Norwegian University
of Life Sciences

Master's Thesis 2023 30 ECTS
Faculty of Science and Technology

Tackling Lower-Resource Language Challenges: A Comparative Study of Norwegian Pre-Trained BERT Models and Traditional Approaches for Football Article Paragraph Classification

Eirik Duesund Helland
Data Science

Abstract

In lower-resource language settings, domain-specific tasks such as paragraph classification of football articles present significant challenges. Traditional machine learning models face difficulties in effectively capturing the linguistic complexities inherent in the paragraphs, emphasizing the need for more advanced approaches to overcome these obstacles. This thesis investigates the potential of Norwegian pre-trained BERT (Bidirectional Encoder Representations from Transformers) models for paragraph classification tasks in the context of Norwegian football articles, a domain requiring a nuanced understanding of the Norwegian language. BERT is a powerful model architecture for language-specific processing tasks, which learns from the context of words in a sentence in both directions. Specifically, this thesis compares the performance of Transformer-based BERT models with traditional machine learning models in multi-class and multi-label classification tasks. An existing dataset of about 5,500 football article paragraphs is utilized to evaluate multi-class classification results. In addition, a newly annotated multi-label dataset of just over 2,000 samples is introduced for the multi-label classification assessment. The results reveal promising performance for the Norwegian pre-trained BERT models in both classification tasks, achieving an accuracy of ~ 0.88 and a weighted-average F1-score of ~ 0.87 in the multi-class classification task and accuracy of ~ 0.40 and a weighted-average F1-score of ~ 0.58 in the multi-label classification task, significantly outperforming the results of the traditional machine learning models. This study highlights the effectiveness of Transformer-based models in lower-resource language settings. It emphasizes the need for continued research and development in Natural Language Processing for underrepresented languages.

Contents

List of Figures

List of Tables

Nomenclature

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
GELU	Gaussian Error Linear Unit
GPT	Generative Pre-Trained Transformer
GRU	Gated Recurrent Unit
GUI	Graphical User Interface
KNN	K-Nearest Neighbors
LAMB	Layer-Wise Adaptive Moments
LSTM	Long Short-Term Memory
LSVC	Linear Support Vector Machine
MLM	Masked Language Modelling
MLP	Multilayer Perceptron
NER	Named Entity Recognition
NLP	Natural Language Processing
NoReC	Norwegian Review Corpus
NSP	Next Sentence Prediction
POS	Part Of Speech
ReLU	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

SVM Support Vector Machine

TF-IDF Term Frequency - Inverse Document Frequency

VAR Video Assistant Referee

Chapter 1

Introduction

In today's data-driven world, the capability to process vast quantities of textual information is crucial for making informed decisions and unlocking valuable insights across various industries and sectors. As a result of the ever-growing textual information available, Natural Language Processing (NLP) has emerged to make sense of unstructured data, facilitating advancements in fields such as sentiment analysis, machine translation, and information extraction, among others [1–3]. These advancements have primarily been introduced to solve English NLP problems. Still, there have recently been efforts to extend state-of-art techniques to languages such as Norwegian, which have significantly fewer language resources available. Norwegian is defined as a lower-resource language in this thesis as Norway has a population of only 5.5 million people and even fewer native speakers, and the language approximately only makes up 0.1 % of the content on the internet, compared to the English language making up an estimate of 55.6% [4, 5].

This thesis explores the potential of Norwegian pre-trained Transformer models in tackling text classification on paragraphs from Norwegian football articles. By investigating and comparing the performance of these Transformer models with traditional machine learning techniques, this research seeks to demonstrate the effectiveness of Transformer models for lower-resource languages on a domain-specific task and contribute to the growing body of knowledge in the field.

1.1 Motivation

NLP is an interdisciplinary field in computer and data science focusing on the interaction between human language and machines. In recent years, NLP has gained immense popularity and importance due to its ability to process and understand large amounts of text, opening up new opportunities for governments, businesses, and individuals alike. The recent popularity in the field of NLP can be credited to the emergence of powerful Transformer models. The most known of these models is ChatGPT, which recently set the record for being the fastest-growing consumer application in history [6].

Transformer architectures, such as Bidirectional Encoder Representations from Transformers (BERT), Generative Pre-Trained Transformers (GPT), and T5, have revolutionized the field of NLP by achieving state-of-art performance on a variety of language tasks. These tasks include sentiment analysis, text classification, text summarization, and machine translation, among others [7–9]. Pre-training these models on large-scale corpora enables them to learn complex linguistic features and relationships. Initially, these models were predominantly pre-trained on English corpora, leading to superior performance in English NLP tasks. However, recent efforts have also been directed towards training Transformer models for lower-resource languages, such as Norwegian [10, 11].

The importance of Transformer models for lower-resource languages boils down to their ability to improve performance and efficiency in NLP tasks for languages with limited available data for modeling. Transformer-based models can learn from unannotated data by training on self-supervised learning tasks such as Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) [7]. These tasks ensure that the model acquires a general understanding of a language and its context. Pre-trained models can be further fine-tuned on smaller labeled datasets to perform specific language tasks.

Paragraph classification of Norwegian sports articles is a specific task that involves a general understanding of the Norwegian language and its context to be achieved efficiently. A model capable of accurately classifying the different types of information found in football article paragraphs can impact the writer, editor, and reader of such articles. These impacts range from enhanced content personalization and improved search functionality for the reader to streamlined content curation and advanced analytics and insights for the news outlet. The application of Transformer models has the potential to unlock these benefits by optimizing performance in the paragraph classification task.

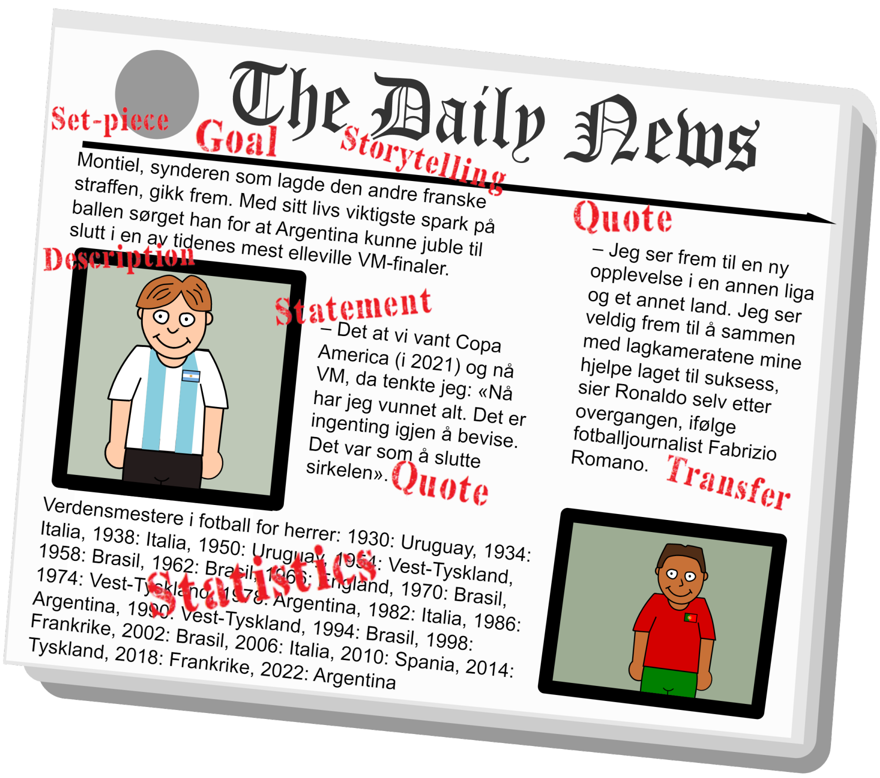


Figure 1.1: Illustration of paragraphs from news articles and their suggested labels.

Paragraph- or text classification, in the context of classifying information types, can be categorized in two ways:

- **Multi-class classification:** In multi-class classification, each text can be classified as multiple different output classes, but each text can only be assigned precisely one class. Multi-class classification is optimal when all labels are mutually exclusive.
- **Multi-label classification:** In multi-label classification, there are multiple possible output classes for each text. Each text can be assigned to one or multiple output classes simultaneously. Multi-label classification is particularly suitable when categorizing a text into a single label is difficult. In these situations, multiple labels can more accurately represent the diverse aspects of the text content.

Figure 1.1 illustrates football article paragraphs labeled with multiple labels as in multi-label classification. Reducing labels to only one for each paragraph results in multi-class classification. This study will include both the tasks of multi-class classification and multi-label classification of the different types of information that can be retrieved from a paragraph in a football article. The multi-class classification will be carried out on the Arx dataset introduced by Nordskog et al. [12], comprising of football article paragraphs from TV2.no and VG.no [13, 14]. For the multi-label classification, a new dataset, referred to as the Multi-Label Dataset, will be annotated and introduced, containing football article paragraphs from VG.no [14].

The use of Transformer models on lower-resource languages is in its early stages. This thesis aims to explore and demonstrate the use of different BERT models pre-trained on Norwegian data at various text classification tasks. The objective is to showcase the versatility and effectiveness of Transformer models for a lower-resource language, such as Norwegian, using datasets comprising paragraphs from online football articles. This research will further validate the progress made in NLP in recent years and provide substance to Transformers as the solution to solving natural language tasks for lower-resource languages.

1.2 Related Work

Natural Language Processing, as a field, has had significant advancements in recent years, primarily driven by the introduction and development of Transformer-based models. The introduction of the Transformer by Vaswani et al. [15] has paved the way for multiple new model architectures such as BERT, GPT, and T5 [7, 9, 16]. These models have all achieved state-of-art performance on various NLP tasks. The application of these Transformer models has been explored at multiple tasks, including sentiment analysis, text classification, text summarization, and machine translation [17–20]. These models have been successful at learning the complex linguistic relationships in texts from large corpora, applying the generally learned language representations to downstream tasks in NLP, and improving the accuracy of those tasks in the process.

Recent efforts have been made to improve the results of NLP tasks in lower-resource languages, such as Finnish, Sundanese, Swedish, and Norwegian, with the help of large Transformer models trained in the respected languages [10, 11, 21–23]. These models are often tested on token classification tasks such as PoS tagging and NER in their introduction. Still, fewer models are tested on other NLP tasks, such as text classification. There are some exceptions, such as the effort by Holmer and Jönsson [24] to test the Swedish BERT models at Swedish text classification and the actions of Virtanen et al. [21] to test the Finish model for Finnish text classification.

The Norwegian BERT models NB-BERT and NorBERT, introduced by Kummervold et al. [10] and Kutuzov et al. [11], respectively, have presented a new solution to tackle Norwegian NLP problems. The models have been tested, among others, on sentiment analysis using a subset of the Norwegian Review Corpus (NoReC) called NoReC$_{FINE}$ [25]. The models performed significantly better than the multilingual trained BERT model by Devlin et al. [7] on that specific task.

Various studies have explored the potential of machine learning for text classification in the context of news articles, especially in the sports category. Research has demonstrated the effectiveness of different machine learning algorithms for classifying news articles in specific languages like Uzbek and Bengali [26, 27]. Other tasks have also been targeted, such as classifying sports bloggers' entries into appropriate sports categories [28]. A machine learning-based sport-athlete evaluation model has also been proposed, combining game statistics and qualitative analyses from news articles [29]. These studies underline the growing significance of machine learning techniques in the news article and sports domain.

This thesis tackles text classification of information types found in Norwegian football news articles. The topic has been explored by Nordskog et al. [12], who made a comparison between the deep learning architectures, Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs), versus traditional machine learning algorithms, such as Naive Bayes and Support Vector Machine (SVM). The deep learning algorithms achieved slightly better scores at the evaluation metrics, but the training times of the models were significantly longer, and the setup was more complex, making the results more of a mixed blessing.

The work mentioned in this section makes up the foundation of this thesis. The Transformer architecture and the BERT iteration of the architecture, applied in the large Norwegian pre-trained models, will be utilized in this thesis. The task is to train and compare traditional machine learning models and the Transformer-based deep learning model, BERT, on the task of text classification of Norwegian football articles. The aim is to provide results that can be more definite in the performance gained by harnessing the power of the latest introduced models in deep learning, especially for a lower-resource language such as Norwegian.

1.3 Objectives

The primary goal of this thesis is to demonstrate the potential of Norwegian pre-trained BERT models for text classification on paragraphs from Norwegian football articles. The performance of Norwegian pre-trained BERT models and traditional machine learning models will be compared at multi-class and multi-label classification to explore the potential. The multi-class classification performance will be evaluated on an existing dataset of football article paragraphs, called Arx, introduced by Nordskog et al. [12]. To evaluate the models on multi-label classification, a new dataset containing paragraphs from football articles will be multi-label annotated and presented as the Multi-Label Dataset.

The overall objective is to evaluate the ability of the Transformer-based BERT model to accurately and efficiently process natural language for text classification in a lower-resource language setting where there is limited data available. To achieve the objective, the performance of Norwegian pre-trained BERT models will be compared to traditional machine learning models. The product of this thesis will be an analysis of the model training results at two different football paragraph classification tasks, including evaluation metric scores, resource utilization, and ease of implementation.

1.3.1 Research Question

Given the objectives, the core research question of this thesis is:

> *"How can Norwegian pre-trained BERT models be utilized for effective text classification on paragraphs from Norwegian football articles, and how does their performance compare to traditional machine learning models?"*

This question primarily addresses two aspects. First, it aims to evaluate the effectiveness of Norwegian pre-trained BERT models in the specific task of text classification, focusing on paragraphs from Norwegian football articles. Given the success of BERT models in English NLP tasks, exploring their potential in a lower-resource language like Norwegian can shed light on their universal applicability and versatility.

The second part of the question compares the performance of these models with traditional machine learning models. This comparative analysis can provide valuable insights into whether the cutting-edge BERT models offer significant improvements over established methods in a lower-resource language context.

To answer this question, Norwegian pre-trained BERT models will be fine-tuned to datasets of Norwegian football articles, and their performance will be evaluated in classifying the text. The results will be compared with those from traditional machine learning models applied to the same task.

1.3.2 Scope and Limitations

The scope of the research is limited to paragraph classification within the domain of football articles. Although the findings may have implications for other domains, the specific focus on football articles allows for a targeted investigation into the effectiveness of Norwegian pre-trained BERT models for this particular task.

The research is conducted on limited datasets for both the multi-class and the multi-label classification. The Arx dataset, consisting of approximately 5,500 football article paragraphs, is utilized for multi-class classification evaluation, while the newly annotated Multi-Label Dataset comprising just over 2,000 samples is introduced for the multi-label classification assessment. The datasets' relatively small size may impact the results' generalizability to larger-scale applications.

The research assumes the availability of pre-trained Norwegian BERT models and does not delve into the difficulties in the training process of these models. The pre-trained models are utilized as a starting point for fine-tuning and evaluating these BERT models at the paragraph classification tasks.

A semester-long master thesis inherently carries certain limitations due to its constrained time frame. The short duration limits extensive data collection and thorough model selection and experimentation, necessitating a focus on a narrow research question. The set time frame constrains the possibility of delving into too many additional experiments. Despite these constricts, there are great opportunities to contribute to exploring the potential of Norwegian pre-trained BERT models.

1.3.3 Contributions

This thesis aims to contribute to the small body of research on Norwegian pre-trained BERT models. By exploring the effectiveness of BERT models for lower-resource languages, it will be possible to determine if NLP has reached a satisfactory level for sustainable solutions to comprehensive text processing problems. Hopefully, the efforts of this thesis will inspire further research in the field of NLP for lower-resource languages and further encourage the development of pre-trained Transformer models for multiple other languages.

The specific contributions of this thesis are:

- A demonstration of the potential of Norwegian pre-trained BERT models at text classification of football articles.
- The annotation and introduction of a multi-label dataset containing paragraphs from football articles to present a more challenging and relevant domain-specific NLP problem for the Norwegian pre-trained BERT models.
- An analysis of the performance of Norwegian pre-trained BERT models compared to traditional machine learning models for the tasks of multi-class and multi-label classification of football articles.

Chapter 2

Theory

The theoretical foundation of this thesis provides the essential background knowledge to navigate the research presented in subsequent chapters. By offering an overview of relevant development in the field, models, preprocessing steps, and evaluation metrics, this chapter seeks to create a theoretical framework for this thesis, ensuring the reader is equipped to contextualize the research findings and their significance.

2.1 Natural Language Processing

Natural Language Processing (NLP) is a subfield of computer science and artificial intelligence that concerns the interaction between computers and human language. The goal of NLP is to enable machines to process, understand and generate human language. NLP involves a range of techniques, including rule-based systems, statistical methods, and deep learning, each representing different ways to tackle natural language tasks [30]. NLP is applied to many problems, such as text classification, chatbots, machine translation, and speech recognition [31–34].

The development of NLP can be divided into three waves. The first is the rational wave, which relies on expert domain knowledge and rule-based systems in traditional computer science fashion [35]. Weizenbaum introduced ELIZA, the first chatbot, in the 1960s, a great example of a rule-based system for processing natural language [36]. ELIZA simulates a psychotherapist that tries to engage a patient by reflecting their statements back at them. ELIZA knows nothing of the real world, but by searching for ranked keywords and utilizing pattern transformation, ELIZA can formulate human-like responses to statements.

User:	Men are all alike.
ELIZA:	IN WHAT WAY
User:	They're always bugging us about something or other.
ELIZA:	CAN YOU THINK OF A SPECIFIC EXAMPLE
User:	Well, my boyfirend made me come here.
ELIZA:	YOUR BOYFRIEND MADE YOU COME HERE
User:	He says I'm depressed much of the time.
ELIZA:	I AM SORRY TO HEAR YOU ARE DEPRESSED
User:	It's true. I am unhappy.
ELIZA:	DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
User:	I need some help, that much seems certain.
ELIZA:	WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
User:	Perhaps I could learn to get along with my mother.
ELIZA:	TELL ME MORE ABOUT YOUR FAMILY
User:	My mother takes care of me.

