Deep Learning Algorithms for Mixed-Classification Sentiment Analysis in Norwegian Text

ISAK KILLINGRØD, JON INGVAR JONASSEN SKÅNØY

SUPERVISOR
Mohammad Arif Payenda

University of Agder, 2024
Faculty of Engineering and Science
Department of Information and communication technology

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

We'd first and foremost like to thank our supervisor Mohammad Arif Payenda for introducing us to the world of NLP research and giving us a fascinating direction to go in, as well as interesting ideas to add to our implementation. Due to his guidance, we feel this project has significantly deepened our understanding of many key topics within machine learning, and it has been a very fulfilling experience.
In addition to this thesis, we intend to write about our project as a research paper to submit for an international conference.

Abstract

The field of natural language processing contains many methods for addressing tasks in every subfield, for several different languages with varying amounts of prior literature. This thesis intends to present further results for the specific subdomain of mixed-label Norwegian sentiment analysis, which is less represented in prior research than other languages. In this thesis, we investigate performance on sentiment analysis datasets involving and not involving the mixed label, and apply a range of model types including GRUs, BiGRUs, LSTMs, BiLSTMs, CNNs, transformers trained specifically on Norwegian, and ensembles of these models to see how performance can be improved on the datasets.
We test the impact of using embeddings from a BERT-based model trained on Norwegian as input for RNNs and CNN, and test across both pre-existing datasets and relabeled versions we've manually verified the labels of. We report the average precision, recall, accuracy and F1 score of each model and the more performant ensembles.
Our results consistently show lower performance on datasets that include the mixed label due to increased complexity and increased challenges in dataset creation. We report disproportionately worse performance on mixed-label datasets for RNNs compared to other models, and ensembles performing better than singular models with significant diminishing returns as the number of models in an ensemble increases. We find that a T5-based model performs very well, but multiple T5-based models in an ensemble gives marginal performance increases compared to other models due to this model being more uniform in its outputs when trained on the same hyperparameters . We also find that ensembles using multiple types of models together gives more efficient performance increases than ensembles consisting exclusively of multiple models of the same type, due to increased diversity in outputs. We present model combinations to use for ensembling on datasets with and without mixed-label data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background And Motivation

The field of sentiment analysis within natural language processing is a rapidly growing field with much practical utility. Systems that predict sentiments within text have many potential use cases in collecting useful information without needing to use time-consuming and prohibitively inefficient manual methods. It is a long-standing field with many previous studies applying most machine learning algorithms in different and interesting ways, including both feed-forward and recurrent neural networks, ensemble modeling and other classification algorithms. Particularly since the introduction of the transformer architecture has the field of sentiment analysis seen rapid leaps forward in performance.
English has been the most popular language to study within sentiment analysis due to its role as the dominant lingua franca of the world, but this leaves some comparative gaps in the literature for other languages. Norwegian, for instance, is naturally less studied than English due to many factors including much fewer speakers and lesser international relevance. There are therefore several unknowns regarding the performance of various methods when applied to Norwegian, which could warrant investigation.
Our reasons for choosing to research a subfield within Norwegian sentiment analysis are twofold. First, we believe there are significant gains to be made in performance in Norwegian sentiment analysis, and that there are generally underexplored areas where more studies would be beneficial. Our second reason is oriented towards our own learning outcomes. It just so happens to be that sentiment analysis touches on many different interesting fields within machine learning, and we'd like to develop and further our understanding of RNN models, CNN models, transformers, ensemble modeling and particularly language models.
We've chosen to particularly look at mixed-label sentiment analysis due to the field currently being particularly underexplored in Norwegian, and allows us to learn about the impact of the labels used on a dataset on model performance, which is another fascinating topic.

1.2 Problem Statement

We've chosen to define the problem statement we intend to solve as "Investigate the effects of introducing a mixed label into a ternary sentence-level sentiment analysis dataset on performance of common contemporary machine learning models and ensemble models".

1.3 Objectives of The Study

To paraphrase our problem statement, we intend to implement and test several machine learning models and ensembles of models on data that either labels a given sentence as positive/negative/neutral (hereby referred to as "ternary") or positive/negative/neutral/mixed (hereby referred to as "mixed-label"). We intend on trying to optimize the results for both ternary and mixed-label variants of the same dataset by fine-tuning singular models, and by creating various ensemble models and testing those. After these tests, we should have comparable results for both ternary and mixed-label datasets, and can make conclusions regarding the effects of introducing mixed-label data, and which models and ensembles are better suited for either dataset.

1.4 Scope And Limitations

We are limited both by time and computational resources, and there are therefore a limited number of models we can properly fine-tune and a limited number of ensembles we can create while still being thorough. We have chosen to use 7 different singular models, which are NorT5-Large, NorBERT-Large, convolutional neural networks, long short-term memory networks, bidirectional LSTMs, gated recurrent units, and bidirectional GRUs.
We will create ensembles using various combinations of the models previously mentioned, and test those as well.
We will test these algorithms on 4 total datasets. The first two are ternary and mixed-label datasets which are identical save for one of them including the datapoints labeled as mixed whereas the other contains only positive/negative/neutral labels. The other two datasets are similar to the prior two, but we've relabeled them ourselves to verify the accuracy of the labels, and have applied some post-processing to remove duplicates and such. We report the results on the first two datasets to be able to compare results with prior research, and the latter two datasets due to believing the labeling to be more accurate and our post-processing to have beneficial effects on performance.
All final results of models on the datasets will be trained on the same hyperparameters, to ensure they are comparable. We also intend to train each model many times with the same hyperparameters, to ensure that they are consistently performant and not a product of luck.

1.5 Thesis Organization

Chapter 2 - Literature review This section discusses prior literature on sentiment analysis.
Chapter 3 - Methodology The methodology chapter details our process in relabeling the dataset, our chosen algorithms, theoretical knowledge of these algorithms and our reasons for choosing them.
Chapter 4 - Implementation The implementation provides details regarding how we implemented the models in practice, the processes we took to settle on those details, and our most performant hyperparameters for all models.
Chapter 5 - Results And Discussion The results and discussion chapter details all the various results of our experiments regarding singular models and ensembles across datasets, and contains our explanations for these results.
Chapter 6 - Conclusion And Future Work In our conclusion, we look over all our results to propose certain ensembles to use for mixed-label sentiment analysis and ternary sentiment analysis, as well as the limitations of our study and how the models and topics we've researched could be researched further.

Chapter 2

Literature Review

2.1 Overview of Sentiment Analysis

Sentiment analysis is a subfield within natural language processing in which the goal is to accurately determine the sentiment held by the issuer of a statement towards the subject of the statement. When processing text, sentiment analysis can be further categorized into multiple types of problems depending on the type of labelling one wants to predict, if intensity of polar expression is of interest, and whether one wishes to predict sentiments from individual sentences, from paragraphs or from whole documents.
Each kind of sentiment analysis task has its own set of challenges. With sentence-level sentiment analysis, one might encounter problems with missing context that might be crucial to understanding whether a descriptor is good or bad when applied to the subject. When doing document-level sentiment analysis, there are notable practical constraints around the input size a sentiment analysis model can take. The popular BERT class of models is only capable of receiving 512 tokens, which might not be sufficient for many documents.[6]

2.1.1 Sentiments

A sentiment is a classification of a writer's opinion, which gets simplified to fit within one of a few classes. The most basic form of classification is a binary classifier, in which all input data can only be categorized with one of two labels, typically positive and negative. A functioning binary classifier would consider a sentence such as "This makes me happy" to be positive, and "This makes me angry" to be negative. However, two classifiers is often not sufficient for capturing all nuance in real-world data, and models can be taught to recognize further nuances with text. Ternary classification is the next most basic form of classification next to binary. The most common labels used are positive, negative and neutral. The addition of neutral as a label enables the models to recognize that not all statements are polar expressions with a sentiment.
There are still more complexities in natural language that cannot be clearly labeled as positive, negative or neutral, and introducing another label to capture the nuance might have practical benefits. For some use cases it might be too reductive to classify a sentence such as "This has many benefits, but also some notable drawbacks" as either positive, negative or neutral. To classify these more complex multipolar statements, a "mixed" label can be introduced to indicate the presence of both positive and negative sentiments within the same statement. [13]

2.1.2 Embeddings

Neural networks generally do not perform well on plaintext as input data. Plaintext and plain language are not encoding information in a way that can be easily understood by a neural network, as very similar input strings can have very different meanings.

An embedding algorithm is a common way of converting the plaintext to a format which is better suited for a machine learning model to learn to interpret. Early embedding methods such as word2vec embedded input words as multidimensional vectors in which the vector's dimensions relative to other word vectors indicate its meaning.[12] A more recent embedding method is contextual embedding in BERT[6], which instead of words processes sub-word tokens for its embeddings.

2.1.3 Tokens

The first requirement of an embedding algorithm is that the input natural language is converted to numbers before it is passed to the embedding. These numbers are called tokens, which represent the sub-words that together make up the words in the natural language data. Each token is a number that represents a portion of a word or other piece of information from the text, such as a particular token indicating the beginning or end of a sentence. A tokenizer, the program converting language to input tokens, has a predefined limit on how many unique tokens it can process, called its vocabulary size. In the paper introducing BERT, the WordPiece tokenizer was used, with a vocabulary size of 30,000 tokens[6]. In more recent papers focusing on the Norwegian language, the most notable transformers NorBERT and NorT5 use a vocabulary size of 50,000.[22]. In addition to encoding textual information, a tokenizer can mask some of its input tokens so a model can attempt to predict the correct ones that ought to be there during pre-training.

2.1.4 Cohen's Kappa

Cohen's Kappa is a metric to evaluate inter-annotator agreement in NLP, but can also be used in many other situations. It is a normalized score between -1 and 1 where 0 is random chance and 1 represents perfect agreement. The measurement is used when there is no objective truth. For example there is no definition of when exactly a sentence contains a positive sentiment, natural language can be interpreted in many ways, so we need some measurement of how much different annotators agree with each other. To measure Cohen's Kappa the annotators need to annotate a number of data points independently and then apply the Cohen's formula $\kappa = \frac{\Pr(a)-\Pr(e)}{1-\Pr(e)}$. for the interpretation of the results "Cohen suggested the Kappa result be interpreted as follows: values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement"[11]. [5]

2.2 Machine Learning

2.2.1 Dropout

Dropout is a powerful and cost-efficient regularization technique which reduces overfitting[8, p. 258]. The technique consists of removing a stochastically selected portion of the outputs of the hidden units. The model will be trained as if the units whose output are dropped are not part of the model. By doing so, the model can build redundancy in its structure.
If a node is capable of consistently detecting a pattern in the input data of the training data, there is little incentive for the model to learn to identify other indicators of said feature as long as that singular node performs well. Therefore, if the single node encounters data it does not perform well on, it might significantly drag down model accuracy. Due to dropout, the hypothetical single node that predicts a feature is not always present in training, which gives incentive for multiple nodes in the model to learn to recognise the same feature.
In a sense, dropout works in a conceptually similar manner to bootstrap aggregation, (see section 2.3.2) where each possible combination of neurons that are not dropped out each are a sub-model of the whole, and the sub-models are sufficiently independent that they might

cover the classification weaknesses of one another. Through dropout the whole model can generalize better. [8, p. 267]

2.2.2 Learning Rate

The learning rate is a particularly significant hyperparameter in the training process which determines the extent to which model weights are updated at each iteration of the training loop. By scaling down the changes made to weights during the model training, a good learning rate can prevent a model from overcorrecting itself after running into a handful of mistakes, while still allowing each training iteration to make a meaningful difference. If a model makes changes to its weights that are too large, the model might go past the ideal weight and end up with another poor-performing weight that gets put through more excessive weight changes. However, if a model makes insufficiently large updates to its weights, it may get trapped in a local minimum instead of approaching the global minimum, as well as needing many more epochs to learn meaningful patterns.[8, p. 295] There are no fixed rules for selecting a good learning rate, necessitating thorough experiments to ensure model performance is fine-tuned.

2.2.3 Batch Learning

During the training of deep neural networks, calculating the gradient error is very computationally expensive. To reduce the computational cost one can apply batch learning, a learning process involving training the model on a batch of data points at a time and calculating a gradient error of the model on the batch for the model to learn from. Doing so is significantly more efficient compared to calculating gradients for the entire dataset and updating model weights in accordance. The main trade-off of batch learning is that a whole batch needs to be kept within memory at once, which increases the minimum required amount of VRAM to train a model on a GPU. Therefore, batch sizes are often not very large when training large language models due to the size of the model and the space needed for its computations[8, p. 277].
Batch learning can alter the training process of a model in notable ways. By training on batches of data, some create gradients that can go in different directions than the dataset as a whole would, and a large gradient from a batch could cause the model to not converge. To limit the effects of this problem, a lower learning rate can be employed. The ideal hyperparameters for training are therefore impacted by the batch size. The training on individual batches can also help the model generalize itself better by not fitting itself to the whole training data at a time[8, p. 279].

2.2.4 Patience Factor

Model training can often lead to overfitting to the training data as the training epochs goes on, where in addition to patterns that are generally useful for the task it's being trained on, a model also adapts to patterns that are only present in the training data. By adapting model weights in a manner exclusively useful within the training data, a model can reduce its training loss significantly, while performing much worse on unseen data. One way to address the overfitting problem is with early stopping, which is a group of methods for terminating the training process early if a given condition is met. One such early stopping method is to use a patience factor.
When using a patience factor, after each training epoch the model is evaluated on the validation data set to gauge how well it performs outside of the training data. After each epoch, it is noted down whether the model performs better or worse on the patience data. The patience factor itself is an integer threshold, which is compared with a counter. Every time a model performs worse on validation data from epoch to subsequent epoch, the counter

is incremented. Every time the model performs better on validation data from epoch to subsequent epoch, the counter is reset. If the counter becomes equal to the patience factor, the training process terminates early. Higher patience factors result in longer training processes and a higher risk of overfitting, whereas an insufficiently small patience factor might terminate the training before a model converges. [19]

2.2.5 Activation Functions

Within each layer of a neural network, most mathematical operations done are linear. In order to model complex systems, a neural network needs to be able to output very different results for potentially very similar data, and therefore nonlinear operations are a necessity. Neural networks therefore commonly have hidden units, which perform non-linear operations on the outputs of the linear operations in a layer. These non-linear operations are called activation functions, and commonly used activation functions include the rectified linear unit function shown in equation 2.1, which replaces negative values with 0, and the sigmoid function shown in equation 2.2, which compresses numbers into the range [0,1]. By allowing the model to change outputs significantly for small input changes, these functions enable the model to learn complex patterns[8, p. 2019].

$$f(x) = max(0, x) \tag{2.1}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

2.2.6 Foundation Models

Foundation models are a category of model which are trained on a large amount of broad data on a self-supervised task. In the case of foundation models, this is done to train a model with a wide base of foundational knowledge applicable to many tasks in a field. The intention behind these foundation models is then to fine-tune the model to perform well on a related but different task, which would be more efficient than training a model from scratch. These models are particularly prominent in the field of natural language processing, such as BERT[6] and GPT4[16]. These models have been given a vast training corpus of natural language and have trained on tasks that are generally useful for encoding an understanding of the training data in the model, such as masked language modeling. In recent years, we've seen significant improvements in hardware, leading to greater possibilities in regards to the scale of models[3, p. 4]. However, larger models are also more demanding to train, which is an important reason as to why having a pre-trained foundation model that can be further specialized on a task through transfer learning is very appealing for many use cases. These foundation models being trained on self-supervised tasks also allows easier scaling, as data labeling is not necessary for the training of the foundation model, making it viable to train them on billions of words.

2.2.7 Transformers

First applied for the task of language translation, transformers have become a major presence in many machine learning fields, and especially so in natural language processing, where many state-of-the-art solutions are built on transformer models such as BERT[3]. The most notable characteristics of a transformer model are the attention mechanism used for contextual embedding, as well as a division of tasks between converting inputs into context-sensitive model-understood representations of language in the encoders, and making use of these representations for generating output sequences in the decoders. This division of tasks also allows for transformer models to use encoders and decoders based on their use cases, such as BERT using exclusively encoders, GPT using exclusively decoders, and T5 making use of both encoders and decoders.

Attention

Attention is one of the most significant aspects of transformer models, as it allows for a deep understanding of the input language. The input matrix is initially divided into three separate matrices known as key, query and value. The query matrix is multiplied by the transpose of the key matrix to create a matrix where every token has a number associating it with every other token. This matrix is then scaled by dividing each element by the square root of the dimensions of the query and key matrices and apply a softmax function to normalize it. The resulting matrix now represents a series of weights for how much impact each token has on each other token. Next, the weight matrix is multiplied by the value matrix to weight every input token by its various weights, and therefore updates the embeddings of each token by taking into account the other tokens surrounding it[28].

Multi-Headed Attention Multi-headed attention is a modified approach to the attention mechanism, which consists of splitting the key, query and value matrices into a number of separate matrices and run through the self-attention mechanism independently before being concatenated together again. Each group of key, query and value matrices is refered to as a "head", and using multi-head attention allows each head to update the weights of a group of selected tokens in relation to one another without having to look at every single token at a time, which makes it possible for heads to find specific relations in one representation subspace each[28].

Encoders

Encoders consist of a number of layers, where each layer applies multi-layer attention to the layer input, then normalizes and forwards it to a fully connected feed-forward neural network. The output of the FNN is then forwarded to the next layer, whether that be another encoder layer, a decoder layer, or a fine-tuned output layer of some kind.

Decoders

The decoder is necessary for the generation of variable-length output sequences of a transformer. Each decoder layer uses two layers of multi-head attention. The first layer takes in the previous outputs of the decoder, if there are any. These are then the value matrix used for the second layer, which in an encoder-decoder transformer, gets its query and key matrices from an encoder layer. The final value matrix after the second attention section is then normalized and sent to a FNN, whose output is put through either another decoder layer or linear and softmax filters. The output of the final softmax filter are the probabilities of every possible next item in the sequence. The transformer can then either stop the sequence on some stop condition such as sequence length or the presence of a stop token, or the sequence can be passed to the start of the decoder again to keep appending to it[28].

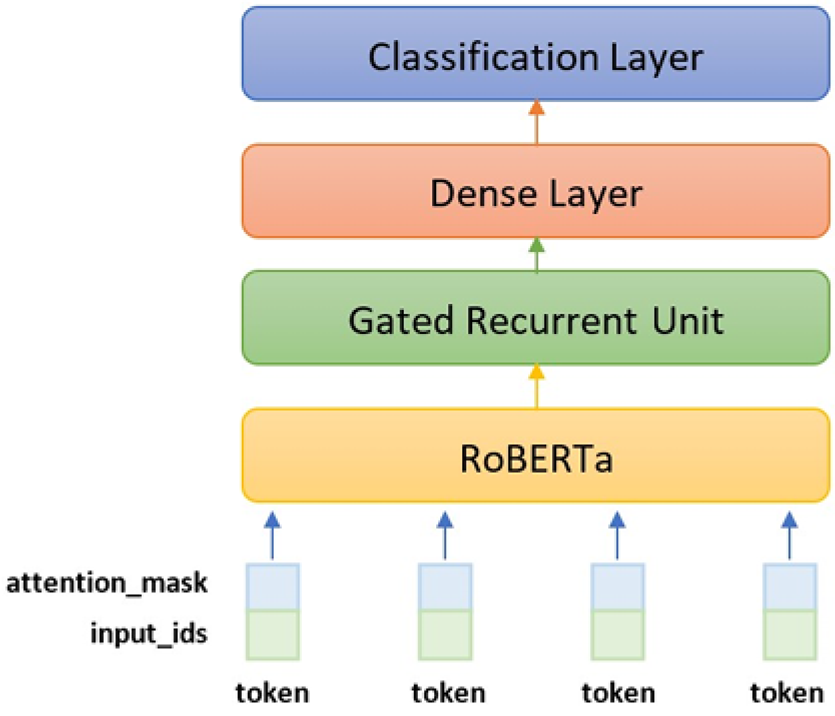
2.3 Previous Studies on Sentiment Analysis

2.3.1 BERT Embeddings

A well-studied approach in English sentiment analysis is to make use of a model based on the BERT architecture as an embedding for other model types, such as the RNN-based LSTM and GRU models, or a convolutional neural network. Several studies have achieved high performance with this approach, including the studies "RoBERTa-GRU: A Hybrid Deep Learning Model for Enhanced Sentiment Analysis"[24], which made use of GRUs, "Sentiment classification with modified RoBERTa and recurrent neural networks"[4], which used RNNs, "RoBERTa-LSTM: a hybrid model for sentiment analysis with trans- former and recurrent neural network"[25], which made use of LSTMs. An example of a RoBERTa-GRU model is depicted in figure 2.1, an illustration from the study regarding the use of GRUs.

[24], [4] and [25] And shown in figure 2.1.

Figure 2.1: Illustration of a RoBERTa GRU [24]



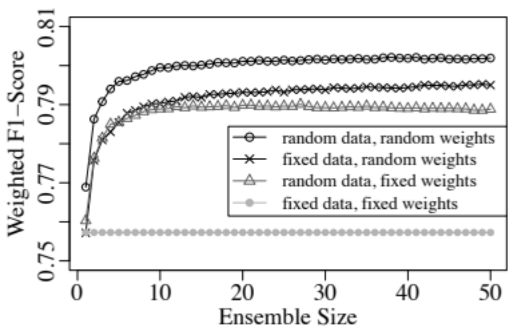
2.3.2 Ensembles and BERT

Ensembling is a common concept for seeing performance increases by involving multiple models. The general idea of ensembling in the case of a classification problem is that there's greater certainty in labels if multiple models agree on the same labeling, and in cases where some models disagree, the majority is the safer bet. Ensembles are consistently good options for use cases where small increases in performance matter much more than large increases in computational costs[8, p. 257].

