried, but found no results. Although it now was possible to use modAL natively, it serves little purpose when the training itself fails.

This experimentation led to the final solution for this project. The Sklearn pipeline was scrapped and the native Hugging Face trainer API was used, as shown in Figure 4.3. The project group still wanted to implement the modAL package - or at least the source code for it. Therefore, the AL sampling methods from the package were rewritten by the project group to support the Hugging Face model instead. This resulted in a Hugging Face API training process with supporting AL sampling methods originating from modAL. Sklearn provided, as mentioned earlier, accuracy metrics used in the results from training, based on the predictions made by the Hugging Face model. Everything else in the source code for the project is either related to Hugging Face or modAL.

4.3.2 Training Arguments

To reproduce the results in this project, it is important to examine the training arguments defined. As previously mentioned, the models were trained using the Trainer API from Hugging Face. The *TrainingArguments* class can be found in the same API and is used as an argument for the Trainer. This class defines several aspects of the training process. The project group arrived at these arguments after extensive trial and error. The final training arguments reflect the final results from the models and can be seen in Figure 4.5.

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    num_train_epochs=5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    gradient_checkpointing=True,
    gradient_accumulation_steps=8,
    learning_rate=5e-5,
    optim='adafactor',
    weight_decay=0.01,
    tf32=True
)
```

Figure 4.5: The training arguments for each model in the project.

To go into more detail about the training arguments, they can be looked at one by one:

- **num_train_epochs**: This is the most significant argument, as it specifies the number of times the model will be trained on the dataset provided. For the project at hand, this means how many times each iteration of data will be trained on. One epoch is defined as one complete pass through the entire training dataset. The duration of training was influenced by limited time and resources, as mentioned in Chapter 4.1. As a result, the project group

chose a middle ground for the number of epochs used in each iteration of training. This decision led to 5 epochs per iteration.

- **per_device_train_batch_size**: This argument specifies the number of training samples per batch per GPU.
- **per_device_eval_batch_size**: This argument specifies the number of evaluation samples per batch per GPU.
- **gradient_checkpointing**: This argument specifies whether or not to use gradient checkpointing to reduce memory usage. It is a technique that trades compute for memory by recomputing intermediate activations during backpropagation.
- **gradient_accumulation_steps**: This argument specifies the number of steps to accumulate gradients before performing an optimizer step. This can be used to simulate larger batch sizes without increasing memory usage. Both of the gradient arguments are used to save memory because of limited memory access and out-of-memory errors during training.
- **learning_rate**: This argument specifies the learning rate for the optimizer. The learning rate determines how much the weights of the model are updated during training. The same learning rate is kept throughout the training process.
- **optim**: This argument specifies the optimizer to use. An optimizer is an algorithm that updates the weights of a model during training based on the gradients computed during backpropagation. The *adafactor* optimizer is an adaptive learning rate optimizer that focuses on fast training of large scale neural networks.
- **weight_decay**: This argument specifies the weight decay for the optimizer. Weight decay is a regularization technique that adds a penalty term to the loss function to encourage smaller weights.
- **tf32**: This argument specifies whether or not to use TF32 precision. TF32 is a mixed precision format that uses 10-bit mantissa and 5-bit exponent. A NVIDIA Ampere hardware with a compute capability of 8.0 is required for this file format.

Chapter 5

Results

The results are presented in this chapter and reflect the methods described in Chapter 4. Each chapter is related to one of the three research questions presented in Chapter 1. Specifically, Chapter 5.1 corresponds to RQ1, Chapter 5.2 corresponds to RQ2, and Chapter 5.3 corresponds to RQ3.

5.1 Fine-Tuning Methods and Accuracy Metrics

As previously mentioned, the models in this project are trained using two methods: iteratively training the same model with new data and training new models for each iteration with a continuously larger dataset. This chapter presents a comparison of how each training method scores on different accuracy metrics from Scikit-learn (Sklearn). The research question for this chapter is as follows:

RQ1 *What are the differences in the accuracy metrics of Norwegian natural language processing models for classification problems using different fine-tuning methods while incorporating active learning?*

In this chapter, the project group presents the results as graphs from the training process, in addition to tables containing the progression of accuracy metrics throughout the training. For each of the fine-tuning training methods, a table is complemented by a description of the different active learning (AL) sampling methods, in addition to traditional random sampling. This is also displayed in the graphs. Chapter 5.1.1 displays accuracy metrics for the iterative training method, while Chapter 5.1.2 displays accuracy metrics for the new-models training method.

5.1.1 Iterative Training Method

Accuracy Score

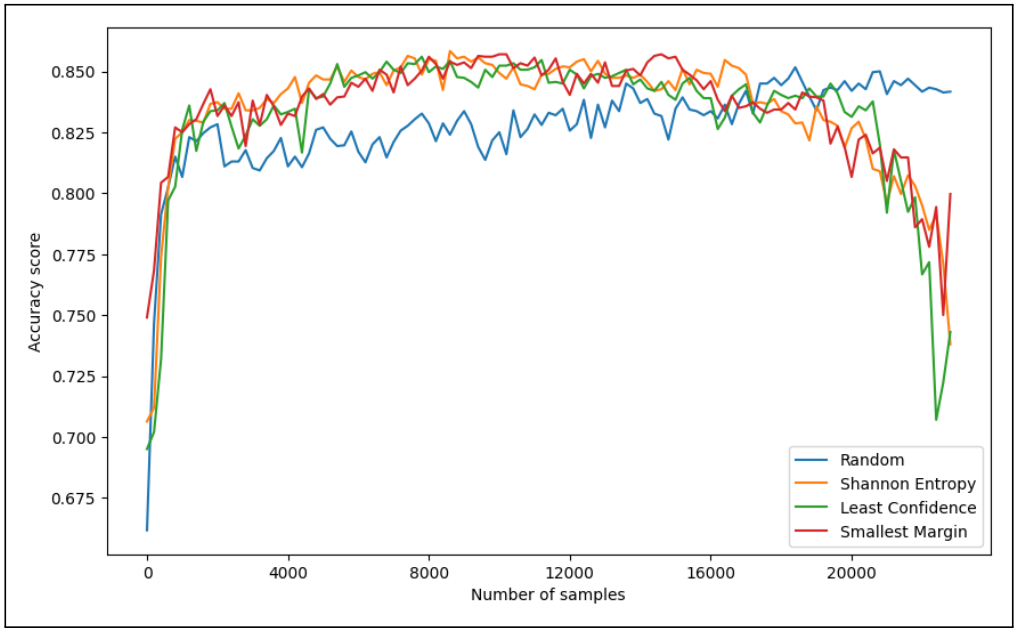


Figure 5.1: All training runs for the iterative training method using the accuracy score metric.

Table 5.1: The accuracy score metric for Figure 5.1 is shown in this table. Each active learning method used, in addition to traditional random sampling, is displayed by the interval of number of samples.

Samples	Least Confidence	Smallest Margin	Shannon Entropy	Random
0	0.695	0.749	0.706	0.662
2,000	0.834	0.832	0.837	0.828
4,000	0.833	0.833	0.843	0.811
6,000	0.848	0.844	0.848	0.817
8,000	0.850	0.856	0.855	0.829
10,000	0.852	0.857	0.849	0.825
12,000	0.851	0.840	0.852	0.826
14,000	0.847	0.848	0.849	0.837
16,000	0.839	0.846	0.849	0.834
18,000	0.840	0.834	0.834	0.844
20,000	0.831	0.807	0.827	0.842
22,000	0.767	0.789	0.795	0.842

Balanced Accuracy

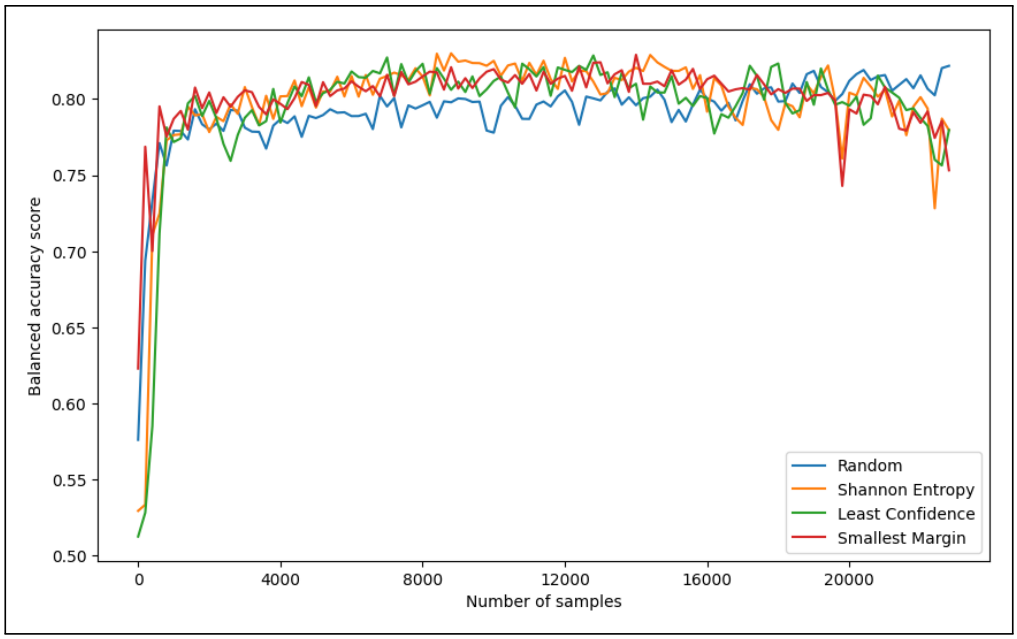


Figure 5.2: All training runs for the iterative training method using the balanced accuracy score metric.