Figure 2.1: A snippet of dialogue with ELIZA, the first rule-based chatbot, simulating a psychotherapist, demonstrated by Weizenbaum [36].

The second wave of development in NLP, the empiric wave, is characterized by the use of machine learning and the statistical analysis of natural language to discover patterns in texts [37]. The idea of representing text documents as unordered collections of words became a part of the NLP field, and methods such as Bag-of-Words (BoW) became fundamental concepts, which, in short, is a way to represent texts as frequency distributions of words [38]. Maron introduced the Naive Bayes algorithm in 1961 to use the occurrence of cue words in text documents to classify the subject categories of each text [39]. The advent of Naive Bayes exemplifies the empiric approach to NLP, where the statistical properties of words in texts are leveraged. This methodology was state-of-art in NLP until very recently [35]. Still, it has limitations in handling complex linguistic structures and large-scale data, which has prompted further development of more advanced techniques.

The third and current wave in NLP is the application of deep learning. Deep learning has made an impact on NLP by mimicking biological neural networks. Deep learning models such as Recurrent Neural Networks (RNNs) and, more recently, Transformers, can handle a language's complex linguistic structures by capturing more contextual information [38]. Their approach goes beyond the shallow treatment of words as numbers, as in techniques such as BoW. Instead, deep learning models learn a representation of words and phrases from large corpora, which allows them to capture the semantics of sentences and long-range dependencies of words [38]. This provides properties of generalization that can result in significant advances in NLP for lower-resource languages such as Norwegian. Deep learning is used to train models to understand the contextual properties of words, similar to how a person would learn them from reading a corpus. Deep learning models have become more common to use as the availability of large datasets has risen and high-resource computational units have become more obtainable [35].

The first breakthrough during the third wave came with the introduction of the Word2Vec model, which significantly reduced the resources needed to train deep learning models [40, 41]. Word2Vec was able to capture syntactic and semantic relationships between words using word embeddings. These embeddings are short, dense vectors that enable classifiers to learn fewer weights than earlier. These dense vectors proved to work better at a large variety of NLP tasks thrown at them [42].

Word embedding models proved very powerful, but it was the introduction of Transformer models, such as BERT [7] and GPT [16] that represents a significant advance in NLP in recent times. Transformer models allow for modeling longer sequences of texts and capturing their contextual information into word representations. Transformer models utilize a self-attention mechanism that weighs the importance of different words in context, allowing the model to capture long-range dependencies and complex relationships within the text effectively.

This third wave of development has made the use of NLP models mainstream. The popular chatbot, ChatGPT [43] has spectacularly entered the market. ChatGPT, by reaching 100 million monthly active users in two months, has become the fastest-growing consumer application throughout history [6]. This further states the importance of Natural Language Processing and the continued research in the field.

2.1.1 Common challenges in NLP

NLP has been rapidly evolving, and the most recent models in the field seem to understand and generate human language easily. Despite the achievements accomplished in recent years, NLP still faces several challenges that must be overcome to make these models better, especially for lower-resource languages.

One of the significant challenges in NLP is handling large, unstructured datasets. NLP models require vast amounts of data to learn linguistic patterns and capture the context of human communication. The data used to train NLP models often comes from books, web pages, or speeches. These unstructured collections of texts can be inconsistent, lack context, or be multilingual, to name some challenges. All of these challenges can impact the model, as they are hard to discover and fix.

Another major challenge in NLP is the high dimensionality of language data. To represent text numerically, the only way a machine learning model can understand it, the text is usually represented as a high-dimensional vector [38]. The text can be defined as a fixed-length vector where each of the dimensions of the vector corresponds to a word in the corpora of texts. A short sequenced text would have the same length vector as a longer text, but all words from the vocabulary which are not used will be represented as 0. This makes the vector space large, making learning and generalizing from the data much more complex for models.

2.2 Machine Learning

Machine learning refers to computer programs designed to learn from experience to complete specific tasks. Their performance is measured by their ability to do so. In the context of machine learning, the "experience" an machine learning model learns from is represented by a training dataset, which includes input and output pairs. By analyzing these examples, the model learns to recognize patterns and generalize the patterns to new, unseen data. Machine learning can be compared to how people learn and adapt to new situations. Some typical machine learning applications include self-driving cars, fraud detection, and personalized ads and recommendations [44–47].

Key differences distinguish the abilities of humans and machines to learn. Machines require specific training data to perform different tasks. The higher quality examples it uses to learn, the more adept the machine learning model will be at handling the task. Machine learning is exceptionally proficient in situations where hindsight is valuable, as it can identify patterns that indicate potential outcomes far earlier than humans, resulting in greater predictive performance [48].

2.2.1 Statistical Methods in NLP

Statistical methods have been widely used in the field of NLP. Some common techniques used are Naive Bayes, Support Vector Machines, Logistic Regression, and Hidden Markov Models. These models are all based on mathematical models that use statistical interference to learn patterns in the language data.

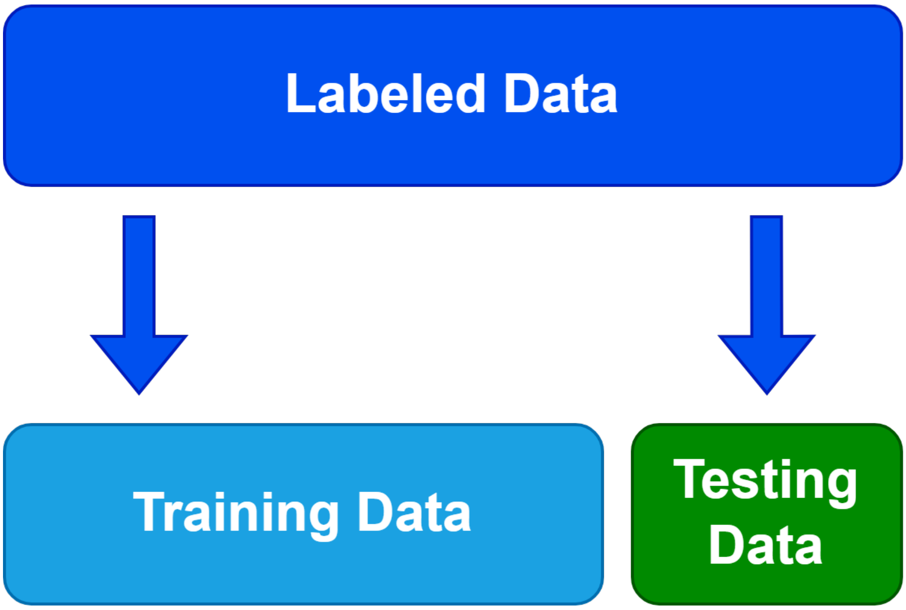


Figure 2.2: A dataset partitioned into a training and testing split.

The models are trained on a large amount of labeled data, which are input and output pairs, split into training data and testing data as seen in Figure 2.2. This is called the train-test-split and is a fundamental technique in machine learning. Typically 70/80 % of the data is used for training, with the remaining data being used for testing. The training data is fed to the model, and each model's parameters are adjusted to minimize prediction errors. Afterward, the model is assessed by making predictions on the test set, which helps evaluate its performance on data it has not encountered before. This is the process for both text data and other types of data, but text data must be represented in a format that the statistical models can receive. This is where text vectorization comes into play.

Text Vectorization

Text vectorization is a processing step that converts textual data to numerical vectors that can be used as input for machine learning algorithms. These vectors are usually high-dimensional, where each dimension represents a feature from the text dependent on which text vectorization technique is used. The most common techniques in NLP have been the Bag-of-Words (BoW) technique and the Term Frequency-Inverse Document Frequency (TF-IDF) technique.

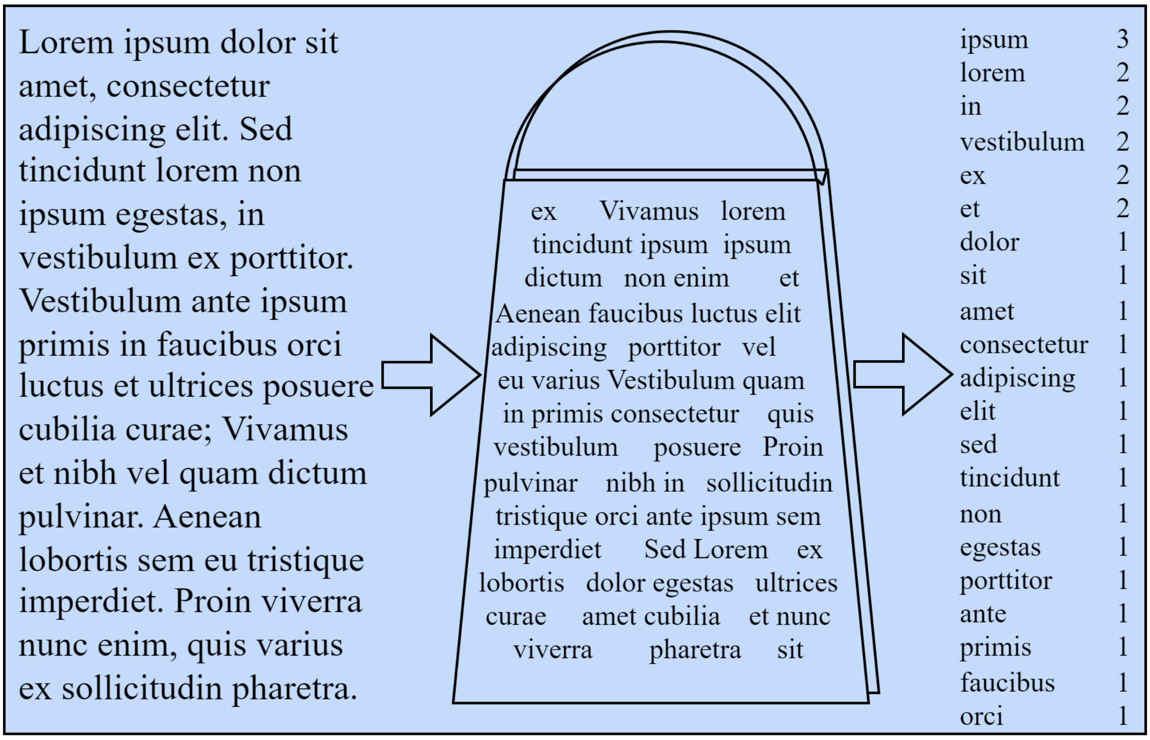


Figure 2.3: A demonstration of the Bag-of-Words technique. Figure showing how a text goes from ordered to scramble, like in a bag, to a list of word frequencies. Figure inspired by Jurafsky and Martin [49].

BoW is a simple vectorization technique that involves representing each text as a vector of word frequencies [38]. As Figure 2.3 illustrates, BoW shuffles a structured text as though placing the words into a bag and then stores the word occurrences in a vector. The vector dimension is equal to the vocabulary size, and each value in the vector represents the number of occurrences of each word in the text.

TF-IDF is similar to BoW, but it captures the importance of a word in a document relative to its frequency of occurrence in other documents [38]. The TF-IDF score is the product of how often a word appears and the rarity of the word across the whole document collection. The higher the TF-IDF score of a word in a text, the more likely the word is to be essential and informative.

Overfitting

Overfitting is a general issue in machine learning, characterized by the model learning the training data so thoroughly that it struggles to generalize to new, unseen data [38]. This results in the model not learning underlying patterns in the data but adapting to learning the noise and obscurities of the specific training data instead. In the case of overfitting, the model has become too complex, which means it will not generalize to new data. This is why the dataset used for training a machine learning model is split into a training and a testing set.

A less common but problematic issue is when the model overfits the testing data. This can happen due to excessive parameter tuning, leading to a set of parameters and a model that performs extraordinarily well at evaluating the specific test split. These parameters might translate poorly to performance on other unseen data.

The best way to prevent overfitting is to employ cross-validation [38]. A good choice of cross-validation is k-fold cross-validation, which evaluates the model using different subsets of the data. K-fold cross-validation limits the use of the test data for finding the best model parameters. This method divides data into training-, validation-, and testing data. The testing split is set aside for the ultimate testing of the model. The other two divisions are chosen from the remaining data in a user-specified number of iterations where the validation and training splits differ. Models are trained using the different training- and validation splits and the average performance of the various models is used to find parameters. Using k-fold cross-validation ensures that the parameters are not overfitting the model to a specific data split. The best parameters found are used to train a model on the combination of the training- and validation data. The resulting model is evaluated on the testing data to check if the model can be generalized to unseen data.

The Sparsity Problem

Another challenge that arises for statistical machine learning methods is the sparsity problem [38]. The sparsity problem, which is frequent in NLP, occurs because the feature space is large, but the data is typically sparse. This can be explained as some words in a text dataset occurring so infrequently that the model cannot learn the patterns they form. This causes many words to have low frequency and, therefore, an insignificant impact on the model. This can cause the model to overfit the training data, learning the noise in the data instead of the patterns, resulting in poor performance on new, unseen data. The sparsity in the vector space, which is the numerical representation of the text data, also makes the model more computationally expensive to train and evaluate, mainly when dealing with many features.

Addressing the sparsity problem in traditional machine learning is done by carefully modifying the features inputted into the model. Stemming, lemmatization, and stop word removal are all methods that can help reduce the dimensions of features. Stemming reduces words to their base form by removing suffixes and prefixes. Lemmatization reduces words to their dictionary form (lemma) rather than the simple root form produced by stemming. For example, given the word "babies", stemming might produce "babi" as the root, whereas lemmatization would produce "baby". Stop word removal removes commonly used words that do not contribute to understanding the context, such as "but", "if" and "or". All these methods help reduce the dimensionality of the feature space and aim to improve the quality of the features. These methods are only suited for statistical machine learning models that do not consider word order.

2.2.2 Deep Learning

Deep learning is a technique that enables models to identify patterns in raw data by using non-linear modules to transform the data at multiple levels of abstraction [50]. This can be interpreted as deep learning models finding and learning different data features without human intervention. This provides universal learning, robustness, generalization capabilities, and scalability benefits, making it suitable for multiple domains without requiring precise feature engineering [51].

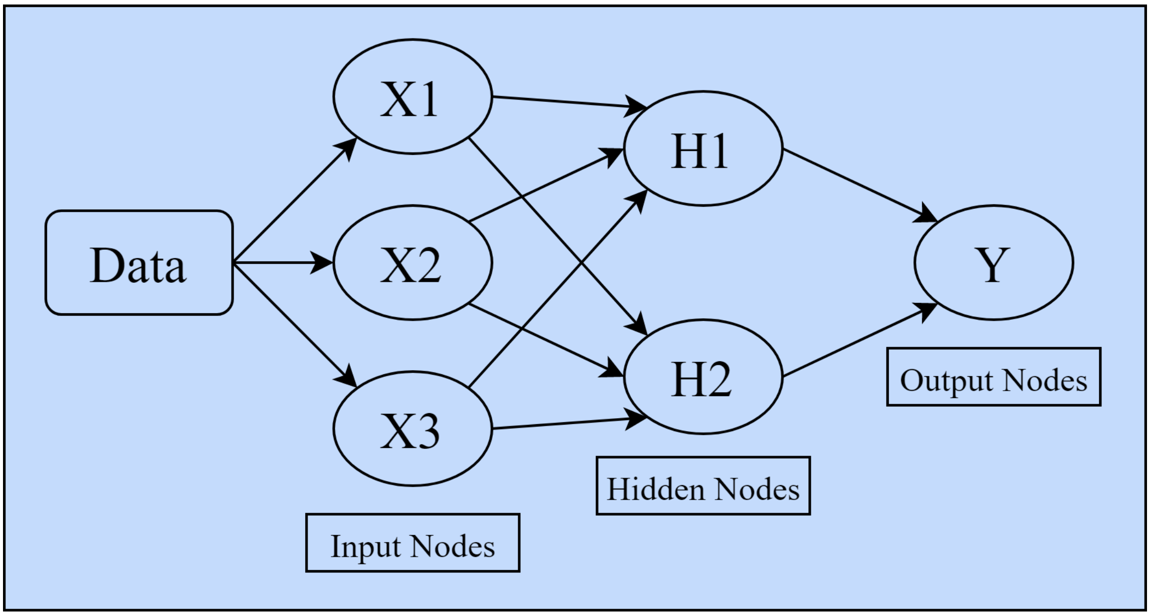


Figure 2.4: Raw data is processed through the different levels of a two-layer neural network. Figure inspired by Dinov [52].

A deep neural network, as depicted in Figure 2.4, is a computational model that consists of multiple layers of artificial neurons or nodes, inspired by the biological brain's functionality [52]. These layers process input data and generate an output, mimicking the brain's response to various stimuli and its ability for parallel processing.

The fundamental processing unit in this network is the artificial neuron [53]. The neuron is responsible for receiving inputs, processing them, and generating output. The inputs to a neuron are multiplied by corresponding weights, which are parameters within the network that define the strength of the connections between neurons. Weights are initially set to random values and are adjusted during the training process to reduce prediction errors [38].

Each neuron also has a bias, a scalar value that allows the activation function's output to be shifted. This bias plays a crucial role in improving model accuracy, as it allows the neuron to make better predictions.

The weighted inputs and bias are summed up and passed through an activation function. This function introduces non-linearity into the network, which helps the network learn complex patterns in the data [38]. The output of the activation function is given by:

$$a = f\left(\sum(\text{weights} \times \text{inputs}) + \text{bias}\right) \tag{2.1}$$

where a is the activation, f is the activation function, the sum is over all inputs to the neuron, *weights* is the weight parameters, and *bias* is the bias parameter.