Many sentiment analysis papers find excellent performance with BERT-based models. Ensembles of BERT-based models have also been studied, as they might have even higher performance scores. A limiting factor in the process of creating ensembles of BERT-based models is that BERT, as an LLM, is very large and has significant computational costs in training and inference. One way to save on computational costs in training the multiple BERTs needed for a model is to extract the base learners for different epochs of the training of a single model. Doing so might allow one to still get the improvements of an ensemble at much lower cost[30].

The change in performance of BERT ensembles on English sentiment analysis over number of models in the ensemble has been previously researched by the paper "Improving BERT Fine-Tuning via Self-Ensemble and Self-Distillation"[30], which found that there were diminishing returns as more BERT models were added into the ensemble, with significantly greater increases in performance in going from 1 model to 2 models in the ensemble than going from 20 to 21 models. The change in performance documented in the study is shown in figure 2.2.

Figure 2.2: Graph Displaying The Impact of Ensemble Size on F1-Score [21]



Bagging

Bootstrap aggregation, or bagging for short, is an ensembling method which involves training multiple models on the same dataset, or different subsets of the same dataset. In instances where subsets of the dataset are used, it's common to not remove previously sampled datapoints when sampling for the next subset. Through the slightly different order of training data and potentially different subset each model is given, the models will learn different patterns to classify data with, and in working together in an ensemble these patterns will on average cover each other's weaknesses and allow the ensemble better performance than the most performant singlular model[8, p. 258].
For these ensembles, there are two main categories of majority voting one can implement. The first of these is known as hard majority voting, where whichever label is considered most likely by the most models is the label chosen by the ensemble. The second voting scheme is soft majority voting, in which each model outputs its certainty for each label given the input data, and the label with the highest average certainty is chosen by the ensemble[21].

Diversity of Models

Due to the principle behind ensembling being that the majority can cover the occasional errors of single models, a key determinant of ensemble performance is the overlap between the output of the errors of each model in the ensemble. If one were to train 10 models which each had 90% accuracy, but the datapoints mislabeled by each model were the same, there would be no benefit to ensembling, whereas if the mislabeled portion was entirely different for each model, the ensemble would be 100% accurate. In practice, there will be some overlap between errors of individual models, but also errors that do not overlap.
If one were to theoretically be able to train a set of models on a 4-class classification problem, and each model were 30% accurate, but each model had minimal overlap in the classification errors, one would approximate 100% accuracy with enough models in the ensemble. Therefore, diversity in the ensemble is considered an important indicator of the performance improvements that can be gained through ensembling[1].

2.3.3 LSTM and GRU on Word Embedding

Typical LSTM and GRU models utilize some kind of word embedding, which does not take into account the context of each word, and instead intends for similar words to have similar embeddings regardless of the preceding and subsequent words. This kind of embedding is expected for LSTM and GRU models[14]. For context-insensitive embeddings, commonly used options include pretrained embeddings like Word2Vec and GloVe word embeddings. These pre-trained embeddings are an appealing choice for many due to the expensive training process of the training a new set of word embeddings on a sufficiently large dataset of millions or billions of words.

These pre-trained embeddings can be used as static embeddings that remain as they were after pre-training, or can be fine-tuned to update in response to the data they are given during training, which can increase performance at a cost of increased computation[14].

Chapter 3

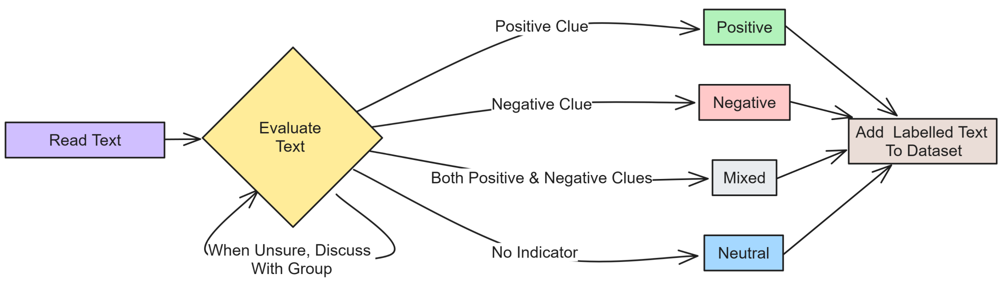
Methodology

3.1 Dataset Creation

At the start of this project, we came upon one untargeted sentence-level sentiment dataset in Norwegian. The untargeted dataset was initially created by looking at the dataset norecfine[18], the targeted sentiment analysis portion of the Norwegian Review Corpus[29], a dataset of labeled newspaper reviews of various products and pieces of media. To create the Norec sentence dataset first used in [9], the targeted sentiments in norecfine were translated to untargeted sentence-level sentiments in the following way: A sentence with one or more instance of a sentiment of a single polarity were labeled as having that sentiment, while sentences without any polar sentiments were labeled as neutral. If both positive and negative sentences were present in a sentence, it would be discarded. Herein lies our main contention, which is that these sentences with both positive and negative polarities are a common part of the language, and there is uncharted value in attempting to classify these more complex sentences.

Starting once again with norecfine, we decided to create a dataset which includes these sentences. While we could try labeling them using the preexisting labels of positive, negative and neutral, we do not believe these labels are appropriate for describing the nuances in the data. We have therefore decided to introduce a fourth label, 'Mixed' to the data, representing sentences with both negative and positive polar expressions. This was our primary motivation for choosing to work with this dataset, as introducing a mixed-label would be comparatively simple, and as the non-mixed dataset has been used for other research papers, we have good performance benchmarks to compare with. We initially downloaded the dataset, and converted each of the four possibilities to an initial label in accordance with the way the previous sentence-level dataset was created, but accounting for the multi-polarity data. Next, we began manually verifying whether the initial label for a given sentence was a good fit, or whether we thought another label fit better, in a process outlined in figure 3.1.

Figure 3.1: Diagram Displaying The Annotation Process



Read Text
Evaluate Text
Positive Clue
Negative Clue
Both Positive & Negative Clues
No Indicator
When Unsure, Discuss With Group
Positive
Negative
Mixed
Neutral
Add Labelled Text To Dataset

Labels	Description
Positive	"There is an explicit or implicit clue in the text suggesting that the speaker is in a positive state" [13]
Negative	" There is an explicit or implicit clue in the text suggesting that the speaker is in a negative state" [13]
Mixed	"There is an explicit or implicit clue in the text suggesting that the speaker is experiencing both positive and negative feelings"[13]
Neutral	"There is no explicit or implicit indicator of the speaker's emotional state" [13]

Table 3.1: Description of Labels

Challenges

When manually reviewing the dataset, we have used the definition of each label shown in the table 3.1. There are many challenges when deciding the label of a sentence, with the most notable challenge being a lack of context. In reality, sentence polarity is often dependent on the sentences that come before or after. For instance, a sentence from the dataset like "Men det merkes og det høres ." (English: "But it's noticed and heard") has its polarity dependent on the following or preceding sentences. If the preceding sentence was something akin to "The corporation producing this product has tried to cut costs, while stating that it would not affect the quality of the product", the sentence "But it's noticed and heard" would imply a negative sentiment towards whatever "it" is, whereas if the preceding sentence were similar to "The corporation producing this product has committed to an improvement in quality, which some might be skeptical to.", the sentence "But it's noticed and heard" implies a positive sentiment. While these sentences indicate the presence of a polar sentiment, the polarity is not clear without knowing preceding or following sentences, which motivates us to label these sentences as neutral. By converting the sentence "Men det merkes og det høres." from the original norecfine dataset to a non-targeted dataset, the sentence originally ended up with a positive label, which is something we disagree with. we have been unable to find any guidelines for the annotation process in norecfine that describes how to handle this lack of context.

Another significant challenge occurs when we encounter sentences of ambiguous sentiment whose context is clear from the sentence itself. In our experience, this challenge is particularly frequent among sentences that can be interpreted as either neutral or positive, as it can be ambiguous whether something is merely a statement of fact or carries a sentiment. For instance, the sentence "Den er rundere i kantene og har en større skjerm ." (English: "It is rounder around the edges and has a larger screen.") is a sentence we consider a statement of facts, but is annotated as positive in norecfine. It would therefore seem that the one annotating norecfine considered a larger screen and/or rounder edges to be a positive trait, and for the sentence to carry a positive sentiment towards the item that had a larger screen. We've disagreed with this and labeled the sentence as neutral due to it only describing objective parts of the design of an object, and yet we must concede that there's room for debate on whether this sentence should be positive or not. A larger screen is something that is considered a direct upgrade, in most cases, and there might be surrounding context in the original review for why the writer might intend the statement in a positive way. The ambiguousness of some sentences is a cause for concern, as annotators might have significantly different ideas on the labeling of ambiguous sentences as compared to others. Therefore, there's some uncertainty as to whether one annotator is teaching a model useful

patterns, or whether their personal biases in annotation leaves patterns in the dataset that would not generalize well, or would not be useful for sentiment analysis applications. A study which corroborates this concern is "A study of inter-annotator agreement for opinion retrieval"[2] by Bermingham et al, which states that there might be cause to believe that sentence-level annotation is too granular to accurately label statements in a manner that can be agreed upon.
In the process of annotating the sentiment we've relabeled a significant amount of sentences for varying reasons. Table 3.2 shows four examples of labels we've changed during the relabeling process, one datapoint we've relabeled to each label. Figure 3.2 shows the total count of labels we've changed from one label to another.

Relabeled sentence	Original Label	New Label	Note
Omar Elabdellaoui 5	Positive	Neutral	The phrase "Omar Elabdellaoui 5" is simply a name followed by a number, which we consider to be a neutral statement. Perhaps the number 5 had a positive connotation in the original context, but that is not clear to us now.
De siste 15 minuttene redder konserten fra total katastrofe , men som Jacob Bannon selv sa uten å mukke : neste gang kommer de tilbake og spiller en liten klubb	Positive	Mixed	The statement "The final 15 minutes saves the concert from being a disaster" implies a negative sentiment towards the concert itself, and that there was a positive sentiment towards the final 15 minutes. Stating the performers will likely perform in a smaller club when they next come back also implies a negative sentiment towards the performance as a whole.
Det er feil , feil og atter feil .	Positive	Negative	Describing something as "wrong, wrong and wrong again" seems to us to indicate a negative sentiment.
« Drøm videre ! » er en av de platene som vokser og blir bedre jo mer du spiller den.	Neutral	Positive	Describing a record as "getting bigger and better the more you play it" seems to us to be a subjective statement that carries a positive intention behind it.

Table 3.2: Examples of Relabeled Sentiments

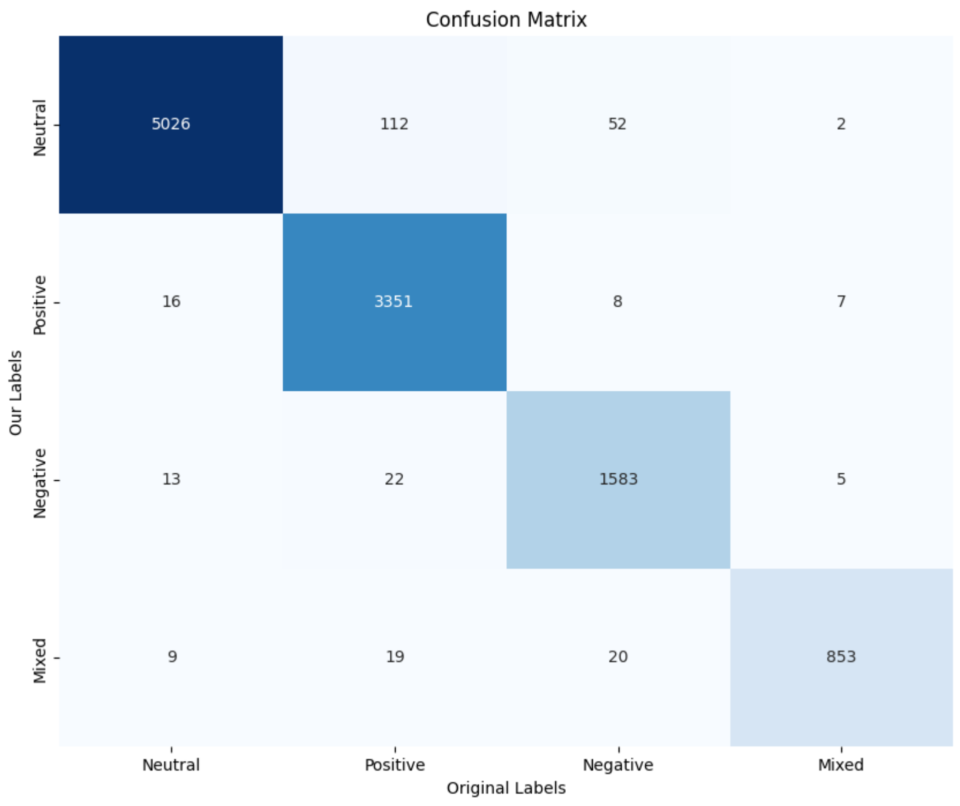


Figure 3.2: Confusion Matrix Displaying the Change in Labeling

3.2 Datasets

3.2.1 Relabeled Datasets

After manually reviewing and relabeling the datasets, we ended up with a mixed-label dataset with a total of 11,098 text datapoints. After updating the labeling with the mixed class in mind, we then created a separate ternary dataset by excluding all the datapoints we consider to be mixed-label in order to be able to compare model results on ternary data as well.
For both the ternary and mixed-label datasets, we have all the exact same number of unique datapoints as the datasets with the original labels. However, we've also removed duplicates from the relabeled datasets and removed datapoints from training data that were also present in test and validation data, resulting in slightly smaller numbers of total datapoints.

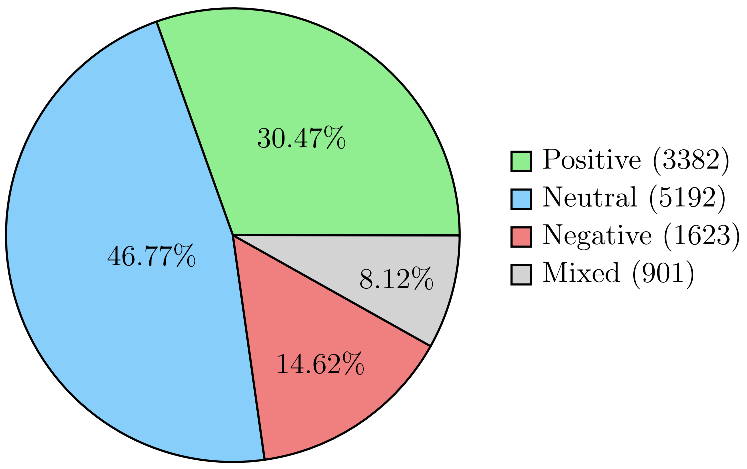
Mixed-label

The relabeled mixed-label dataset has an uneven label distribution as shown in table 3.3 and in the figure 3.3. In particular, the mixed-label consist of only 901 of 11 098 samples this will make learning this dataset more difficult.

Label	Positive	Neutral	Negative	Mixed	Size
Train	2,521	3,869	1,266	682	8,338
Validation	482	689	198	127	1,496
Test	379	634	159	92	1,264
Total	3,382	5,192	1,623	901	11,098

Table 3.3: Label Distribution of the Relabeled Mixed-label Dataset

Figure 3.3: Diagram of Label Distribution of Relabeled Mixed-label Dataset



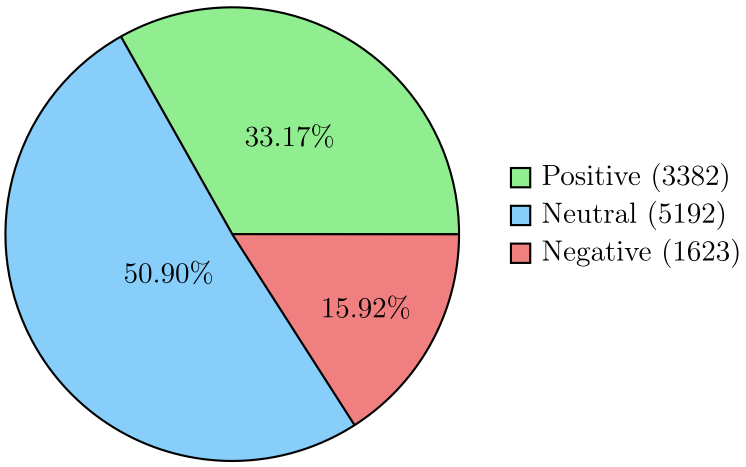
Ternary

The relabeled ternary dataset was created through filtering out sentences with the mixed label from the mixed-label dataset, and therefore includes some sentences we've relabled from mixed to another sentiment, and does not include other sentences we've relabeled from another sentiment to mixed. The resulting dataset distribution is depicted in table 3.4 and visualized in figure 3.4.

Label	Positive	Neutral	Negative	Size
Train	2,521	3,869	1,266	7,656
Validation	482	689	198	1,369
Test	379	634	159	1,172
Total	3,382	5,192	1,623	10,197

Table 3.4: Label Distribution of Relabeled Ternary Dataset

Figure 3.4: Diagram of Label Distribution of Relabeled Ternary Dataset



3.2.2 Original Dataset

At one point after we'd completed our process of data labeling, the Language Technology Group coincidentally also updated their dataset to include a mixed-label variant, by reintroducing the data with both positive and negative labeling from norecfine to norec_sentence, and labeling it with the mixed-label. This dataset is available on HuggingFace at https://huggingface.co/datasets/ltg/norec_sentence.

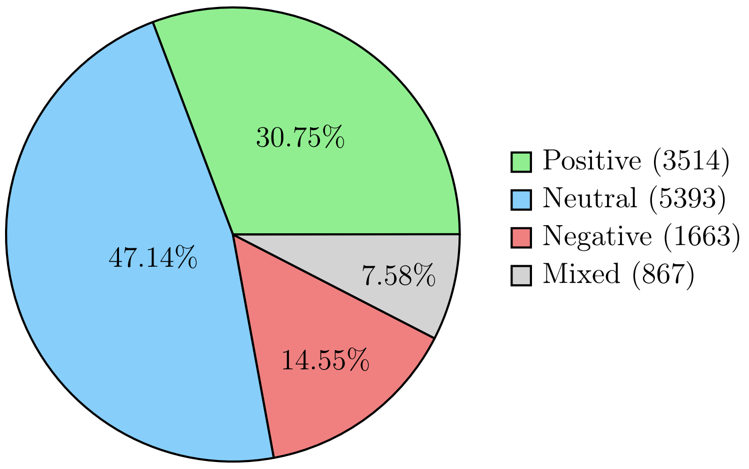
Mixed-label dataset

The LTG mixed-label dataset contains 11.437 sentences with an uneven label distribution and some duplicates. The label distribution is shown in table 3.5 and in the figure 3.5

Label	Positive	Neutral	Negative	Mixed	Size
Train	2,623	4,085	1,270	656	8,634
Validation	490	710	211	120	1,531
Test	401	598	182	91	1,272
Total	3,514	5,393	1,663	867	11,437

Table 3.5: Label Distribution LTG Mixed-label Dataset

Figure 3.5: Diagram of Label Distribution LTG Mixed-labeled Dataset



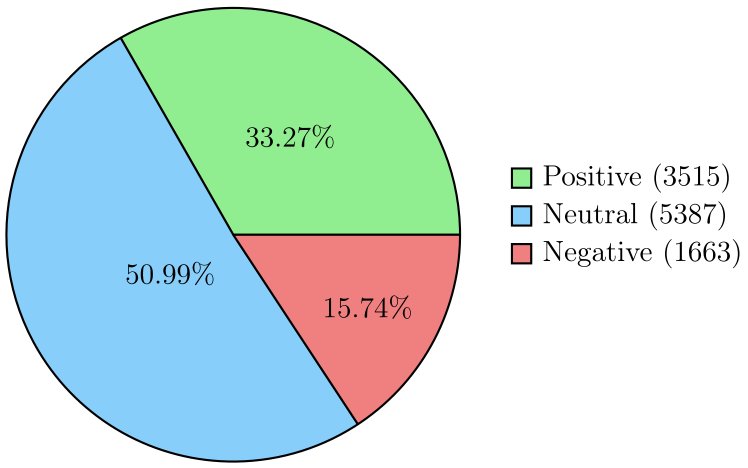
Ternary Dataset

The LTG ternary dataset contains 10,565 sentences with some duplicates and an uneven label distribution. The label distribution is shown in table 3.6 and in figure 3.6

Label	Positive	Neutral	Negative	Size
Train	2,624	4,079	1,270	7,973
Validation	490	710	211	1,411
Test	401	598	182	1,181
Total	3,515	5,387	1,663	10,565

Table 3.6: Label Distribution LTG Ternary Dataset

Figure 3.6: Diagram of Label Distribution LTG Ternary Dataset



3.3 Feature Extraction

In order to extract information from plaintext and convert it into a format that can be understood by NorT5 and NorBERT, we utilize tokenizers. These foundation models are already trained on specific tokenizers that are available through HuggingFace, so the pre-training these models went through to make them useful foundation models is dependent on the input data being tokenized through these tokenizers. A key benefit of these pre-existing tokenizers is that they are specific to Norwegian, which means they are capable of handling the letters that are exclusive to the Norwegian language quite well. All the models we will present make use of the tokenizer for NorBERT-3 and NorT5.

3.4 Model Selection

For the problem of mixed-label sentiment analysis in Norwegian, we will at the end of this thesis propose an ensemble model aggregating the votes of a combination of NorBERTs, NorT5s and NorBERT-embedded CNNs. For ternary sentiment analysis, we will propose an ensemble model consisting of NorT5s and NorBERT-embedded GRUs. In addition to these models, we've also done several experiments involving the use of NorBERT-embedded LSTMs, BiLSTMs and BiGRUs. Due to the comparatively small number of results that exist for these models on Norwegian sentiment analysis, we believe there is value in including them in our method and report our results, even if they were not the most performant models on the datasets.