Table 5.2: The balanced accuracy score metric for Figure 5.2 is shown in this table. Each active learning method used, in addition to traditional random sampling, is displayed by the interval of number of samples.

Samples	Least Confidence	Smallest Margin	Shannon Entropy	Random
0	0.513	0.623	0.529	0.576
2,000	0.799	0.804	0.778	0.780
4,000	0.785	0.797	0.802	0.787
6,000	0.818	0.812	0.815	0.789
8,000	0.823	0.815	0.815	0.796
10,000	0.812	0.820	0.825	0.778
12,000	0.819	0.815	0.827	0.806
14,000	0.810	0.829	0.821	0.796
16,000	0.800	0.813	0.792	0.800
18,000	0.823	0.807	0.780	0.798
20,000	0.796	0.793	0.804	0.812
22,000	0.788	0.784	0.801	0.816

F1 Score

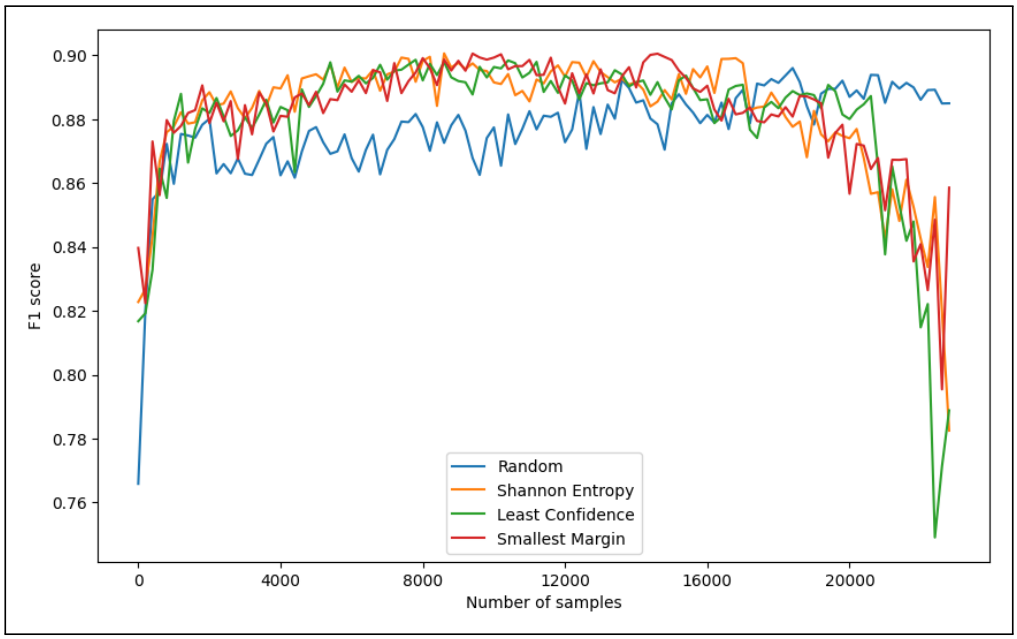


Figure 5.3: All training runs for the iterative training method using the F1 score metric.

Table 5.3: The F1 score metric for Figure 5.3 is shown in this table. Each active learning method used, in addition to traditional random sampling, is displayed by the interval of number of samples.

Samples	Least Confidence	Smallest Margin	Shannon Entropy	Random
0	0.816	0.840	0.823	0.766
2,000	0.882	0.879	0.888	0.880
4,000	0.884	0.881	0.890	0.862
6,000	0.892	0.889	0.892	0.868
8,000	0.892	0.899	0.898	0.878
10,000	0.896	0.899	0.892	0.877
12,000	0.894	0.885	0.893	0.873
14,000	0.892	0.890	0.892	0.885
16,000	0.886	0.890	0.897	0.881
18,000	0.883	0.881	0.885	0.891
20,000	0.880	0.857	0.874	0.887
22,000	0.815	0.841	0.843	0.886

5.1.2 New-Models Training Method

Accuracy Score

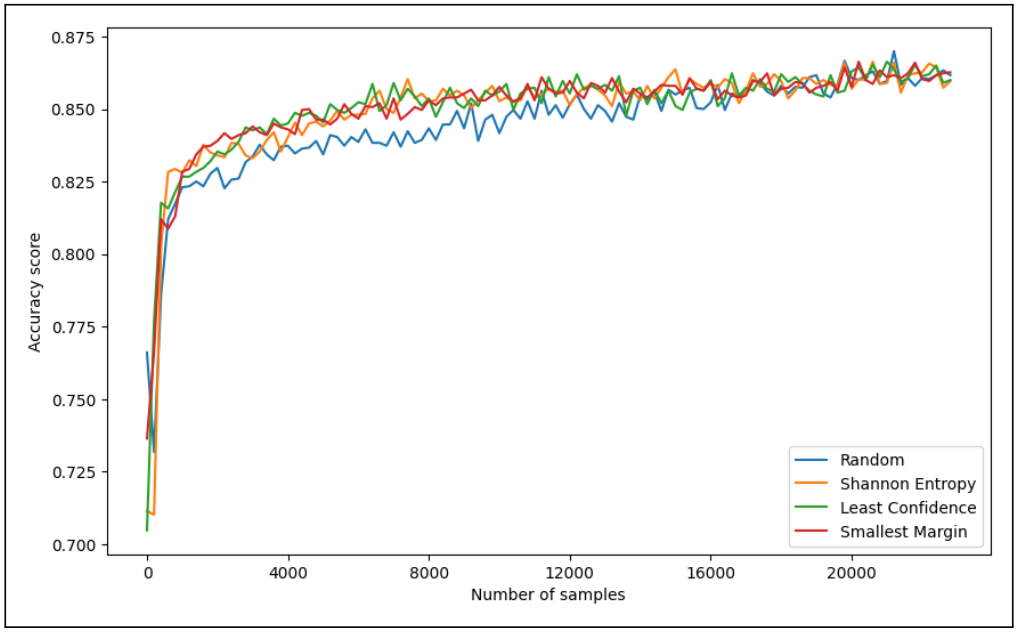


Figure 5.4: All training runs for the new-models training method using the accuracy score metric.

Table 5.4: The accuracy score metric for Figure 5.4 is shown in this table. Each active learning method used, in addition to traditional random sampling, is displayed by the interval of number of samples.

Samples	Least Confidence	Smallest Margin	Shannon Entropy	Random
0	0.705	0.736	0.711	0.766
2,000	0.835	0.839	0.834	0.830
4,000	0.845	0.843	0.840	0.837
6,000	0.852	0.847	0.848	0.839
8,000	0.854	0.853	0.853	0.843
10,000	0.856	0.858	0.853	0.842
12,000	0.855	0.860	0.851	0.852
14,000	0.857	0.855	0.853	0.855
16,000	0.860	0.859	0.859	0.852
18,000	0.862	0.857	0.860	0.858
20,000	0.863	0.858	0.857	0.861
22,000	0.861	0.860	0.863	0.861

Balanced Accuracy

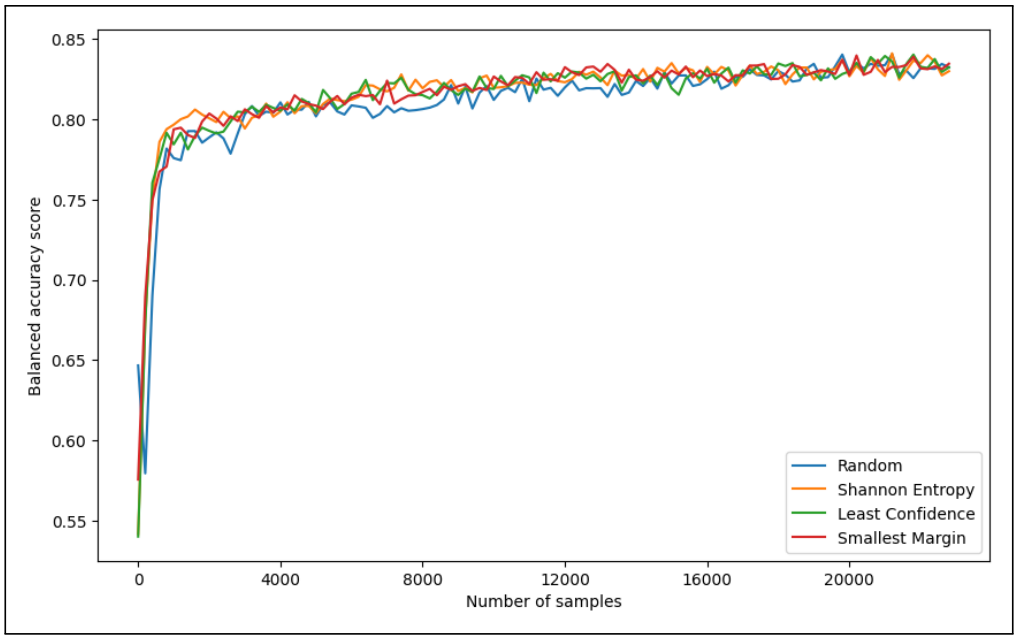


Figure 5.5: All training runs for the new-models training method using the balanced accuracy metric.

Table 5.5: The balanced accuracy score metric for Figure 5.5 is shown in this table. Each active learning method used, in addition to traditional random sampling, is displayed by the interval of number of samples.