Training a deep neural network involves repeatedly feeding data through the network for a specified number of iterations, commonly referred to as epochs [38]. At each epoch, the network's current output is compared to the targeted output, and the variation, quantified by a loss function, is utilized to update the network's parameters. The loss function L, which measures the difference between the network's predictions y_{pred} and the true labels y_{true}, in this case cross-entropy loss, is given as follows:

$$L = -\frac{1}{N} \sum_{i=1}^{N} y_{\text{true}_i} \log(y_{\text{pred}_i}) + (1 - y_{\text{true}_i}) \log(1 - y_{\text{pred}_i}) \tag{2.2}$$

where N is the number of samples, y_{true_i} is the true label, and y_{pred_i} is the predicted probability of the positive class.

An optimization algorithm that adjusts the weights and biases in the network is used to minimize this loss [38]. One of the key algorithms used in this optimization process is called backpropagation. Backpropagation calculates the gradient of the loss function with respect to each weight and bias in the network. The gradient, denoted by ∇, gives the direction in which the weights and biases should be adjusted to decrease the loss. The term "backpropagation" refers to how the adjustments are propagated across all the layers of the network, starting from the output layer and moving backward to the input layer [38]. This method ensures that each layer contributes to the adjustments relative to its contribution to the total error.

The general update rule for a weight vector w in the network using a learning rate α is:

$$w_{\text{new}} = w_{\text{old}} - \alpha \nabla L \tag{2.3}$$

where ∇L is the gradient of the loss function with respect to the weights in w. The learning rate α is a hyperparameter that controls the step size during each iteration in the gradient descent process. A suitable learning rate allows the network to converge to a minimum of the loss function efficiently [38].

An optimizer uses these gradients to perform the actual updates to the weights and biases. Different optimizers have different ways of applying these updates, often to achieve faster convergence or better generalization [38].

The Activation Function

The activation function is a crucial part of a neural network. The activation function activates the neurons in the network if the input is large enough and transforms the weighted and aggregated inputs to a new output, making the neural network capable of learning non-linear relations and complex tasks [52].

The simplest activation function is called the threshold function. The threshold function, represented by $f(x)$, operates based on the value of the input x, which is the aggregated weighted input to a neuron. The function is defined as follows:

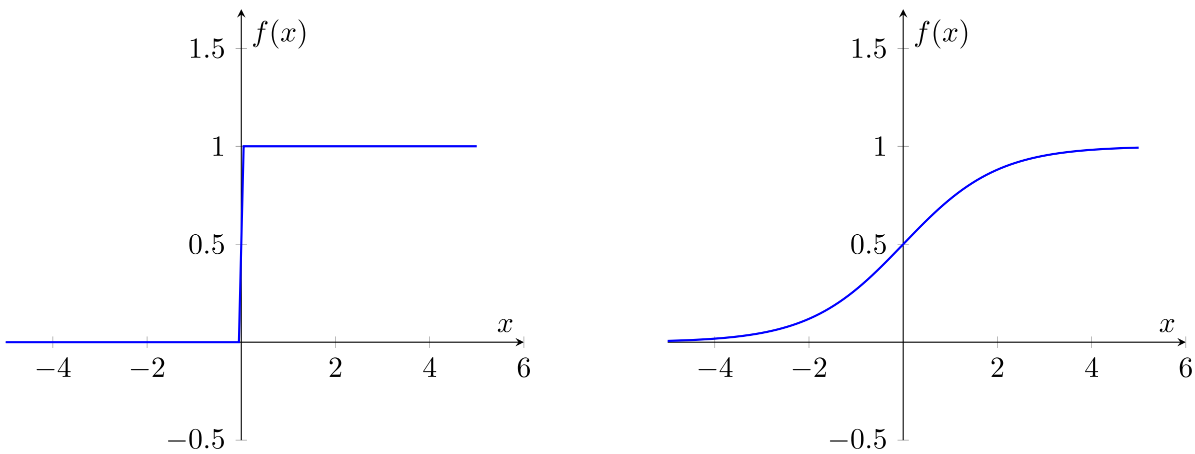
$$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases} \tag{2.4}$$

In the context of the neural network, this function is applied to the input of each neuron, and it outputs a binary result based on whether the input is less than or equal to zero, or greater than zero. The threshold function's basic binary decision serves as an essential foundation for comprehending more intricate activation functions.

The sigmoid function, represented by $f(x)$, smoothly transitions its output from 0 to 1 as the input x increases. This function is expressed as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

The sigmoid function is utilized to manage the neuron's output in a way that keeps values between 0 and 1, facilitating the interpretation of the output as a probability [38]. The smooth gradient of the sigmoid function also assists in the effective training of the neural network via backpropagation, as it provides a clear direction for the update of the weights.



(a) Threshold Activation Function

(b) Sigmoid Activation Function

Figure 2.5: Activation Functions

Equation 2.4 displays the simplest form of an activation function: The threshold function. While the threshold function does activate a neuron once a certain threshold is met, its binary nature limits its usefulness in many machine learning scenarios. This limitation arises from its inability to differentiate between input values that are far from or near the threshold, which can be critical in complex problems that require nuanced interpretations of the input data. More commonly used are continuous functions such as the sigmoid function (Equation 2.5), the tanh function, and the rectified linear unit (ReLU) function. These functions provide smoother changes to the inputs, causing smoother decision boundaries in classification and improved nonlinear mapping [54]. The difference between the binary step function and the sigmoid function is demonstrated in Figure 2.5.

The Optimizer

The optimizer is a numerical algorithm used to adjust the weights and biases of the model, to minimize the loss function. The optimizer utilizes the gradients to determine how to optimally update the weights and biases. An appropriate optimizer can significantly impact the performance and convergence speed achieved during training, gaining both model accuracy and efficiency.

The most basic optimizer is Stochastic Gradient Descent (SGD) which updates weights and biases by stepping proportional to the negative of the calculated gradient. The learning rate of the SGD optimizer, which is set before training, scales the negative gradient to control the step size. The chosen learning rate substantially impacts the convergence of the model and may hinder the optimization by getting stuck in local minima. Convergence means that the model has found a set of parameters that minimizes the loss functions.

More advanced optimizers, such as the Adaptive Moment Estimation (Adam) optimizer, reduce the risk of non-convergence by applying an adaptive learning rate [55]. The Adam optimizer also calculates a moving average of previous gradients that can be utilized to give momentum to the model that can help it pass through local minima and arrive at convergence faster. Another feature of Adam is using a moving average of the squared gradient to adjust weights and biases adaptively. This causes the Adam optimizer to be more efficient at finding the optimal weights and biases for the model. The Adam optimizer can also be more robust if a non-optimal learning rate is chosen from the start [55].

Even though Adam is the more robust choice of optimizer compared to SGD, it is still essential to choose an appropriate learning rate for the problem at hand and the model architecture. Choosing poorly can still lead to slow convergence or divergence. However, in general, the Adam optimizer provides better performance and faster convergence than the SGD optimizer.

The Vanishing Gradient Problem

The vanishing gradient problem is an issue that may occur in deep learning during training. The gradients calculated during backpropagation indicate how to adjust the weights and biases to achieve optimal performance. The vanishing gradient problem occurs when the gradient becomes very small as it passes through the network layers. The early layers of the network receive too small gradients to effectively update its weights and biases, leading to inefficient training and poor model performance [38].

One approach to address the vanishing gradient problem is using advanced activation functions, such as ReLU and its variants [38]. Careful weight initialization strategies, like Xavier or He initialization, and batch normalization can also help maintain stable gradients during training. Another solution is incorporating residual or skip connections that mitigates the problem by allowing gradients to flow directly to earlier layers. The Transformer architecture also addresses the vanishing gradient issue by employing the self-attention mechanism and layer normalization. This is further explained in Section 2.2.3.

Transfer Learning

Transfer learning is a method in machine learning which allows a model to be pre-trained on a large dataset to be able to be later adapted to related but different problems [56]. The idea is to leverage general knowledge about a domain during pre-training, improving performance when learning a new task where less data is available.

Transfer learning is most commonly associated with deep learning, where the lower layers of a network can learn to extract many valuable features from the data. This results in a pre-trained model, which often generalizes well when fine-tuning the models with new data for other tasks. Using pre-trained models can save significant time and computational resources, often leading to better generalization ability, especially when labeled data is limited [57].

Recurrent Neural Network

Recurrent Neural Network (RNN) is a neural network specialized in processing data sequences, such as time series or text data [38]. It can be characterized as a predecessor to the Transformer architecture for NLP tasks. It can capture sequential relationships and dependencies within sentences and paragraphs, but it also possesses some limitations. The contextual ability of RNNs is achieved by having hidden states, which are updated. At the same time, the network processes each element in the sequence in its original order, implementing a memory-like functionality. The name "Recurrent Neural Network" highlights that the information in the network is passed through several times in the memory of the hidden states, giving the model a sense of recurrence.

The major limitation of RNNs is the difficulty of capturing long-term dependencies due to the vanishing gradient problem [58]. The result is that the model forgets context from earlier parts of the text, which might be essential. The other limitation is that RNNs depend on processing the input data sequentially, which is computationally expensive and not parallelizable, resulting in an increased runtime [59].

The limitations of RNNs have been addressed with variants of RNNs such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. LSTM introduces a memory cell structure allowing more extended access to previously processed information [60]. This structure enables the LSTM network to selectively remember essential details and forget irrelevant information. GRU is a simplification of LSTM, which with fewer computations, achieves the same goal of remembering only the critical information [61]. These architectures provide RNNs with better tools for solving NLP problems, but the Transformer architecture has several advantages compared to LSTM and GRU [62].

2.2.3 Transformers

The Transformer architecture was introduced by Vaswani et al. [15] and has been groundbreaking due to its excellent performance and ability to process sequential data non-sequentially with the help of the attention mechanism. The attention mechanism was initially developed to enhance the performance of RNNs, by addressing the vanishing gradient problem by helping the model to focus on specific, relevant parts of the text sequence when generating output. It was introduced by Bahdanau, Cho, and Bengio [2]. The attention mechanism weights the importance of each token in a text in the context of the word being generated as output. The paper by Vaswani et al. [15] was called "Attention is all you need" and proposed to abandon the recurrent connections and only rely on the attention mechanism, with a few modifications. The Transformer models proved capable of being trained on vast amounts of textual data and learning general contextual representations of word sequences from almost any text before being fine-tuned to specific NLP tasks.

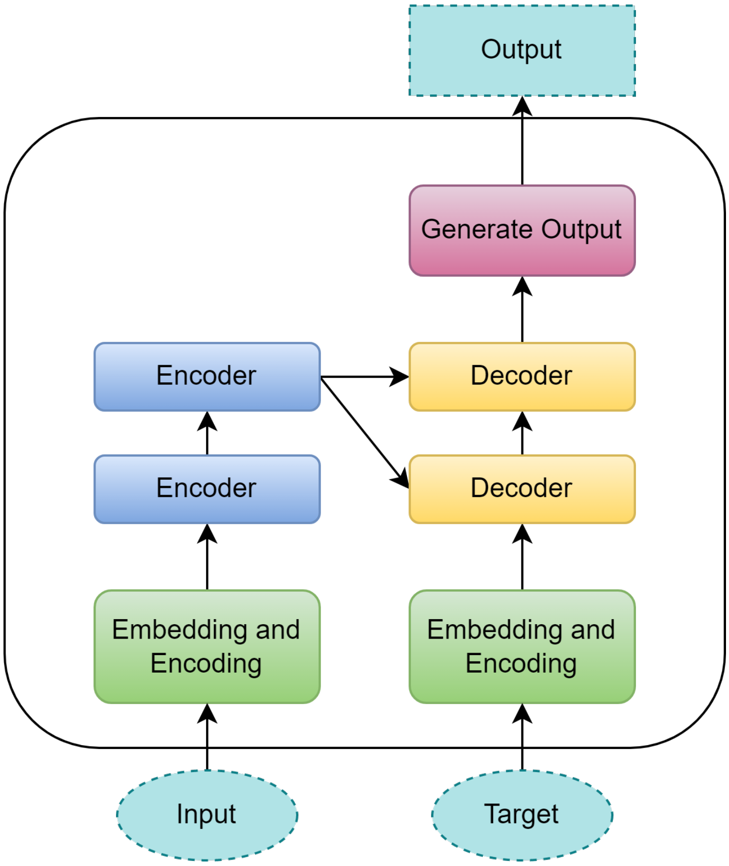


Figure 2.6: The general architecture of a Transformer as presented by Vaswani et al. [15].

The Transformer architecture comprises a series of Encoder and Decoder layers forming a structure as seen in Figure 2.6. The stack of Encoders and Decoders can comprise an arbitrary amount of each. Still, the original architecture by Vaswani et al. [15] utilizes six Encoders and six Decoders instead of the pair of two seen in Figure 2.6. Optimizing the numbers of Encoders and Decoders is done through hyperparameter tuning. Figure 2.6 also shows how the input of both the Encoder stack and the Decoder stack has its own Embedding and Encoding layer, which converts the inputted sequence of text into a vector space of a fixed length and adds positional information using positional encoding. The general task of the Encoder stack is to generate a sequence of hidden representations that encapsulates the contextual information of the input sequence. In the original Transformer architecture, the output of the Encoders is passed to the Decoder stack, generating an output sequence that aims to be contextually appropriate based on the information in the passed sequence. In short, the Transformer utilizes the Encoder and Decoder structure for the Encoder to ingest the inputted text and the Decoder to generate new text using the Encoder output.

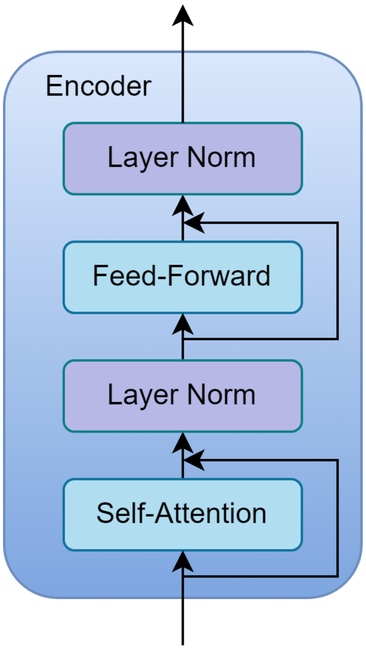


Figure 2.7: The layers and connections that make up the Encoder.

Each Encoder contains multiple layers responsible for processing the sequence and generating the contextual representations, as seen in Figure 2.7. The main components are the self-attention mechanism and a feed-forward neural network. The self-attention mechanism adapts the attention mechanism created by Bahdanau, Cho, and Bengio [2]. The self-attention mechanism works by computing a weighted representation of the inputted elements for each element in the sequence. It means that all elements get a numeric representation of the relationship between itself and all other elements in the sequence. This enables both parallel processing and contextual modeling.

The self-attention mechanism computes the weighted representation of the query (Q), the key (K), and the value (V) vectors, which are high-dimensional numeric representations derived from the input data. The function is expressed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.6}$$

The query contains the token in focus, the key comprises representations of the remaining token of the context, and the value vector carries the information to be extracted based on the similarity between the query and the key vectors. The dot-product is calculated between Q and the transposed K (K^T) to find the similarity between them, with a higher value indicating a higher similarity. The dot-product is divided by the square root of the dimensionality of K (d_k) to prevent it from being too large, which could lead to the vanishing gradient problem. The product of the computations until now is normalized using the softmax function, keeping the sum of similarity scores adding up to 1. This ensures that the self-attention mechanism focuses on the critical keys. The last step is multiplying the V vector with the normalized similarity scores from the previous steps. This is the self-attention mechanism's output, a relevance-weighted value vector. In practice, the vectors Q, K, and V are matrices with differently weighted versions of the vector. This enables the model to simultaneously calculate multiple attention scores that weigh different aspects of the input. This helps the model learn more complex patterns and relationships in the input sequence.

The feed-forward neural network in the Encoder further processes the attention weights created in the self-attention layer by applying non-linear transformations to the representations, which allows for capturing complex patterns and relationships in the input sequence. The architecture of the Encoder also contains a residual connection between the self-attention layer and the feed-forward layer, followed by a normalization layer. The residual connection is a layer bypass that allows the model to learn residual information and retain the input information better. The normalization layer is applied to improve the training and generalization performance of the model.

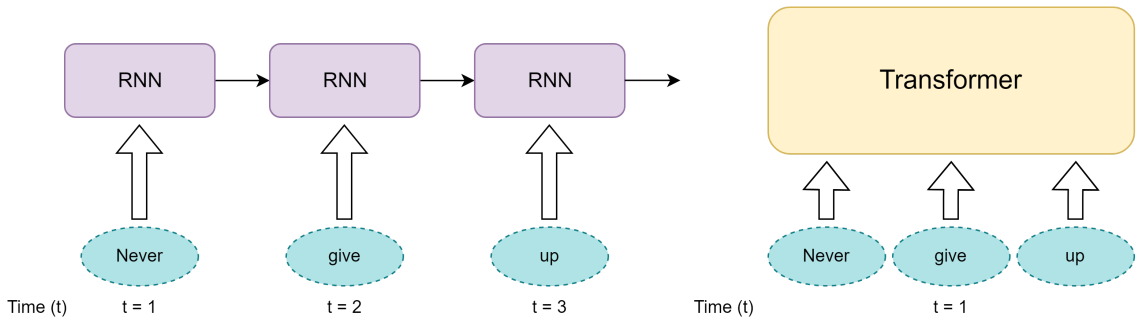


Figure 2.8: The sequential computations of an RNN model versus the parallel computations of a Transformer model showcased with the example sequence; "Never give up".