3.4.1 NorBERT

The first model we settled on testing was NorBERT, as prior literature had it listed as the most performant model for Norwegian sentiment analysis [22]. Our proposed version of NorBERT applies transfer learning to NorBERT-3 Large, using optimized hyperparameters and a patience factor during the training process.

BERT

The BERT model was first introduced in the paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" in 2019[6]. The BERT architecture builds on the generic transformer architecture, with the specific intent of being able to pre-train on large amounts of unlabeled text to act as a foundation model that can be fine-tuned through extra output layers and transfer learning. Since its introduction, it has become a staple in NLP, with many models building on it such as RoBERTa[10] and XLNet[31]. "Almost all state-of-the-art NLP models are now adapted from one of a few foundation models, such as BERT, RoBERTa, BART, T5" to quote the paper "On the Opportunities and Risks of Foundation Models", which looked over the state of foundation models in 2022 [3]. The original model of BERT was primarily trained on English and works best within that domain, but other variants have also been trained on multiple languages to work across several domains, such as mBERT [6]. When working specifically within the domain of Norwegian NLP, however, a foundation model trained in particular on Norwegian is preferable.

Architecture As implied by the name, BERT is a bidirectional encoder based on the transformer architecture[28]. For further information on the transformer architecture, see section 2.2.7. BERT's role as a model consisting exclusively of encoders is primarily meant to generate a language embedding which can be further applied to specific tasks during fine-tuning of the model. For instance, to generate text with a BERT model, one would need to append a decoder to the model, which could make use of the embeddings for good performance on the task.
The bidirectionality of the encoder is a key feature of BERT. When encoding, the attention mechanism will weight each token based on its relation to both prior and subsequent tokens. Therefore, each word can be encoded using context from both the words prior and the words after it in the sequence, which deepens the model's understanding of language.

Pre-Training BERT has traditionally been pre-trained on two particular tasks, the first of which is masked language modeling. As BERT is bidirectional, training it on the typical task of predicting the next word in a sequence would not work well, as it would have access to the word. Therefore, masked language modeling is employed instead, which involves masking words in a sequence and having the model try to fill in the masked words. This task is used with the intention of teaching BERT the relations between words. The other task BERT has been trained on is next sentence prediction, in which the model is given two sentences and has to predict whether the second sentence follows the first or is unrelated to it. By teaching the model to perform well on this task, the hope has been that the model would learn the relationship between sentences. However, this task is no longer in use, as it does not significantly improve model performance[10].

Fine-Tuning Fine-tuning BERT is intentionally simple and computationally inexpensive due to the adaptability of the base foundation model. In order to begin fine-tuning BERT models, one needs to adapt the input and output layers to the specific needs of the task, then begin the training on the foundation model, which is significantly computationally cheaper than training a model from scratch.

NorBERT 3

NorBERT 3 is a set of BERT models specifically trained on Norwegian data, including Norwegian Wikipedia, online libraries, news and other sources, totaling approximately 25 billion tokens. These models were first introduced in the paper "Introducing Norbench"[22], and trained according to a recipe first introduced in the paper "Trained on 100 million words and still in shape: BERT meets British National Corpus" [23].

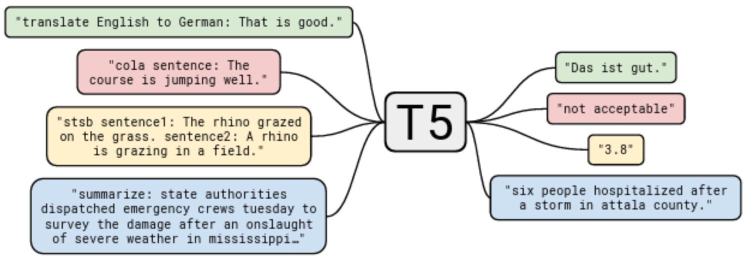
3.4.2 NorT5

Another model we've chosen to use is NorT5 Large, a model first introduced in the paper "Introducing Norbench", where it was shown to have good performance on sentiment analysis[22]. NorT5 is a set of models using the T5 model architecture, trained on the same dataset that NorBERT 3 is. The changes we've implemented when training NorT5 on the task of sentiment analysis include optimization of hyperparameters and a different way of representing the task as a text-to-text task.

T5

T5 is an abbreviation of the phrase Text-to-Text Transfer Transformer. This is a model using the transformer architecture, as an encoder-decoder transformer[20]. By using both encoders and decoders, T5 takes tokens as input and outputs tokens as well. When working with T5, problems must therefore be constructed in a way where both the input and output are text as shown in figure 3.7.

Figure 3.7: Diagram Demonstrating the Capabilities of the Text to Text Framework [20]


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

The differentiating factors between T5 and a generic transformer include a modified attention function and activation function. The encoders are bidirectional, like NorBERT. The decoder however is not bidirectional, and only has access to current and prior tokens.
The pre-training for T5 models works similar to that of BERT models. As with NorBERT, the base training with sentence prediction is no longer done, and masked language modeling with spans is the main task it is trained on. The original T5 models are trained on C4, the Colossal Clean Crawled Corpus, which contains a wide variety of data scraped from the web. Fine-tuning T5 models is relatively simple. As long as the problem it needs to be trained for can be framed as a text-to-text problem, it can be trained to output the correct text with minimal modifications.

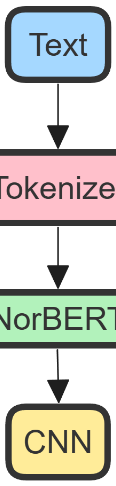
3.4.3 NorBERT-CNN

We propose a model we call a NorBERT-CNN, which consists of a convolutional neural network that is fed the final hidden states of a NorBERT 3 Large model as its input, as depicted in figure 3.8. Input data is first fed through the tokenizer for NorBERT, which has a vocabulary of 50,000 tokens. These tokens are then fed through the encoders in NorBERT to generate contextualized embeddings for the CNN to use to classify the sentiment of the text.

Training

When training the CNN, we have the choice of whether to use NorBERT as a static embedding, which is to use it as-is without training it further on our task, or to train it on the input data alongside the CNN. To use a static embedding would save significant computational costs during training, as there would be no need to update the NorBERT weights throughout the training process. Another factor in the increased computational cost of not using a static embedding is that since the output of a static embedding is the same every time, one would not need to calculate the embeddings every training epoch. For a non-static embedding, the embedding process would give a different output every time the NorBERT weights were updated, causing one to need to redo the forwarding of the input tokens through the

Figure 3.8: Diagram of Our NorBERT-CNN Model



NorBERT each epoch. Training the NorBERT still leads to improved performance, as it is further specialized on the particular task.

CNN Architecture

A convolutional neural network is a machine learning model particularly suited for learning patterns in data where datapoints matter in relation to one another, a set of attributes which makes it popular for various fields such as image processing and natural language processing. Unlike the RNN models discussed previously, a convolutional neural network is a type of feed-forward neural network (FNN), characterized by taking in all inputs at once, and forwarding and processing all inputs simultaneously through hidden layers, where each subsequent layer can be different in weights, neuron counts or activation function when compared to the prior layer. A typical FNN is not commonly used for NLP, as it struggles with data where the key points can be in different input fields, such as if inputs 1 and 2 are "not good", it'd have to learn that those indicate a negative sentiment, whereas it would not be able to apply this knowledge if "not good" were the 2nd and 3rd inputs. Another similar example is that an FNN could learn to classify one image, but would fail at classifying the same image if it were shifted by one pixel. Due to exclusively using matrix multiplication at every layer, the typical FNN cannot consider translational symmetry, which is when data is shifted but ought to retain the same significance to the output. Rather than relying on matrix multiplication exclusively, convolutional neural networks do as their name implies, and uses layers where the input data is convolved with kernels in addition to layers where the input to the layer is subject to matrix multiplication. The use of the convolution operator has many benefits for the network, and is what makes a CNN capable of handling input data with translational symmetry[8, p. 330].

Input Parameters And Kernels A CNN typically uses kernels of weights smaller than the input size for its layer. This is done to limit the amount of outputs to the next layer are impacted by any given input. For instance, ina convolution with a 3x1 kernel, an input might influence up to three outputs of the convolution layer, as there are three positions in the kernel for the input to be multiplied with during convolution. However, at subsequent layers, those three outputs might each impact multiple subsequent outputs, allowing the original input to propagate its influence further in later layers. In practice, this means that each neuron in deeper layers take into consideration more of the input data than neurons in earlier layers, which make them capable of picking up more complex patterns. This indirect influence creates a funnel where earlier nodes pick up and propagate simple patterns to further layers, where they are used together to detect more complex and nuanced patterns. Additionally, having fewer outgoing connections from each neuron allows the convolutional layer to have fewer weights, which has the added benefit of making the model more memory efficient. When a kernel smaller than the input matrix of its layer is convolved with the input, the same input parameter will often be multiplied with the kernel multiple times, with several different weights. This way, every weight in the kernel is applied to multiple input parameters. Through parameter sharing, weights that learn to detect certain features are going to encounter the portion of the input containing said feature if it is present in the input, and output the expected result. In the event that two inputs "not" "good" are shifted by one, the kernel would still calculate the correct connection that it implies a negative sentiment, and the calculation would be output one output index later in the output matrix, and be otherwise unaffected. Therefore, the convolution operator's inherent compatibility with parameter sharing allows for the model to be equivariant to translation[8, p. 330].

Convolutional Layer Structure As well as the convolution itself, there are two other stages to a convolutional layer. The first is an activation function layer, which works similarly to in an FNN. The third stage is the pooling stage, in which the outputs of the activation function layer are subject to some function combining multiple outputs together, creating a new condensed output. An example of a pooling function is max pooling, which involves applying a filter to sections of a set size, such as 2x2, of the activation function output at a time, and taking the maximum value from each section. For instance, with a 2x2 filter, the pooling output would be half the width and height of the activation function output. A key advantage to pooling is that if equivariant output is only shifted by a few indexes, then the output from the pooling layer is entirely unaffected, which allows the convolution layer to have a limited translational invariance. It's also worth noting that the translational invariance being restricted somewhat is not necessarily negative in NLP, as phrases might have somewhat different intents when used at the start of the phrase compared to the end[8, p. 330].

3.4.4 NorBERT With RNN Variants

Similarly as with CNNs, we have chosen to use NorBERT for embeddings in our implementations of various RNNs. Doing so allows us to leverage the utility of contextualized embeddings together with RNN models' ability to process time series data.
We've chosen to test four separate RNN models that we embed with NorBERT. These are LSTMs, BiLSTMs, GRUs and BiGRUs. Our reason for choosing to use NorBERT embeddings for these models is that there have been reports of performance improvements in papers regarding BERT-embedded LSTMs[25] and BERT-embedded GRUs[24]. We've also chosen to implement bidirectional models of both models for the sake of potential performance increases, further comparison points, and in part due to the relative ease of implementation when we already have the non-bidirectional versions implemented.

3.4.5 LSTM

LSTM, Long Short Term Memory networks, are a commonly used subcategory of RNNs, recurrent neural networks. Recurrent neural networks are a type of neural network commonly used for processing time series data by receiving inputs sequentially and processing them one by one. Each input in a sequence is used for calculating an output, which is then weighted and forwarded to the next layer in the RNN. There, the prior output is used alongside the current input for calculating the next output, which is forwarded on. Thus, the previous pieces of data in the time series are used for additional context in each subsequent layer. The additional data from prior inputs is known as the hidden state of the model. [8, p. 410]

For NLP purposes, sentences can be considered time series data, as a sentence is a sequence of words, and the meaning of words are often contingent on the prior words. However, a typical RNN is limited in key ways, including being subject to the vanishing/exploding gradient problem, where depending on how the output of a layer is weighted when forwarded, it will over time either vanish in comparison with more recent inputs by being reduced to being infinitesimally small through many multiplications, or become so large as to overshadow new inputs. This problem has a greater impact on an input's effect on the final model output the longer the following remaining portion of the time sequence is, as the output from the input is passed through many weights. For the case of a vanishing gradient, this would mean that earlier inputs are essentially forgotten after a large number of following inputs, which is impractical for many use cases one would want to apply RNNs to, including sentiment analysis, due to the potentially tremendous influence of prior words.[8, p. 410]
The difference between an LSTM and an ordinary RNN lies in the memory cell mechanism, which is an attempted solution for the vanishing/exploding gradient problem. The memory cell keeps track of a memory separate from the hidden state called the cell state, intended to remember mostly key features without the distortion caused by the vanishing/exploding gradient problem.
When a new input in the time series is entered into the LSTM, it is like in a typical RNN inputted alongside the hidden state, however there's a third input value as well, which is the memory for the cell state.

Forget Gate

The first step of the memory cell is the forget gate, which determines what portion of the cell memory should be discarded, typically using a sigmoid activation function on the weighted sum of input and hidden state. [8, p. 410]

Input Gate

The next gate of the memory cell is the input gate, which determines how the information from the current input and hidden state should impact the cell state. In one portion of the input gate, the values to add to the cell are calculated by weighting the input and hidden state and adding them together, then passing them through a tanh function to normalize them in the range of -1 to 1. In another part of the input gate, the input and hidden state are weighted and summed with a different set of weights, then passed through a sigmoid function to normalize the values in the range of 0 to 1, which represents percentage weights for the values that are to be added to the cell state. The vector to be added to the cell state and its weight vector are combined into their Hadamard product, and then added to the cell state.[8, p. 411] Output Gate At the end of the memory cell is the output gate, which calculates the next hidden state for the next input. It gets this next hidden state by first sampling the cell state and passing it through a tanh function to normalize it. Then the gate uses the current hidden state and input by weighting them and adding them together, then passing the sum through a sigmoid function to create a vector of weights for the output of the tanh. The outputs of the sigmoid and the tanh are then combined into their Hadamard product, becoming the hidden state for the next layer.[8, p. 411]
At the final layer, the final hidden state output by the model is what determines the predictions of the model.

3.4.6 BiLSTIM

A bidirectional LSTM is a modified LSTM model capable of using the context of prior data to evaluate current data, like an ordinary LSTM, and also capable of taking into account all the remaining data in the time series for the same evaluation. In NLP, this would mean that a BiLSTM can account for not only the previous words, but also all the words that will follow at each step. This is achieved by creating two separate LSTM models, one of which handles the time series from the earliest datapoint to the latest as usual for an LSTM, while the other handles the time series in reverse. At the end, each model has a separate output vector, which are then combined to form the final prediction of the network.

3.4.7 GRU

Similarly to an LSTM, a GRU is a subtype of recurrent neural networks, meaning it is built for processing time series data in order and passing a hidden state through each layer of the model. The distinguishing feature of a GRU is the way it processes the hidden state at each layer, and calculates the next one. While a GRU only keeps track of a hidden state, the hidden state is treated somewhat similarly to the cell state of an LSTM, in that it is weighted dynamically using the prior hidden state and current input. As indicated by the name "gated recurrent unit", each layer of a GRU has gates with separate functions, which are applied sequentially to the hidden state and input for calculating the output. [8, p. 412]

Reset Gate

The reset gate serves a similar function to the forget gate in an LSTM. This gate is responsible for handling the updating of the prior hidden state, and calculates what percentages of the hidden state vector that should be forgotten. It accomplishes this using the input and prior hidden state, weighting them and passing them through a sigmoid activation function to create a vector of values ranging from 0 to 1, which gets multiplied with the hidden state vector. By doing so rather than weighting the hidden state the same every iteration through the network, information in the hidden state is retained until the model chooses to forget it, rather than prior information decaying by being weighted constantly. A properly trained reset gate will then forget irrelevant information while retaining important context regardless of how many time steps ago the important context was input, which is an important factor in how a GRU handles the vanishing/exploding gradient problem.[8, p. 412]

Update Gate

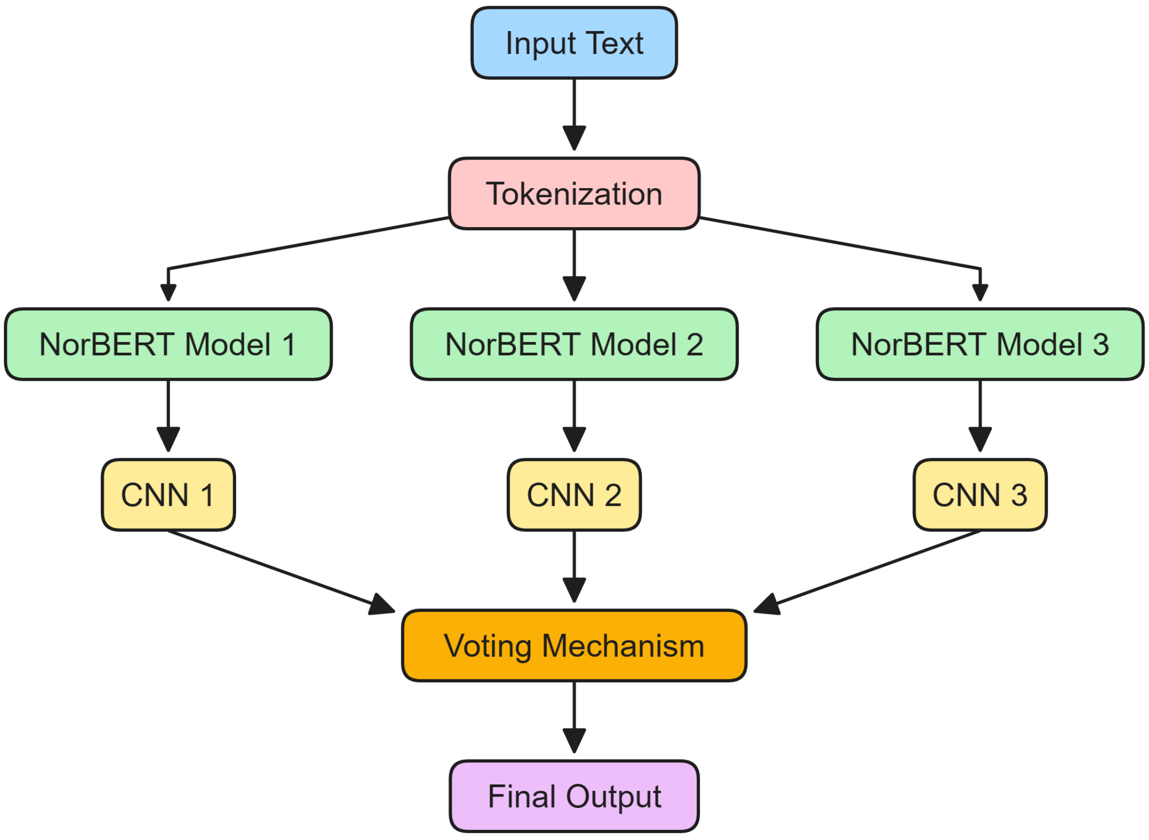
The function of the update gate is to append new information to the hidden state, and it does so in two parts. First, it calculates the percentage significance of new information by passing the prior hidden state and the current input through weights and a sigmoid function which gives a vector for update values ranging from 0 to 1, which is then used to scale the prior hidden state further, and scale the information to be added further.
Second, output of the reset gate and the input are then weighted and passed to another activation function, such as tanh, to normalize the new information in a sensible range. This new information is then used alongside the vector for update percentages, and the final new information is the Hadamard product of these two vectors.
Then, the prior hidden state is used alongside 1 - the vector for update values, and the Hadamard product is calculated. Then, the two Hadamard products calculated have been multiplied by the significance vector or its inverse, so they add up to 1, or 100% significance. The final step is then to combine these two vectors with addition to create the final hidden state, which is the output of the layer, or used for prediction at the final layer.[8, p. 412]

3.5 Ensemble

In addition to the implementation of single models, we've also implemented ensembles to potentially increase performance, at a cost of lessened interpretability and increased computational costs as shown in figure 3.9.

For our ensembling, we have chosen to aggregate the outputs of several models together using a soft majority voting scheme. The underlying idea is that we allow models to cast votes proportional to how confident they are in a label, and choose the label with the highest amount of total confidence among our individual models. Through this voting scheme, the hope is that the more confident models will determine the label for each datapoint and cover each other's weaknesses, improving performance. We make use of both ensembles consisting of multiple individual models of the same algorithm, and ensembles consisting of multiple different algorithms.

Figure 3.9: Aggregation of Votes for Ensemble Model of Three NorBERT-CNNs



3.6 Model Evaluation Metrics

To estimate the performance of a model, points of comparison are necessary in order to know whether something is an improvement or a degradation. Therefore, models are evaluated using several different metrics dependent on the problem they are intended to solve. The standard approach in machine learning research is to divide the dataset in three subsets, a training subset, a validation subset and a test subset [8, p. 104]. The training data is typically the largest subset, and is given to the model for it to be trained on. Most models are capable of excellent performance on the training data they are given, and therefore the other two datasets are used to test model performance on data it has not seen before, to gauge generalizability. When training a model, one analyzes its performance on the validation set to estimate whether the model generalizes well or not. Typically, the model that performs best on validation data is chosen for further testing. Later in the process, the generalizability is verified further with the test dataset, to ensure that the model wasn't trained on hyperparameters that happened to perform well on the validation by chance[8, p. 120].

3.6.1 Confusion Matrix

A confusion matrix is an illustration of the accuracy of the predicted labels of a model on a dataset, compared with the correct labels. Along one axis are the true labels and along the other are the labels the model predicted. A confusion matrix typically shows either the absolute number of datapoints that fit into a category (e.g. "17" in the overlap of Actual Positive/Predicted Negative to show that the model incorrectly classified 17 positive datapoints as negative), or the percentage of each true label that was assigned each label by the model (e.g. "5%" in the overlap of Actual Positive/Predicted Negative to show that the model incorrectly classifies 5% of positive data). Table 3.7 shows an example confusion matrix for a binary classifier[26, p. 209].