Samples	Least Confidence	Smallest Margin	Shannon Entropy	Random
0	0.540	0.576	0.542	0.647
2,000	0.793	0.804	0.801	0.789
4,000	0.805	0.807	0.805	0.811
6,000	0.816	0.813	0.812	0.809
8,000	0.815	0.817	0.819	0.806
10,000	0.819	0.827	0.820	0.812
12,000	0.826	0.832	0.823	0.820
14,000	0.827	0.825	0.825	0.824
16,000	0.831	0.825	0.833	0.830
18,000	0.835	0.826	0.831	0.830
20,000	0.830	0.828	0.827	0.830
22,000	0.833	0.832	0.835	0.832

F1 Score

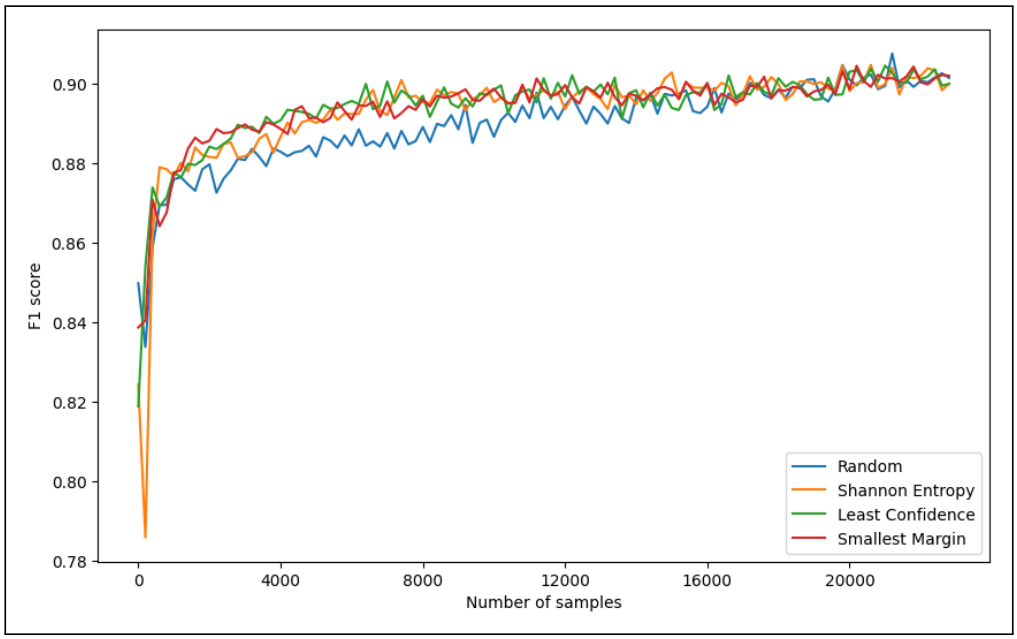


Figure 5.6: All training runs for the new-models training method using the F1 score metric.

Table 5.6: The F1 score metric for Figure 5.6 is shown in this table. Each active learning method used, in addition to traditional random sampling, is displayed by the interval of number of samples.

Samples	Least Confidence	Smallest Margin	Shannon Entropy	Random
0	0.8188	0.8387	0.8243	0.8498
2,000	0.8842	0.8857	0.8816	0.8797
4,000	0.8908	0.8887	0.8867	0.8829
6,000	0.8957	0.8910	0.8924	0.8845
8,000	0.8970	0.8961	0.8955	0.8892
10,000	0.8985	0.8988	0.8954	0.8867
12,000	0.8968	0.8997	0.8937	0.8945
14,000	0.8984	0.8970	0.8948	0.8965
16,000	0.9001	0.9002	0.8991	0.8942
18,000	0.9014	0.8984	0.9000	0.8986
20,000	0.9030	0.8986	0.8982	0.9011
22,000	0.9011	0.9004	0.9020	0.9007

5.2 Comparing Active Learning Sampling Methods

The three AL sampling methods that are presented in the results are *Least Confidence*, *Smallest Margin* and *Shannon Entropy*. In this chapter, each of the methods is presented and compared by their peak metrics, and how they compare to the baseline of traditional random sampling. All of the metric numbers used in this chapter are generated from the same code that produced the graphs and metric numbers in Chapter 5.1. The research question for this chapter is as follows:

RQ2 *Which of the well-known active learning methods score highest in accuracy metrics when selecting informative samples for labeling in Norwegian natural language processing models, compared to the traditional random sampling method? Additionally, how many samples can be potentially saved by employing active learning?*

5.2.1 Baseline with Traditional Sampling

To compare the different AL sampling methods, traditional random sampling will first be presented. This baseline, which is trained by both presented fine-tuning methods, iterative and new-models, is measured in accuracy score, balanced accuracy score, and F1 score. In the later subchapter, each AL sampling method is compared to this baseline in terms of peak accuracy metric reached and how many samples this took to accomplish. The traditional sampling baseline is presented in Table 5.7.

Table 5.7: Baseline with traditional sampling method. Both of the fine-tuning methods are displayed, in addition to their respective **peak** metric value and how many samples it took to reach this metric.

Fine-tuning $_{\textbf{Metric}}$	**Peak metric value**	**Samples**
Iterative $_{\text{Accuracy}}$	0.851716	18,400
Iterative $_{\text{Balanced}}$	0.821800	22,800
Iterative $_{\text{F1}}$	0.896004	18,400
New-Models $_{\text{Accuracy}}$	0.870043	21,200
New-Models $_{\text{Balanced}}$	0.840520	21,200
New-Models $_{\text{F1}}$	0.907670	21200

The following chapters will have expanded tables. These tables contain comparisons to the baseline. This includes two distinct comparisons:

- ***(+/-) Metric***: The first column compares the peak accuracy metric of the baseline model to that of the AL sampling method. This comparison involves examining the sample count required by the AL sampling method to reach its peak and then checking the metric value for the baseline model at that sample count. The results show the difference between the AL sampling method and the baseline model in terms of increased or decreased metric value.
- ***(%) Samples***: The second comparison column examines the number of samples required by the AL sampling method to reach its peak compared to the number of samples required by the baseline model to reach its peak. This comparison shows how many potential samples one can avoid labeling by hand, and is measured in percentages. Also, to emphasize, these peaks do not mean that the compared metrics have the same value, as alluded to in the previous bullet point.

5.2.2 Least Confidence Sampling

The first AL sampling method presented is the *Least Confidence* sampling method in Table 5.8:

Table 5.8: Results regarding the *Least Confidence* sampling method. Both of the fine-tuning methods are displayed, in addition to their respective **peak** metric value and how many samples it took to reach this metric. The two rightmost columns are comparisons to the baseline in Chapter 5.2.1. The first column compares the peak metric value of the *Least Confidence* sampling method to the metric value of the baseline at the sample count where the AL method peaked. The second column displays how many samples the *Least Confidence* sampling method requires to reach its peak compared to the baseline.

Fine-tuning $_{\text{Metric}}$	**Peak metric value**	**Samples**	**(+/-) Metric**	**(%) Samples**
Iterative $_{\text{Accuracy}}$	0.856048	7,800	+0.023326	-57.6%
Iterative $_{\text{Balanced}}$	0.828616	12,800	+0.028175	-43.9%
Iterative $_{\text{F1}}$	0.898592	7,800	+0.016988	-57.6%
New-Models $_{\text{Accuracy}}$	0.866378	21,000	+0.006665	-7.9%
New-Models $_{\text{Balanced}}$	0.840311	21,800	+0.014647	+2.8%
New-Models $_{\text{F1}}$	0.904592	21,000	+0.005093	-0.9%

5.2.3 Smallest Margin Sampling

Next up are the results for *Smallest Margin* sampling in table 5.9:

Table 5.9: Results regarding the *Smallest Margin* sampling method. Both of the fine-tuning methods are displayed, in addition to their respective **peak** metric value and how many samples it took to reach this metric. The two rightmost columns are comparisons to the baseline in Chapter 5.2.1. The first column compares the peak metric value of the *Smallest Margin* sampling method to the metric value of the baseline at the sample count where the AL method peaked. The second column displays how many samples the *Smallest Margin* sampling method requires to reach its peak compared to the baseline.

Fine-tuning $_{\textbf{Metric}}$	**Peak metric value**	**Samples**	**(+/-) Metric**	**(%) Samples**
Iterative $_{\text{Accuracy}}$	0.857048	10,000	+0.03199	-45.7%
Iterative $_{\text{Balanced}}$	0.829079	14,000	+0.033172	-38.6%
Iterative $_{\text{F1}}$	0.900531	9,400	+0.03268	-48.9%
New-Models $_{\text{Accuracy}}$	0.866378	20,200	+0.024659	-4.7%
New-Models $_{\text{Balanced}}$	0.839774	20,200	+0.004848	-4.7%
New-Models $_{\text{F1}}$	0.904547	20,200	+0.004929	-4.7%

5.2.4 Shannon Entropy Sampling

Lastly, the results for *Shannon Entropy* sampling, are displayed in table 5.10:

Table 5.10: Results regarding the *Shannon Entropy* sampling method. Both of the fine-tuning methods are displayed, in addition to their respective **peak** metric value and how many samples it took to reach this metric. The two rightmost columns are comparisons to the baseline in Chapter 5.2.1. The first column compares the peak metric value of the *Shannon Entropy* sampling method to the metric value of the baseline at the sample count where the AL method peaked. The second column displays how many samples the *Shannon Entropy* sampling method requires to reach its peak compared to the baseline.