Transformers can perform parallel processing as opposed to RNNs, due to the application of the self-attention mechanism, eliminating the need for recurrent connections. This is beneficial, as seen in Figure 2.8, as it can reduce the training time of the model by utilizing multiple processing units by avoiding the recurrent connections [15]. Using multiple processing units means that the model can take advantage of modern hardware, such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), significantly speeding up computation times. The Transformer architecture is also better at capturing long-range dependencies than the LSTM and GRU architectures because it does not lose information over longer sequences [15]. Transformers are also easier to train than RNNs because they require less hyperparameter tuning or changes to the model architecture to achieve good results, compared to RNN architectures such as LSTM and GRU. All of these factors make Transformers a good choice for transfer learning. By pre-training the model on large corpora and learning general contextual language understanding, the Transformer models can easily be fine-tuned for more specific NLP tasks. These are all reasons why the NLP research field now considers Transformer models as the new de facto solution to many NLP problems [7, 15, 16].

2.2.4 BERT

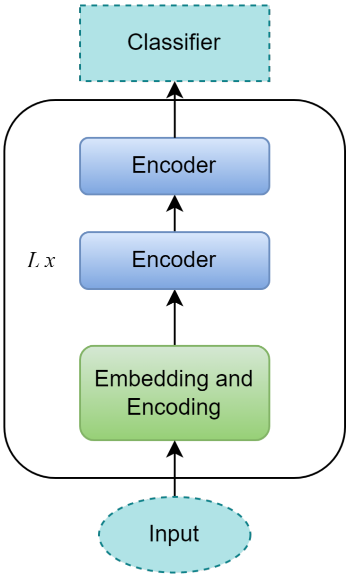


Figure 2.9: The general architecture of a BERT model. The number of Encoder layers can be changed.

The BERT model, short for the Bidirectional Encoder Representations from Transformers model, introduced by Devlin et al. [7] takes the concept of the Transformer and its architecture into their implementation of only the Encoder stack, as seen in Figure 2.9, in a new pre-trained deep learning model. The BERT model stands out from the original Transformer model by training to capture the bidirectional context of the sequence, thus evaluating the sequence in both directions. The bidirectional context understanding is learned through Masked Language Modeling (MLM). MLM is a self-supervised task where some tokens in a sequence are randomly masked before the model tries to predict the hidden token based on the surrounding tokens and the contextual information they present. The MLM task also makes the pre-trained BERT model easier to fine-tune for many NLP tasks that involve token classification, such as Named Entity Recognition (NER) and Part-of-Speech (POS) Tagging. The BERT model is also trained at Next Sentence Prediction, which involves learning the relationship between sentences by trying to predict if two given sentences are proceeded by each other. This makes the model achieve a better understanding of sentence-level semantics and coherence of text, which also provides benefits to different downstream tasks that the model can be fine-tuned for.

The BERT model presented by Devlin et al. [7] comes in two versions. The BERT$_{BASE}$ comprises 12 Encoder layers with a hidden size of 768 and a total of 110 million parameters. The BERT$_{LARGE}$, on the other hand, comprises 24 Encoder layers with a hidden size of 1,024, meaning there is a total of 1,024 neurons in each hidden layer. The model contains a total of 340 million parameters. The more significant number of layers and hidden size can increase the model's capacity and allow it to learn more intricate patterns at the cost of more computational resources. Both the original BERT model was trained on English text from books and the English Wikipedia [63]. Devlin et al. [7] also introduced a model named mBERT, which was trained on a multilingual corpus of 104 languages, including Norwegian. The exact size of the Norwegian portion of the dataset is unknown, but it corresponds to the size of the Norwegian Wikipedia pages. The Norwegian Wikipedia is relatively small compared to other high-resource languages causing a disadvantage when fine-tuning the mBERT model for Norwegian NLP tasks. The mBERT also comes in two versions, mBERT$_{BASE}$ and mBERT$_{LARGE}$, with the same configurations as the original models.

NB-BERT

The Norwegian National Library embarked on a project to develop a BERT model trained on Norwegian text. The Norwegian National Library has a massive Norwegian corpus for training such a model called The Colossal Norwegian Corpus [64]. The corpus includes Norwegian newspapers, journals, and books spanning the last 200 years, which makes it a perfect subject for being the basis of a pre-trained BERT model that represents general contextual information about the Norwegian language. The distribution of the texts between the two language variants in Norwegian, "Bokmål" and "Nynorsk", is estimated to be about 83 % Norwegian Bokmål and 12 % Norwegian Nynorsk, with the remaining 5% being English or other Nordic languages. The corpus contains over 18 billion words.

The NB-BERT is developed by Kummervold et al. [10] using the same architecture as the original BERT model [7]. The goal of the model was to create a robust model that excelled in all Norwegian NLP tasks. The NB-BERT model is initialized using the weights derived from the mBERT model to harness the cross-lingual capabilities of the mBERT model and give the NB-BERT model a better starting point than random weights would accomplish. This is theorized to improve the robustness of the model when coming across new, unseen words [10]. The model was enhanced beyond the initial training of the mBERT model by employing techniques introduced by You et al. [65], which involved increasing the batch size during pre-training and using a layer-wise adaptive moments base (LAMB) optimizer. The LAMB optimizer addressed problems associated with large learning rates with the standard Adam optimizer, such as instability and divergence. NB-BERT comes in two versions, NB-BERT$_{BASE}$ and NB-BERT$_{LARGE}$, where the base version has a total of 177 million parameters and the larger version has a total of 355 million parameters.

NorBERT

The NorBERT model was developed simultaneously as the NB-BERT by Kutuzov et al. [11] at the University of Oslo in the NorLM initiative. The model had the same goals as the NB-BERT model but utilized different training data. The NorBERT model was trained using the "Norwegian News Paper Corpus" from Språkbanken containing 160 million words [66]. The model was also trained on a dump of the Norwegian Wikipedia of 40 million words. The architecture used for NorBERT was the same as the original BERT model with 12 Encoder layers and a hidden size of 768. The NorBERT model contains in total 111 million parameters.

The NorLM initiative later introduced a new version of this model called NorBERT2 at Hugginface.co. The new model is trained on the Norwegian Colossal Corpus as the NB-BERT model [64]. The NorBERT2 also has 12 Encoder layers and a hidden size of 768 but contains 124 million parameters.

2.3 Evaluation Metrics

Metrics are used to evaluate the performance of a machine learning model. They are used to assess the quality of the resulting predictions, to compare different models, and to tune models for optimal performance. There are various metrics for different types of machine learning problems. The choice of metrics can impact the optimization process, the model selection, and the general interpretation of the model's capabilities. Choosing the wrong metrics can also lead to a biased model, which aligns differently with the goals of the modeling project.

2.3.1 Accuracy

Accuracy is a widely used evaluation metric for classification problems. It measures the proportion of correctly classified samples out of the total number of samples in the prediction. Accuracy can be expressed as follows:

$$\text{Accuracy} = \frac{\text{Number of correctly classified instances}}{\text{Total number of instances}} \tag{2.7}$$

Accuracy is suitable for balanced datasets as it gives a realistic impression of the model's capabilities and performance. In the case of imbalanced datasets, accuracy can be misleading and harder to interpret [67]. This can be caused by one label in the dataset accounting for most of the samples; thus, predicting all samples to the dominant label can still give a reasonable accuracy. This does not mean it is a good model because it does not consider the lesser common but still important labels. In the case of multi-label classification, accuracy only considers the samples where all labels are correctly classified. This makes the application of accuracy as a metric for multi-label classification strict, less informative, and less preferred to assess the performance of multi-label models.

2.3.2 Precision

Precision is another evaluation metric utilized in classification problems. Precision is better at evaluating imbalanced datasets where it is essential to avoid false positives [67]. Precision measures the proportion of true positive prediction out of all the actual positive samples in the dataset as illustrated in the equation as follows:

$$\text{Precision} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{Number of false positives}} \tag{2.8}$$

A high precision indicates that the model is good at predicting positive instances correctly [67]. Precision can be formulated as how many of the samples predicted as positives are indeed positive. Precision is important, but it should not be interpreted alone as it is closely related to recall.

2.3.3 Recall

The recall is the measure of the proportion of correct positive instances out of all samples predicted as positive, as seen in the equation as follows:

$$\text{Recall} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{Number of false negatives}} \tag{2.9}$$

The recall can be considered as how many relevant samples are actually in the positive predictions. A high recall score indicates that the model can identify positive samples and minimize false negative samples [67]. To quickly identify the trade-off between precision and recall, F1-score is introduced.

2.3.4 F1-score

The F1-score is an evaluation metric used especially for imbalanced datasets when both false positives and false negatives must be considered. The F1-score is the harmonic mean of the precision and recall score, which means that it weights both metrics equally to evaluate the balance between them [38]. The calculation of the F1-score is illustrated in the following equation:

$$\text{F1 score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2.10}$$

The F1-score, as a result of being a product of precision and recall, is high when both precision and recall are high. This is a valuable metric for tuning a model if the balance between precision and recall is more important than getting the best accuracy, which is often the case in real-life scenarios.

While the F1-score efficiently balances precision and recall, it doesn't account for true negatives, which could be a limitation in scenarios where both classes in a binary classification problem are equally vital. Therefore, depending on the situation, accuracy might be more suitable.

2.3.5 Precision, Recall and F1-score in Multi-Class Problems

To utilize metrics such as precision, recall, and F1-score to evaluate multi-class and multi-label classification models, the metrics must be calculated by class and then averaged to give a general model score. This is achieved by multiple means, two of which are given by the micro-average and the macro-average.

The following equations depict the different micro-average methods for the precision, recall, and F1-score:

$$\text{Micro-average precision} = \frac{\sum \text{TP}}{\sum \text{TP} + \sum \text{FP}} \tag{2.11}$$

$$\text{Micro-average recall} = \frac{\sum \text{TP}}{\sum \text{TP} + \sum \text{FN}} \tag{2.12}$$

$$\text{Micro-average F1-score} = 2 \times \frac{\text{Micro-average precision} \times \text{Micro-average recall}}{\text{Micro-average precision} + \text{Micro-average recall}} \tag{2.13}$$

In these equations, TP stands for the total true positives, FP for the total false positives, and FN for the total false negatives across all classes. The metric is computed globally by counting the total true positives, false negatives, and false positives.

The micro-average method can be used to score the overall performance of a multi-class or multi-label classifier [38]. This is achieved by aggregating all instances where the model predicted a sample to either belong or not belong to each class, as seen in Equation 2.11, Equation 2.12, and Equation 2.13. The micro-average assigns equal importance to all predictions, which can be crucial in imbalanced datasets.

The equations for the macro-average methods for the same metrics are given as follows:

$$\text{Macro-average precision} = \frac{\sum_{i=1}^{n} \text{Precision}_i}{n} \tag{2.14}$$

$$\text{Macro-average recall} = \frac{\sum_{i=1}^{n} \text{Recall}_i}{n} \tag{2.15}$$

$$\text{Macro-average F1-score} = \frac{\sum_{i=1}^{n} \text{F1-score}_i}{n} \tag{2.16}$$

Here, n is the total number of classes, and Precision_i, Recall_i, and F1-score_i are the precision, recall, and F1-score for each individual class i, respectively.

The macro-average method calculates the different metrics independently for each label. Then it takes the average as seen in Equation 2.14, Equation 2.15, and Equation 2.16. This results in the evaluation treating all classes equally, not considering the class size [38]. This ensures that larger classes do not overshadow smaller classes, but it can also give too much importance to classes that are so rare that they should instead be removed from the model.

Weighted-average can also be used to evaluate classifiers. The weighted-average considers the size of each class and weights the impact of the calculated scores according to their occurrences in the data. This is useful to account for imbalanced datasets and still evaluate the classwise performance of the model [38]. The equations for the weighted-average methods are as follows:

$$\text{Weighted-average precision} = \sum_{i=1}^{n} w_i \times \text{Precision}_i \tag{2.17}$$

$$\text{Weighted-average recall} = \sum_{i=1}^{n} w_i \times \text{Recall}_i \tag{2.18}$$

$$\text{Weighted-average F1-score} = \sum_{i=1}^{n} w_i \times \text{F1-score}_i \tag{2.19}$$

In these equations, w_i is the weight for each class i, which can be calculated as the proportion of instances in the class i out of all instances in the dataset, n is the number of classes, and Precision$i$, Recall$i$, and F1-score_i are the precision, recall, and F1-score for the class i, respectively.

Chapter 3

Data Exploration

This chapter introduces the football article datasets used to train and evaluate the models presented in this thesis. This chapter also includes assumptions and limitations made during the exploratory data analysis. All data exploration is carried out in Python and published to GitHub.

Dataset	Classification Task	Samples	No. of Labels	Data Source
Arx	Multi-Class	5526	12	Simula.no
Multi-Label Dataset	Multi-Label	2009	22	Github.com

Table 3.1: Dataset Information

The data used for this thesis is based on the two datasets displayed in Table 3.1. The first and original Arx dataset [12] was collected by Aanund Nordskog, which inspired this thesis and the further collection of the supplementary dataset presented in this thesis. The new dataset is meant to tackle and solve apparent problems with the first dataset and give the task of classifying paragraphs from football articles a new and more complete approach. Both datasets contain annotated paragraphs tagged as football articles from Norwegian online news providers.

3.1 The Arx Dataset

The Arx dataset is a collection of paragraphs from football articles fetched from VG.no, a major Norwegian newspaper, and TV2.no, a Norwegian commercial TV channel and news provider, introduced by Nordskog et al. [12]. The data was collected using the built-in Python library `Requests` [68] to fetch the HTML code from all articles tagged with "football". The relevant paragraphs from each article were extracted using the library `Beautiful Soup` [69] in Python and stored in a database. The Arx dataset comprises articles published in the interval between 22.07.18 and 17.09.18. Note that the short period of the publications included in the dataset can impact models trained with the data. Events such as international cups and transfer markets are seasonal and do not appear consistently throughout a year of football. Some events might not be represented realistically in a model trained with the Arx dataset.

The Arx dataset is labeled through the Arx web app, a graphical user interface (GUI), also created by Nordskog et al. [12]. The GUI made the filtering and labeling of the dataset easier.

The dataset consists of 5,526 labeled paragraphs and is available online at Simula.no [70]. The labeling is stated at Simula as being performed by a single person who is familiar with football [70]. Each paragraph is labeled as one of the following 12 classes;

- Goal/Assist
- Quotes
- Transfer
- Irrelevant
- Ignore
- Player Detail
- Club Detail
- Chances
- Injuries
- Red/Yellow Card
- Club Drama
- Personal Drama

"Sist 26-åringen hadde både mål og målgivende i en og samme Premier League-kamp var mot Newcastle i mars 2016, opplyser VGs faktaguru, Geir Juva. "	Goal/Assist
"– Bittert. Det er mye mer bittert enn å tape drittkamp der du blir rundspilt og ikke fortjener noe, men i dag var det glød i øynene på gutta. Det var tro, hardt arbeid og masse offensivt spill. Det var en herlig ramme med fantastiske supportere, sier Vålerenga-trener Ronny Deila til TV 2."	Quote
"Avis: – Agenten har lovet Pogba overgang til Spania"	Transfer

Figure 3.1: Three examples of paragraphs and classes from the Arx dataset by Nordskog et al. [12].

The difference between paragraphs can be seen in Figure 3.1. Some paragraphs are short and concise, such as those labeled with the transfer class. Other paragraphs are descriptive and long, such as the paragraph labeled quote in Figure 3.1, painting a picture of the match and the surrounding atmosphere. The total variation of paragraphs is vast and makes labeling difficult.

The Arx dataset offers a diverse range of paragraphs, providing valuable data for training models to recognize different information types in football articles. However, a limitation is the assignment of only one label per paragraph, even though a paragraph could contain information relevant to multiple classes. This single-label approach could oversimplify complex texts and potentially limit the model's ability to accurately understand and classify multi-topical paragraphs. Moreover, the fact that a single individual performed the labeling might introduce bias and inconsistency in the dataset. Lastly, the dataset's relatively short timeframe might only represent some football events realistically, affecting the model's generalizability.

3.1.1 Label Distribution in The Arx Dataset

A challenge with labeling an inhomogeneous dataset is the imbalance in the labels used. To catch the essence of each paragraph into a single class, the person labeling must be very decisive and obedient to the labeling guidelines, which in many cases, causes some labels to be more frequent than others.

The Arx dataset contains a total of 12 different mutually exclusive classes. Each paragraph is labeled with one class, and the total sum of labeled paragraphs is 5,526.

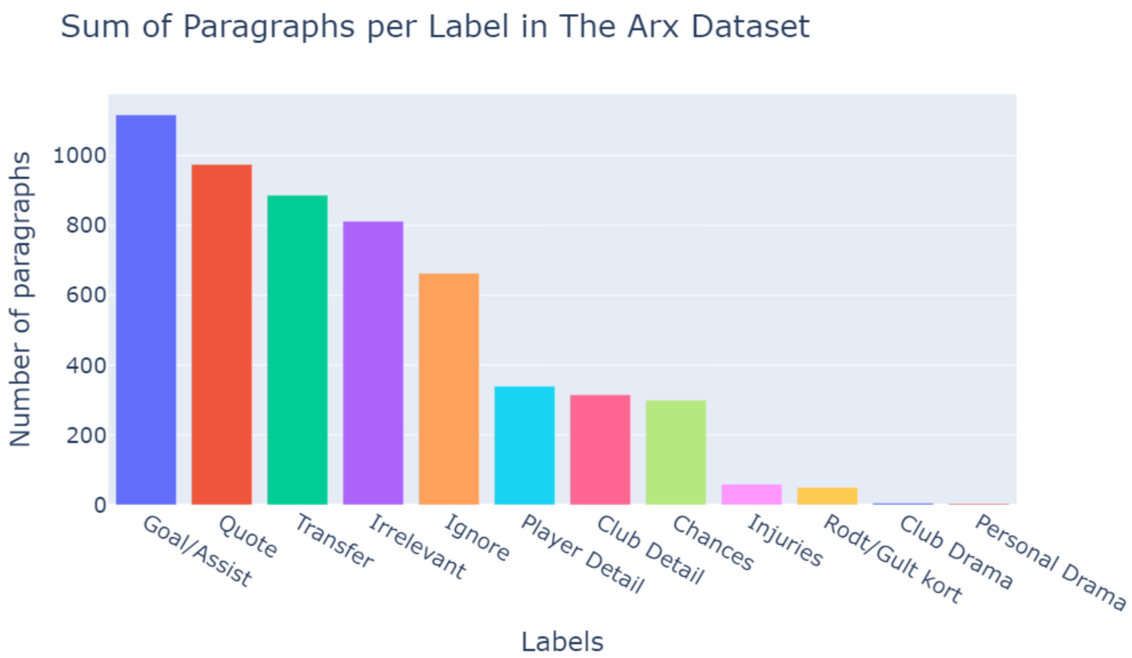


Figure 3.2: Distribution of labels in the Arx dataset.