	Predicted True	Predicted Negative
Actual True	True Positive	False Negative
Actual Negative	False Negative	True Positive

Table 3.7: Example Confusion Matrix For a Binary Classifier

Confusion matrices are not necessarily restricted to two classes. Each axis can be extended with as many labels as needed. With our use cases, we will create confusion matrices with either three or four classes, depending on whether models are trained on ternary or mixed-label data. These matrices allows one to tell at a glance which labels cause models more problems and which classes are difficult to differentiate.

3.6.2 Precision

Precision is an evaluation metric used in classification problems for describing how often the model is correct when predicting a datapoint is of a given label. There's an individual precision score for each class, which is the percentage of each label applied by the model that is a true positive. If a model labels 100 datapoints with label 1, and 90 of them were meant to be labeled with label 1, the precision score would be 0.9 for label 1. Precision is given with formula 3.1 [27, p. 780].

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{3.1}$$

3.6.3 Recall

Recall is a similar metric to precision, looking at how many of each true label was classified correctly instead of how many of each label assigned by a model were correct. Recall is given with formula 3.2[27, p. 781]. If a dataset contains 100 datapoints with label 1, and a model assigns label 1 to 90 of them, the recall score for label 1 would be 0.9.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.2}$$

3.6.4 F1

F1 score is a metric that combines both precision and recall for a label, weighting them equally. This metric grants an overview of how the model performs on each label, and is a good measure on general performance, as creating a model that performs well on either recall or precision is easier and less useful than a model that performs well on both metrics. The F1 score of a label can be calculated with formula 3.3[17].

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.3}$$

3.6.5 Multi-Class Metrics

F1, recall and precision are all metrics used for binary classification, and calculated for one label at a time. For multi-class classification we need to combine these measurements to a single combined score for each label, and ideally a single score for each model for ease of comparison. For this, we utilize macro-averaging of the metrics, which involves using formula 3.4 to average the F1 scores for every label. Taking the average score of each label has the benefit of ensuring each label is weighted equally when it comes to judging performance, which is beneficial in the event that certain labels are significantly less represented in the dataset than others. Macro-averaged F1 therefore gives incentive for models to learn to classify the less represented labels as well as the overrepresented labels.

$$Macro\ F1 = \frac{1}{N}\sum_{i=1}^{N} F1_i \tag{3.4}$$

Chapter 4

Implementation

4.1 Hardware and Software

4.1.1 PyTorch

PyTorch is a popular python library for machine learning, which has been integral to our implementations. PyTorch offers many important features, including a very useful object type called Tensor. The major benefit of the Tensor object for storing numbers is the built-in compatibility with CUDA for running them on GPUs rather than CPUs, which saves significant time. More significant than the Tensor are the PyTorch modules that enable users to declare and train machine learning models, which we use in some capacity for the majority of the models we've trained. Our choice of PyTorch over similar options like Keras was made due to NorBERT and NorT5 being already implemented in PyTorch, meaning we would already be using PyTorch for some parts of the project.

4.1.2 HuggingFace

In addition to storing our re-labeled datasets privately on the HuggingFace website, several HuggingFace libraries have been very useful for us. For one, the library called "transformers" is a library allowing us to conveniently download and use open source tokenizers and transformer models hosted on HuggingFace including NorBERT and NorT5. These models all come loaded in with a framework compatible with PyTorch, which has been quite convenient. The datasets we've tested on have also been loaded with the library called "datasets", which also contains several useful objects for storing the datasets with useful methods, such as Dataset for storing the datasets and DatasetDict to connect multiple datasets (e.g. separate train, validation and test datasets) in one object.

4.1.3 Hardware

The hardware we've used for training and testing models are our own home computers. The main hardware of note are the graphics cards, which have done essentially all the heavy computation. They are an Nvidia RTX 3090 with a VRAM capacity of 24GB and an Nvidia RTX 4070ti with a VRAM capacity of 12GB. For most of the model training, either card has had sufficient VRAM to train and test any model, with the exception of NorT5 Large, which can be tested on the 12GB VRAM card but required greater than 12GB VRAM to run the training loop. When discussing VRAM requirements, it's worthwhile to note that the models themselves take up only a small portion of the memory, and the rest is taken up by the training dataset, values being forwarded through the model and gradient calculation. Therefore, our memory cost has been greatly reduced due to working on sentence-level data, with a fairly small maximum token length of 90 tokens.

4.1.4 Use of ChatGPT and Copilot

We've made use of both ChatGPT and Copilot for generating portions of the code we've used for the project[15], [7]. Generative AI has not been used in the writing process for our report in any capacity.

4.2 Details of The Machine Learning Models Implemented

4.2.1 NorBERT

The NorBERT model is implemented initially by loading the base model from HuggingFace, after which we begin applying transfer learning. We've tested many sets of hyperparameters for NorBERT through the span of this project, and found the hyperparameters in table 4.1 4.1 to perform quite well.

Hyperparameter	Value
Learning Rate	6e-6
Batch Size	14
Patience Factor	2
Max Epochs	10
Dropout	0.1
Hidden Dropout	0.1
Attention Dropout	0.1
Max Seq Length	90

Table 4.1: Hyperparameters For Fine-Tuning NorBERT Model

Token Length

When deciding the maximum token length for the tokenizer, our main concern was the memory usage on the GPU. The longest datapoint in the dataset had a length of 128 tokens, which seemed like a good initial choice. However, doing so constrained us to using very small batch sizes when training certain models and the average length of a datapoint in the dataset is approximately 23 tokens, meaning that the vast majority of tokens in a given piece of tokenized data are likely meaningless padding tokens. After further testing, we decided to settle on a maximum token length of 90, which allows us to train with higher batch sizes without significantly impacting the final model performance. In total, there are 22 sentences within our 11,098 sentence dataset that are longer than 90 tokens. Truncating these sentences have not caused any noticeable performance changes by itself, which makes the training time improvements from higher batch sizes and smaller input data worthwhile.

Patience Factor

To reduce overfitting to the training data, we've implemented a patience factor for the models. After each training epoch, we check against a stop condition and see whether or not the model has seen no improvement for a certain number of epochs, at which point we terminate the training. When implementing the patience factor, determining the stop condition can have a big impact. Therefore, we've created a set of graphs containing the progression of four key metrics across epochs, to act as a visual guide, se figure 4.1. The graphs shows training loss, validation loss, accuracy on validation set and macro-averaged F1 score on validation set, for a NorBERT model being trained. For each model, we've chosen to save it and overwrite the older version every epoch where it outperforms the previous best score it had on our chosen metric. We've also done some quick, informal experiments on using each metric as a stop condition and getting the average results of models trained with these stop conditions, and have decided that evaluation loss is the metric we'd see better results with, especially when combining models trained using evaluation loss as stop condition into an ensemble. For most of our process, the patience factor we've used is 2, which is to say that a model would stop training after two epochs of the model performing worse in a row.

Learning Rate

To train a model properly, one of the most important hyperparameters is the learning rate, which must be high enough to avoid local minima and not be too high so as to avoid being unable to stabilize on a good solution. With large language models like NorBERT, they will often find some solution when trained with any reasonable learning rate, so we are more concerned about a model potentially overfitting to the training data and generalizing poorly. From our experiments, we have therefore settled on a relatively lower learning rate of $6e^{-6}$, which has performed well. When deciding learning rate, we've also had to take into consideration the batch size. A lower learning rate is also needed when training on smaller batch sizes, such as a batch size of 14, which we chose due to GPU memory constraints.

Scheduler

The HuggingFace library called transformers made it relatively simple to implement a scheduler using the get_scheduler method in transformers. We've chosen a linear scheduler with no warm-up steps, using the AdamW optimizer from PyTorch, and a total number of training steps equal to the number of epochs times the number of batches. The get_scheduler method also allows for other non-linear schedulers, such as constant, cosine, or cosine with restarts. We've done multiple tests with each of these, and found a linear scheduler to work the best.

4.2.2 NorBERT-CNN

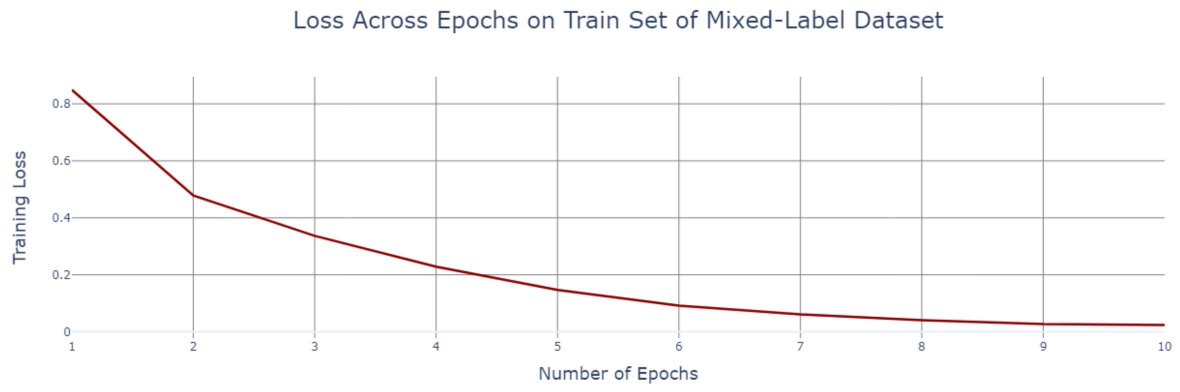
The NorBERT-CNN model is a model making use of a CNN module implemented in PyTorch, which gets fed the output of the embedding layers of a NorBERT model. This process takes advantage of the attention mechanism in the embedding layers of the NorBERT to make the input data fed to the CNN more context-sensitive. Doing so significantly improves the accuracy of the CNN on the classification problem, at the cost of increased model size, time to train and a slightly more complex training process. For the general structure of a NorBERT-CNN, see figure 4.2.

CNN Class

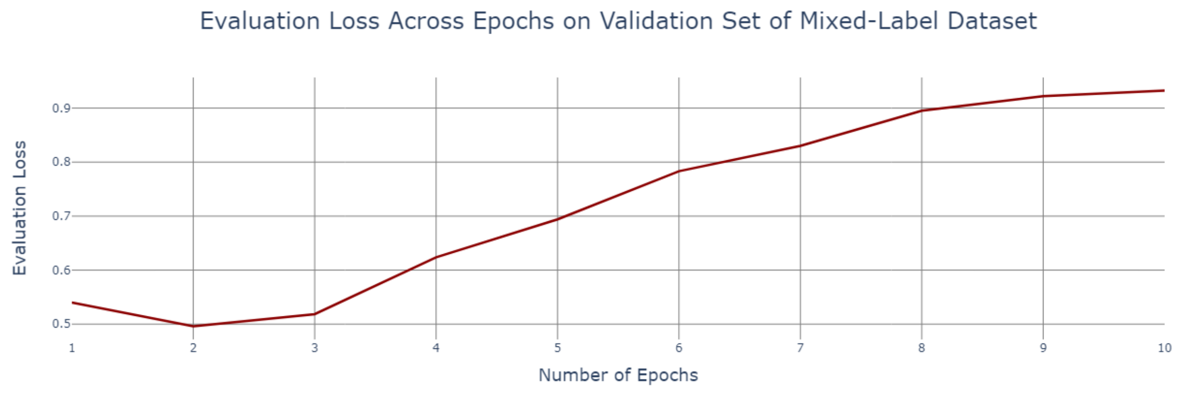
The CNN model is implemented as a custom module in PyTorch. The base of the model is the torch.nn.Module module from PyTorch, with some attributes of its own that use PyTorch functions and wrap them to work with the input data and hyperparameters of the model. The model initializes its list of convolutional filters by creating a nn.ModuleList containing a nn.Conv1d convolutional filter with the correct kernel size and number of out channels from the given hyperparameters. A fully connected layer is implemented as the output of a nn.Linear filter that takes in all convolutional filter outputs and outputs a matrix with four values, one representing each of the possible labels. Dropout is implemented with nn.Dropout.
When input data is fed into the CNN from the NorBERT embeddings, it is forwarded with the model's forward method. The forwarding method applies all the kernels to the input data, then the activation function. Our activation function of choice is ReLU for this model. Next, the pooling phase is applied, in which we use max pooling. After pooling, the outputs are concatenated together into a single matrix, which is then forwarded through dropout

Figure 4.1: NorBERT Model Analysis Over 10 Epochs

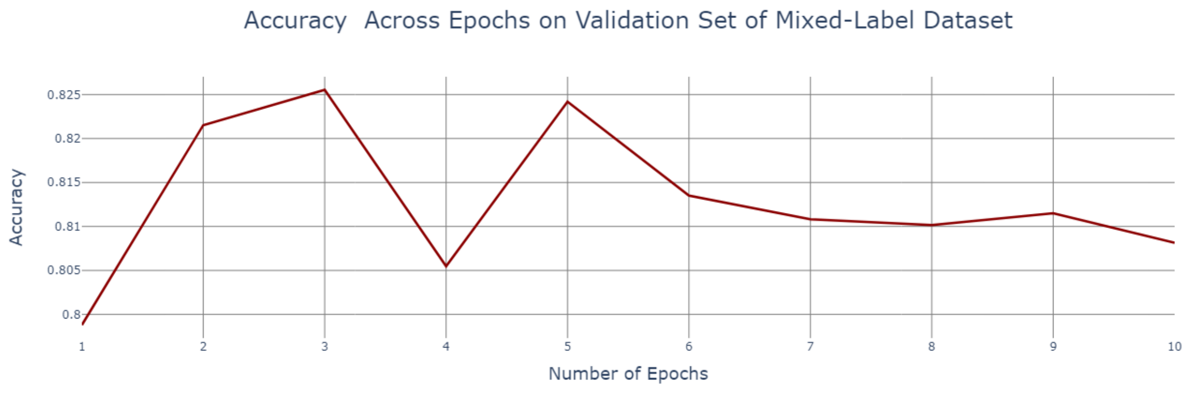
(a) Training Loss



(b) Validation Loss



(c) Accuracy Score



(d) F1 Macro Score

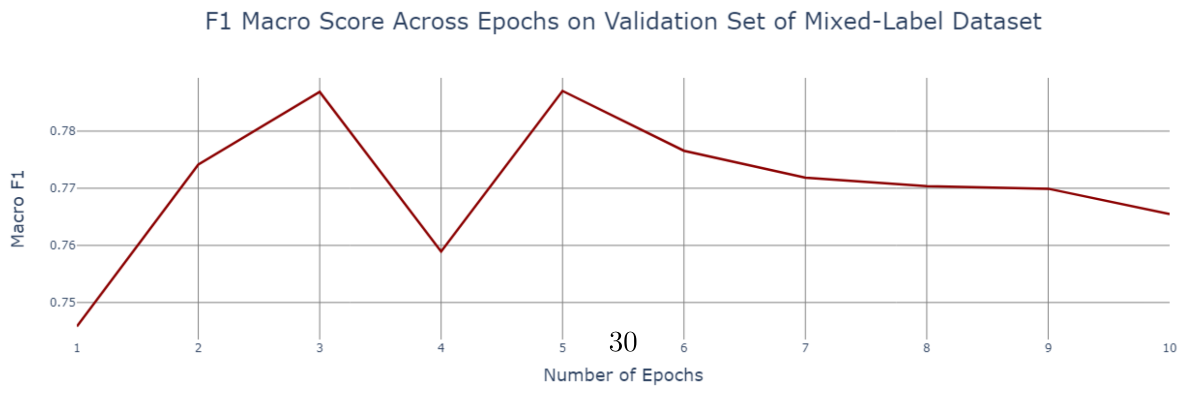
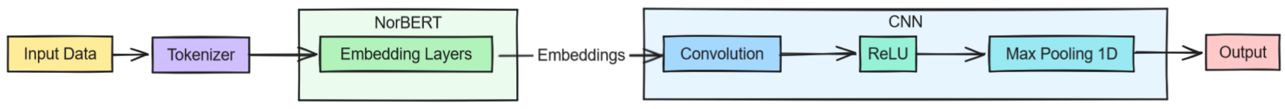


Figure 4.2: The General Structure of a NorBERT-CNN

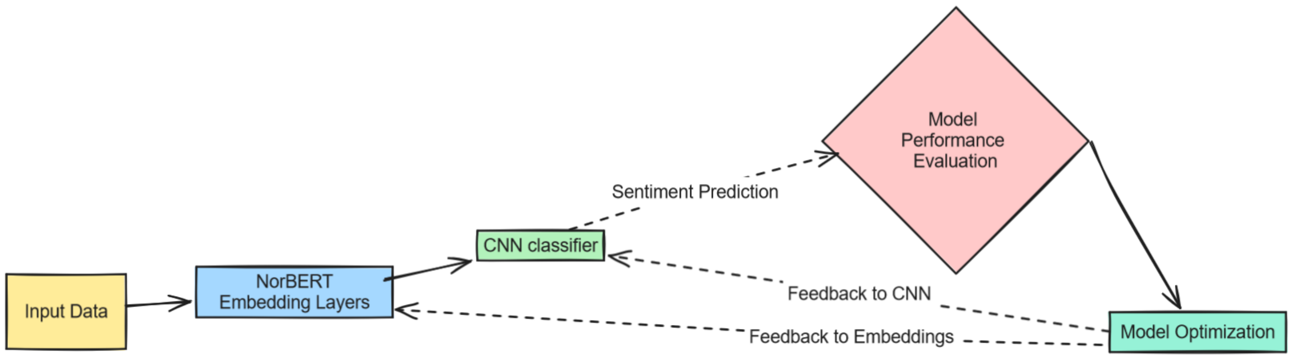


and then the fully-connected layer. The output of the fully-connected layer is the output of the model.

Model Training

As the NorBERT-CNN consists of two separate models, it is a necessity to apply two separate training processes to create a performant model. When implementing these models, we initially did not apply training to the embedding layers, and saw significantly worse results due to the embedding layers not improving. Therefore, we make sure to take into account both the embedding layers and the CNN when updating our model weights, as depicted in 4.3.

Figure 4.3: Training of a NorBERT-CNN



Our final hyperparameters for the NorBERT-CNN are as described in table 4.2

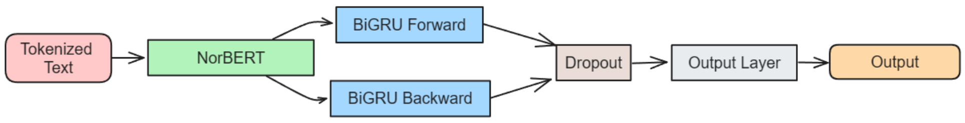
Parameters	**Value**
Max Epochs	10
Patience Factor	2
Batch Size	14
NorBERT Learning Rate	6e-6
CNN Learning Rate	1e-5
Embedding Dimensions (BERT Hidden Size)	1024
Dropout Rate	0.5
Filter Sizes	[2, 3, 4]
Number of Filters	[200, 200, 200]
Scheduler	Linear
Number of Training Steps	10 * Length of Data Loader
Number of Warm up Steps	0

Table 4.2: Hyperparameters For NorBERT-CNN Model

4.2.3 NorBERT-RNN Variants

Similarly to the NorBERT-CNN, we have implemented four different RNN models that have been embedded with NorBERT. For LSTMs, BiLSTMs, GRUs and BiGRUs, we have tried to optimize the hyperparameters for each model, and report our final iteration in tables 4.3, 4.4, 4.5 and 4.6. Each model has been implemented as a class in Python using PyTorch in a similar manner to the CNN class, and they are illustrated in figures 4.4, 4.5, 4.6 and 4.7. Each model receives their input from the NorBERT's final hidden layers, and uses it for their classification. For all four models, we found significantly better results by training the NorBERT as well rather than use it as a static embedding. Separate hyperparameters have been tested for both the RNNs and the NorBERT to find sets of hyperparameters that work well together.

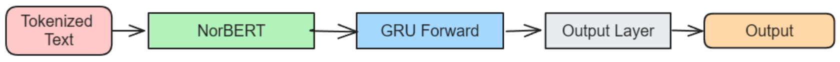
Figure 4.4: BiGRU Model Illustration



Parameters	Value
Max Epochs	10
Patience Factor	2
Batch Size	14
Learning Rate BiGRU	5e-6
Learning Rate NorBERT	4e-6
Embedding Dimension (BERT Hidden Size)	1024
Dropout After GRU Layers	0.3
Dropout Between GRU Layers	0.2
Number of GRU Layers	3
GRU Hidden Size	512
Max Seq Length	90
Scheduler	Linear
Number Training Steps	10 * Length of Data Loader
Number of Warm up Steps	0

Table 4.3: Hyperparameters For NorBERT-BiGRU Model

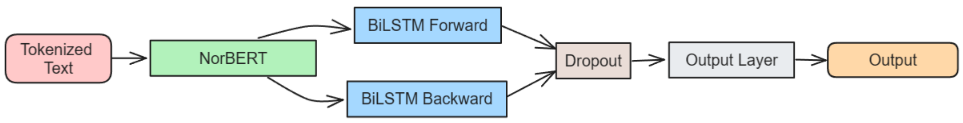
Figure 4.5: GRU Model Illustration



Parameters	Value
Max Epochs	10
Patience Factor	2
Batch Size	14
Learning Rate GRU	6e-6
Learning Rate NorBERT	1e-5
Embedding Dimension (BERT Hidden Size)	1024
Dropout Between GRU Layers	0.3
Number of GRU Layers	3
GRU Hidden Size	256
Max Seq Length	90
Scheduler	Linear
Number Training Steps	10 * Length of Data Loader
Number of Warm up Steps	0

Table 4.4: Hyperparameters For NorBERT-GRU Model

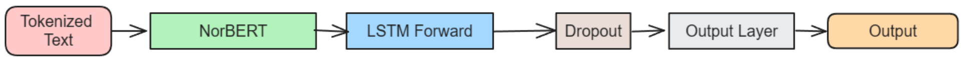
Figure 4.6: BiLSTM Model Illustration



Parameters	Value
Max Epochs	10
Patience Factor	2
Batch Size	14
Learning Rate BiLSTM	5e-6
Learning Rate NorBERT	4e-6
Embedding Dimension (BERT Hidden Size)	1024
Dropout	0.4
Number of LSTM Layers	1
LSTM Hidden Size	512
Max Seq Length	90
Scheduler	Linear
Number Training Steps	10 * Length of Data Loader
Number of Warm up Steps	0

Table 4.5: Hyperparameters For NorBERT-BiLSTM Model

Figure 4.7: LSTM Model Illustration



Parameters	**Value**
Max Epochs	10
Patience Factor	2
Batch Size	14
Learning Rate LSTM	6e-6
Learning Rate NorBERT	6e-6
Embedding Dimension (BERT Hidden Size)	1024
Dropout	0.3
Number of LSTM Layers	1
LSTM Hidden Size	256
Max Seq Length	90
Scheduler	Linear
Number Training Steps	10 * Length of Data Loader
Number of Warm up Steps	0

Table 4.6: Hyperparameters For NorBERT-BiLSTM Model

4.2.4 NorT5

Due to being a text-to-text transformer, the implementation of NorT5 for classification is necessarily different from the others. In order to be able to predict the correct label, we had to recontextualize the problem as a text-to-text problem. To do so, we simply have the model predict a token representing a single number instead, so instead of classifying with integer labels 0/1/2/3, it uses text labels "0"/"1"/"2"/"3". We initially tried having it predict a text label such as "Positive" or "Mixed", but had less success with those predictions due to needing to predict multiple tokens. Predicting only a single token worked quite well due to being able to extract the output confidences for each of the four tokens, therefore being able to include NorT5 in ensembles more easily. Hyperparameters for NorT5, displayed in table 4.7, were found through testing on the validation datasets. It's worth noting that while we did have a patience factor implemented, it to our knowledge never activated and terminated training, so it had no impact on the model.