Fine-tuning $_{\textbf{metric}}$	**Peak metric value**	**Samples**	**(+/-) Metric**	**(%) Samples**
Iterative $_{\text{accuracy}}$	0.858381	8,600	+0.034322	-53.3%
Iterative $_{\text{balanced}}$	0.830019	8,800	+0.032159	-61.4%
Iterative $_{\text{F1}}$	0.900585	8,600	+0.027998	-53.3%
New-Models $_{\text{accuracy}}$	0.866378	20,600	+0.003332	-2.8%
New-Models $_{\text{balanced}}$	0.841101	21,200	+0.000581	0%
New-Models $_{\text{F1}}$	0.904773	20,600	+0.002421	-2.8%

5.3 Performance of Fine-Tuning & Active Learning

Accuracy metrics are not the only thing to measure model training on. This chapter will present other metrics, such as training time and training loss. The training time can be separated into the training itself and the process of picking samples to train on. This combined time will result in a total time for the whole *training job*. Training loss is also an important metric for understanding the training process for a model, particularly when analyzing potential over- or underfitting. The research question for this chapter is as follows:

RQ3 *How do different active learning methods, combined with different fine-tuning methods, affect the performance, in terms of training time and training loss, when training Norwegian natural language processing models, compared to traditional data sampling methods?*

5.3.1 Train Time

Training time in terms of metrics generated from the HuggingFace application programming interface (API) is measured in seconds and describes how much time the trainer uses in each iteration of the training process. In the case of this project, this is either each time the iterative method trains on each batch of samples or when the new-models method trains on the continuously increasing pool of data. In this chapter, the training time will be displayed in graphs, one which will describe time used in each iteration of training, and the other as a cumulative visualization of seconds used. Lastly, the total time of the whole training job will be explored. This includes initialization of models and datasets, in addition to picking samples from the dataset to train on.

Train Time Per Iteration

In Figure 5.7, the training time, in seconds, is displayed for each iteration of the training process. The main takeaway from this graph is that the iterative training method uses approximately the same amount of time throughout the whole training process, while the new-models method uses linearly more time for each iteration.

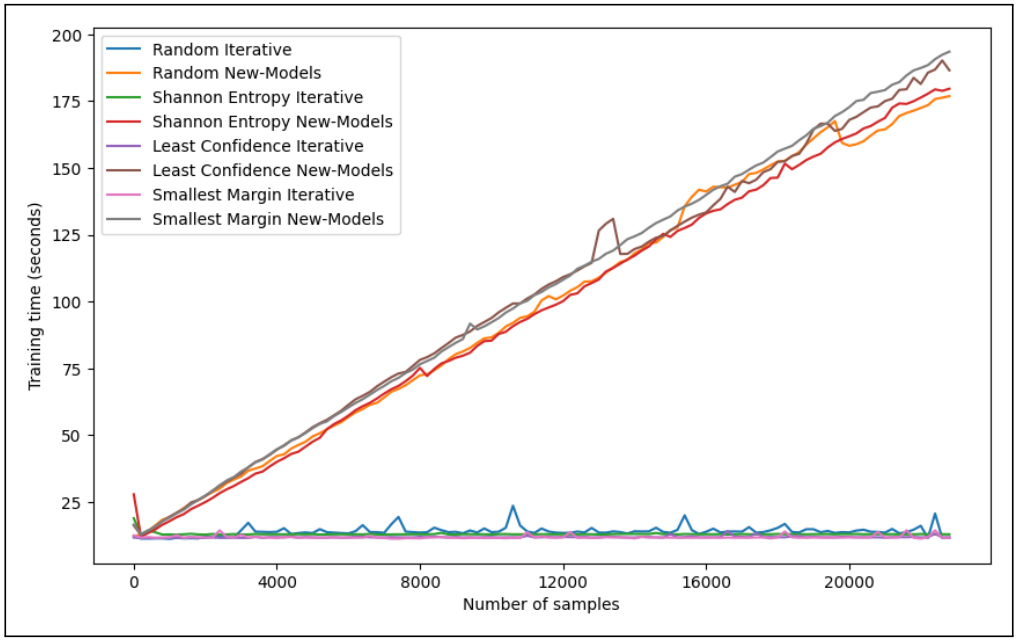


Figure 5.7: Training time in seconds for each iteration of each trained model. Both fine-tuning methods are displayed, with the different active learning sampling methods for each training method.

Cumulative Train Time

Figure 5.8 displays the total time used by each model with each AL sampling method by accumulating all the previous entries of seconds to make a graph. This graph shows the development of time usage during the whole training process. The main takeaways from this graph are that the new-models method uses continuously more time, while the iterative version uses linear amounts of time during training.

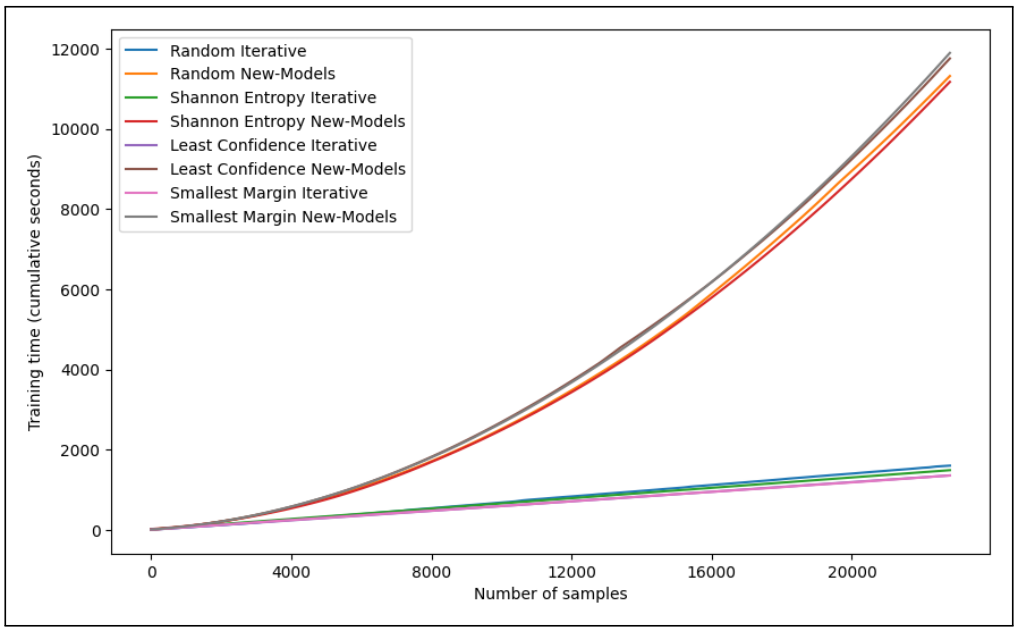


Figure 5.8: Cumulative train time for all methods, including both fine-tuning training methods, with the different active learning sampling methods.

Total Time on Training Job

Table 5.11 displays how much time each of the sampling methods and fine-tuning training methods use on training and the job in total. As previously mentioned, it does not only take computational power to train the model itself but also for the sampling methods to pick informative samples for the model to train on. The total time will express this difference. In particular, with AL sampling methods, the model has to predict each sample in the dataset to pick the most informative samples to train on. Furthermore, there is overhead in terms of initializing models/datasets for both fine-tuning methods, and creating new models in the new-models method. The value is measured in seconds for both train time and total time.

Table 5.11: Train time and total time for each training job. Each sampling method is represented by both fine-tuning training methods. The difference is displayed in the rightmost column. All values are measured in seconds.

Fine-tuning $_{\textbf{Sampling}}$	**Train time**	**Total time**	**Increase**
Iterative $_{\text{Random}}$	1,607	1,824	217
Iterative $_{\text{Shannon Entropy}}$	1,493	2,449	956
Iterative $_{\text{Least Confidence}}$	1,363	2,329	966
Iterative $_{\text{Smallest Margin}}$	1,360	2,102	742
New-Models $_{\text{Random}}$	11,312	17,936	6,624
New-Models $_{\text{Shannon Entropy}}$	11,166	18,317	7,151
New-Models $_{\text{Least Confidence}}$	11,750	19,404	7,654
New-Models $_{\text{Smallest Margin}}$	11,888	19,926	8,038

5.3.2 Training Loss

The last performance metric that will be displayed in this chapter is training loss. Figure 5.9 shows each sampling method, with both fine-tuning training methods, and their corresponding training loss across the whole training session.

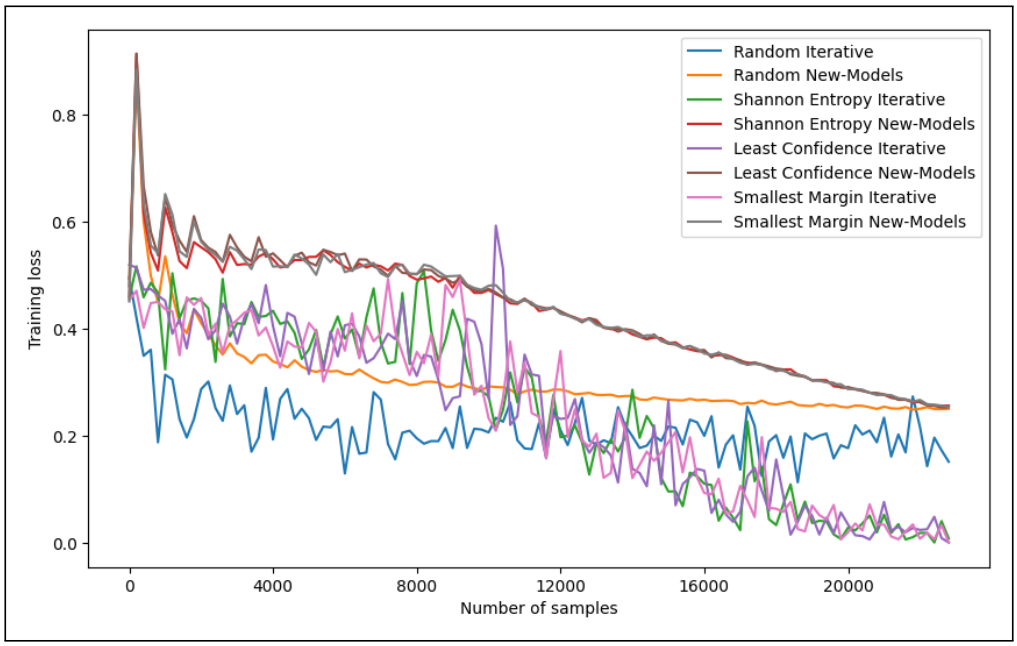


Figure 5.9: The training loss for every model. This includes both fine-tuning methods with every sampling method.