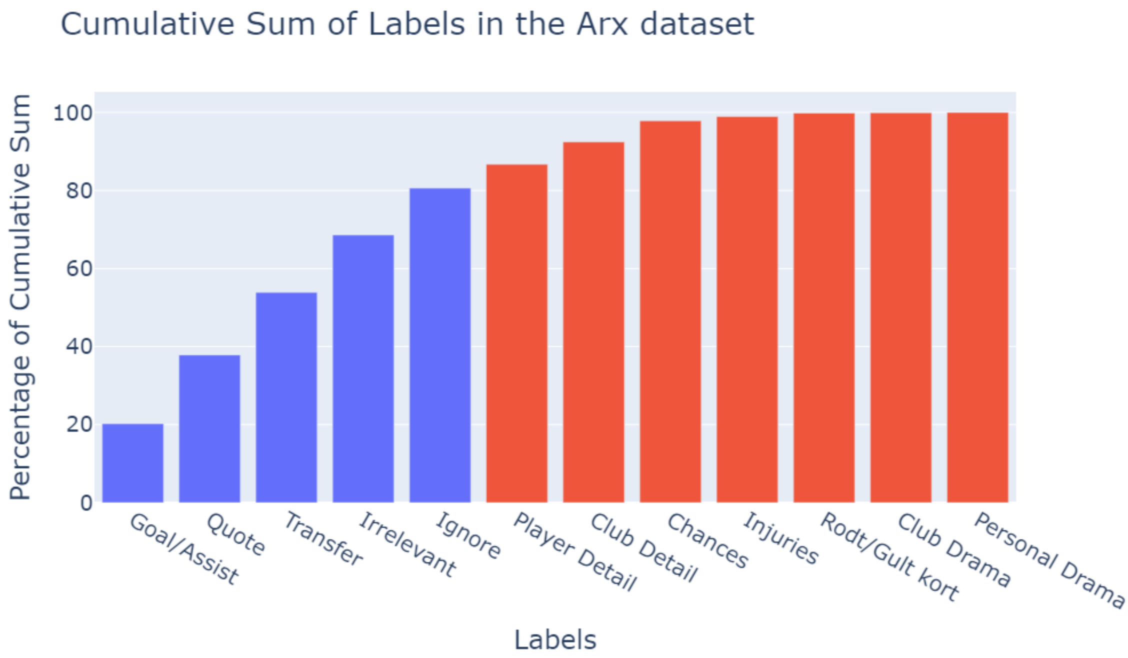


Figure 3.3: Cumulative sum of labels in the Arx dataset ordered by the number of samples the label supplies the sum in decreasing order. The labels that compose 80.3 % of the data are displayed in blue. The remainder of the data is displayed in red.

Figure 3.2 shows the distribution of the different classes. Figure 3.3 indicates the imbalance in the distribution in the Arx dataset in a cumulative percentage graph. The five most extensive classes, Goal/Assist, Quote, Transfer, Irrelevant, and Ignore, make up a total of 80.3 % of all the classes in the dataset. The remaining seven classes only make up 19.7 % of the labels causing a significant challenge to the classification problem with the introduction of low-sampled classes.

3.2 The Multi-Label Dataset

The new data is collected to create a more challenging classification task for assessing the models in this thesis. The dataset consists of paragraphs from football articles at VG.no published between 01.01.2022 and 01.01.2023 and is referred to as the Multi-Label Dataset. The total data collected is 10,147 paragraphs; of these, 2,005 have been annotated. The paragraphs have been annotated in a random order to prevent seasonal football events from being too prevalent in the data. The data have been annotated with multiple labels, making the task associated with the dataset multi-label classification instead of multi-class classification, in contrast to the Arx dataset.

A new set of labels has been chosen for the Multi-Label Dataset compared to the Arx dataset. This decision is primarily motivated by a desire to capture the multi-dimensional nature of the football articles more accurately. The original Arx labels were somewhat restrictive, limiting their ability to represent these articles' complexity adequately. A more comprehensive labeling schema allows for more nuanced classification, acknowledging the multifaceted elements within the paragraphs. Consequently, this enhances the understanding of the articles and broadens the potential applications of the dataset. The possible labels for each paragraph are as follows:

- **Booking**: Instances when a player receives a yellow or red card during a football match.
- **Chance**: Potential goal-scoring opportunities or significant plays during a game.
- **Commentary**: Play-by-play narrations or remarks that describe the events happening in a game.
- **Description**: Detailed accounts or explanations about events, scenarios, or subjects related to football.
- **Garbage**: Irrelevant or nonsensical text that does not contribute to the context of the dataset.
- **Goal**: Instances when a goal is scored during a football match.
- **Injury**: Situations where a player gets hurt during a match or describe a scenario affected by an injury.
- **Link**: Instances where a URL or hyperlink is present in the text.
- **Next game**: Text that discusses or mentions upcoming football matches.
- **Odds**: Predicted probability of specific outcomes in a game, commonly used in betting contexts.
- **Opinion**: Personal views, feelings, or attitudes towards a particular subject.
- **Quote**: Text that contains a direct citation from a person involved in football, such as a player or a coach.
- **Rumour**: Unverified information or speculation, typically about player transfers or upcoming matches.
- **Set-piece**: Controlled restarts during a football match, such as free kicks, corner kicks, or penalty kicks.
- **Statement**: Formal or unofficial announcements made by clubs, players, or expert commentators.
- **Statistics**: Numerical data providing information about various aspects of football, such as player performance, team performance, or match outcomes.
- **Storytelling**: Text that provides a narrative or descriptive account of events.
- **Substitution**: Instances when one player is replaced by another during a football match.
- **Summary**: Condensed reports or overviews of match events or other football-related topics.
- **Table**: Instances where league standings, match results, or other tabulated data are discussed.
- **Transfer**: Discussions about the movement of players between different clubs.
- **VAR**: Video Assistant Referee, a technology used in football to review decisions made by the head referee with the help of video footage.

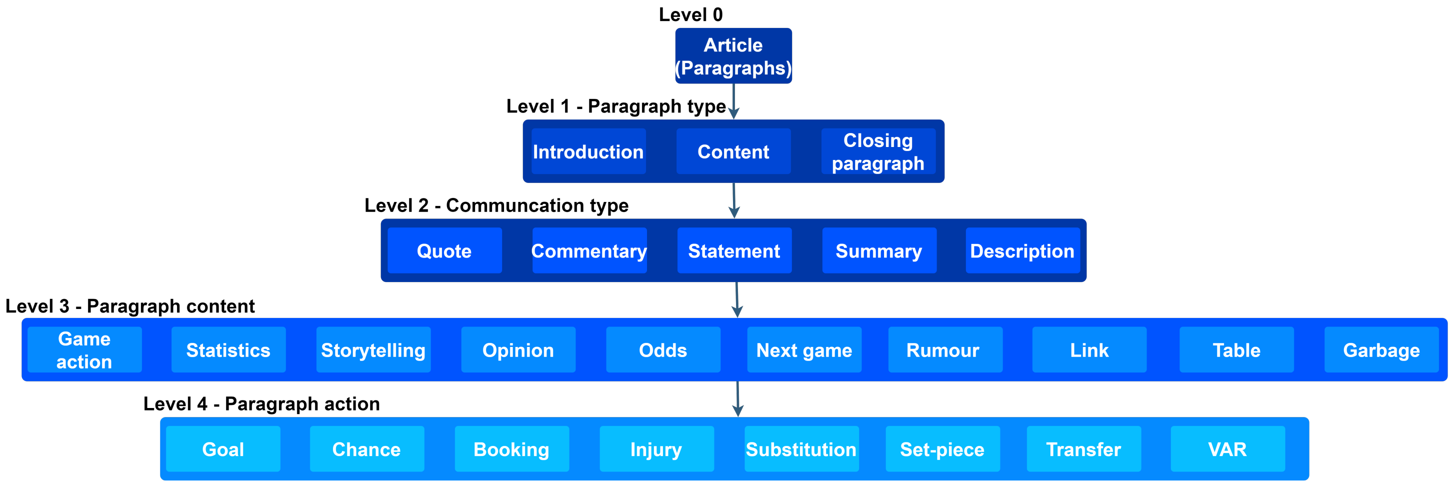


Figure 3.4: The different levels of information the labels aims to capture.

There are a total of 22 different labels that each paragraph can be assigned to. The labels are split into different levels of information types. The levels are paragraph type (level 1), communication type (level 2), paragraph content (level 3), and paragraph action (level 4), and together they form a hierarchy, as seen in Figure 3.4. The annotated labels only include the communication type, paragraph content, and paragraph action labels, as they are the ones that can be distilled from a paragraph. The hierarchy of the labels is not utilized to restrict the annotation in this thesis, as paragraphs can be labeled with multiple labels from the same level. Still, it shows that the paragraphs are more likely to be annotated with lower-level labels as they are more important for the composition of a paragraph. Note that all of the labels in level 4 can be aggregated to "Game action" in level 3, and therefore "Game action" is not included in the labels. None of the labels are mutually exclusive because the paragraphs sometimes contain multiple levels of information.

The Multi-Label Dataset differs from the original Arx dataset in two crucial aspects. Firstly, it encompasses an additional period, precisely, articles published from 01.01.2022 to 01.01.2023, broadening the temporal scope of the data. Secondly, it introduces a distinct set of 22 labels that capture varied levels of information types as depicted in Figure 3.4. With these labels, the annotated samples have been multi-labeled, accommodating the multi-dimensional nature of the paragraphs, which often encapsulate more than one type of information. Thus, the Multi-Label Dataset offers an expanded and more nuanced understanding of football articles' content from VG.no.

3.2.1 Label Distribution in The Multi-Label Dataset

The Multi-Label Dataset contains 2,005 annotated samples. Each of those samples is at least annotated with one of 22 labels that vary from broader concepts to more specific events in football matches.

The distribution of the Multi-Label Dataset can be viewed in two different ways. The distribution can be considered as the count of each combination of labels, known as label sets. A label set is a specific combination of the labels assigned to a paragraph. For instance, one paragraph might have the labels "Goal", "Description", and "Commentary", which would form one distinct label set. Another paragraph might be labeled with "Injury", "Statement", and "Summary", making up another unique label set. The distribution referred to here is a count of how often each unique label set appears across the dataset. Alternatively, the distribution can be understood as the cumulative count of each individual label across the entire dataset, regardless of the label combinations in which they appear.

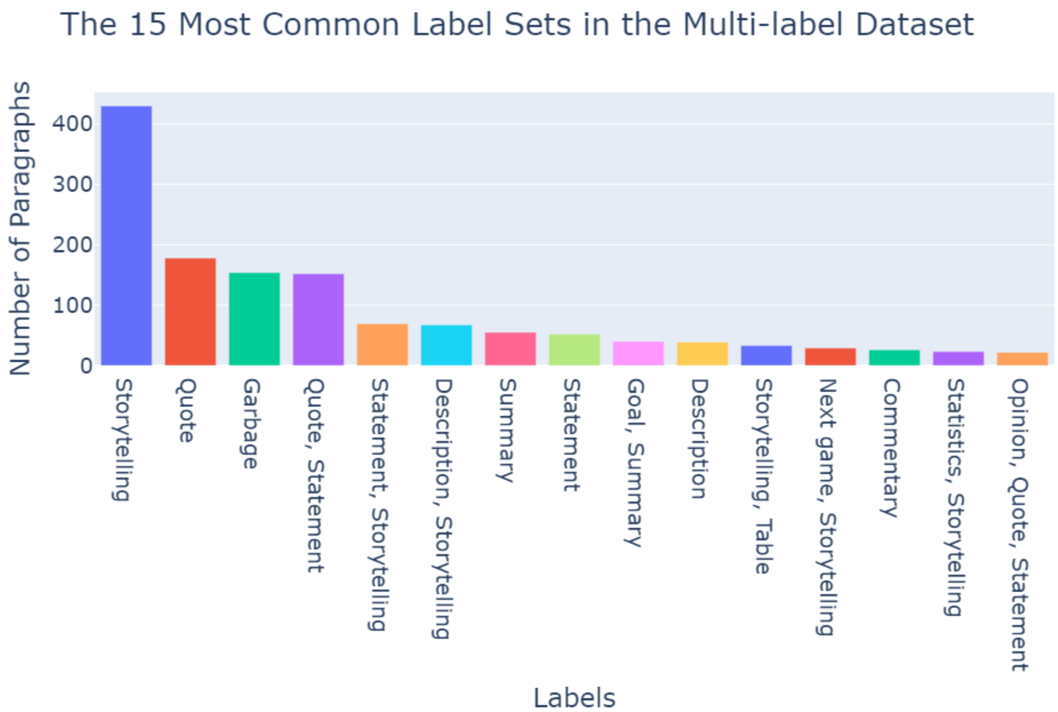


Figure 3.5: The 15 most common label sets in the Multi-Label Dataset. Some label sets include only one label, as the paragraphs are only annotated with one label.

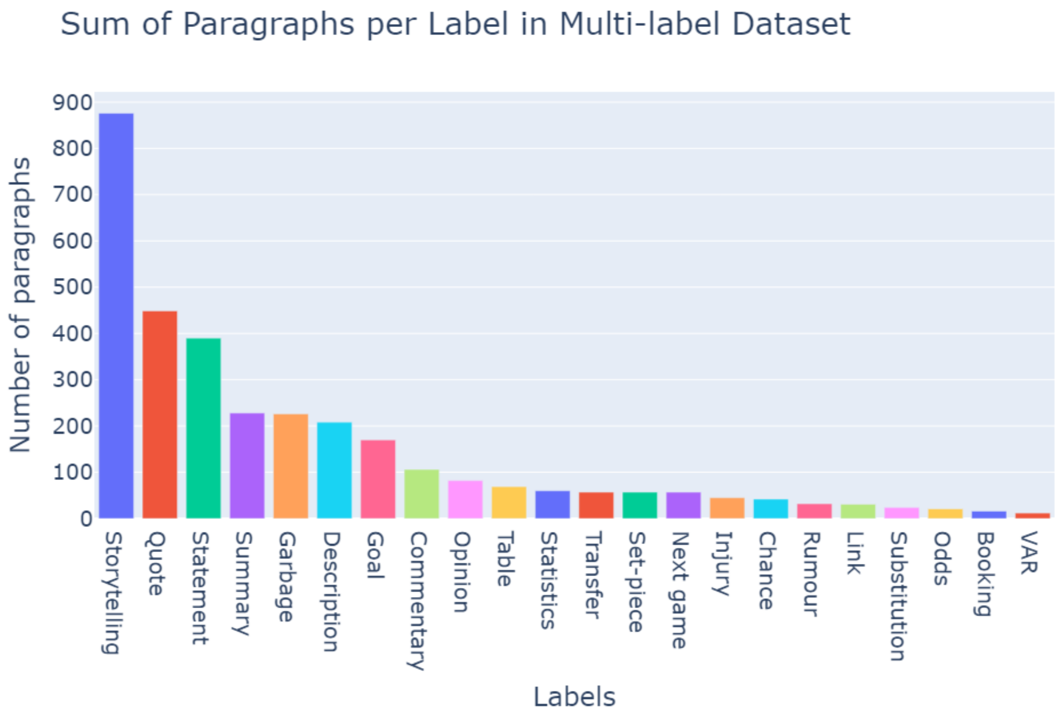


Figure 3.6: Label distribution for paragraphs from the Multi-Label Dataset. The y-axis differs from Figure 3.5 as this includes both solo labels and labels annotated in a co-occurrence with other labels.

There are a total of 226 different label sets in the Multi-Label Dataset. Figure 3.5 shows the 15 most common label sets in the 2005 samples. Figure 3.6 shows the distribution of all individual labels used in annotating the Multi-Label Dataset. The total number of labels surpasses the number of samples because each sample can have multiple labels.

The most frequent label is "Storytelling", as it is one of the broadest terms, and most paragraphs, if not all, are there to broadcast a story in some way. The labels "Quote" and "Statement" occur often and are usually inclusive of each other. It can be noticed that "Goal/Assist" occurs far more often in the Arx dataset than in the Multi-Label Dataset. This can be caused by the difference in seasonality in the data. The Multi-Label Dataset contains samples that are randomly sampled throughout the year, as opposed to the Arx dataset, which is annotated over a shorter period.