Hyperparameter	Value
Learning Rate	3e-6
Batch Size	16
Patience Factor	2
Max Epochs	10
Hidden Dropout	0.3
Max Seq Length	90

Table 4.7: Hyperparameters For Fine-Tuning NorT5 Model

4.3 Ensemble

4.3.1 Voting Mechanism

Our chosen method of implementing ensemble classification is through aggregating the output of many individual models to make the classification decision for each datapoint. A classification model outputs a one-dimensional tensor called its "logits", where each value

corresponds to a potential label to apply to the input data it was given. The higher the value for each number in the logits, the more confident the model is that the corresponding label is correct. The lower the value for each number is, which is worth noting can be less than 0, the more certain the model is that a given label does not apply to the input data. The further away from 0 a value is, in either direction, the more confident it is. To determine which label a model wishes to apply to the input data, we simply take the label corresponding to the highest value in the logits.
When creating an ensemble, we run the validation dataset on multiple models that have previously been trained independently, and take the average of each value in the logits for each model to create a set of logits for the whole ensemble. A benefit of the way logits work is that certainty is built into the values, so a less certain model will have less impacton the ensemble decision regarding the label in question than very certain models. Similarly, if there are two models, and the first model is very certain of both label 1 and label 2 being good descriptors, and the other model is ambivalent towards label 1 and certain label 2 does not fit, the ensemble would end up choosing label 1. Doing so allows models to cover for each other's weaknesses.

NorT5: A Special Case

Due to NorT5 being a text-to-text transformer, its output logits have a different shape to that of the other models. NorT5 logits consist of 3 sets of confidences for each of the 50,000 tokens in its vocabulary. Due to how the tokenizer works by adding a start token and an end token, the model expects to output a 3 token long series, where the first value is minimal for all tokens except the start token, and the third value is minimal for all tokens other than the end token. The value relevant for us are the second of the three values, which represents the actual token to be output, and we're not particularly interested in the logits for the 49,996 tokens that do not correspond to a label in the dataset, so we extract the second values of these 4 indices in the logits to create a set of relevant logits for the ensemble to use.

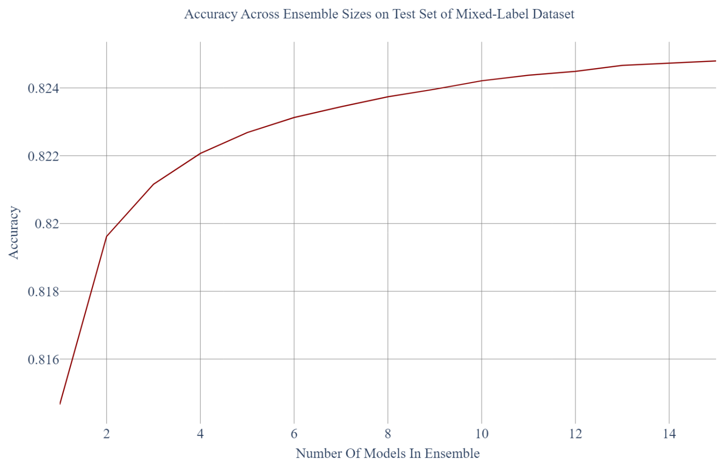
4.4 Experimental Setup

4.4.1 Ensemble Size

When creating an ensemble with multiple discrete models voting, one way to scale its performance is by increasing the number of models you collect votes from. As they all should in theory cover part of each other's weaknesses, increasing model count should have some positive effect on performance at the expense of increased computational cost. However, scaling to a very high number of models would likely be very inefficient, as we expect diminishing returns as more models are added to the ensemble. Taking inspiration from prior experiments done in the paper Bagging BERT Models for Robust Aggression Identification [21], who created a plot showing ensemble performance over number of models in ensemble 2.2, we've decided to try similar experiments for our models.
We have therefore trained 30 models of NorBERT to have a varied pool to pull from, and tested them on the datasets to see how performance improves as more of the same model are added together. For each count of models from 1 to 15, 10,000 different random combinations of specific models were tested, with the average accuracy and macro-averaged F1 score being noted down for each model count. The findings from these initial experiments are displayed in figures 4.8 and 4.9.
As a result of these experiments, we settled on testing ensembles with model counts of 5 and 15, to get a grasp of performance improvements at 5 models where diminishing returns are not yet significant, and improvements at 15 models where diminishing returns start being more relevant.

Figure 4.8: Comparison of F1 And Accuracy Across Ensemble Model Size

(a) Accuracy Score For Ensembles of Norbert's Across Sizes



(b) F1 Macro Score For Ensembles of NorBERT's Across Sizes

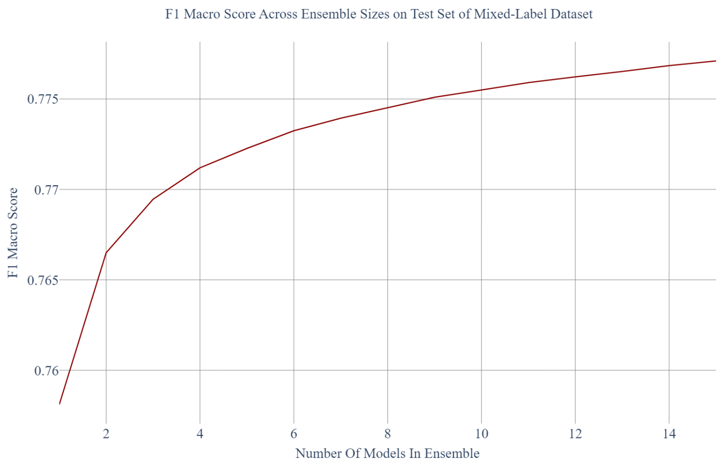
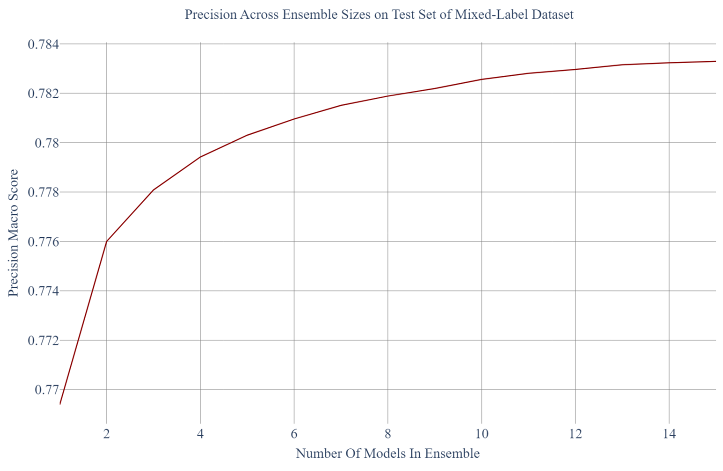
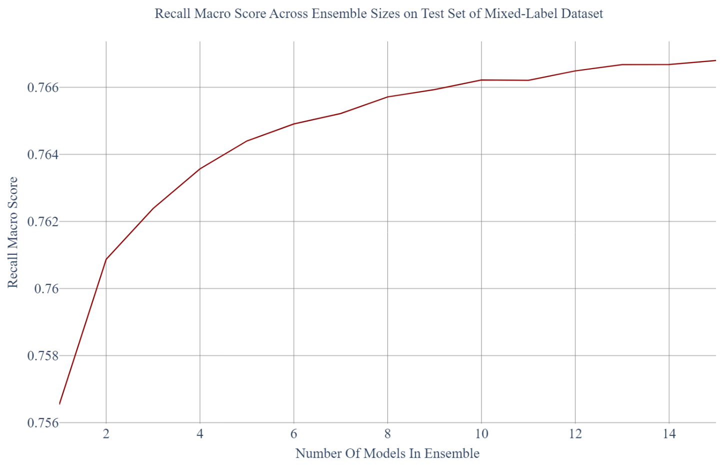


Figure 4.9: Comparison of Precision And Recall Across Ensemble Model Size

(a) Precision Macro Score For Ensembles of NorBERT's Across Sizes



(b) Recall Macro Score For Ensembles of NorBERT's Across Sizes



4.4.2 Random Ensemble Combinations

An area that seems potentially rewarding to us is to use multiple different model architectures within the same ensemble to leverage model diversity for higher performance. One concern when testing such an architecture would be picking two individual models who happen to compliment each other due to the patterns they happened to pick up on during training. Therefore, to ensure reproducibility, we train many models of each architecture using the same hyperparameters and randomly select from them. We also want to avoid the possibility of a certain distribution of models happens to perform well on our dataset (e.g. 2 of NorT5 and 3 NorBERTs happening to be best through luck) without generalising well, so we prefer to sample the distribution randomly instead. When testing ensembles consisting of NorT5, NorBERT and NorBERT-CNNs, for instance, we have a list of 30 NorT5s, 30 NorBERTs and 30 NorBERT-CNNs and randomly select 1-15 models from this distribution. We repeat this random selection 10,000 times for each model count from 1 to 15 and note the average of the performance metrics for each model count for each ensemble combination we wish to try.

4.4.3 Annotation Evaluation

To gauge how consistently our dataset is labeled, we've also done an experiment to see how much us two annotators agree on dataset labeling. For this, we selected 300 random sentences from the dataset and labeled them individually, then compared our labels. We then made use of scikit-learn's implementation of calculating the Cohen's kappa score of our labels. We report our results on both the whole set of 300 sentences and on specifically the subset where we exclude all sentences one or both of us have labeled as mixed. Our percentage agreed upon and our Cohen's kappa score for both categories are displayed in table 4.8

	Cohen's Kappa Score	Accuracy
With Mixed-label	0.6922	80.00%
Without Mixed-label	0.7441	84,42%

Table 4.8: Agreement Between Annotators

Chapter 5

Results and Discussion

5.1 Comparison of Datasets

Given that we've manually relabeled the original ternary and mixed-label datasets, the first point of comparison might be the model performance of models trained on the new datasets in comparison to those trained on the original datasets. Therefore, the first set of results we present are the average performances of a fine-tuned NorBERT model on all four datasets. For each dataset, 15 NorBERTs were trained with the same hyperparameters, and we show the average performance, as well as the standard deviation in table 5.1. Note that the performance on the relabeled mixed-label dataset is the average of 30 models, as we happened to train 30 of those for another experiment.

Datasets	F1 Macro	Accuracy	Recall Macro	Precision Macro
Original Label Mixed-label	$72.9621^{\pm 0.6905}$	$79.2531^{\pm 0.6391}$	$72.0326^{\pm 0.9830}$	$74.4133^{\pm 1.1864}$
Relabeled Mixed-label	$76.0413^{\pm 0.9949}$	$81.4610^{\pm 0.6773}$	$75.5733^{\pm 1.4048}$	$77.0924^{\pm 1.1370}$
Original Label Ternary	$79.1765^{\pm 0.6387}$	$82.1846^{\pm 0.6089}$	$78.5250^{\pm 0.8706}$	$80.1503^{\pm 1.1992}$
Relabeled Ternary	$82.1018^{\pm 0.8147}$	$84.5307^{\pm 0.7642}$	$82.0135^{\pm 1.1228}$	$82.3545^{\pm 1.0088}$

Table 5.1: Results of NorBERT Large Trained on Different Datasets, Average Performance of 15-30, With Standard Deviation

Across both ternary and mixed-label datasets, the relabeled datasets see substantially higher model performance for NorBERT, which suggests that the patterns in the relabeled dataset are clearer than those in the original datasets. This is also the first set of results comparing ternary and mixed-label datasets, and they show a trend that will reappear throughout the rest of this chapter. That is, the models trained on the mixed-label datasets perform worse by a wide margin.

5.2 Presentation of Results

5.2.1 Singular Models

We now present the results of singular models on the relabeled mixed-label dataset. For most of the models, their performance when trained on the same hyperparameters varied from individual model to individual model by a small but noteworthy amount. Therefore, we have trained each model 30 times with the exact same hyperparameters, to calculate the average of the performance metrics and their standard deviations. Our hope is that by taking the average of many models, we can account for outliers for each architecture and give a better view of their expected performance if our methods were to be replicated.
Across all metrics, NorT5 is the best performing model. The second best performing model is NorBERT, which notably has less than half as many parameters as NorT5, at 323M instead of 808M. NorBERT-CNNs perform slightly worse on average, and all four NorBERT-RNNs

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN	$75.6117^{\pm 0.8923}$	$81.3977^{\pm 0.5955}$	$75.1168^{\pm 1.4098}$	$76.7504^{\pm 1.4123}$
NorBERT-LSTM	$74.2377^{\pm 1.3268}$	$80.9309^{\pm 0.6254}$	$74.1176^{\pm 1.8037}$	$75.0192^{\pm 1.1869}$
NorBERT-BiLSTM	$74.0981^{\pm 1.1164}$	$81.0047^{\pm 0.5129}$	$73.8273^{\pm 1.6644}$	$75.1781^{\pm 1.1962}$
NorBERT-GRU	$75.3016^{\pm 0.8307}$	$81.5981^{\pm 0.6201}$	$74.7112^{\pm 1.1427}$	$76.6043^{\pm 1.3280}$
NorBERT-BiGRU	$74.8472^{\pm 1.2810}$	$81.1234^{\pm 1.0791}$	$74.6313^{\pm 1.3786}$	$75.7788^{\pm 2.0212}$
NorBERT Large	$76.0413^{\pm 0.9949}$	$81.4610^{\pm 0.6773}$	$75.5733^{\pm 1.4048}$	$77.0924^{\pm 1.1370}$
NorT5 Large	$\mathbf{76.6761^{\pm 0.5061}}$	$\mathbf{81.7868^{\pm 0.2886}}$	$\mathbf{76.1686^{\pm 0.6388}}$	$\mathbf{77.3930^{\pm 0.6492}}$

Table 5.2: Model Results on Relabeled Mixed-label Test Dataset, Average of 30 With Standard Deviation

perform relatively poorly. The bidirectional RNNs in particular perfrom worse than their non-bidirectional counterparts. The only performance of note amonth the NorBERT-RNNs are the NorBERT-GRUs, which have a slightly higher accuracy on average than NorBERTs and NorBERT-CNNs. The higher accuracy despite lower recall, precision and F1 score indicates that NorBERT-GRUs perform worse on the less represented labels.
To be able to compare our fine-tuning with previous performance results on the datasets, we've also tested our models on the ternary dataset with original labels, as the paper introducing NorBERT 3, NorT5 and the NorBench dataset had reported results on the dataset for sentence-level sentiment analysis[22].

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN	$79.3984^{\pm 0.8059}$	$82.6616^{\pm 0.6244}$	$78.1204^{\pm 1.2244}$	$81.3331^{\pm 1.1712}$
NorBERT-BiGRU	$79.5746^{\pm 0.4712}$	$82.3539^{\pm 0.3150}$	$78.9485^{\pm 0.7133}$	$80.3984^{\pm 0.5150}$
NorBERT-BiLSTM	$79.6812^{\pm 0.2861}$	$82.4555^{\pm 0.4397}$	$78.6282^{\pm 0.4951}$	$\mathbf{81.3721^{\pm 1.2529}}$
NorBERT-GRU	$79.9449^{\pm 0.7817}$	$82.7942^{\pm 0.7941}$	$79.1473^{\pm 1.0245}$	$81.1543^{\pm 1.5964}$
NorBERT Large	$79.1765^{\pm 0.6387}$	$82.1846^{\pm 0.6089}$	$78.5250^{\pm 0.8706}$	$80.1503^{\pm 1.1992}$
NorT5 Large	$\mathbf{80.3564^{\pm 0.4478}}$	$\mathbf{83.0773^{\pm 0.3182}}$	$\mathbf{79.6629^{\pm 0.5065}}$	$81.2610^{\pm 0.4795}$
(Prior Result) NorBERT Large[22]	$78.4^{\pm 0.6}$	-	-	-
(Prior Result) NorT5 Large[22]	$76.9^{\pm 2}$	-	-	-

Table 5.3: Select Models Results on the Original Ternary Dataset, Average of 30 With Standard Deviation

Once again, NorT5 is the best performer. This time, however, NorBERT-embedded CNNs and RNNs all outperform NorBERT across all metrics, with NorBERT-GRUs being the best performer behind NorT5, and NorBERT-BiLSTMs actually being the best performers on precision. The models we've trained also regularly outperform the model results reported in the paper introducing NorBench by a small margin.

5.2.2 Ensembles with a Single Type of Model

Before we present our results from testing diverse ensembles consisting of multiple model architectures, we've also looked at ensembles consisting of multiple models of the same architecture. We've chosen to do this experiment to get a comparison benchmark to test the importance of model diversity in an ensemble and get an idea of how increasing the count of models of the same architecture might perform in an ensemble that uses multiple model architectures.
We present results for ensembles with a single type of model for ensembles with 5 models voting, and ensembles with 15 models voting. First, we present the results of testing 5-model ensembles on the relabeled mixed-label dataset in table 5.4. To ensure that we simply do not pick 5 or 15 models that work unusuall well together, we trained 30 models of each kind and

chose 5 or 15 random ones to include in the ensemble 10,000 times, and took the average, and have included standard deviation on each metric.

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN	**76.9993$^{\pm 0.5797}$**	**82.2392$^{\pm 0.3614}$**	76.2280$^{\pm 0.7450}$	**78.1887$^{\pm 0.7136}$**
NorBERT-LSTM	75.7804$^{\pm 0.7220}$	81.8791$^{\pm 0.3580}$	75.2340$^{\pm 0.8705}$	76.8033$^{\pm 0.7322}$
NorBERT-BiLSTM	75.4574$^{\pm 0.6870}$	81.8047$^{\pm 0.3193}$	74.9647$^{\pm 0.9083}$	76.5030$^{\pm 0.5955}$
NorBERT-GRU	76.3392$^{\pm 0.5968}$	82.1811$^{\pm 0.3825}$	75.6092$^{\pm 0.8624}$	77.5407$^{\pm 0.8204}$
NorBERT-BiGRU	76.3285$^{\pm 0.5893}$	82.1764$^{\pm 0.3856}$	75.6066$^{\pm 0.8599}$	77.5227$^{\pm 0.8289}$
NorBERT Large	76.5376$^{\pm 0.6250}$	81.9170$^{\pm 0.3566}$	75.8329$^{\pm 0.9095}$	77.5522$^{\pm 0.7512}$
NorT5 Large	76.9840$^{\pm 0.3858}$	81.8443$^{\pm 0.2407}$	**76.5596$^{\pm 0.4057}$**	77.5675$^{\pm 0.4752}$

Table 5.4: 5-Model Ensemble Performance on Relabeled Mixed-labeled Dataset

We now present results for 15-model ensembles with a singular model type. These ensembles perform marginally better, at a tripled computational cost. The results, presented in table 5.5. The most performant 15-model ensembles are the ones consisting of NorBERT-CNNs and NorT5s, both of which have very similar results to their 5-model ensembles in terms of F1 score, but have reached a larger increase in accuracy.

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN	**77.1836$^{\pm 0.3039}$**	82.2613$^{\pm 0.2098}$	76.3943$^{\pm 0.3436}$	**78.3324$^{\pm 0.3628}$**
NorBERT-LSTM	76.0543$^{\pm 0.4316}$	82.1239$^{\pm 0.2315}$	75.4283$^{\pm 0.4662}$	77.1965$^{\pm 0.4531}$
NorBERT-BiLSTM	75.6863$^{\pm 0.3293}$	81.9734$^{\pm 0.1710}$	75.2365$^{\pm 0.3869}$	76.6147$^{\pm 0.3193}$
NorBERT-GRU	76.6868$^{\pm 0.3621}$	**82.4793$^{\pm 0.2250}$**	75.8355$^{\pm 0.4581}$	77.9881$^{\pm 0.4893}$
NorBERT-BiGRU	76.6796$^{\pm 0.3595}$	82.4766$^{\pm 0.2204}$	75.8315$^{\pm 0.4642}$	77.9810$^{\pm 0.4827}$
NorBERT Large	76.6776$^{\pm 0.4271}$	82.0120$^{\pm 0.2621}$	75.9288$^{\pm 0.4936}$	77.6578$^{\pm 0.5533}$
NorT5 Large	77.0621$^{\pm 0.2487}$	81.8227$^{\pm 0.1511}$	**76.6923$^{\pm 0.2646}$**	77.5807$^{\pm 0.2746}$

Table 5.5: 15-Model Ensemble Performance on Relabeled Mixed-labeled Dataset

For comparability with prior data, we've also tested these ensembles in the same way on the ternary dataset with original labels, and present our results in the following tables.