Chapter 6

Discussion

The findings from Chapter 5, Results, suggest that using active learning (AL) could be beneficial. This is an exciting discovery in and of itself. Based on published work, Norwegian Bidirectional Encoder Representations from Transformers (BERT) models haven't been utilized much for this type of task. Although AL has several positive and exciting factors, it also has some downsides. Therefore, it's crucial to assess one's own needs and available resources. What stands out in this work is the unconventional approach of iteratively training language models during the AL cycle. This chapter will be organized similarly to Chapter 5, Results, with Chapter 6.1 associated with RQ1, Chapter 6.2 covering RQ2, and the same with Chapter 6.3 and RQ3.

6.1 Examining Training with Fine-Tuning Methods

This chapter will focus on the first phase of training, which is the most relevant to discuss when using AL since the main focus of this approach is to reduce the amount of data needed. The benefits of using AL are clearly shown in Figures 5.1 and 5.4. In Figure 5.1, the advantage is even more apparent compared to Figure 5.4. In Figure 5.4, where the iterative training method is used, *random* requires considerably more data to achieve the same accuracy. In fact, random does not even reach as high a peak in accuracy score as *Shannon Entropy*, *Least Confidence*, and *Smallest Margin*. This can also be seen in the other score metrics: *balanced accuracy* and *F1 score*. When looking at the corresponding graphs, the scores converge around 8,000 to 10,000 samples. Compared to a more traditional approach to random training, which does not reach the convergence point, even with 22,000 samples. It is worth noting that the iterative way of training when using random is not exactly the same as the normal traditional way of training. On the contrary, creating new models in each training cycle with random is closer to the approach of traditional supervised training.

When looking closer at the results of the training that used the more traditional approach, making new models in each training cycle, the difference is not as sig-

nificant as with iterative training when compared to the random baseline. Despite this, a clear improvement is still seen with AL compared to random selection when creating new models.

If one compare the accuracy of the different AL sampling methods from iterative and new-models, they show quite similar results when looking at accuracy score at 8,000 samples. However, if one looks at the interval from 4,000 to 6,000, new-models is slightly higher. This argues for using new-models over iterative.

It is worth noting that the iterative approach requires far less computational power compared to creating new models in each training cycle. This can be a decisive factor in determining the most feasible solution in a given situation. This will be further discussed in Chapter 6.3. On the other hand, the new-model approach achieves a higher peak in accuracy score compared to its iterative counterpart, with samples in the higher region. This will be discussed further in the next Chapter, 6.2.

6.2 Active Learning versus Traditional Sampling

This chapter reviews the findings presented in Chapter 5.2, which addresses RQ2. There is not always a perfect way to present the results. One must therefore do it as they deem most appropriate. Chapter 5.2 emphasized peak values, which are easily misrepresented in fluctuating graphs. As such, it is possible to achieve a high accuracy score with fewer samples. In some cases, a marginally lower accuracy score may be acceptable if it significantly reduces the number of necessary samples. Despite this, the project group believes that the data is appropriately represented. However, to allow for further improvement, all data is available on a public **GitHub** repository [45].

Table 5.7 in Chapter 5.2.1 indicates that between 18,400 and 22,800 samples are required to reach peak values for all of the metrics using *random* sampling. New-models require more samples but have higher metric values, with the exception of the balanced accuracy score on iterative which has a lower value but still requires more samples. Figure 5.2 shows that random on balanced accuracy score has three peaks at approximately 18,000, 21,000, and 22,800, all of which could be representative for random as a baseline. Ultimately, the table demonstrates that a significant amount of data and samples are required to train the model.

In Chapters 5.2.2, 5.2.3, and 5.2.4, the findings from the sampling methods are reviewed. The baseline from random is compared with each of the AL sampling methods used in the research. Each of these chapters contains a Table (5.8, 5.9, 5.10), that shows the percentage of samples that can be saved based on when the AL sampling methods reach their respective peaks. A quick overview of these tables clearly shows that all iterative methods show significant savings compared to the iterative baseline. New-models also show savings, but not as significant. This is because the metrics for new-models improve marginally throughout the training process. At first glance, it may seem that new-models do not have a significant effect, but this does not provide an accurate picture. Training with

new-models results in higher scores earlier than the baseline, which is the main principle of AL. Therefore, it is important to examine the graphs that show development and focus on the beginning of the graph. Both methods have advantages compared to the baseline, including requiring fewer samples and improved accuracy and runtime. The latter will be discussed in Chapter 6.3. Since the purpose of using AL is to reduce the need for samples, the focus will be on the results early in the training process when comparing all sampling methods.

When examining the various sampling methods, there are no significant differences between them. If these training runs were conducted multiple times and an average was taken, it could potentially be easier to see and recognize if any of the methods proved to be better than the others. When looking at the sampling methods with the iterative approach, Shannon Entropy possibly performs slightly better. Shannon Entropy has fewer low dips and remains relatively high. When looking at the new-models approach, Shannon Entropy also stands out here. It has a slightly lower score until 5,000 samples, then it transitions to being in the top tier at around 8,000 samples. Note that the differences are small and it is difficult to distinguish the AL sampling methods from each other.

6.3 Model Performance Review

This chapter reviews the findings presented in Chapter 5.3, which addresses RQ3. The findings from Chapter 5.1 and 5.2 conveyed that there were many similarities between the different fine-tuning methods, but this is not the case in Chapter 5.3. Here, the difference between the iterative and new-model methods becomes clear.

6.3.1 Time Used

The first thing displayed in Chapter 5.3 is the runtime of the training itself. Note that this only includes the training itself, and not the retrieval of datasets, initialization of models, and calculations related to sampling methods. Chapter 5.3 has two graphs that show the runtime of training, where Figure 5.7 shows the runtime of each training instance, while Figure 5.8 shows the accumulated runtime, i.e., the total runtime for training. In both of these graphs, two distinct trends can be seen, and these two differ to a large extent. These trends are quite understandable considering the approaches used in training. In Figure 5.7, it can be seen that training using new-models has a linear increase, while iterative training has a constant development. The reason for this large difference lies in the size of the dataset that language models train on. In new-models, the dataset increases by 200 samples in each training cycle, and it therefore takes longer to fine-tune the model. The same correlation between runtime and dataset size can be seen in the iterative approach. This is completely constant and does not increase because it always trains on 200 samples. Figure 5.7 is a very good graph for understanding the relationship between runtime and choice of approach. Figure 5.8 shows the total runtime and provides a very good overview of the training time. In this graph, the difference between training methods is even more evident. This difference is so

large that it is worth considering what one's greatest need is, what resources are available, or how one wants to prioritize these resources. If one accepts a slightly lower score, it may be appropriate to use the iterative approach. If one absolutely wants to maximize the score and computing power is not a problem, then it may be appropriate to use new-models.

Another factor that may influence the choice of training approaches is power consumption. A correlation can be drawn between the time spent on training and power consumption. Implementing fine-tuning on a large scale can result in significant power savings when choosing the iterative approach compared to the new-models approach. Overall, these two approaches require little power compared to creating the language models themselves. This resonates with Chapter 2.8, which explains the environmental impact of language model training. Although the fine-tuning methods has less power requirement than training language models from scratch, the difference between the methods are significant. For instance, the iterative version of the Smallest Margin AL method uses about 89% less time than the new-models version.

In Table 5.11, the total time for training is presented. This data includes everything from the initialization of models, retrieval of datasets, the training itself, and the time spent on sampling methods. In this table, one can easily compare the training time and the total time spent on training. Here, one can see that the difference from iterative random on *train time* to *total time* is 217 seconds, a 13.5% increase, but with AL sampling methods, the difference is greater. This makes sense as it takes time to use these methods to select new samples, compared to random where some arbitrary samples are chosen. Of the three AL sampling methods on iterative, there is a slight difference in how large the increase is. It is uncertain whether this information can be used to conclude anything, and the project group does not have enough information to conclude why this is so. But in this case, Smallest Margin uses slightly less time compared to Shannon Entropy and Least Confidence.

Looking at new-models in Table 5.11, one can see that there is quite a bit more overhead happening alongside the training itself. One cannot be sure what this is due to, but it probably comes from initializing new models before each training and possibly deleting old models. So it makes sense that new-models will have a larger difference than iterative. Looking at the AL sampling methods closest to random in iterative and new-models, one ends up with fairly similar numbers. Iterative Smallest Margin at 742 minus iterative random at 217 gives 525 seconds. New-models Shannon Entropy at 7,151 minus new-models random at 6,624 gives 526 seconds. Based on this, one can assume that introducing AL sampling methods is less time-consuming than introducing training using new-models. A very simple estimate suggests that the process of creating new models and deleting old ones takes at least 6,098 seconds, in this project. This is calculated by taking 6,624 minus 526 It should be noted that this estimate is not precise and it must be taken into consideration that other AL sampling methods had longer training times. The most important thing here is to show that the overhead of creating and deleting used/old language models is more time-consuming than using AL sampling methods.