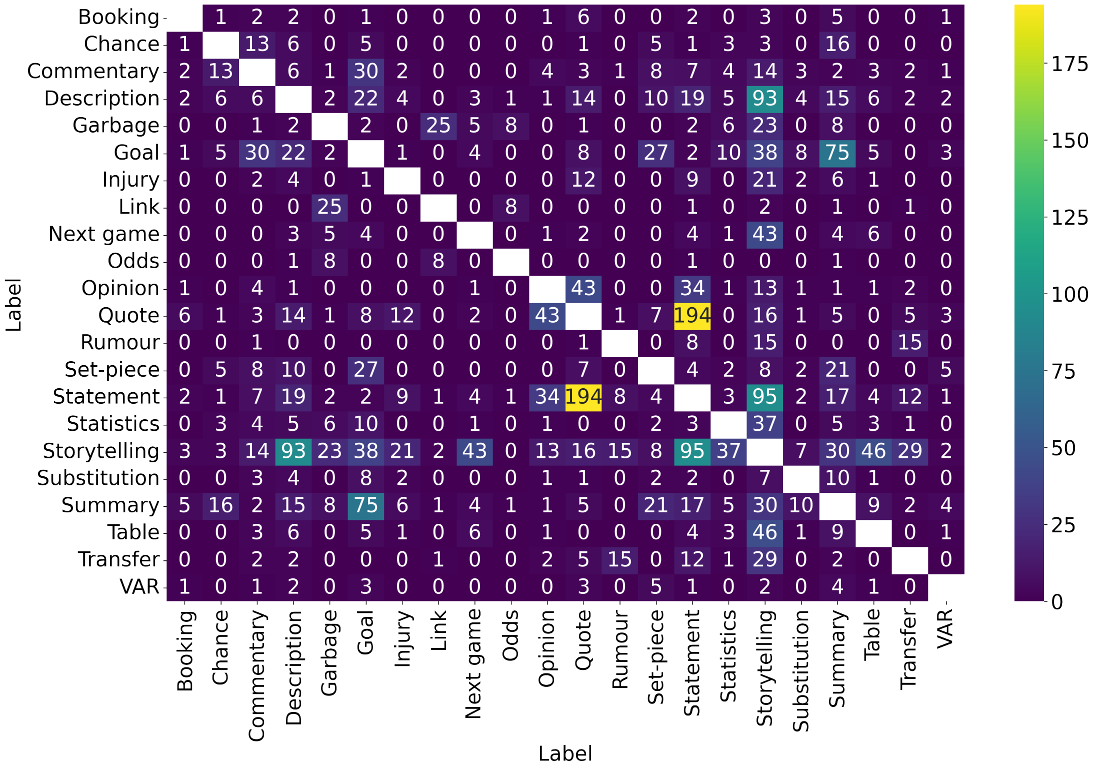


Figure 3.7: Co-occurrence matrix of labels in the Multi-Label Dataset. Each cell shows the number of times a pair of labels co-occurs in the dataset. Brighter colors represent higher co-occurrence frequencies.

A co-occurrence matrix represents the number of times two labels occurs together in a label set in a dataset. Figure 3.7 shows the co-occurrence matrix for the Multi-Label Dataset. The co-occurrence matrix displays that "Statement" and "Quote" has the highest co-occurrence in the dataset by occurring 194 times together. The "Storytelling" label has a high co-occurrence with several labels, such as "Description" (95 times), "Statement" (93 times), and "Table" (46 times). The "Storytelling" label also co-occurs with all labels except the "Odds" label. The "Goal" and "Summary" labels also have high co-occurrence, labeled together 25 times, indicating that summaries often mention goals. "Garbage" and "Link" co-occur 25 times. This suggests that many of the links are considered irrelevant in the context of this dataset. Some labels, such as "Booking", "Chance", "Injury", and "Odds", have low co-occurrence with other labels. This can indicate that the labels are distinct, non-overlapping categories or simply occur infrequently in the dataset.

Chapter 4

Method

This chapter outlines the steps taken to address the research questions posed in this thesis. This chapter aims to provide a comprehensive description of the data collection, annotation, preprocessing, and model tuning to achieve the results to be presented. This section should explain the procedures to ensure reproducibility and give the reader a fair foundation for assessing the use of Norwegian pre-trained BERT models on text classification tasks.

4.1 Data Collection

Data collection is the most critical step in all research projects and is the foundation of machine learning. The quality of the data obtained impacts the overall findings of the thesis. The data used in this paper is also the source of the research question that this paper seeks to address; whether Norwegian pre-trained BERT models outperform traditional machine learning models at text classification on Norwegian texts. The question calls for a Norwegian dataset labeled for text classification in a diverse topic. This is where the Arx dataset [12] and the Multi-Label Dataset is introduced.

4.1.1 Arx Dataset Collection

The collection of the Arx dataset is described in Section 3.1. To summarize the collection, the Arx dataset was collected from the Norwegian news sources VG.no and TV2.no, comprising football articles published between 22.07.18 and 17.09.18. The data, fetched using Python libraries `Requests` and `Beautiful Soup` , consists of 5,526 labeled paragraphs.

The Arx dataset proved to be a perfect match to test out Norwegian pre-trained BERT models at multi-class text classification. Nordskog et al. [12] tested the Arx dataset similarly as in this thesis, by comparing CNNs and RNNs with traditional machine learning models with minimal text preprocessing but with some label manipulation. The 12 original labels were merged into five bigger labels. Initial attempts at replicating the label manipulation by Nordskog et al. [12] proved unsuccessful. This makes the work in this thesis not based on a comparison with the models and results from Nordskog et al. [12]. Benchmarks from the Norwegian pre-trained BERT models were instead compared with new traditional machine learning models trained on all the initial 12 labels not to compromise the results of this thesis.

4.1.2 Multi-Label Dataset Collection

The BERT model is created to be fine-tuned for multiple downstream tasks in NLP [7]. To check if this is applicable and effective for the Norwegian pre-trained BERT models, the pre-trained models must also be trained on another classification task. The multi-label classification was chosen because the original dataset labels are not mutually exclusive and, therefore, hard only to give one label.

The Multi-Label Dataset was created with some of the challenges of fitting the Arx dataset for machine learning in mind. The Multi-Label Dataset should solve problems such as the annotator having to decide between multiple non-mutually exclusive labels for a paragraph and the issue of the annotated data being poorly distributed throughout a year, causing the distribution of events to be incorrectly represented in the dataset.

The Multi-Label Dataset was collected from VG.no, also using the Python library `Requests` fetching all HTML code for articles tagged with "football" from 01.01.2022 until 01.01.2023, ensuring updated paragraphs. The HTML code was parsed using the Python library `Beautiful Soup` [69].

Annotation

For constructing the Multi-Label Dataset, one annotator, the author of this thesis, annotated 2,009 samples using 22 non-mutually exclusive labels, ensuring consistency in the annotation outcomes. The 2,009 samples are distributed throughout the whole time series, the collected data, to make the distribution of events in the annotated data more realistic.

Figure 4.1: The GUI `Doccano` [71] used to annotate the Multi-Label Dataset with an example paragraph and possible labels.

The annotation was done using the Python library `Doccano` [71] as seen in Figure 4.1. `Doccano` provides a GUI for managing annotation projects and supports the task of multi-label text classification and data shuffling. `Doccano` made the process of annotating the data simple and intuitive.

`Doccano` can integrate an active learning model, which was implemented during this annotation process. The active learning model helps the annotator by suggesting the labels for each paragraph. Each paragraph labeled by the active learning model must be approved by the annotator to be marked as correctly annotated. The annotator changes the labels if necessary before approving the labels for each annotated paragraph.

Active Learning

Active learning is a semi-supervised machine learning technique that can be utilized to streamline the annotation of a dataset. Instead of manually annotating a large set of samples, active learning enables using an initial model trained on a small set of randomly chosen labeled data to assist in the annotation. As annotators label the rest of the samples in the dataset, the model is iteratively updated and improved. By combining human annotators and an active learning model in the labeling process, active learning can reduce the time spent on annotating each sample, resulting in a more efficient annotation process.

Active learning was integrated into the annotation to speed up the process. For every hundred annotated sample, a model was trained using all the until now annotated data at multi-label classification. The newly trained model was used to predict the labels for each new paragraph being annotated forward from introducing a new iteration of the active learning model.

The model used for active learning was a Binary Relevance model with a LinearSVC estimator with a radial basis function kernel as the base classifier from `Scikit-Learn`. `Scikit-Learn` is an open-source Python library that provides tools for machine learning tasks, including data preprocessing and classification [72]. The hyperparameters C and *gamma* were set to 2 and 0.55, respectively, and were not tuned during training. The classifier was chosen because of the resource-effective results provided by Nordskog et al. [12] on the Arx dataset. The annotator evaluated the model's results during training, and the active learning model could be turned off if it did not prove effective at helping the annotator. The active learning model was used during the annotation process with careful consideration and surveillance.

4.2 Model Selection

This section describes the process of choosing among the multiple options of Norwegian pre-trained BERT models to benchmark versus traditional machine learning models at multi-class and multi-label classification. The goal is to get a diverse set of models and compare the performance of the different models on the Arx Dataset and the Multi-Label Dataset.

4.2.1 Norwegian BERT Model Selection

The selection of the Norwegian BERT model is the foundation of this thesis. The two large Norwegian BERT models released are NB-BERT and NorBERT [10, 11]. Both these models are relevant for the classification tasks and will be trained and tested for comparison. Both these models come in multiple versions. The NB-BERT version utilized in this thesis is the NB-BERT$_{LARGE}$ built on the digital collection at the National Library of Norway [10]. The other BERT model in this thesis is NorBERT, introduced by the University of Oslo [11]. In the case of NorBERT, the second iteration of the model was used; NorBERT2.

Both models are easily integrated into a machine learning pipeline through the `Transformers` library from Huggingface [73]. The library is used in the thesis to load the pre-trained models and for fine-tuning the models on downstream classification tasks with `PyTorch` integration. `PyTorch` is an open-source Python library for deep learning developed by Facebook, which supplies tools for building and training neural networks [74]

4.2.2 Traditional Machine Learning Model Selection

In the selection of models for comparison, a consideration of a range of machine learning algorithms that are widely used for classification tasks and have demonstrated success in similar natural language processing tasks is performed. The final set of models includes Random Forest, AdaBoost, LinearSVC, Multilayer Perceptron, K-Nearest Neighbor Classifier, and Gradient Boosting. Each selected model has unique strengths and weaknesses that make it suitable for different machine learning problems.

Random Forest is a popular and common ensemble learning algorithm implemented in most machine learning packages. The Random Forest model has the benefit of not needing much tuning [75]. Random Forest models are also not prone to overfitting if the depth of the trees is restricted and, therefore, make for a great model to add to the benchmark.

GradientBoosting is another popular ensemble learning algorithm that differs from RandomForest in incrementally adding decision trees. GradientBoosting often produces better results than Random Forest but is more prone to overfitting [76].

AdaBoost is another ensemble method that utilizes many weaker learners with different weights on the classification. AdaBoost also often outperforms RandomForest at making accurate classifications [76].

LinearSVC is a linear model which is highly effective at text classification with high-dimensional data, which is the case in this thesis [77]. This makes it a great supplement to the benchmark models.

Multilayer Perceptron (MLP) is a type of neural network that can learn complex nonlinear relationships between the features [78]. The MLP is a straightforward approach to testing out neural networks for text classification and is a natural algorithm to append to the benchmark models.

K-Nearest Neighbor Classifier is a simple and intuitive algorithm. K-Nearest Neighbor Classifier is based on clustering the data lying in a similar place in the feature space [79]. Texts with similar structures can often be located in similar areas in the feature space.

The models that do not natively support multi-label classification are added as estimators in either the OneVsRestClassifier or the MultiOutputClassifier. The OneVsRestClassifier trains a separate model for each class, which in turn treats each label as the positive class and the rest as the negative class, thereby transforming the problem into multiple binary classification problems. The MultiOutoutClassifier fits one classifier per target, but each classifier can have multiple outputs, rather than binary, as opposed to OneVsRestClassifier.

All of the mentioned models are easy to implement through `Scikit-Learn` and are adaptable for multi-class and multi-label classification. This is the main reason for the selection of models. The following sections will refer to these models as the traditional machine learning models.

4.3 Data Preprocessing

This section presents the data preprocessing to ready the datasets for the text classification task for the different machine learning models. Data preprocessing is a vital part of machine learning and impacts the performance of any model trained. The models trained in this thesis require different preprocessing techniques and will be explained separately.

4.3.1 Data Preprocessing for BERT Models

Transformer models, such as BERT, are in the deep learning architectures category, designed to process sequential data effectively. The positional information found in a textual structure is required for optimal performance. To keep the positional information in the model, minimal data preprocessing is utilized. This approach ensures that the model does not have to transform the raw text into a simplified representation, such as a token frequency count. The only crucial step in data preprocessing for the BERT models is to tokenize the plain text data. Tokenization ensures that the input text is consistent across different samples while keeping the text within the fixed-size vocabulary of the pre-trained BERT model. Tokenization also processes words outside the vocabulary into smaller known subwords, enabling the positional information for each token to be stored.

Both BERT models in this thesis utilize the AutoTokenizer from the Huggingface library `Transformers` [73]. The AutoTokenizer finds the correct tokenizer for the Huggingface model the data is being prepared for. The main features of the tokenization process are input feature generation and padding/truncation. The AutoTokenizer generates token IDs, attention masks, and token-type IDs. The token ID is a numerical representation of the word linked to the representation of the model vocabulary. The attention mask indicates which tokens that should be paid more attention to by the model. The token type IDs help the model distinguish between the different segments in the input sequence. Padding/truncation is making the sequence of tokens into the desired length required as the input of the model. The AutoTokenizer makes no distinction between capitalized and non-capitalized words by default.

4.3.2 Data Preprocessing for Traditional Machine Learning Models

The traditional machine learning models often need more specific preprocessing to achieve optimal results. These models do not consider the positional information when extracting information from the text being processed. The lack of an attention mechanism makes them prone to being polluted with noise which can reduce the model's effectiveness. A few steps in the data preprocessing are usually introduced to tackle this problem and to ensure optimal performance: These steps are stopword (words that are too common to add value to the class of the text) and special character removal, stemming (word reduction to the root form) and tokenization.

To compare the BERT models and the traditional machine learning models as fair as possible, the same amount of preprocessing was applied to the data used in training both models. The data fed to the traditional machine learning models were tokenized using the TfidVectorizer (TF-IDF) from the `Scikit-Learn` library. The TfidVectorizer makes no distinction between capitalized and non-capitalized words by default.

4.3.3 Train/Test Split

To evaluate the models fairly, the datasets must be split into training and testing data. Splitting the data allows the model performance to be assessed on unseen data, indicating if the model is overfitting or generalizing well. The ultimate goal of the model is to predict new data accurately, so this step is essential.

Different methods were used to split the datasets for the different classification tasks because of the different nature of the multi-class and multi-label datasets. To keep the benchmark as fair and replicable as possible, each dataset was only split once, and all models were trained and tuned using the same split.

The Arx dataset was split using the `Scikit-Learn` train/test split. The Arx dataset was distributed into 80 % training data and 20 % testing data. The data were stratified by class, meaning the class proportion is maintained in both the training and testing data.

The Multi-Label Dataset was split using IterativeStratification from the `Scikit-Multilearn` library [80–82]. This was used instead of the Scikit-Learn train/test split due to its ability to handle class imbalance in multi-label data. The `Scikit-Learn` train/test split would have treated each of the 226 label sets as a class during stratification, causing many label sets to be unique, which would ruin the possibility of maintaining the class proportion in both training and testing data. IterativeStratification fixes this by ensuring that the imbalance ratio is kept in each split. The Multi-Label Dataset was also split into 80 % training data and 20 % testing data.

4.4 Model Tuning

Model tuning is an essential step in the development of machine learning models. Model tuning involves optimizing the hyperparameters of each model to achieve the best results for the problem at hand. Appropriate hyperparameters are crucial for the model's performance and generalization abilities.

The hyperparameter tuning process was designed to maximize the weighted-average F1 score, a metric that effectively balances precision and recall while considering label imbalances. In the process of hyperparameter tuning for the traditional machine learning models implemented via `Scikit-Learn`, two-fold cross-validation was performed on the training set. Subsequently, the models were evaluated on the testing set. In the case of the BERT models, the entire training set was employed for hyperparameter tuning, followed by an evaluation using the testing set. The Multi-Label Dataset proved challenging to split into multiple folds due to some labels only occurring a few times. This is the reason for excluding cross-validation from the tuning of the BERT models.

Hyperparameter tuning involves training the models multiple times using different hyperparameters while logging the different results for comparison. `Weights & Biases` [83] is a popular tool for experiment tracking and hyperparameter optimization with the ability to store, monitor and visualize the training process of each of these models. `Weights and Biases` was utilized in this thesis to track the different models' hyperparameter tuning and log the resources used for training the different models.

4.4.1 Bayesian Search

Bayesian search is an efficient method to utilize while tuning hyperparameters [84]. It leverages the information from previous evaluations to guide the search process. Bayesian search can result in quicker convergence to optimal hyperparameter settings and relieves resources used for tuning by being effective.

The alternatives to Bayesian Search are grid search and random search. Grid search tries all possible combinations of specified hyperparameters, which is computationally expensive for high-dimensional search spans. Random search is more efficient but chooses the hyperparameters randomly for a selected number of iterations. Compared to the other methods, the benefit of Bayesian search is its ability to learn from past evaluations, directing the search into promising regions.

The Bayesian search was used in tuning each of the models presented in this paper. All models were trained using 30 combinations of hyperparameters from spans defined specifically for each model. The spans utilized for the Bayesian search can be found in the subsequent tables.

NB-BERT and NorBERT2

Hyperparameter	Range/Search Space
epoch	{3, 4, 5}
learning_rate	uniform(1e-6, 1e-4)
per_device_train_batch_size	{16, 32}
warmup_ratio	uniform(0, 0.5)
weight_decay	uniform(0, 0.3)

Table 4.1: Hyperparameter search space for the NB-BERT model and the NorBERT2 model

The hyperparameter span for the NB-BERT and NorBERT2 models, as shown in Table 4.1, was chosen based on the fine-tuning hyperparameters suggested by Devlin et al. [7].