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN	80.1535$^{\pm 0.3150}$	83.2833$^{\pm 0.2749}$	78.7050$^{\pm 0.4064}$	82.1671$^{\pm 0.5238}$
NorBERT-GRU	**80.9257$^{\pm 0.3831}$**	**83.7083$^{\pm 0.3814}$**	**79.8389$^{\pm 0.4964}$**	**82.3847$^{\pm 0.6350}$**
NorBERT Large	79.9318$^{\pm 0.3851}$	82.8836$^{\pm 0.3507}$	79.1618$^{\pm 0.4046}$	80.8806$^{\pm 0.6394}$
NorT5 Large	80.4228$^{\pm 0.3090}$	83.1925$^{\pm 0.2353}$	79.6853$^{\pm 0.3453}$	81.3941$^{\pm 0.3064}$

Table 5.6: 5-Model Ensemble Performance on Original Ternary Dataset

model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN	$80.1751^{\pm0.2082}$	$83.3223^{\pm0.2029}$	$78.7351^{\pm0.2308}$	$82.1525^{\pm0.2780}$
NorBERT-GRU	$\mathbf{81.0969^{\pm0.2321}}$	$\mathbf{83.8659^{\pm0.2094}}$	$\mathbf{79.9441^{\pm0.2777}}$	$\mathbf{82.6297^{\pm0.2989}}$
NorBERT Large	$80.2060^{\pm0.2489}$	$83.1067^{\pm0.2172}$	$79.4137^{\pm0.2143}$	$81.1550^{\pm0.3998}$
NorT5 Large	$80.4989^{\pm0.0000}$	$83.4039^{\pm0.0000}$	$79.7514^{\pm0.0000}$	$81.4847^{\pm0.0000}$

Table 5.7: 15-Model Ensemble Performance on Original Ternary Dataset

5.2.3 Ensembles of Multiple Model Types

In this section, we present results on our most performant ensemble models, which are ensembles consisting of multiple different model types. We've tested many different combinations of models, and report the most performant combinations on the relabeled mixed-label dataset and the original ternary dataset.

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN + NorT5	$78.1778^{\pm0.6282}$	$82.9823^{\pm0.5190}$	$77.2616^{\pm0.6173}$	$79.3586^{\pm0.8219}$
NorBERT + NorT5	$78.0209^{\pm0.6173}$	$82.8533^{\pm0.4626}$	$77.0038^{\pm0.7538}$	$79.3615^{\pm0.8475}$
NorBERT-GRU + NorT5	$77.8541^{\pm0.7490}$	$82.9144^{\pm0.4872}$	$77.0361^{\pm0.8643}$	$78.9984^{\pm0.8479}$
NorBERT + NorBERT-CNN + NorT5	$\mathbf{78.2793^{\pm0.7134}}$	$\mathbf{83.0614^{\pm0.5155}}$	$\mathbf{77.3016^{\pm0.7603}}$	$\mathbf{79.5694^{\pm0.9007}}$
NorBERT + NorBERT-CNN	$77.7049^{\pm0.4303}$	$82.6831^{\pm0.2751}$	$76.9415^{\pm0.5545}$	$78.7578^{\pm0.5334}$

Table 5.8: 5 Models Results on Relabeled Mixed-label Dataset

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN + T5	$78.5173^{\pm0.3915}$	$83.2513^{\pm0.3874}$	$77.5878^{\pm0.3941}$	$79.6676^{\pm0.5177}$
NorBERT + NorT5	$78.6557^{\pm0.4112}$	$83.2386^{\pm0.3634}$	$\mathbf{77.7053^{\pm0.4158}}$	$79.8457^{\pm0.5293}$
NorBERT-GRU + NorT5	$78.4278^{\pm0.4970}$	$83.3802^{\pm0.3085}$	$77.5338^{\pm0.5829}$	$79.6134^{\pm0.4838}$
NorBERT + NorBERT-CNN + NorT5	$\mathbf{78.7444^{\pm0.4950}}$	$\mathbf{83.4553^{\pm0.3970}}$	$77.5992^{\pm0.5163}$	$\mathbf{80.2124^{\pm0.6004}}$
NorBERT + NorBERT-CNN	$77.2302^{\pm0.6390}$	$82.3776^{\pm0.3968}$	$76.4968^{\pm0.8684}$	$78.3194^{\pm0.7833}$

Table 5.9: 15 Models Results on Relabeled Mixed-label Dataset

LTG Ternery

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN + NorT5	$80.9681^{\pm 0.5687}$	$83.8651^{\pm 0.4450}$	$79.7245^{\pm 0.6689}$	$82.6698^{\pm 0.6284}$
NorBERT + NorT5	$80.7988^{\pm 0.5512}$	$83.5131^{\pm 0.4412}$	$79.8274^{\pm 0.5485}$	$82.0622^{\pm 0.7166}$
NorBERT-GRU + NorT5	$\mathbf{81.3396^{\pm 0.4585}}$	$\mathbf{84.0212^{\pm 0.4065}}$	$\mathbf{80.2856^{\pm 0.5129}}$	$\mathbf{82.7624^{\pm 0.6574}}$
NorBERT + NorBERT-CNN + NorT5	$80.7034^{\pm 0.5889}$	$83.5716^{\pm 0.4650}$	$79.5811^{\pm 0.6128}$	$82.1933^{\pm 0.7767}$
NorBERT + NorBERT-CNN	$80.1785^{\pm 0.3855}$	$83.2117^{\pm 0.3406}$	$79.1103^{\pm 0.4555}$	$81.5724^{\pm 0.6800}$

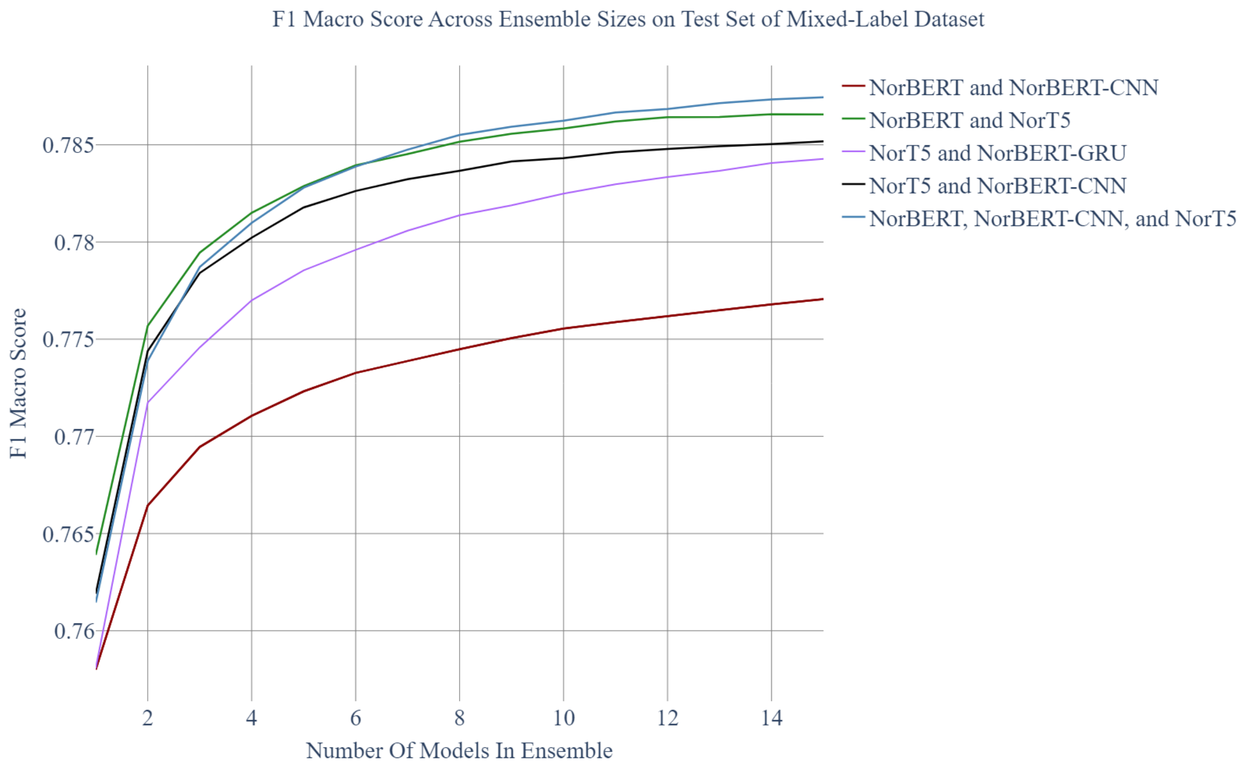
Table 5.10: 5 Models Results on Original Ternary Dataset

Model	F1 Macro	Accuracy	Recall Macro	Precision Macro
NorBERT-CNN + T5	$81.2600^{\pm 0.4627}$	$84.1238^{\pm 0.3248}$	$79.9987^{\pm 0.5163}$	$82.9719^{\pm 0.4393}$
NorBERT + NorT5	$81.1395^{\pm 0.3705}$	$83.7372^{\pm 0.2792}$	$80.0878^{\pm 0.3905}$	$82.5005^{\pm 0.4153}$
NorBERT-GRU + NorT5	$\mathbf{81.6273^{\pm 0.2999}}$	$\mathbf{84.2425^{\pm 0.2573}}$	$\mathbf{80.5189^{\pm 0.3331}}$	$\mathbf{83.1099^{\pm 0.3789}}$
NorBERT + NorBERT-CNN + NorT5	$80.9115^{\pm 0.4031}$	$83.7453^{\pm 0.3077}$	$79.7323^{\pm 0.4217}$	$82.4638^{\pm 0.4502}$
NorBERT + NorBERT-CNN	$80.4696^{\pm 0.2678}$	$83.4767^{\pm 0.2250}$	$79.3945^{\pm 0.2992}$	$81.8389^{\pm 0.3958}$

Table 5.11: 15 Models Results on Original Ternary Dataset

Figure 5.1: Macro F1 And Accuracy from Ensemble Combinations Across Ensemble Size On Relabeled Mixed-label Dataset

(a) F1 Macro Score



(b) Accuracy Score

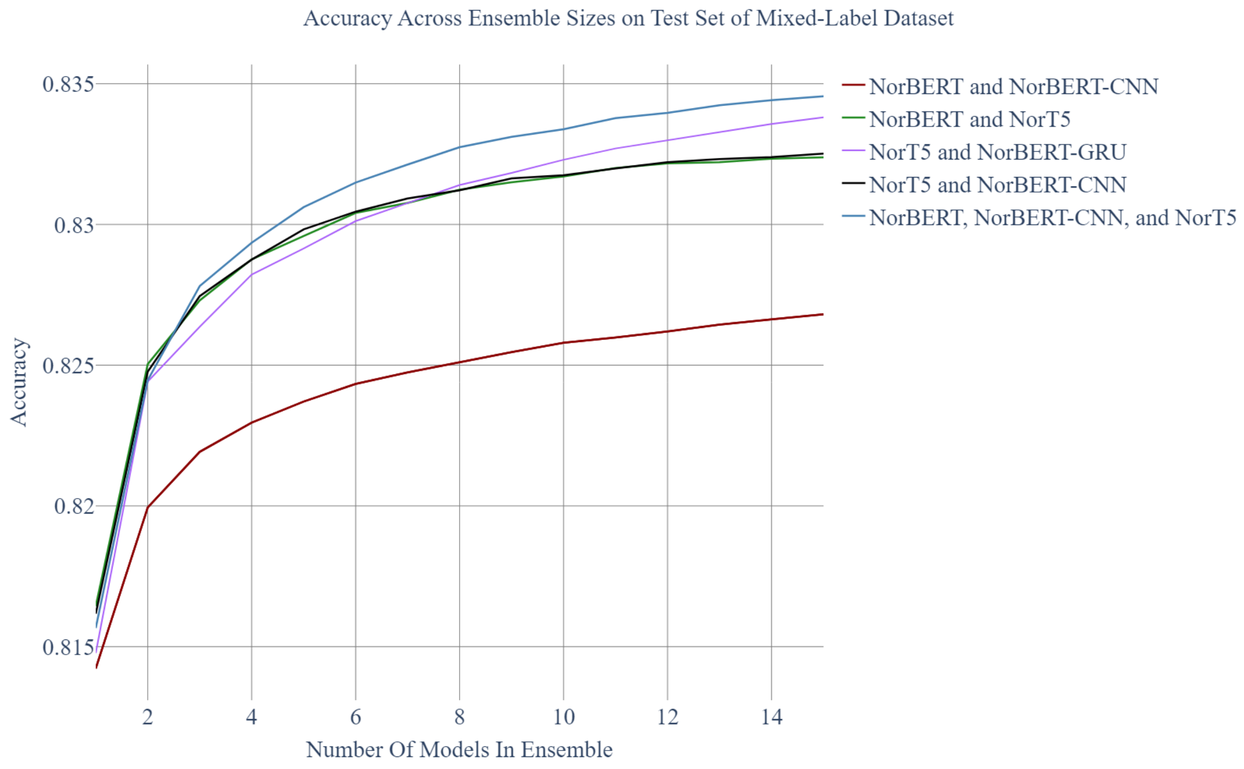
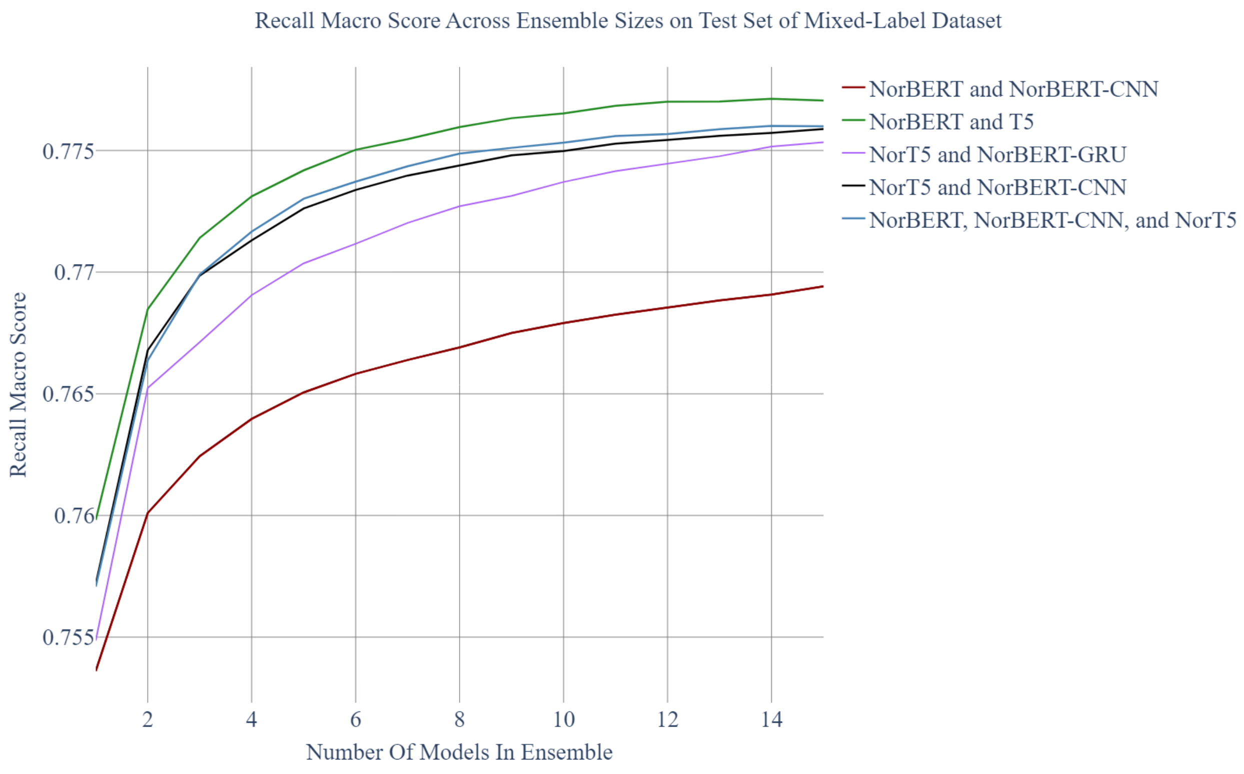


Figure 5.2: Macro Recall And Macro Precision From Ensemble Combinations Across Ensemble Size On Relabeled Mixed-label Dataset

(a) Recall Macro Score



(b) Precision Macro Score

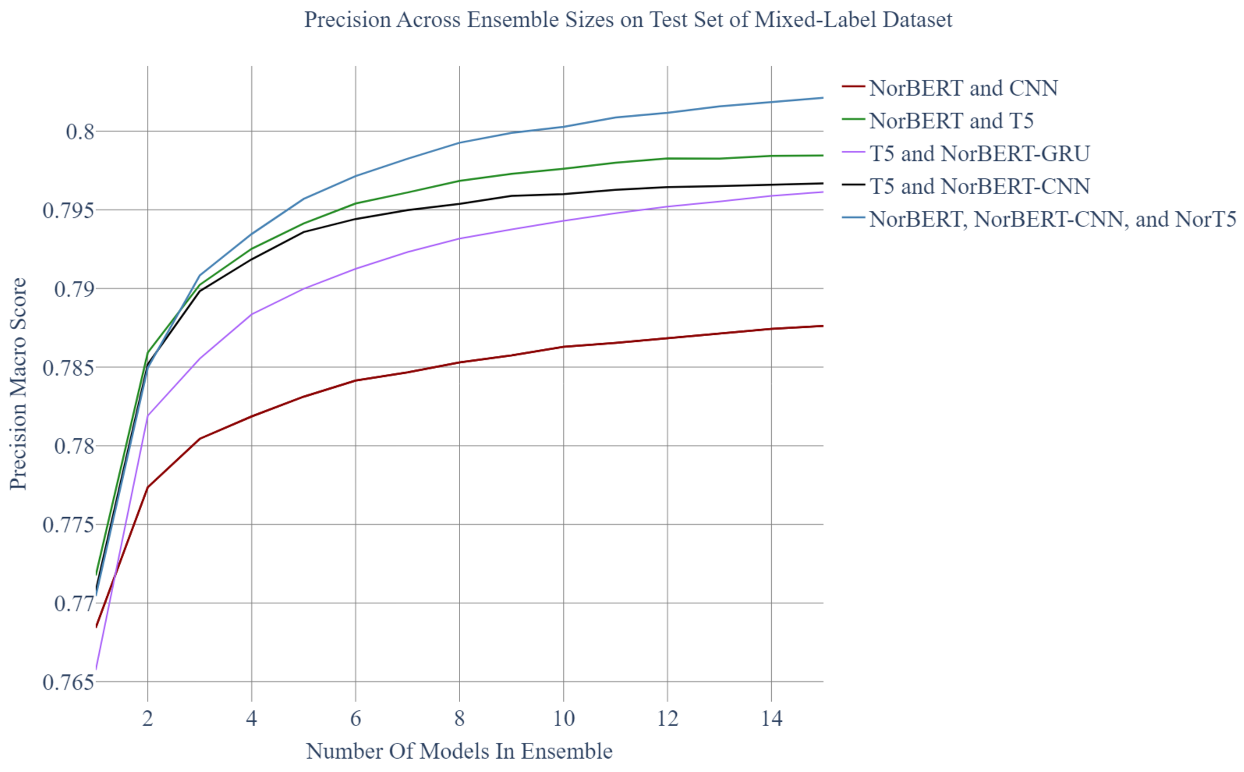
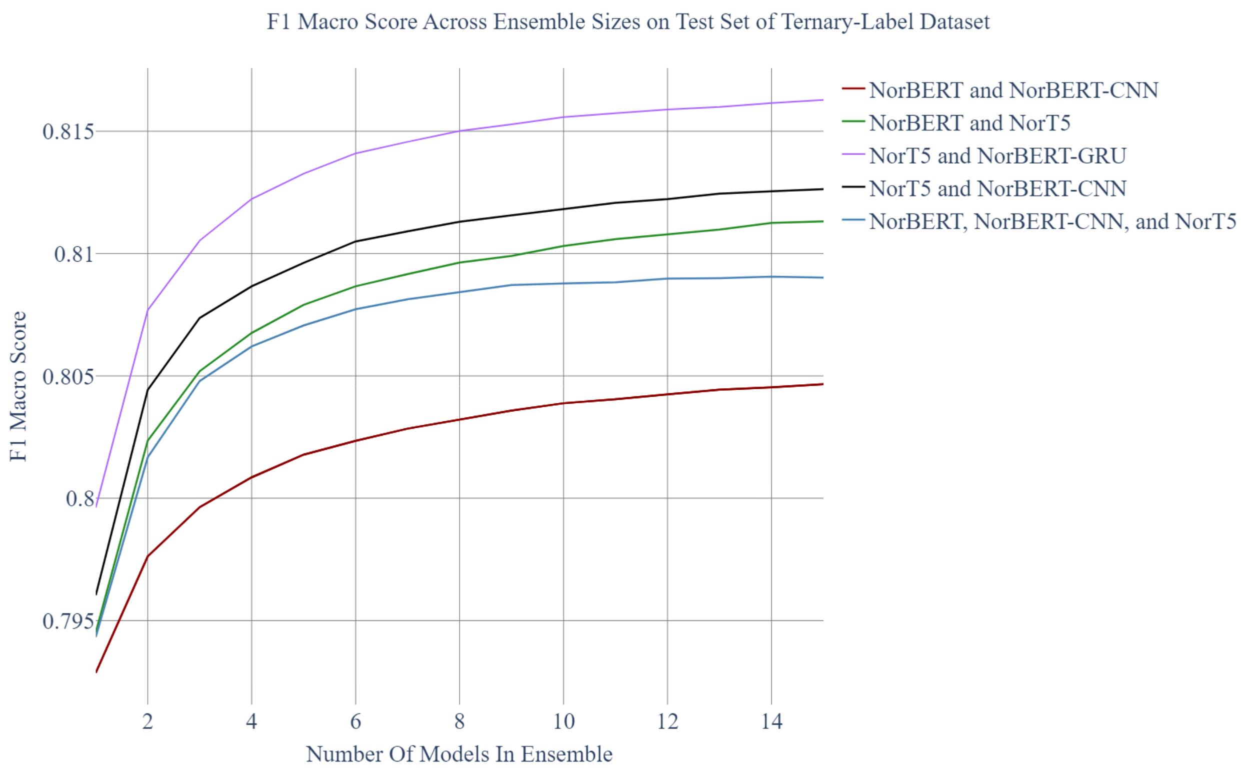