6.3.2 The Importance of Training Loss

Chapter 5.3.2 introduces Figure 5.9, which illustrates the train loss for each model during the training process. This metric is crucial to understand how the model performs and for researchers to consider possible flaws or intricacies using AL. One clear correlation between all AL sampling methods for both iterative and new-models fine-tuning is that the training loss is significantly higher than random sampling in the first half of the training process. This can be explained by the type of samples each method trains on. The point of AL is to select samples that provide the most value for the model. These samples happen to be the "hardest" to classify, naturally. Therefore, the training loss would be higher earlier in the training process, contrary to random sampling, which selects samples from all across the dataset.

The more interesting part of Figure 5.9 is the clear difference between the iterative and new-models fine-tuning methods in terms of training loss. Firstly, the new-models method has a very steadfast development both for random sampling and AL sampling methods. The only difference between them is that the AL sampling methods have a higher loss earlier in the training process. The training loss for both sampling methods naturally ends up at the same value since they ultimately train on the same data, in addition to training on the whole dataset in one go. The same cannot be said with the iterative fine-tuning method. In this case, the random sampling ends up with a training loss around 0.2, while the AL sampling methods end up with zero loss. This is a worrying sign for the training process but can also explain the massive dip in accuracy metrics for the iterative method in Chapter 5.1.

There might be multiple reasons for the low training loss for the iterative fine-tuning method at the end of the training. One reason might be the fact that the remaining samples in the later stage of training do not offer much value to the model or, in other words, are "easy" for the model to classify. One can also consider these samples as being of the lowest echelon because of their lack of value. So when the model gets to the point of trying to learn from these samples, it might largely generalize on them and "forget" about the valuable samples it learned from in the beginning.

Another reason for the loss might be because of the class disparity in the dataset. There are significantly more positive samples than negative ones. Therefore, at the end of the training process, there might be a case where there are samples from only one class left to train on. This leads to the same problem as in the previous case, where the model tries to generalize on samples that do not offer any significant value from only one class. In either case, there is a strong case for overfitting because of the significantly low training loss. This case is strengthened by the theory that the remaining samples in the dataset are very alike in some way. Either by giving no significant value to the model by being "easy" classifiable samples or by being from one class in particular. The model tries to generalize on a data pool that does not fit being generalized in this case, because it does not represent a large part of the dataset.

One potential solution to prevent the training loss from reaching zero is to weigh the training against the level of uncertainty. This can be achieved, for example, by adjusting the learning rate during training so that models learn more at the beginning and less over time. However, this is only speculation without further information. In hindsight, the project group recognizes that more data should have been stored from the training. It would have been useful to know which classes of samples were selected and how confident the model was in these selections. As mentioned in Chapter 4.2.1, the training dataset consists of several different types of feedback. The proportion of each type is quite skewed, so it would have been interesting to store which types of samples were selected during training. Collecting all this data could have provided a better understanding of what is happening.

It is worth mentioning that in real-life scenarios where active learning would be implemented in training natural language processing (NLP) models, one would never reach the lower echelon of samples because of the practical use of the method. This is because an organization or business would probably classify a batch of samples every week or even every month. The sampler would only sample "good" samples and not an entire dataset which has been conducted in this project. Therefore, this intricacy would probably not affect real-life scenarios, but the project group found it interesting and valuable to highlight.

Chapter 7

Conclusion

Active learning (AL) has been a concept of interest in practical scenarios and has proven useful in this project for training Norwegian natural language processing (NLP) models. The objective of this thesis is to investigate the effectiveness of AL and its employment using different fine-tuning methods for transformer models from Hugging Face. The Norwegian NorBERT 2 model has been the focus of this project as it has proven to be exceptional for different Norwegian text tasks such as sentiment analysis, which is the task at hand in this master thesis. This thesis has three research questions that focus on different aspects of the implementation of AL when training NLP models. The first research question focuses on the difference between fine-tuning language models using an iterative method and a new-models method. The second research question looks at the different AL sampling methods and how they compare to a baseline using traditional random sampling. The AL sampling methods used in this project are *Least Confidence*, *Smallest Margin*, and *Shannon Entropy*. The last research question focuses on the general performance of the different fine-tuning methods using the different sampling methods, both random and AL. The performance is measured in time used in the training process in addition to the training loss for each fine-tuning method with all sampling methods. These research questions will be answered in this chapter.

RQ1 *What are the differences in the accuracy metrics of Norwegian natural language processing models for classification problems using different fine-tuning methods while incorporating active learning?*

The two methods for fine-tuning AL models are vastly different in both accuracy metrics and performance during training. The iterative method is efficient and volatile, while the new-models method is inefficient but steadfast. When training the same model iteratively, one can see a big difference between random sampling and AL sampling. While using AL sampling, the model converges quickly but drops off in the later stages of training. The accuracy metrics are higher than when using random sampling. The difference between random sampling and AL sampling is more clear with the iterative method than with new-models, but the latter also

benefits from AL as this method also achieves higher accuracy at earlier stages of training in comparison to using random sampling. This shows that AL works as it is supposed to. When comparing the two fine-tuning methods, new-models achieves higher accuracy metrics but at a cost. The iterative method is much more time-efficient, which will be explored more in RQ3.

RQ2 *Which of the well-known active learning methods score highest in accuracy metrics when selecting informative samples for labeling in Norwegian natural language processing models, compared to the traditional random sampling method? Additionally, how many samples can be potentially saved by employing active learning?*

When comparing the different AL sampling methods, two clear conclusions can be drawn. Firstly, there is a significant possibility to save resources on labeling data. Secondly, the accuracy metrics are always better when using AL in the experiments conducted by the project group. There are some nuances to the results though. The conclusions can clearly be drawn when using the iterative fine-tuning method. When using AL sampling, it converges using between 38% to 61% fewer samples than traditional random sampling. At the same time, the iterative method shows higher accuracy metrics using the same sample count as the traditional sampler. The results are not as promising using the new-models fine-tuning method. The accuracy metrics are higher in this case as well, but it requires about the same amount of samples as traditional sampling. On the other hand, the new-models method achieves higher metrics across the board compared to the iterative method. When it comes to finding the best AL sampling method, it is hard to conclude. The differences are marginal, but Shannon Entropy might take the marginal lead. Because of the limited number of training runs, this is inconclusive

RQ3 *How do different active learning methods, combined with different fine-tuning methods, affect the performance, in terms of training time and training loss, when training Norwegian natural language processing models, compared to traditional data sampling methods?*

Lastly, the training performance is very interesting, as alluded to earlier. The training time is vastly different when using the two different fine-tuning methods. The new-models method requires a lot of training time compared to the iterative method. This is not surprising, as the dataset the new-models method trains on gets increasingly bigger for each iteration, while the dataset the iterative version trains on is static with 200 samples each iteration. There are more factors to consider though, when using the new-models method. Firstly, for each iteration, a new model has to be initialized, and the previous model has to be deleted. This creates a lot of overhead. So much that one has to carefully consider the importance of slightly higher metrics compared to computational resources. Also, different kinds of tasks require different methods.

When comparing traditional random sampling to AL sampling in terms of performance, one can see that random sampling requires less time. This is not surprising, as AL sampling requires the model to predict on the remaining samples in the dataset to pick valuable samples for labeling. Also here, it is inconclusive which

AL sampling method is the most efficient in terms of total time, as the differences are marginal. After calculating the differences between random sampling and AL sampling, one can conclude that AL sampling uses a relatively short amount of time compared to other overhead factors. In particular, one can see this overhead in the new-models fine-tuning method. The sampling takes around an estimated time of 525 seconds on average. This is not so prevalent when comparing it to the total time of 18-20,000 seconds when fine-tuning using new-models.

The training loss is also important to highlight for the performance of the models. When using AL sampling, the training loss is much higher in the beginning, contrary to using traditional random sampling. This supports the theory that AL picks samples that are "harder" to classify and can provide more value. The new-models method shows steadfast development of training loss, but the same cannot be said about the iterative method. This method has a very irregular loss, in particular with AL sampling, where the loss ends up at zero. This is most likely a sign of overfitting. The reasons for this can be class disparity, samples with low value to the model, or that the samples are very alike and should not be generalized upon. In the real world, this might not be a problem, as the samples in the very end can be classified as lowest echelon, as they are picked last. In a real-world scenario, one would only label samples that provide real value to the model at hand.

Chapter 8

Further Work

8.1 Multiple Classification Classes

In this project, it was decided quite early in the process that a binary dataset for classification would be used. The Norwegian Review Corpus (NoReC) [35] seemed to fit well with the goal of this thesis, in addition to another dataset from Hugging Face [43] that was concatenated with the NoReC dataset. Originally, the NoReC dataset is labelled in dice ratings from 1-6. It would be interesting, for further work, to see how well the Norwegian natural language processing (NLP) models, with active learning (AL), performs with multiple classes.

8.2 Cross Validation

Cross validation is a widely used technique in machine learning (ML) for evaluating model performance and tuning model hyperparameters. When fine-tuning Bidirectional Encoder Representations from Transformers (BERT) language models, cross validation can be particularly useful for several reasons [50].