Random Forest

Hyperparameter	Range
n_estimators	[50, 500]
max_depth	[5, 30]
min_samples_split	[2, 10]
min_samples_leaf	[1, 5]
max_features	[0.1, 0.5]

Table 4.2: Hyperparameter search space for RandomForestClassifier

The hyperparameter span for the Random Forest models is shown in Table 4.2. All the hyperparameters in the span can range between the two set values for each hyperparameter.

Linear SVC

Hyperparameter	Range/Search Space
C	log-uniform(1e-6, 1e+6)
tol	log-uniform(1e-6, 1e-1)
class_weight	{balanced, None}

Table 4.3: Hyperparameter search space for LinearSVC

The hyperparameter span used for tuning the Linear SVC model is shown in Table 4.3. The log-uniform distribution is used because the optimal hyperparameters might span several orders of magnitude.

AdaBoost

Hyperparameter	Range/Search Space
n_estimators	[10, 1000]
learning_rate	log-uniform(0.001, 10)
algorithm	{SAMME, SAMME.R}

Table 4.4: Hyperparameter search space for AdaBoostClassifier

The hyperparameter span used for tuning the AdaBoostClassifier is shown in Table 4.4.

Multilayer Perceptron

Hyperparameter	Range/Search Space
hidden_layer_sizes	log-uniform(10, 200)
activation	{relu, logistic, tanh}
solver	{adam, sgd}
alpha	log-uniform(1e-6, 1e-2)
learning_rate	{constant, invscaling, adaptive}
learning_rate_init	log-uniform(1e-4, 1e-2)
max_iter	[100, 1000]
tol	log-uniform(1e-6, 1e-2)

Table 4.5: Hyperparameter search space for MLPClassifier

The hyperparameter span used for tuning the MLPClassifier is shown in Table 4.5.

KNeighbors

Hyperparameter	Range/Search Space
n_neighbors	[3, 20]
weights	{uniform, distance}
algorithm	{auto, kd_tree}
leaf_size	[10, 50]
p	[1, 3]

Table 4.6: Hyperparameter search space for KNeighborsClassifier

The hyperparameter span used for tuning the KNeighborsClassifier is shown in Table 4.6.

GradientBoosting

Hyperparameter	Range/Search Space
n_estimators	[100, 1000]
max_depth	[3, 10]
min_samples_split	[2, 10]
min_samples_leaf	[1, 10]
max_features	{sqrt, log2}
learning_rate	log-uniform(0.001, 0.1)

Table 4.7: Hyperparameter search space for GradientBoostingClassifier

The hyperparameter span used for tuning the GradientBoostingClassifier is shown in Table 4.7.

4.5 Model Configurations

The hyperparameter search only optimizes some of the hyperparameters available for the models. The rest have the default value set for the different implementations of each model. This section presents the complete model configurations after the hyperparameter search for each classification task. The final configuration for each model leads to the results presented in Chapter 5. The explanation of all the features introduced in this section precedes the scope of this thesis. Still, it can be explored more thoroughly in the documentation of the packages presented in Appendix A.

All models are trained using either the packages from Huggingface or the `Scikit-Learn` package. Values not explicitly mentioned are all set to the default values from the affiliated package, and the packages' version can be found in Appendix A.

4.5.1 Model Configurations for the Multi-Class Classification

NB-BERT

Parameter	**Value**
Model	NbAiLab/nb-BERT-large
Training epochs	3.0
Learning rate	8.969444903069361e-05
Weight decay	0.038330404806479645
Warm-up ratio	0.32604822325361665
Per device train batch size	32
Evaluation strategy	epoch
Hidden size	1024
Number of attention heads	16
Number of hidden layers	24
Adam beta1	0.9
Adam beta2	0.999
Adam epsilon	1.0e-08
Seed	100
Transformer architecture	BERT
Position embedding type	absolute
Hidden activation function	GELU
Hidden dropout probability	0.1
Attention probabilities dropout probability	0.1
Maximum position embeddings	512
Vocabulary size	50,000

Table 4.8: The model configuration used for the NB-BERT model for text classification on the Arx dataset.

The Nb-BERT configuration for the model trained on multi-class classification using the Arx dataset is given in Table 4.8. The key setting includes three training epochs, a learning rate of $\sim$ 8.97e-05, and a weight decay of $\sim$ 0.038. The NB-BERT model has 1,024 hidden units, 16 attention heads, 24 hidden layers, and a vocabulary of 50,000 words. Other settings include a training batch size of 32, the GELU activation function, and maximum position embeddings at 512.

NorBERT2

Parameter	Value
Model	ltg/norBERT2
Training epochs	3.0
Learning rate	4.140603251643812e-05
Weight decay	0.25327001499496754
Warm-up ratio	0.140475173723635
Per device train batch size	32
Evaluation strategy Per	epoch
Hidden size	768
Number of attention heads	12
Number of hidden layers	12
Adam beta1	0.9
Adam beta2	0.999
Adam epsilon	1.0e-08
Seed	100
Transformer architecture	BERT
Position embedding type	absolute
Hidden activation function	GELU
Hidden dropout probability	0.1
Attention probabilities dropout probability	0.1
Maximum position embeddings	512
Vocabulary size	50,104

Table 4.9: The configure of the NorBERT2 model for text classification on the Arx dataset.

Table 4.9 shows the configurations of the NorBERT2 model used for multi-class text classification on the Arx dataset. The model includes 12 hidden layers, 12 attention heads, and a hidden size of 768. The model was trained for three epochs with a learning rate of $\sim$ 4.141e-05 and a weight decay of $\sim$ 0.253. The warm-up rate was set to $\sim$ 0.140. The model used the GELU activation function and utilized 512 as maximum position embeddings and a vocabulary size of 50,104. The training batch size was set to 32.

Random Forest

Parameter	Value
bootstrap	true
ccp_alpha	0.0
criterion	gini
max_features	sqrt
min_impurity_decrease	0.0
min_weight_fraction_leaf	0.0
max_depth	30
max_features	0.1177403669337236
min_samples_leaf	1
min_samples_split	9
n_estimators	498
random_state	100
warm_start	false

Table 4.10: The configuration used for the Random Forest model for text classification on the Arx dataset.

The Random Forest model configuration for multi-class classification is displayed in Table 4.10. The model was trained using bootstrap sampling and the Gini criterion. When splitting nodes, the maximum number of features was set to the square root of the total number of features. The maximum depth of the tree was set to 30, and the maximum features in each split were set to ~ 0.118. The number of trees in the forest was set to 498.

Linear SVC

Parameter	Value
dual	true
fit_intercept	true
intercept_scaling	1
loss	squared_hinge
max_iter	1000
C	0.3764727960989543
class_weight	balanced
tol	0.1
multi_class	ovr
penalty	l2
random_state	100

Table 4.11: The model configuration used for the LinearSVC model for text classification on the Arx dataset.

Table 4.11 shows the configuration of the LinearSVC model for multi-class text classification on the Arx dataset. The model was trained with the dual parameter set to true. The loss function utilized was squared hinge, and the maximum iterations were set to 1,000. The regularization strength indicated by C was set to ~ 0.376, and the class weights were balanced. The tolerance was set to 0.1, and the multi-class strategy used was one-vs-rest (ovr). The penalty used for regularization was L2.

AdaBoost

Parameter	**Value**
base_estimator	deprecated
algorithm	SAMME
learning_rate	0.09456593019309933
n_estimators	1000
random_state	100

Table 4.12: The model configuration used for the AdaBoost model for text classification on the Arx dataset.

The Adaboost model configuration for multi-class text classification is presented in Table 4.12. The algorithm used in AdaBoost is SAMME. The learning rate was set to ~ 0.095, and the number of estimators was set to 1,000.

Multilayer Perceptron

Parameter	**Value**
batch_size	auto
beta_1	0.9
beta_2	0.999
early_stopping	false
epsilon	1.0e-08
max_fun	15000
activation	logistic
alpha	0.01
hidden_layer_sizes	200
learning_rate	constant
learning_rate_init	0.001019551167116848
max_iter	1000
solver	adam
tol	1.0e-06
momentum	0.9
n_iter_no_change	10
nesterovs_momentum	true
power_t	0.5
random_state	100
shuffle	true
validation_fraction	0.1
warm_start	false

Table 4.13: The model configuration used for the Multilayer Perceptron model for text classification on the Arx dataset.

Table 4.13 includes the model configuration for the Multilayer Perceptron model for multi-class text classification on the Arx dataset. The model was trained using the Adam optimizer. The epsilon used was 1.0e-08. The maximum number of function evaluations was set to 15,000, and the activation function used was logistic. The regularization parameter alpha was set to 0.01, and the hidden layer size was set to 200. The learning rate was constant with a value of ~ 0.001, and the maximum number of iterations was 1,000. The tolerance was set to 1.0e-06.

KNeighbors

Parameter	**Value**
metric	euclidean
metric_params	null
algorithm	kd_tree
leaf_size	50
n_neighbors	17
p	1
weights	distance

Table 4.14: The model configuration used for the KNeighborsClassifier model for text classification on the Arx dataset.

The KNeighborsClassifier model configuration for the multi-class classification is given in Table 4.14. The configurations include utilizing the Euclidean distance metric, the algorithm kd tree, and a leaf size of 50. The number of neighbors used was 17, and the distance metric parameter p was set to 1. The weighting- scheme used was distance-based.

GradientBoosting

Parameter	**Value**
ccp_alpha	0.0
criterion	friedman_mse
loss	log_loss
min_impurity_decrease	0.0
min_weight_fraction_leaf	0.0
learning_rate	0.020194194038099945
max_depth	10
max_features	sqrt
min_samples_leaf	10
min_samples_split	10
n_estimators	544
random_state	100
subsample	1.0
tol	0.0001
validation_fraction	0.1
warm_start	false

Table 4.15: The model configuration used for the GradientBoostingClassifier model for text classification on the Arx dataset.

Table 4.15 shows the model configuration for the GradientBoosting model for the multi-class text classification on the Arx dataset. The model was trained using the Friedman mean squared error criterion, using a loss function of log loss. The minimum impurity decrease required for a split was set to 0, and the minimum fraction of the samples needed to be at a leaf node. The learning rate used was ~ 0.020, and the maximum depth of the tree was set to 10. The maximum number of features to consider when splitting nodes were assigned to the square root of the total number of features. The minimum number of samples required to be at a leaf node was set to 10. The number of estimators was set to 544, and the tolerance was set to 0.0001.

4.5.2 Model Configurations for the Multi-Label Dataset

NB-BERT

Parameter	**Value**
Model	NbAiLab/nb-BERT-large
Fine-tuning task	Multi-label classification
Training epochs	3
Learning rate	7.470003153100673e-05
Weight decay	0.06450301502111656
Warm-up ratio	0.4721188986762204
Per device train batch size	16
Evaluation strategy	Per epoch
Hidden size	1024
Number of attention heads	16
Number of hidden layers	24
Adam beta1	0.9
Adam beta2	0.999
Adam epsilon	1.0e-08
Seed	100
Transformer architecture	BERT
Position embedding type	absolute
Hidden activation function	GELU
Hidden dropout probability	0.1
Attention probabilities dropout probability	0.1
Maximum position embeddings	512
Vocabulary size	50,000

Table 4.16: The model configuration used for the NB-BERT model for text classification on the Multi-Label Dataset.

The model configuration for the NB-BERT model for multi-label text classification is presented in Table 4.16. The model was trained for three epochs using a learning rate of $\sim 7.470\text{e-}05$ and a weight decay of ~ 0.065. The warm-up ratio was set to ~ 0.472, and the model used absolute position embedding and the GELU activation function. The maximum position embeddings were 512, and the vocabulary size was 50,000. The training batch size was 16, and the model included 24 hidden layers, with a hidden size of 1,024 and 16 attention heads.

NorBERT2

Parameter	Value
Model	ltg/norBERT2
Fine-tuning task	multi-label classification
Training epochs	3.0
Learning rate	8.900593382516312e-05
Weight decay	0.06534504506200942
Warm-up ratio	0.050168004828590906
Per device train batch size	16
Evaluation strategy	epoch
Hidden size	768
Number of attention heads	12
Number of hidden layers	12
Adam beta1	0.9
Adam beta2	0.999
Adam epsilon	1.0e-08
Seed	100
Transformer architecture	BERT
Position embedding type	absolute
Hidden activation function	GELU
Hidden dropout probability	0.1
Attention probabilities dropout probability	0.1
Maximum position embeddings	512
Vocabulary size	50,104

Table 4.17: The configuration used for the NorBERT2 model for text classification on the Multi-Label Dataset.

Table 4.17 shows the configuration of the NorBERT2 model used for multi-label text classification on the Multi-Label Dataset. The model was trained for three epochs with a learning rate of $\sim$ 8.901e-05. The weight decay was set to $\sim$ 0.065, and the warmup ratio was set to $\sim$ 0.050. The model used the GELU activation function and had 512 as the maximum position embeddings. The vocabulary size was 50,104, and the batch size was set to 16. The model included 12 hidden layers, with a hidden size of 768 and 12 attention heads.

Random Forest

Parameter	Value
bootstrap	true
ccp_alpha	0.0
criterion	gini
min_impurity_decrease	0.0
min_weight_fraction_leaf	0.0
max_depth	22
max_features	0.4763
min_samples_leaf	1
min_samples_split	2
n_estimators	50
random_state	100
warm_start	false

Table 4.18: The model configuration used for the Random Forest model for text classification on the Multi-Label Dataset.

The configuration of the Random Fores model used on multi-label classification on the Multi-Label Dataset is given in Table 4.18. The model was trained using bootstrap sampling and a criterion of Gini. The maximum number of features was set to ~ 0.476, and the maximum tree depth was set to 22. The minimum number of samples to split an internal node was two 2. The number of trees in the forest was set to 50.

Linear SVC

Parameter	Value
model_type	OneVsRestClassifier
estimator	LinearSVC
estimator__dual	true
estimator__fit_intercept	true
estimator__intercept_scaling	1
estimator__loss	squared_hinge
estimator__max_iter	1000
estimator__multi_class	ovr
estimator__penalty	l2
estimator__random_state	100
estimator__C	0.2629
estimator__class_weight	balanced
estimator__tol	1.0e-06

Table 4.19: The model configuration used for the LinearSVC model for text classification on the Multi-Label Dataset.

Table 4.19 gives the configuration of the LinearSVC model used for multi-label text classification. The model was trained using the OneVsRestClassifier as a wrapper. The model used a squared hinge loss function, and the maximum iterations were set to 1,000. The regularization penalty was L2, and the regularization parameter C was set to ~ 0.263. The class weights were balanced, and the tolerance was set to 1.0e-06.

AdaBoost

Parameter	**Value**
model_type	MultiOutputClassifier
estimator	AdaBoostClassifier
estimator__base_estimator	deprecated
estimator__random_state	100
estimator__algorithm	SAMME.R
estimator__learning_rate	0.0432
estimator__n_estimators	817

Table 4.20: The model configuration used for the AdaBoost model for text classification on the Multi-Label Dataset.

The AdaBoost model configuration for multi-label text classification on the Multi-Label Dataset is presented in Table 4.20. The model was trained with the MultiOutputClassifier wrapper. The algorithm used in AdaBoost was SAMME.R, and the learning rate applied was ~ 0.043. The number of estimators used was 817.

Multilayer Perceptron

Parameter	**Value**
model_type	MLPClassifier
batch_size	auto
beta_1	0.9
beta_2	0.999
early_stopping	false
epsilon	1.0e-08
max_fun	15000
activation	relu
alpha	0.01
hidden_layer_sizes	200
learning_rate	adaptive
learning_rate_init	0.0022
max_iter	518
solver	adam
tol	1.0e-06
momentum	0.9
n_iter_no_change	10
nesterovs_momentum	true
power_t	0.5
random_state	100
shuffle	true
validation_fraction	0.1
warm_start	false

Table 4.21: The model configuration used for the Multilayer Perceptron model for text classification on the Multi-Label Dataset.

Table 4.21 shows the configuration of the MultiLayer Perceptron model for multi-label classification on the Multi-Label Dataset. The batch size of the model was set to auto. The maximum number of function evaluations was set to 15,000, and the activation function used was RELU. The L2 regularization parameter alpha was set to 0.01. The hidden layer size was 200, and the learning rate was set to adaptive, with an initial rate of $\sim$ 0.002. The maximum number of iterations was 518, and the optimizer used was Adam. The tolerance was set to 1.0e-06. The power_t parameter for the learning rate schedule was set to 0.5

KNeighbors

Parameter	Value
metric	euclidean
metric_params	null
algorithm	kd_tree
leaf_size	27
n_neighbors	3
p	1
weights	uniform

Table 4.22: The model configuration used for the KNeighborsClassifier model for text classification on the Multi-Label Dataset.

The KNeighborsClassifier model used for multi-label classification is displayed in Table 4.22. The model used the Euclidean distance metric. The algorithm used was kd-tree, and the leaf size was set to 27. The number of neighbors was set to 3, and the weights were assigned to uniform. The parameter p was set to 1.

GradientBoosting

Parameter	Value
model_type	MultiOutputClassifier
estimator	GradientBoostingClassifier
estimator_ccp_alpha	0.0
estimator_criterion	friedman_mse
estimator_loss	log_loss
estimator_min_impurity_decrease	0.0
estimator_min_weight_fraction_leaf	0.0
estimator__learning_rate	0.05386130750443591
estimator__max_depth	5
estimator__max_features	sqrt
estimator__min_samples_leaf	6
estimator__min_samples_split	9
estimator__n_estimators	753
estimator_random_state	100
estimator_subsample	1.0
estimator_tol	0.0001
estimator_validation_fraction	0.1
estimator_warm_start	false

Table 4.23: The model configuration used for the GradientBoosting model for text classification on the Multi-Label Dataset.