Figure 5.3: Macro F1 And Accuracy from Ensemble Combinations Across Ensemble Size on Original Ternary Dataset

(a) F1 Macro Score



(b) Accuracy Score

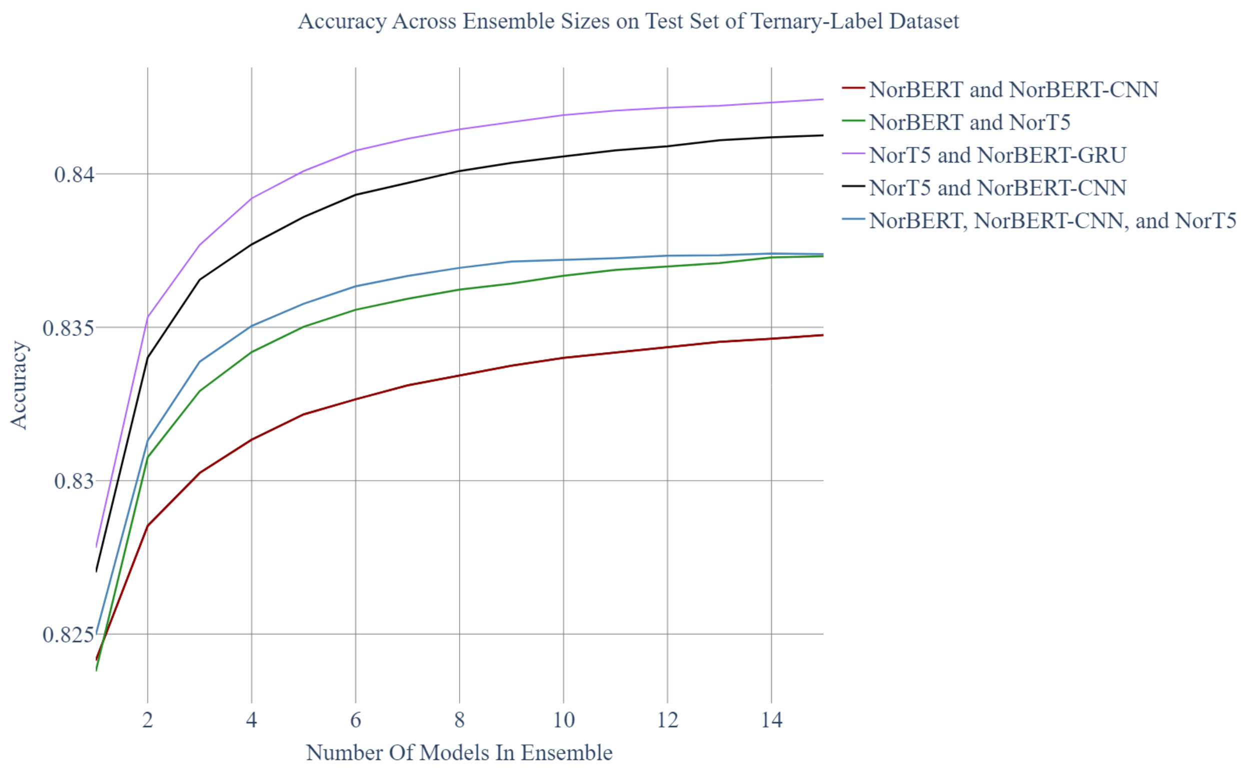
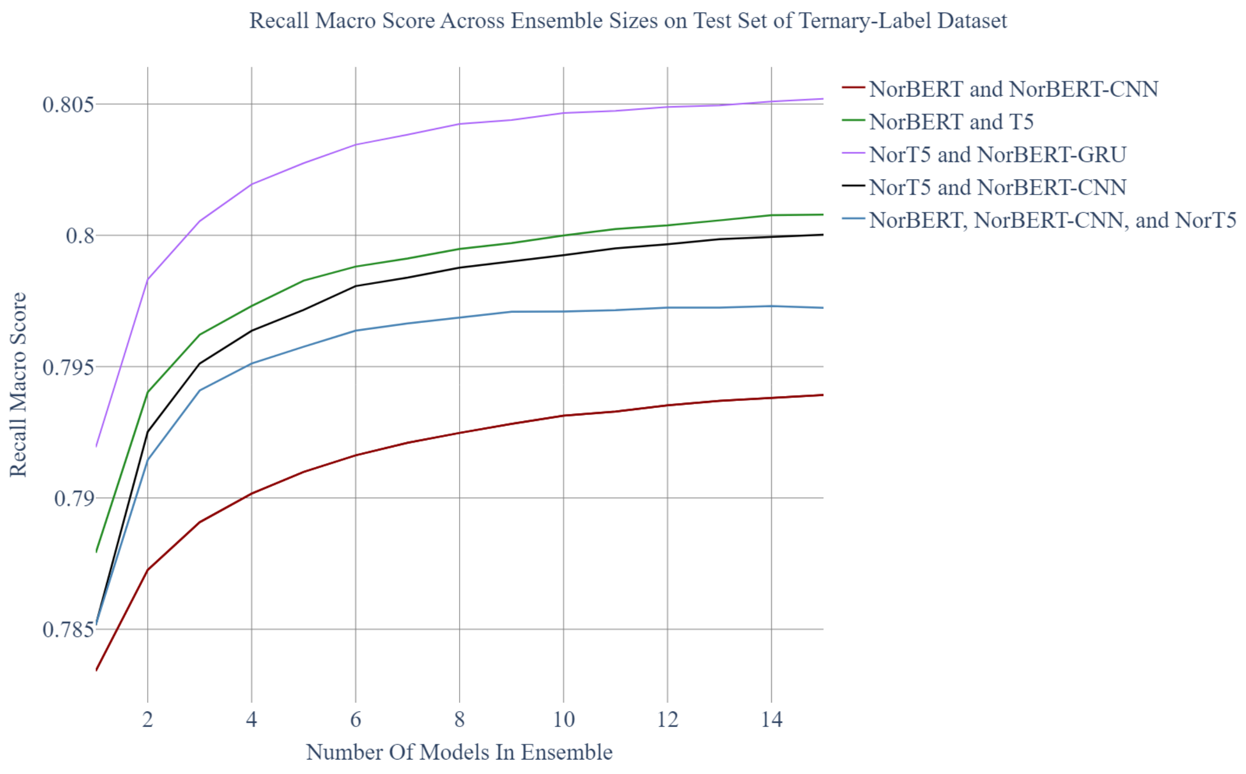
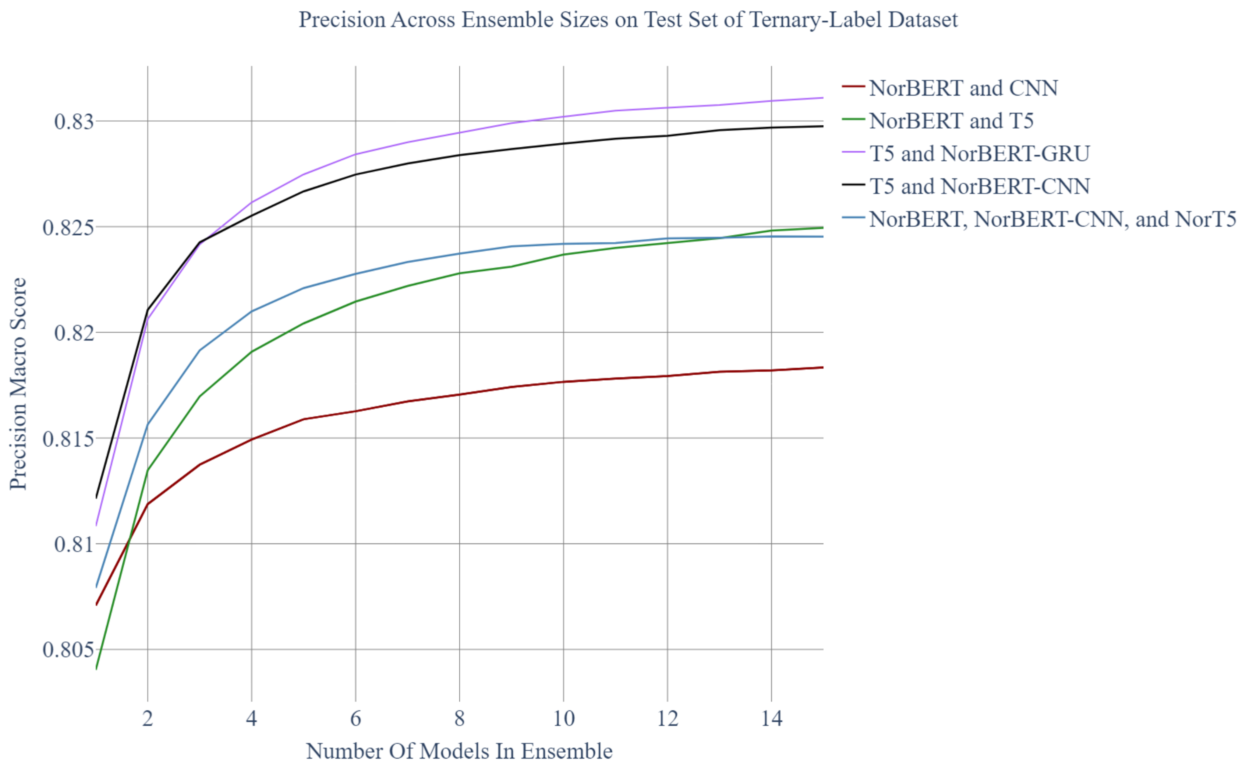


Figure 5.4: Macro Recall And Macro Precision From Ensemble Combinations Across Ensemble Size on Original Ternary Dataset

(a) Recall Macro Score



(b) Precision Macro Score



5.3 Discussion on Findings

5.3.1 Dataset Creation

For the relabeling process, each of us have labeled a separate portion of the dataset with only some overlap. A concern caused by this is low inter-annotator agreement, which would be that we consistently label things with a different thought process, resulting in multiple possible labels for any piece of data depending on who labeled it. From table 4.8, we consistently label 80% of data the same, resulting in a Cohen's Kappa score of 0.6922, which significantly indicates that we label data similarly. In reality, we believe the full dataset to have a higher level of agreement on the labels, as when we've labeled data independently, each of us also labeled more challenging data as "in need of discussion", to mark it as something we need to discuss together before labeling. When discussing as a group, we've often come to a much more confident conclusion, which we were certain we agreed on.
The most disproportionate challenge during the labeling process in terms of inter-annotator agreement was consistently the mixed-label, and whether or not a datapoint should be labeled with it. Excluding the mixed-label, we agreed on 84.42% of data have a Cohen's kappa score of 0.7441, which is a notable increase compared to labeling including the mixed-label. We believe the reason to be that the mixed-label introduces significant possibility for nuance that leaves much more room for interpretation, causing disagreement in the labeling.

Comparison of Datasets

In table 5.1, for both mixed and ternary datasets, we've achieved higher scores on the relabeled datasets than on the datasets with original dataset. We believe this to in part be due to our labeling being done by only two people who regularly discussed what to label more nuanced sentences as, leading to clearer patterns in the labeling. Additionally, the original dataset had a number of duplicates, and we believe the removal of duplicates reduces the model's overfitting on the duplicated datapoints, improving performance marginally.
The model performances on mixed-label datasets are also noticeably worse. We attribute the decline in performance in part to the increase in nuance in the data by introducing a label with content overlap with both the positive and negative labels and in part to the decrease in inter-annotator agreement, which makes the training data less clear for the model. Notably, the model performs worse on all labels on the mixed-label datasets, and not just the mixed-label datapoints. We suspect that the ambiguities between mixed-label and the other labels makes it more challenging for the model to categorize negative data as negative just because there's an indicator for a negative sentiment present, and that this source of uncertainty causes the model to perform worse.

5.3.2 Findings on Singular Model Performances

Our Mixed-labeled Dataset

In our findings, the best performing non-ensemble model on mixed-label datasets is the NorT5-Large, which exceeded our expectations by a wide margin. We initially expected NorBERT to outperform it due to the reported findings of NorBERT-3 Large performing well in the paper introducing NorBench[22]. The possible reasons we point to for our high NorT5 performance are twofold. The first is that we've trained it to predict a number label rather than a full word, i.e. "0" rather than "negative", which is just a single token. The second reason is simply that we've further optimized the hyperparameters by changing the learning rate to be lower and increasing dropout, which has had a big impact.
All NorBERT-embedded RNN models perform relatively poorly on the mixed-label dataset. NorBERT-GRUs stand out among them as the best performers, but still perform worse across most metrics than the FNN-based models. We are led to believe that the RNN-based

models perform particularly poorly at accounting for the complexities introduced by the mixed label, as they perform rather well on the ternary datasets.
An additional challenge of the mixed-label datasets, which all models have to deal with, is that our label distribution is very uneven, with mixed as the least represented label, and this disparity might contribute to the decreased model performance on mixel-label datasets. NorBERT performs decently on this dataset.

Original-Label Ternary Dataset

Among the models we've tested, the one that performs worst on the not-relabeled ternary dataset labels was NorBERT, as seen in table 5.3. By optimizing hyperparameters and introducing a patience factor, we've still achieved a marginally higher score on the dataset than was reported in the paper introducing NorBench[22].
On all metrics, a NorBERT-CNN outperforms the NorBERT model by small margins.
Of particular interest are the results achieved by the NorBERT-embedded RNNs, which all perform significantly better when compared to the mixed-label dataset. Both bilinear LSTMs and bilinear GRUs perform on par with or marginally better than CNNs, with the sole exception of BiGRU models having slightly worse accuracy than the CNNs. The high performance on the ternary datasets reinforce our opinion that the NorBERT-embedded RNNs struggle significantly more with classifying mixed-label data, and work well on classifying ternary data.
The highest performing model across most metrics is the fine-tuned NorT5 here as well, with a macro-averaged F1 score of 80.3564, which is a significant increase from the reported NorT5 results in the paper introducing NorBench. NorT5 seems to perform generally well on Norwegian sentiment analysis, regardless of if the mixed label has been introduced to the datasets. NorT5 is significantly outperforming the other models on F1 score, accuracy and recall, with it being narrowly behind NorBERT-BiLSTM on recall.

5.3.3 Findings on Ensemble Model Performances

Ensembles With A Singular Model Type

The key factors we believe impacts the performance of ensembles consisting of the same type of models are the performance of a single model and the diversity in patterns learned among individual models trained on the same hyperparameters.

5-Model Ensembles On the relabeled mixed-label dataset, NorBERT-CNNs marginally outperform NorT5 as the best model on average in a 5-model ensemble across all metrics except recall, as shown in table 5.4. This result is somewhat unexpected, as a single NorT5 outperforms a single NorBERT-CNN by a notable margin on the same dataset, as shown in table 5.2. NorBERT-CNNs have therefore seen greater performance gains when ensembled with more NorBERT-CNNs than NorT5s have seen when ensembled with more NorT5s.
On the original ternary dataset, the NorBERT-RNNs perform significantly better when ensembled as well, with the best model across all metrics for a 5-model ensemble, as shown in table 5.6, is the NorBERT-GRU. From table 5.3, NorBERT-GRU was consistently the second best performer for most metrics as a singular model on the original ternary dataset as well, so it's not surprising to see it exceed the NorT5 models when in 5-model ensembles, after we've already seen NorT5 achieve very marginal performance increases in ensembles.

15-Model Ensembles On the relabeled mixed-label dataset, the model seeing the highest performance in a 15-model ensemble are according to table 5.5 NorBERT-CNNs in terms of F1 score, and NorBERT-BiGRUs in terms of accuracy. NorBERT-CNNs have seen even

further increases in performance compared to NorT5, which has seen only marginal improvements. We believe the better scaling of NorBERT-CNNs to be due to each model picking up on slightly different patterns and having more distinct outputs from one another than the NorT5 models, which are performant on their own, but are more often in agreement when trained on the same hyperparameters.
On the original ternary dataset, NorBERT-GRUs see marginal improvements in every metric, but the difference in contents between table 5.6 and table 5.7 is slight, especially considering that one table contains three times as many parameters as the other.

Findings on Ensembles With Multiple Model Types

5 Models Table 5.8 shows our best results for ensembles of 5 different models on the relabeled mixed-label dataset. The performance of the most performant ensemble has increased significantly, with an increase in macro-averaged F1 score of over 1 point, an increase in recall of over 0.7, an increase in precision of over 1.3, and an increase in accuracy of 0.82. Another noteworthy change from 5-model ensembles with only one type of model is that our best performing ensemble with multiple different model types is the best performer across all metrics. This ensemble, consisting of NorT5s, NorBERT-CNNs and NorBERTs, gains both the high recall of NorT5, and benefits from the performance and diversity of NorBERT-CNNs and NorBERTs.
Table 5.10 shows our best results for ensembles of 5 models of multiple types on the original ternary dataset. Combining NorBERT-GRUs and NorT5s together makes for a more diverse ensemble consisting of the two most performant singular models on the dataset, and outperforms the 5-model ensemble consisting of only NorBERT-GRUs across every metric.

15 Models Table 5.9 shows the most performant ensembles we've implemented for the relabeled mixed-label datasets, at a very high computational cost. The differences in performances between table 5.8 and table 5.9 are still greater than those between table 5.4 and table 5.5 across all metrics, showing that the more diverse models reach diminishing returns slightly later when introducing more models to the ensemble. Whether these marginal improvements are worth a significantly higher computational cost is another question.
Table 5.11 show the most performant ensembles on the original ternary dataset. The performance increases between table 5.10 and 5.11 are also greater than those of the equivalent ensembles of a singular model type, with the same question regarding whether the small performance increase is worth significantly higher computational cost.

Graphs

Figures 5.1, 5.2, 5.3 and 5.4 show the change in performance metrics over the change in numbers of models in an ensemble. Across all of them, we see immediate improvements when going from one to two models, and lesser but still notable increases for the next few models, and see performance increases taper off the more models are added after the first few, clearly indicating diminishing returns.

5.4 Error Analysis

We've extracted the sentences misclassified by the best performing ensemble, and in looking over the results, a portion of the misclassified sentences could be up for interpretation. Our ensemble is far from perfect, but the ambiguity of the task at hand, with an inter-annotator agreement in the range of 80%, leaves room for disagreement. There are still many sentences the model makes more obvious classification errors on, where it picks up on a sentiment that is not there, or misses a relevant sentiment within the training data. An example of this would be the sentence "Det som skulle bli et penere aktivitetsarmbånd , ble et dyrere , med færre funksjoner ." (English: "What was supposed to be a prettier activity arm-band turned out to be more expensive, with fewer functions"), a sentence which we both believe to be negative due to phrasing around "What was supposed to be prettier", implying a negative sentiment. The model, however, classifies it as mixed.

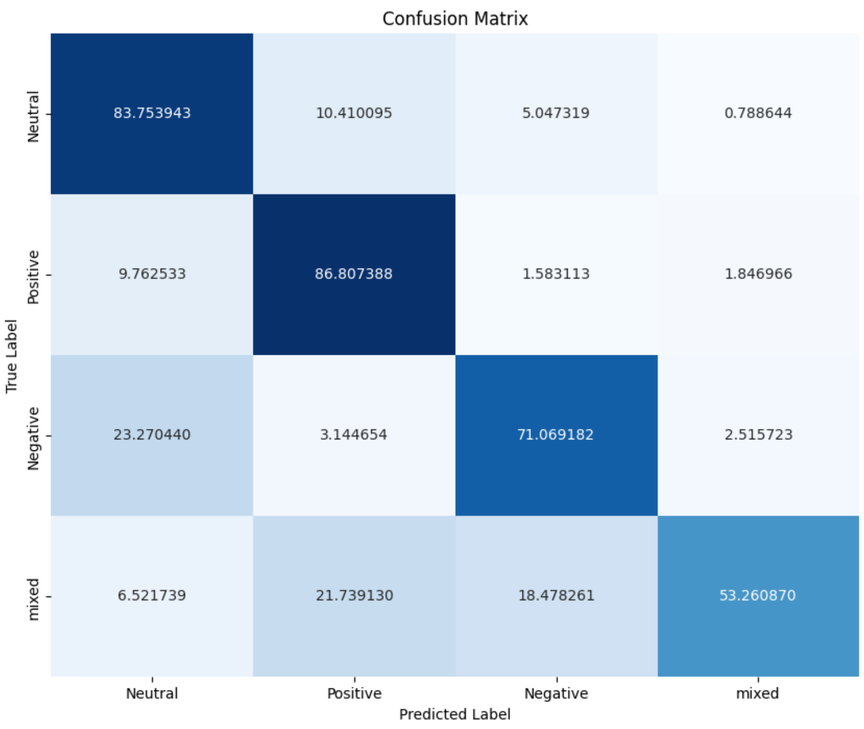
5.4.1 Confusion Matrices

For a clearer overview of how models perform on the datasets, their confusion matrices give a good initial understanding of which labels pose greater challenges than others. For all confusion matrices in figures 5.5 and 5.6, very few negative sentences are labeled as positive, and very few positive sentences are labeled as negative. This is to be expected as they are the most conceptually distinct categories. There are also very few mixed-label sentences labeled as neutral, and even fewer neutral sentences labeled as mixed. This too is not surprising, as there's a vast conceptual difference between text with no polarity and text with both positive and negative polarity.