Firstly, using cross validation can help to reduce overfitting. BERT models contain a lot of parameters, and fine-tuning them on a small dataset can cause overfitting, where the model learns to match the training data too closely and struggles on new, unforeseen data. Cross validation can help by dividing the data into many folds. This will enable training the model on each fold while verifying on the remaining data, and average the performance across all folds. This makes it possible to prevent the model from being too tailored to one subset of the data [50].

Furthermore, cross validation can provide a more precise estimate of model performance. By training and evaluating the model on multiple folds of the data, one can obtain a more robust estimate of how the model is likely to perform on new, unseen data. Given that the performance of BERT models can be significantly influenced by the particular selection of hyperparameters and training data. This

can be especially helpful when fine-tuning BERT models [50].

Finally, cross validation can be useful for selecting the optimal hyperparameters for the BERT model. By testing with different hyperparameters on multiple folds of the data, one may gain a deeper understanding of how different hyperparameters affect model performance. This can help to guide the selection of hyperparameters for the final model [50].

Overall, using cross validation when fine-tuning BERT language models can be a good idea for reducing overfitting, providing a more accurate estimate of model performance, and selecting optimal hyperparameters. It would therefore be a good idea to make use of cross validation if someone is going to further develop the findings in this research report.

8.3 Hyperparameter Tuning/Optimization

Hyperparameter tuning is an essential step in the optimization of BERT models. In order to maximize performance on a specific job, such as text categorization or question-answering, the procedure entails fine-tuning model parameters.

Several studies have shown the importance of hyperparameter tuning for improving BERT model performance. For instance, a study by Devlin et al [13] found that fine-tuning BERT on downstream tasks can significantly improve performance over pre-trained models. Learning rate and batch size are two hyperparameters that one must carefully choose in order to get the best results.

Similarly, Liu et al. [15] found that hyperparameter tuning can significantly improve BERT's performance on a range of natural language understanding tasks. The choice of hyperparameters can significantly affect how well a model represents data.

Additionally, the importance of hyperparameter tuning is understood by the entire ML community. For example, Bergstra and Bengio [51] demonstrated that hyperparameter optimization can significantly improve the performance of deep learning models.

Further study should investigate the effects of various hyperparameters on performance for different tasks, given the significance of hyperparameter tuning for optimizing BERT models. This could involve exploring the effect of different learning rates, batch sizes, and regularization techniques on model performance.

8.4 Comparing Language Models

This research report has only used one language model, NorBERT 2. This was chosen on the basis of the report by Kutuzov et al. [36] and the fact that NorBERT 2 "excels in binary sentiment analysis" [38]. Multiple other Norwegian models were also highlighted that have produced good results, in particular NorBERT

and NB-BERT-Base. The code base used in the project has the ability to use all models from Hugging Face, which both of the aforementioned models are. Due to the time frame and scope of the project, no time and complexity was set aside to test with different models. This is a naive approach, but also a necessity for implementation. It is recommended that others build on the work in this paper and test several different models and find out how a model affects the end result.

8.5 Implementing Parallelism

It is also possible to further develop the research in this article by implementing parallelism in the fine-tuning. This opens up the possibility of using multiple graphics processing unit (GPU)s and cores simultaneously. This can dramatically decrease runtime. Although this is a possibility, the research has other weaknesses and points of improvement than this, which may be more appropriate to work on. If, on the other hand, one were to use a dataset that is considerably larger and or use a model that requires more resources, it may be necessary to use more memory, more cores, and GPUs in the fine-tuning.

8.6 Larger and More Diverse Dataset

The data used for fine-tuning is based solely on feedback on audiovisual media, music, miscellaneous items, literature, products, games, restaurants, stage, and sports. This is a weakness since such feedback is not representative of an entire language. It is possible that the accuracy would have been very low if the language models from this project had been tested against arbitrary sentences that do not originate from a movie review. In that sense, one could argue that the data used in this project is too limited to fine-tune a model that can be used for general Norwegian sentiment analysis. Before drawing conclusions on this, a deeper investigation is required. One possibility is to find or create a more generalized dataset for testing and/or training.

Improved model capacity to capture a wider range of language in addition to sentiment expressions, are two reasons why utilizing a larger dataset for fine-tuning is advantageous. The model performs better on sentiment analysis tasks when it has been fine-tuned on a larger dataset because it can better catch the intricacies of the language used in the relevant area.

Using a larger dataset can help mitigate the effects of bias in sentiment analysis. A study by Kiritchenko et al. (2018) found that sentiment analysis models trained on smaller datasets were more likely to be biased towards the dominant sentiment in the data. In contrast, models trained on larger and more diverse datasets were less susceptible to such biases, leading to more robust and reliable sentiment analysis results [52].

8.7 Number of Samples in Training

When fine-tuning a BERT model with AL, the number of samples added to each training cycle is an important parameter to consider. Adding a higher number of entities per cycle can help to increase the training efficiency. The model can learn more at once. However, it is worth noting that adding a higher number of entities per cycle also requires more computational resources and may lead to slower convergence during training. Therefore, the choice of the number of entities per cycle should be based on a trade-off between performance and computational efficiency. As the primary motive for employing AL is to minimize the labeling requirements during training, it would be worthwhile to experiment with a smaller number of labels added in each training cycle compared to the 200 utilized in this project. This would enable the model to converge faster. Exploring various training sizes could be an intriguing avenue to investigate further.

8.8 Continue Scikit-Learn Solution with *partial_fit()*

As briefly explained in Chapter 4.3.1, a solution using mostly tools from Scikit-learn (Sklearn) was experimented upon. This was halted by a specific factor: the difficulty of iteratively training a model. Sklearn has many tools for training models, where *fit()* is the most popular. This trains the model on the dataset with corresponding labels. The problem with this method in the case of this project is that whenever a "fitted" model receives new data to train on, it discards any earlier fitting of the same model. Therefore, this method could not be used for this project. There is another method which looks promising from Sklearn, which is called *partial_fit()* [49]. This is a method that can incrementally fit on batches of samples and seems to be a solution for using a Sklearn solution. This method is only available on five classification models, but there is absolutely a chance for some of the classifiers to be suitable for the project. If there was not a tight time constraint on the project, this would be valuable to look at. As previously mentioned, the final solution was solely based on the Hugging Face application programming interface (API), but the project group sees the possibility to change this in the future.

Bibliography

[1] F. Tonetti. 'Getting to know chatgpt: Some statistics.' (2022), [Online]. Available: https://blog.invgate.com/chatgpt-statistics (visited on 17/04/2023).

[2] M. R. Anderson. 'Gpt-4 has a trillion parameters.' (2022), [Online]. Available: https://the-decoder.com/gpt-4-has-a-trillion-parameters/ (visited on 17/04/2023).

[3] C. Molnar. 'Active learning: Overview, strategies, and uncertainty measures.' (2020), [Online]. Available: https://towardsdatascience.com/active-learning-overview-strategies-and-uncertainty-measures-521565e0b0b (visited on 17/04/2023).

[4] B. Settles, *Active learning literature survey*, University of Wisconsin-Madison Department of Computer Sciences, 2009. [Online]. Available: https://minds.wisconsin.edu/handle/1793/60660 (visited on 17/04/2023).

[5] V.-L. Nguyen, M. H. Shaker and E. Hüllermeier, 'How to measure uncertainty in uncertainty sampling for active learning,' *Mach. Learn.*, vol. 111, no. 1, pp. 89–122, Jan. 2022, ISSN: 0885-6125. DOI: 10.1007/s10994-021-06003-9. [Online]. Available: https://doi.org/10.1007/s10994-021-06003-9.

[6] M. Oplenskedal, 'Avanserte" maskinlærekonsepter: Aktiv læring,' *Medium*, 2022. [Online]. Available: https://medium.com/kantega/avanserte-maskinl%C3%A6rekonsepyrt-aktiv-l%C3%A6ring-d354fc4e5edf.

[7] M. Goudjil, M. Koudil, M. Bedda and N. Ghoggali, 'A novel active learning method using svm for text classification,' *International Journal of Automation and Computing*, vol. 15, Jul. 2016. DOI: 10.1007/s11633-015-0912-z.

[8] D. Lewis and W. Gale, 'A sequential algorithm for training text classifiers,' *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in Information Retrieval*, vol. 29, Aug. 2001. DOI: 10.1145/219587.219592.

[9] S. Tong and D. Koller, 'Support vector machine active learning with applications to text classification,' *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, Mar. 2002, ISSN: 1532-4435. DOI: 10.1162/153244302760185243. [Online]. Available: https://doi.org/10.1162/153244302760185243.

[10] C. E. Shannon, 'A mathematical theory of communication,' *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[11] T. Danka and P. Horvath, 'ModAL: A modular active learning framework for Python,' available on arXiv at https://arxiv.org/abs/1805.00979. [Online]. Available: https://github.com/modAL-python/modAL.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention is all you need*, 2017. arXiv: 1706.03762 [cs.CL].

[13] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, 'BERT: Pre-training of deep bidirectional transformers for language understanding,' in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. [Online]. Available: https://aclanthology.org/N19-1423.

[14] 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*, 2019. arXiv: 1804.07461 [cs.CL].

[15] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, 2019. arXiv: 1907.11692 [cs.CL].

[16] Scikit-learn, *Accuracy_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html, accessed 2023-04-25.

[17] Scikit-learn, *Recall_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html, accessed 2023-04-25.

[18] Scikit-learn, *Precision_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html, accessed 2023-04-25.

[19] Scikit-learn, *F1_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html, accessed 2023-04-25.

[20] Scikit-learn, *Balanced_accuracy_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html, accessed 2023-04-25.

[21] Hugging-Face, *Trainer_compute_loss*, https://huggingface.co/docs/transformers/main_classes/trainer, accessed 2023-04-25.

[22] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[23] H. Face, *Training performance on gpu with transformers*, https://huggingface.co/docs/transformers/perf_train_gpu_many, accessed 2023-04-25.

[24] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest and A. M. Rush, *Huggingface's transformers: State-of-the-art natural language processing*, 2020. arXiv: 1910.03771 [cs.CL].

[25] Hugging Face, *About hugging face*, https://huggingface.co/about, Accessed on May 2, 2023, Accessed 2023.

[26] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li and M. Sun, 'Graph neural networks: A review of methods and applications,' *AI Open*, vol. 1, pp. 57–81, 2020, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[27] P. Li, J. Yang, M. A. Islam and S. Ren, *Making ai less "thirsty": Uncovering and addressing the secret water footprint of ai models*, 2023. arXiv: 2304.03271 [cs.LG].

[28] A. K. McCallum and K. Nigam, 'Employing em and pool-based active learning for text classification,' 1998.

[29] S. Tong and D. Koller, 'Support vector machine active learning with applications to text classification,' *Journal of Machine Learning Research*, vol. 2, pp. 45–66, 2002.

[30] C. Schröder and A. Niekler, 'A survey of active learning for text classification using deep neural networks,' *arXiv preprint arXiv:2004.06639*, 2020.

[31] J. Ash, C. Zhang, A. Krishnamurthy, J. Langford and A. Agarwal, 'Deep batch active learning by diverse, uncertain gradient lower bounds,' in *Advances in Neural Information Processing Systems*, 2019, pp. 6311–6321.

[32] D. Arthur and S. Vassilvitskii, 'K-means++: The advantages of careful seeding,' in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[33] L. Ein-Dor, A. Halfon, A. Gera, E. Shnarch, L. Dankin, L. Choshen, M. Danilevsky, R. Aharonov, Y. Katz and N. Slonim, 'Active learning for bert: An empirical study,' in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online, Nov. 2020, pp. 7949–7962.

[34] S. Touileb, L. Øvrelid and E. Velldal, 'Gender and sentiment, critics and authors: A dataset of norwegian book reviews,' in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 125–138.

[35] E. Velldal, L. Øvrelid, E. A. Bergem, C. Stadsnes, S. Touileb and F. Jørgensen, 'NoReC: The Norwegian Review Corpus,' in *Proceedings of the 11th edition of the Language Resources and Evaluation Conference*, Miyazaki, Japan, 2018, pp. 4186–4191.

[36] A. Kutuzov, J. Barnes, E. Velldal, L. Øvrelid and S. Oepen, 'Large-scale contextualised language modelling for norwegian,' *Journal of Language Modelling*, vol. 9, no. 1, pp. 163–218, 2021.

[37] A. NORLM, *Norwegian large-scale language models*, Available at http://wiki.nlpl.eu/Vectors/norlm, [Accessed 8.12.2022], 2022.

[38] A. NORLM, *Norbert: Bidirectional encoder representations from transformers*, Available at http://wiki.nlpl.eu/Vectors/norlm/norbert, [Accessed 8.12.2022], 2022.

[39] SchedMD. 'SLURM overview.' (2023), [Online]. Available: https://slurm.schedmd.com/overview.html (visited on 01/05/2023).

[40] High Performance Computing Group, NTNU, *IDUN: A National Infrastructure for High-Performance Computing*, https://www.hpc.ntnu.no/idun/, Accessed 2023.

[41] NVIDIA. 'Nvidia ampere architecture in-depth.' (2020), [Online]. Available: https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/ (visited on 01/05/2023).

[42] NVIDIA. 'Cuda gpus | nvidia developer.' (2023), [Online]. Available: https://developer.nvidia.com/cuda-gpus (visited on 01/05/2023).

[43] A. Sepidmnorozy, *Norwegian sentiment*, Available at https://huggingface.co/datasets/sepidmnorozy/Norwegian_sentiment, 2021.

[44] Language Technology Group (University of Oslo), *NorBERT2: Multilingual BERT for Norwegian and Other Languages*, https://huggingface.co/ltg/norbert2, Feb. 2021.

[45] Simen Tvete Aabol, Marcus Klomsten Dragsten, *Norwegian text classifier optimization with active learning*, https://github.com/simenaabol/Norwegian-text-classifier-optimisation-with-active-learning, Accessed on May 11, 2023, Accessed 2023.

[46] Scikit-learn, *Pipeline*, https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html, accessed 2023-05-05.

[47] Scikit-learn, *Logisticregression*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, accessed 2023-05-05.

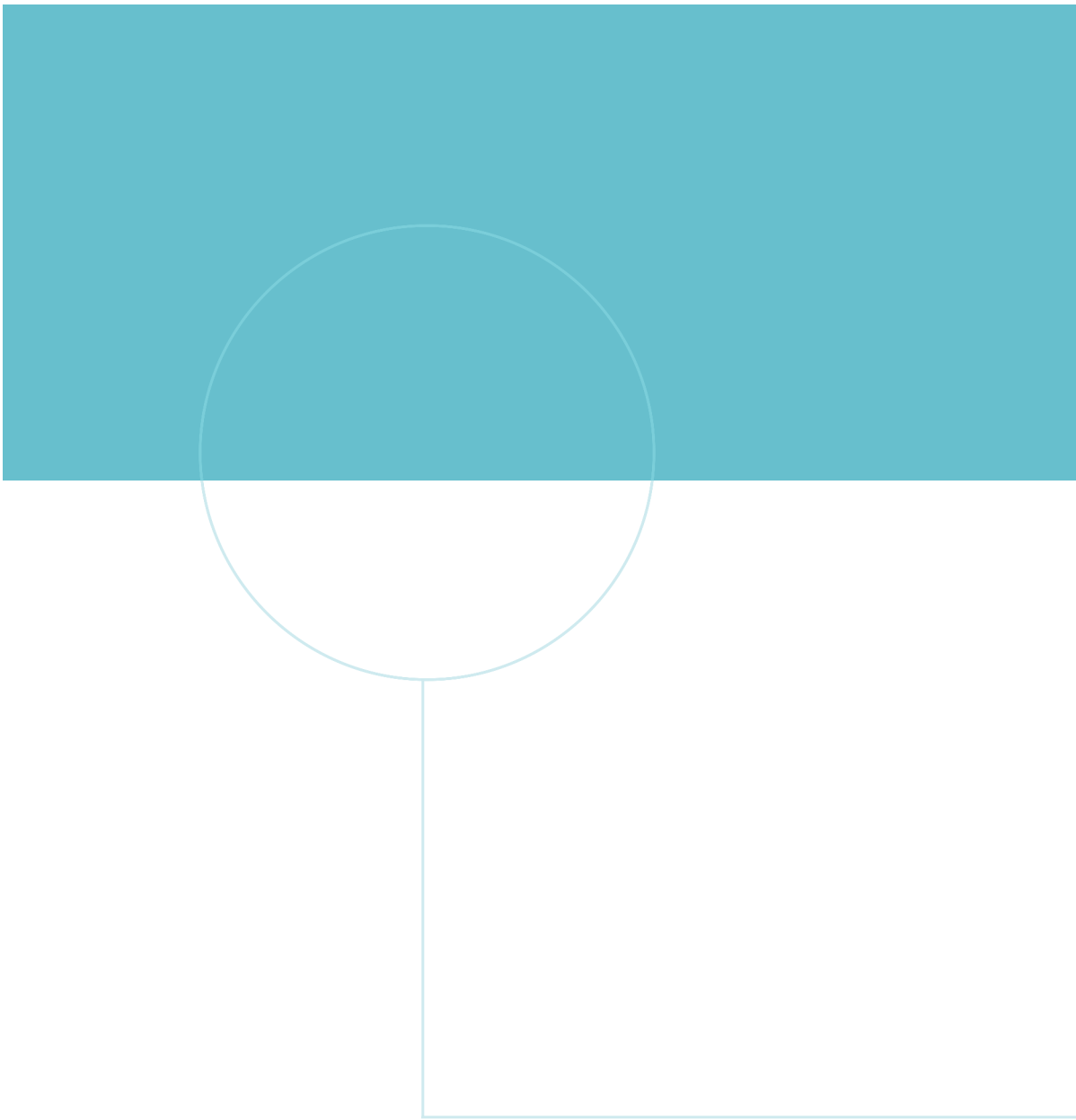
[48] Scikit-learn, *Hugging face transformers with scikit-learn classifiers*, https://huggingface.co/scikit-learn/sklearn-transformers, accessed 2023-05-05.

[49] Scikit-learn, *Strategies to scale computationally: Bigger data*, https://scikit-learn.org/0.15/modules/scaling_strategies.html, accessed 2023-05-05.

[50] J. D. Kelleher, *Deep Learning*. The MIT Press, Sep. 2019, ISBN: 9780262354899. DOI: 10.7551/mitpress/11171.001.0001. [Online]. Available: https://doi.org/10.7551/mitpress/11171.001.0001.

[51] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization,' *J. Mach. Learn. Res.*, vol. 13, no. null, pp. 281–305, Feb. 2012, ISSN: 1532-4435.

[52] S. Kiritchenko and S. M. Mohammad, 'Examining gender and race bias in two hundred sentiment analysis systems,' *CoRR*, vol. abs/1805.04508, 2018. arXiv: 1805.04508. [Online]. Available: http://arxiv.org/abs/1805.04508.



NTNU

Norwegian University of
Science and Technology