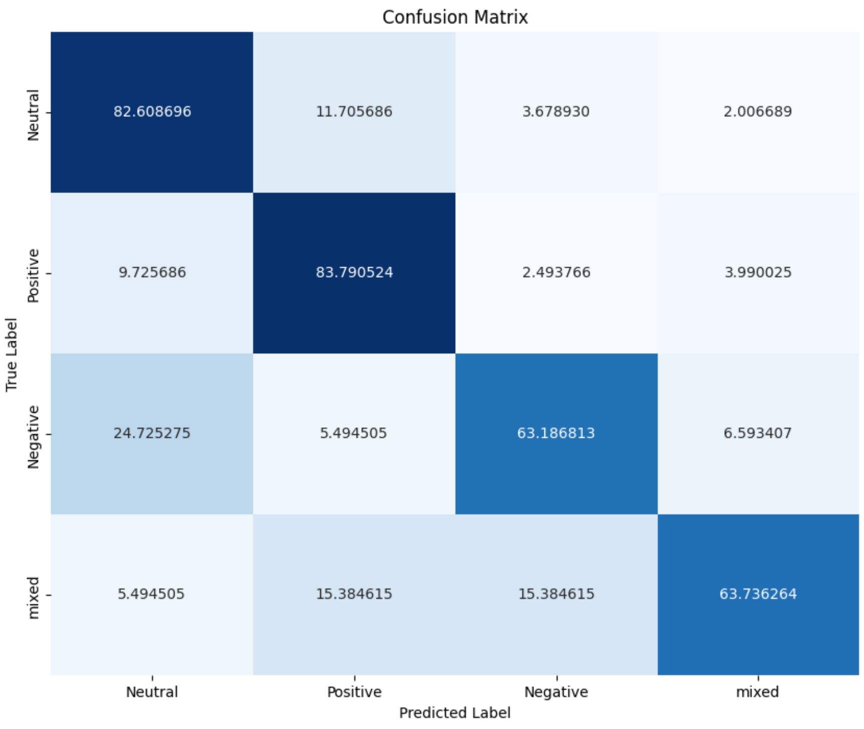
However, the remaining comparisons are less clear-cut. A significant portion of negative sentences are labeled as neutral, around 10% of positive and neutral data has been incorrectly labeled with the other category, and significant portions of mixed-label data are incorrectly labelled as either positive or negative in equal measure.

Interestingly, models trained on the relabeled dataset have a significantly lower recall with regards to the mixed label, but perform better on the other three labels when compared to the original dataset. On the ternary datasets in particular, the increase in recall on the negative label accounts for the vast majority of performance gains.

The confusion matrices also show that the negative label has significantly lower recall when the mixed label is introduced to the dataset, particularly on the relabeled datasets, and that the other two labels are impacted to a lesser extent.

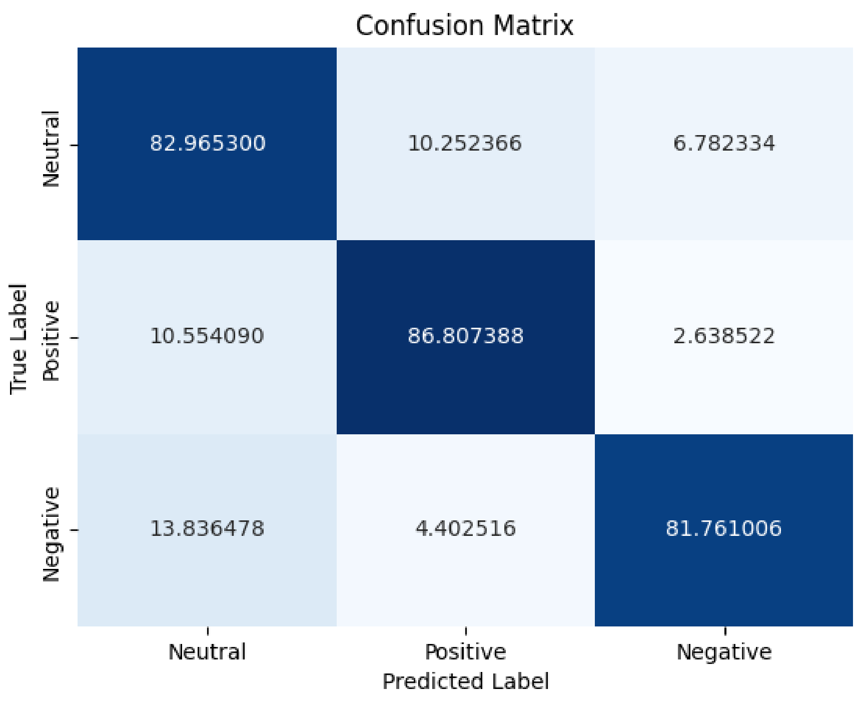


(a) Our Relabeled Version

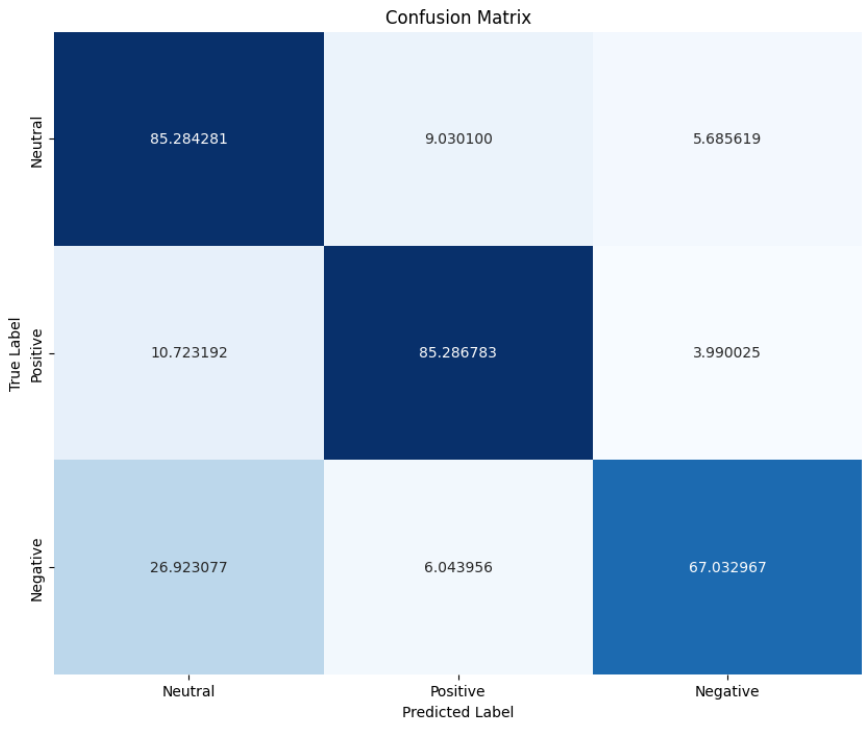


(b) Original Labeled Version

Figure 5.5: Comparison of Mixed-label Datasets



(a) Our Relabeled Version



(b) Original Labeled Version

Figure 5.6: Comparison of Ternary Datasets

Chapter 6

Conclusion and Future Work

6.1 Summary of Key Findings

After implementing, training and testing multiple kinds of algorithms both on their own and in various ensemble combinations on both ternary and mixed-label data, our results strongly indicate that models perform significantly worse across the board on mixed-label datasets. No model architecture has reliably performed better on mixed-label data than on ternary on any metric, and the models perform worse on all labels in mixed-label datasets, rather than just performing disproportionately poorly on the mixed label.
While all model architectures have had significant decreases in performance when trained on mixed-label data rather than ternary, the decrease in performance has not been equal between models. In particular, NorBERT-embedded RNN variants performed significantly worse at the mixed-label data copmpared to ternary data, while NorBERT, NorT5 and NorBERT-CNNs were impacted to a lesser extent.
With our relabeled version of the dataset, we've seen consistent higher model performance scores across most metrics. When labeling data, we've had substantially high inter-annotator agreement, which we believe stems from all labeling being done by two people who discussed the process together regularly. It's worth noting that the label causing the most disagreement between us were the mixed label, by a large margin.
By applying NorBERT embeddings by giving a different model the final hidden states of a NorBERT as input, the other models have seen solid performances. The models that have been given NorBERT embeddings performs better than just using a NorBERT model for classification, on particular tasks and metrics. At other tasks, NorBERT outperforms the NorBERT-embedded models.
In our testing, ensembles where the models' votes are aggregated together performs significantly better than a single model does by itself at a steep increase in computational cost. We've tested many different combinations both on mixed-label and ternary data. On ternary data, with an ensemble of 5 discrete models, having a selection of NorT5s and NorBERT-embedded GRUs was the best performer. For 5-model ensembles on mixed-label data, a selection of NorT5s, NorBERTs and NorBERT-embedded CNNs performed best. We've also tested voting ensembles with 15 models casting votes, and found marginal improvements over 5 model ensembles.

6.2 Conclusion of the Study

6.2.1 Conclusions Regarding Introduction of Mixed-label Data

Models tested on mixed-label datasets perform worse than ones tested on ternary datasets. We put forward the following explanations for this phenomenon, all of which we believe play a part.

Mixed-label data increases the complexity of the classifications significantly, as a negative sentiment being present no longer necessarily mean the negative label is accurate. Increased complexity likely poses a significant challenge to the model, contributing to the decline in accuracy on other labels than the mixed class as well.
Mixed data increasing complexity also makes the annotation process more challenging and prone to disagreement, as we disagreed disproportionately often on whether a datapoint should have the mixed label during our testing for inter-annotator agreement score. Lower inter-annotator agreement also makes classification more challenging for the model, due to more inconsistent labeling in the dataset.
Dataset distribution also seems a likely factor, as mixed sentences are rarer than their positive, negative and neutral counterparts, which means that there's less mixed-label data to train on. Introducing an underrepresented class into the dataset can drag down performance significantly. The other label impacted most by introducing the mixed label was the negative label, which is the second most underrepresented label.

6.2.2 Conclusions on Model Performance

We've found that some models perform quite well on ternary datasets while performing disproportionately worse on mixed-label datasets compared to other models. NorBERT-RNN variants especially outperform NorBERT on ternary datasets, but perform significantly worse on the mixed-label datasets. This brings us to believe that NorBERT-RNNs struggle more with the issues introduced by the mixed label, such as learning to classify an underrepresented class, and the inherent complexities of the mixed label. In general, it seems NorT5 is the best performing singular model.
For ensemble models, we must conclude that there are more important factors than just the performance score of the individual models that determines the ensemble performance. NorT5, the best performing singular model, gets minimal performance increases when put in ensembles with more of itself. An ensemble of either 5 or 15 NorT5 models performed worse than the same number of NorBERT-CNNs, despite singular NorBERT-CNNs performing worse. We believe the reason for this to be that NorBERT-CNNs pick up on different patterns in the data, resulting in higher diversity than NorT5s.
Ensembles consisting of multiple different model architectures performs significantly better than ensembles with a single model architecture, likely due to the increase in model diversity allowing models to cover each other's weaknesses. Some combinations of models consistently result in higher performing ensembles, showing that certain model architectures compliment each other's weaknesses better. In particular, ensembles consisting of NorT5 and other models that singularly perform well on a given dataset seem to perform much better together, leveraging both the high individual performance of NorT5 and the diversity of other architectures.
As ensemble sizes increase, there are diminishing returns on accuracy in exchange for computational cost. We consider an ensemble of 5 models to be a good middle ground of increased accuracy and increased costs. It's worth noting that while computational cost increases with each model added to the ensemble, only one model needs to be loaded in at once, meaning that the memory needed to run the ensemble is equal to the memory needed to run the largest individual model in the ensemble, which is a benefit over a hypothetical single model with the same parameter count as the ensemble.

6.2.3 Final Conclusions and Recommendations

Mixed-label datasets pose a significant challenge for models to adapt to, in part due to an increase in the complexity of the dataset. Depending on use-case, introducing a mixed label to the datasets could have significant practical utility that necessitates its introduction

despite the deprecating effect it has on other labels. For such use cases, we recommend a voting ensemble consisting of a collection of NorT5s, NorBERTs and NorBERT-CNNs.
In the event that the mixed label does not have significant importance for a use case, or it is less important to categorize correctly compared to other labels, we recommend considering a ternary dataset instead. For instance, if one applies sentiment analysis for detecting negative discussions of a product, applying only the negative label for all statements where a negative label is present might have desirable benefits for model performance. For ternary data, we recommend a voting ensemble consisting of a collection of NorT5s and NorBERT-GRUs.
Regardless of if the dataset is mixed-label or ternary, we recommend that the labeling process be done by a small group that communicates regularly about why certain labels apply to the data, to ensure higher inter-annotator agreement, and therefore a clearer dataset for the model to learn from.

6.3 Limitations of the Study

A major limiting factor in the study has been the size of the datasets we've trained on. We believe a larger dataset that contains more than 11,098 unique sentences would lead to better performance. In particular, the uneven label distribution where there are disproportionately few mixed and negative datapoints likely skews the model performance on these labels for the worse.
Another concern limiting the applicability of this study is the uniform type of sources the data comes from. The sentences were all collected from newspaper reviews and articles from several years ago, which is a good source for gauging the manner of speech in the Norwegian media, but does not account for any subcultures or varied types of speech that are used outside of newspapers. We can be confident in the applicability of our findings on sentiment analysis of newspaper excerpts and such, but if one were to for instance do sentiment analysis of current Norwegian teenagers online on social media, our results might be less applicable due to different speech patterns that some other model might better pick up on. We still believe our findings to be useful for sentiment analysis on general Norwegian, but we cannot be certain on our performance outside of newspaper articles due to little diversity in the sources of the training data.
To keep the scope of the project from becoming too wide, we've had to disregard some potentially promising avenues of testing which could've led to us seeing improvements. We've only used NorBERT embeddings on our RNN models and CNN, and have not had the time to research other alternatives such as using traditional word embeddings, which could've potentially performed well. There's still a vast unexplored possibility space for the models we've tested that could improve performance, and low-performing models should therefore not be disregarded entirely due to our results.
Hardware has also been a constraint that has limited our ability to test models. Due to limits in memory, we've only been able to test models using fairly small batch sizes, which might not be optimal. The time needed to train models also limits our ability to search for performant hyperparameters. The possibility still exists that there are hyperparameters that could give a noteworthy increase in performance for any of the models we've tested.

6.4 Recommendations for Future Research

To ensure that our results are not exclusive to the particular brand of Norwegian found in newspaper reviews from years ago and can apply to the entire Norwegian language, we believe the models ought to be tested on other datasets of diverse origin. The creation of further datasets, in particular large ones with a balanced number of datapoints for each label, would likely benefit the state of Norwegian sentiment analysis in general, in addition to potentially increasing performance on this particular task.

The relative performances between models on ternary and mixed-label datasets is also an area worthy of further inquiry. It would be of particular value to compare performance on ternary/mixed-label variants of the same dataset, where in the mixed-label variant the mixed label is represented equally to the other labels.
We also believe there to be more unrealized gains in performance that could be discovered through further testing with ensembling methods such as stacking or boosting, as well as further testing of embedding methods for models.

Appendix A

Python Files

In our submission we include ten jupyter notebook files, which make up all significant code we've used throughout this project:

- V2train_NorBERT.ipynb is the file we've used for training NorBERT models when NorBERT is used for classification and not embedding.
- V2train_GRU.ipynb is the file we've used for training NorBERT-GRUs.
- V2train_BiGRU.ipynb is the file we've used for training NorBERT-BiGRUs.
- V2train_LSTM.ipynb is the file we've used for training NorBERT-LSTMs.
- V2train_BiLSTM.ipynb is the file we've used for training NorBERT-BiLSTMs.
- V2train_CNN.ipynb is the file we've used for training NorBERT-CNNs.
- V2train_NorT5.ipynb is the file we've used for training NorT5s.
- plot.ipynb is the file we've used for plotting results of ensembles.
- dataset.ipynb is the file we've used for post-processing the relabeled dataset.
- V2_BAGGING.ipynb is the file we've used for aggregating votes from individual models to create and test ensembles, and find the average performance of singular models.

We've included one training file per model type, despite testing on four different datasets. Our reason for doing so is that the only code that needs to be changed depending on which dataset one loads in are: Number of classes if going from mixed-label to ternary or vice versa, change the line of code which loads in the particular dataset, and how the columns are relabeled when going from the relabeled dataset to the original dataset or vice versa.

Appendix B

Datasets

The mixed-label and ternary datasets we have relabeled and post-processed are included in the submission.They are included as JSON files within two folders, one folder for the mixed-label version and one for the ternary version. Each dataset is in three separate files, one of which is the test set, one is the training set, and one is the validation set.

Bibliography

[1] Robert E. Banfield et al. “Ensemble diversity measures and their application to thinning.” In: *Information Fusion* 6.1 (2005). Diversity in Multiple Classifier Systems, pp. 49–62. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2004.04.005. URL: https://www.sciencedirect.com/science/article/pii/S1566253504000387.

[2] Adam Bermingham and Alan F. Smeaton. “A study of inter-annotator agreement for opinion retrieval.” In: *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’09. Boston, MA, USA: Association for Computing Machinery, 2009, pp. 784–785. ISBN: 9781605584836. DOI: 10.1145/1571941.1572127. URL: https://doi.org/10.1145/1571941.1572127.

[3] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258 [cs.LG].

[4] Ramalingaswamy Cheruku et al. “Sentiment classification with modified RoBERTa and recurrent neural networks.” In: *Multimedia Tools and Applications* 83.10 (2024), pp. 29399–29417. ISSN: 1573-7721. DOI: 10.1007/s11042-023-16833-5. URL: https://doi.org/10.1007/s11042-023-16833-5.

[5] Jacob Cohen. “A coefficient of agreement for nominal scales.” In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46.

[6] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[7] GitHub and OpenAI. *GitHub Copilot*. AI pair programmer. GitHub, Inc. and OpenAI, 2021.

[8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613. URL: https://books.google.no/books?id=-s2MEAAAQBAJ.

[9] Andrey Kutuzov et al. “Large-Scale Contextualised Language Modelling for Norwegian.” In: *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*. Ed. by Simon Dobnik and Lilja Øvrelid. Reykjavik, Iceland (Online): Linköping University Electronic Press, Sweden, May 2021, pp. 30–40. URL: https://aclanthology.org/2021.nodalida-main.4.

[10] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[11] Mary L McHugh. “Interrater reliability: the kappa statistic.” In: *Biochemia medica* 22.3 (2012), pp. 276–282.

[12] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[13] Saif Mohammad. “A Practical Guide to Sentiment Annotation: Challenges and Solutions.” In: *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Ed. by Alexandra Balahur et al. San Diego, California: Association for Computational Linguistics, June 2016, pp. 174–179. DOI: 10.18653/v1/W16-0429. URL: https://aclanthology.org/W16-0429.

[14] GSN Murthy et al. “Text based sentiment analysis using LSTM.” In: *Int. J. Eng. Res. Tech. Res* 9.05 (2020).

[15] OpenAI. *ChatGPT (Version GPT-4)*. Large Language Model. OpenAI, 2024.

[16] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

[17] Juri Opitz and Sebastian Burst. *Macro F1 and Macro F1*. 2021. arXiv: 1911.03347 [cs.LG].

[18] Lilja Øvrelid et al. "A Fine-grained Sentiment Dataset for Norwegian." English. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Ed. by Nicoletta Calzolari et al. Marseille, France: European Language Resources Association, May 2020, pp. 5025–5033. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.618.

[19] Lutz Prechelt. "Early Stopping — But When?" In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_5. URL: https://doi.org/10.1007/978-3-642-35289-8_5.

[20] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. arXiv: 1910.10683 [cs.LG].

[21] Julian Risch and Ralf Krestel. "Bagging BERT models for robust aggression identification." In: *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*. 2020, pp. 55–61.

[22] David Samuel et al. "NorBench – A Benchmark for Norwegian Language Models." In: *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*. Ed. by Tanel Alumäe and Mark Fishel. Tórshavn, Faroe Islands: University of Tartu Library, May 2023, pp. 618–633. URL: https://aclanthology.org/2023.nodalida-1.61.

[23] David Samuel et al. *Trained on 100 million words and still in shape: BERT meets British National Corpus*. 2023. arXiv: 2303.09859 [cs.CL].

[24] Kian Long Tan, Chin Poo Lee, and Kian Ming Lim. "RoBERTa-GRU: A Hybrid Deep Learning Model for Enhanced Sentiment Analysis." In: *Applied Sciences* 13.6 (2023). ISSN: 2076-3417. DOI: 10.3390/app13063915. URL: https://www.mdpi.com/2076-3417/13/6/3915.

[25] Kian Long Tan et al. "RoBERTa-LSTM: a hybrid model for sentiment analysis with transformer and recurrent neural network." In: *IEEE Access* 10 (2022), pp. 21517–21525.

[26] Kai Ming Ting. "Confusion Matrix." In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 209–209. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_157. URL: https://doi.org/10.1007/978-0-387-30164-8_157.

[27] Kai Ming Ting. "Precision and Recall." In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 781–781. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_652. URL: https://doi.org/10.1007/978-0-387-30164-8_652.

[28] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[29] Erik Velldal et al. *NoReC: The Norwegian Review Corpus*. 2017. arXiv: 1710.05370 [cs.CL].

[30] Yige Xu et al. "Improving bert fine-tuning via self-ensemble and self-distillation." In: *arXiv preprint arXiv:2002.10345* (2020).

[31] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding." In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.