Table 4.23 presents the configuration parameters for the GradientBoosting model for multi-label classification. The model is wrapped in the MultiOutputClassifier. The criterion used was Friedman's mean squared error, and the loss function was log loss. The learning rate was set to ~ 0.054, and the maximum depth of the trees was set to 5. The maximum features considered were assigned to the square root of the total number of features, and the minimum sample for a leaf node was 6. The number of trees in the forest was 753, and the tolerance was set to 0.0001.

4.6 Hardware Specifications

The hardware and software used to train a machine learning model impact the runtime and the possible size of the model. This section will state the hardware and software specifications used to train the machine learning models in this thesis.

Component	Specification
Processor (CPU)	Intel Core i7-1165G7 @ 2.80GHz
Memory (RAM)	16 GB DDR4
GPU	Intel Iris Xe Graphics
Operating System	Windows 11 (64-bit)
Programming Language	Python 3.9.7

Table 4.24: PC Specifications

Table 4.24 presents the specification for the local machine used for training the traditional machine learning models. The setup includes a powerful enough CPU and sufficient RAM to train and evaluate the traditional machine learning models. The programming language used for implementing the models is Python. All the traditional machine learning models were trained using the `Scikit-Learn` package, and none of the implementations support GPU acceleration. This eliminated the need for a dedicated GPU in the local machine setup.

Component	Specification
Processor (CPU)	Intel Xeon @ 2.20 GHz
Memory (RAM)	52 GB
GPU	NVIDIA A100-SXM4-40GB
Operating System	Linux (64-bit)
Programming Language	Python 3.9.16

Table 4.25: Google Colab Pro + Specifications

The local machine setup does not include a powerful GPU that can accelerate the training of a deep neural network. To enable GPU acceleration, Google Colab Pro + was utilized in the model training of the BERT models [85]. The specifications for the Google Colab Pro + environment used to train the BERT models are given in Table 4.25. Google Colab Pro + provides access to powerful GPUs, which are applicable to accelerate the training of the BERT models and ensure enough memory to load and fine-tune a large model such as pre-trained Transformers often are. The GPU was exploited in training all the BERT models and significantly reduced the training times for these models at the cost of drawing more resources to the training task.

Chapter 5

Results

This chapter presents a comprehensive examination of the results derived from the traditional machine learning models applied to the Arx dataset for multi-class classification and the BERT models to the Multi-Label Dataset for multi-label classification. The results are further discussed in depth in Chapter 6. The results include the accuracy, precision, recall, and F1-score performance metrics, as well as training runtimes, for each model. The F1-score provides a more holistic view of the model's performance. However, for a more comprehensive understanding of the model's advantages and shortcomings, each metric is also individually examined in detail.

5.1 Accuracy

Accuracy is one of the most common measures to evaluate a machine learning model for classification problems, as it is easy to understand and interpret, as mentioned in Section 2.3.1.

Model	NB-BERT	NorBERT2	RF	ABoost	LSVC	MLP	KNN	GB
Accuracy	**0.875**	0.863	0.685	0.301	0.757	0.744	0.630	0.706
Runtime (s)	485.009	164.936	38.690	20.776	0.081	1126.448	**0.005**	45.321

Table 5.1: The accuracy and training runtime of the multi-class classification models tested on the Arx dataset.

Table 5.1 shows the performance of the various machine learning models tested on the multi-class classification of the Arx dataset. The models included are NB-BERT, NorBERT2, Random Forest (RF), AdaBoost (AB), Linear SVC (LSVC), Multilayer Perceptron (MLP), KNeighborsClassifier (KNN), and GradientBoosting (GB). The performance is measured in accuracy and runtime.

NB-BERT achieves the highest accuracy (0.875) among all the tested models, but it also uses a significantly longer time to run (485.009 seconds). NorBERT2, the other BERT model, has the second highest accuracy (0.863), but it has a runtime that is only a third of the runtime of NB-BERT (164.936).

The Random Forest model has an accuracy of 0.685 and a runtime of 38.690 seconds. The AdaBoost model performs the worst at the multi-class classification with an accuracy of 0.301

and a runtime of 20.776 seconds.

The Linear SVC model has the best performance of the non-BERT models, with an accuracy of 0.757 and one of the lowest runtimes (0.081). The Multilayer Perceptron has an accuracy of 0.744 and the longest runtime with a time of 1126.448 seconds.

The KNeighborsClassifier has the second lowest accuracy (0.630), though still over double as accurate as the one with the most insufficient accuracy (AdaBoost; 0.301). The runtime of the KNeighborsClassifier has the fastest runtime (0.005). The GradientBoosting model has an accuracy of 0.706 and a runtime of 45.321 seconds.

In summary, the BERT models achieved the highest accuracies in this multi-class text classification task. However, the runtimes of the BERT model are also longer than most of the traditional machine learning methods by quite a margin.

Model	NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB
Accuracy	**0.404**	0.373	0.316	0.283	0.276	0.228	0.226	0.213
Runtime (s)	194.396	74.106	9.968	156.257	0.277	141.924	**0.004**	33.552

Table 5.2: The accuracy and training runtime of the multi-label classification models tested on the Multi-Label Dataset.

Table 5.2 displays the performance of the various models trained for multi-label classification on the Multi-Label Dataset. The model types presented are the same as in the case of the multi-class classification, but all models have been modified to have a multi-label output.

The NB-BERT model achieves the highest accuracy with an accuracy of 0.404, followed by NorBERT2, achieving 0.373. The runtimes of these models are 194.396 seconds and 74.106 seconds, respectively.

The two models with the shortest runtimes are the KNeighborsClassifer model (0.004 seconds) and the LinearSVC model (0.277) seconds. Still, the accuracies are less impressive, with KNeighborsClassifier having an accuracy of 0.226 and LinearSVC having an accuracy of 0.276.

The Random Forest model and the AdaBoost model achieve the best accuracies of the traditional machine learning models with accuracies of 0.316 and 0.283, respectively. Random Forest has a short runtime of 9.968 seconds, while AdaBoost has a relatively long runtime of 156.257 seconds.

The Multilayer Perceptron has a runtime of 141.924 seconds and a low accuracy of 0.228. The GradientBoosting model has a runtime of 33.552 seconds and has the lowest accuracy of the test, with accuracy of 0.213.

To summarize the multi-label classification models, the BERT models significantly outperform the traditional machine learning models, especially when only comparing accuracies. The best BERT model, NB-BERT achieves an accuracy of ~ 0.40 compared to the best traditional machine learning method, Random Forest, which achieves an accuracy of ~ 0.32.

5.2 Precision, Recall and F1-scores

Accuracy is a suitable metric for balanced datasets, but other metrics should also be included, especially in the case of imbalanced datasets, which is the case in this thesis. This section presents the other metrics recorded while evaluating the classification models. The metrics are precision, recall, and F1-score.

5.2.1 Multi-Class Classification Performance on Arx

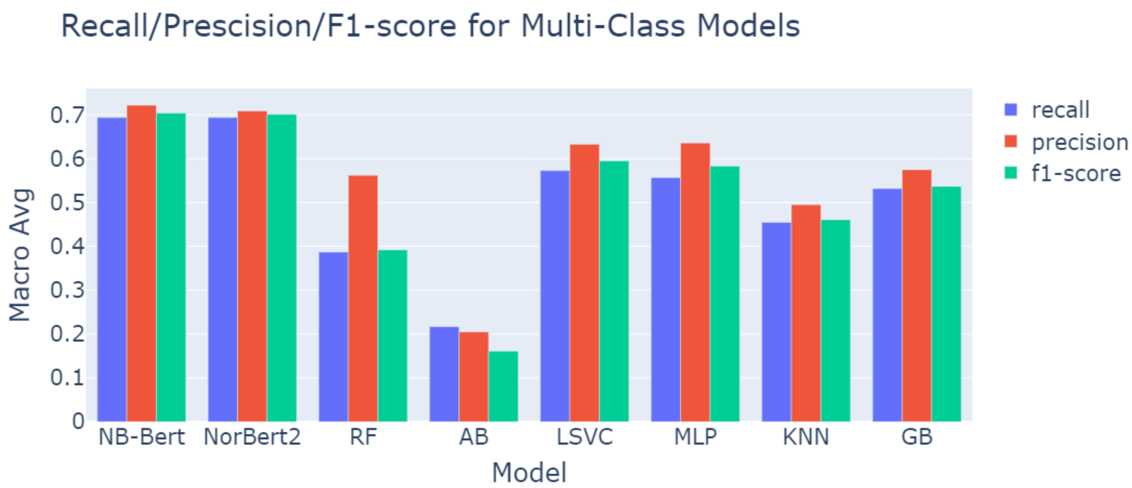


Figure 5.1: The macro-average of each model's precision, recall, and F1-score at multi-class classification on the Arx dataset.

Figure 5.1 gives an overview of the results presented in this section. The macro-average of each model's precision, recall, and F1-scores are showcased.

Model Label	F1-score NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB	Count
Chances	**<u>0.845</u>**	0.804	0.232	0.000	0.566	0.615	0.441	0.517	60
Club Detail	0.776	**<u>0.787</u>**	0.419	0.000	0.574	0.557	0.444	0.388	63
Club Drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1
Goal/Assist	**<u>0.907</u>**	0.899	0.747	0.384	0.820	0.782	0.665	0.802	224
Ignore	**<u>0.947</u>**	0.939	0.875	0.686	0.924	0.905	0.753	0.884	133
Injuries	0.800	**<u>0.833</u>**	0.000	0.000	0.737	0.588	0.250	0.692	12
Irrelevant	**<u>0.720</u>**	0.692	0.450	0.000	0.503	0.510	0.315	0.480	162
Personal Drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1
Player Detail	**<u>0.690</u>**	0.662	0.177	0.000	0.448	0.431	0.441	0.286	68
Quote	**<u>0.985</u>**	0.980	0.842	0.039	0.882	0.883	0.804	0.835	195
Red/Yellow Card	0.857	**<u>0.900</u>**	0.182	0.600	0.842	0.889	0.706	0.727	10
Transfer	**<u>0.921</u>**	0.911	0.782	0.225	0.840	0.836	0.715	0.839	177
macro avg	**<u>0.704</u>**	0.701	0.392	0.161	0.595	0.583	0.461	0.537	1106
weighted avg	**<u>0.873</u>**	0.861	0.645	0.209	0.747	0.737	0.613	0.702	1106

Table 5.3: Classwise F1-scoring for each model on multi-class classification. Macro-average and weighted-average F1-scores for each model. The number of instances in each label.

Table 5.3 gives the individual F1-scores each model achieves for each label. The combined macro-average F1-score and the weighted-average F1-score are also displayed. It is important to note that labels that score 0 are excluded from the averaged F1-scores. The number of instances of each label in the test data is stated in the "Count" column.

For most of the labels, NB-BERT and NorBERT2 have the highest F1-scores, indicating that the models perform better at the multi-class classification task on the Arx Dataset. Some labels, such as "Club Drama" and "Personal Drama", have very few instances in the training and test data making it difficult for the models to learn and classify them correctly. The result is that all models have an F1-score of 0 for these labels. The traditional machine learning algorithms do not beat the F1-score of NB-BERT and NorBERT2 when compared to the highest of the two, but the NB-BERT model is beaten once on the label "Red/Yellow Card" by the Multilayer Perceptron.

The weighted-average F1-score for each model is interesting as it considers the label imbalance. The model with the highest weighted-average F1-score is NB-BERT (0.873). The NorBERT2 model achieves a lower score of 0.861. Both still beat the traditional machine learning models with the best, LinearSVC, achieving a weighted-average F1-score of 0.747.

Model	Precision NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB	Count
Label									
Chances	0.875	**0.915**	0.889	0.000	0.652	0.727	0.448	0.793	60
Club Detail	**0.849**	0.814	0.783	0.000	0.635	0.615	0.417	0.543	63
Club Drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1
Goal/Assist	**0.882**	0.864	0.627	0.240	0.772	0.738	0.633	0.760	224
Ignore	**0.961**	0.953	0.836	0.935	0.938	0.915	0.650	0.948	133
Injuries	**1.000**	0.833	0.000	0.000	**1.000**	**1.000**	0.500	0.643	12
Irrelevant	**0.752**	0.705	0.439	0.000	0.527	0.534	0.435	0.392	162
Personal Drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1
Player Detail	**0.662**	**0.662**	0.636	0.000	0.542	0.452	0.520	0.467	68
Quote	**0.975**	0.965	0.801	0.500	0.866	0.874	0.779	0.839	195
Red/Yellow Card	0.818	0.900	**1.000**	0.450	0.889	**1.000**	0.857	0.667	10
Transfer	0.889	**0.901**	0.739	0.333	0.773	0.777	0.701	0.854	177
macro avg	**0.722**	0.709	0.562	0.205	0.633	0.636	0.495	0.575	1106
weighted avg	**0.873**	0.861	0.692	0.307	0.746	0.738	0.613	0.726	1106

Table 5.4: Classwise precision-scoring for each model on multi-class classification.

It is important to remember that the F1-score is the harmonic mean of the precision and recall and that to understand the F1-score more comprehensively, both precision and recall should be studied separately. Table 5.4 shows the classwise precision scores and the macro-average and weighted-average precision scores. Note that the labels with a score of 0 are still excluded from the averaged precision scores.

NB-BERT demonstrated the highest precision scores for most labels, followed by NorBERT2. The Random Forest model and the MLP surpasses the NB-BERT model and the NorBERT2 model at precision scoring at one label, the "Red/Yellow Card" label. Note that the Random Forest model stands out distinctly with its significantly higher macro-average precision score than its macro-average recall score, as clearly illustrated in Figure 5.1.

The weighted-average precision score of NB-BERT is the highest, with a score of 0.873, followed by NorBERT2, with a score of 0.861. The best of the traditional machine learning models, LinearSVC, has a precision score of 0.746, closely followed by Multilayer Perceptron (0.738) and GradientBoosting (0.726). The AdaBoost model scores the lowest at precision, with a score of 0.307.

Model Label	Recall NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB	Count
Chances	**0.817**	0.717	0.133	0.000	0.500	0.533	0.433	0.383	60
Club Detail	0.714	**0.762**	0.286	0.000	0.524	0.508	0.476	0.302	63
Club Drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1
Goal/Assist	0.933	0.938	0.924	**0.973**	0.875	0.830	0.701	0.848	224
Ignore	**0.932**	0.925	0.917	0.541	0.910	0.895	0.895	0.827	133
Injuries	0.667	**0.833**	0.000	0.000	0.583	0.417	0.167	0.750	12
Irrelevant	**0.691**	0.679	0.463	0.000	0.481	0.488	0.247	0.617	162
Personal Drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1
Player Detail	**0.721**	0.662	0.103	0.000	0.382	0.412	0.382	0.206	68
Quote	**0.995**	**0.995**	0.887	0.021	0.897	0.892	0.831	0.831	195
Red/Yellow Card	**0.900**	**0.900**	0.100	**0.900**	0.800	0.800	0.600	0.800	10
Transfer	**0.955**	0.921	0.831	0.169	0.921	0.904	0.729	0.825	177
macro avg	**0.694**	**0.694**	0.387	0.217	0.573	0.557	0.455	0.532	1106
weighted avg	**0.875**	0.863	0.685	0.301	0.757	0.744	0.630	0.706	1106

Table 5.5: Classwise recall-scoring for each model on multi-class classification.

Table 5.5 introduces the detailed recall scores. NB-BERT continues to be the best performer with most of the best scores at the classwise recall scores and the best scores for macro-average recall (0.694) and weighted-average recall (0.875). NorBERT2 follows a little step behind with the same score for macro-average recall (0.694) and a score of 0.875 for the weighted-average recall. AdaBoost is the only model that manages to beat both the NB-BERT and NorBERT2 model at recall scoring on a label, the "Goal/Assist" label with a score of 0.973. That is compared to NB-BERT with a recall score of 0.933 and NorBERT2 with a score of 0.938.

NB-BERT and NorBERT2 continue to be dominant in performance compared to the other traditional machine learning models by demonstrating high recall scores at most labels. They also have the highest averaged recall scores.

5.2.2 Multi-Label Classification Performance on the Multi-Label Dataset

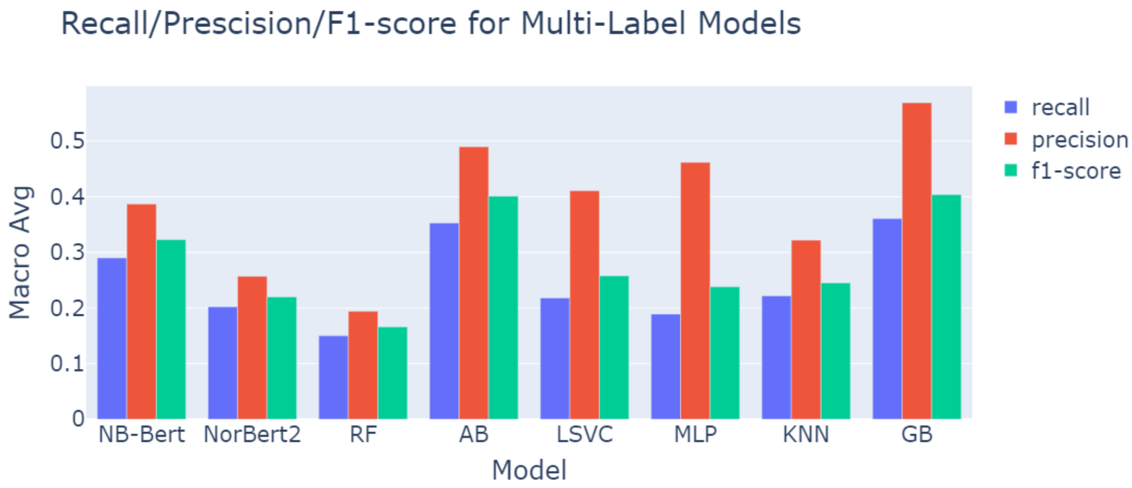


Figure 5.2: The macro-average of each model's precision, recall, and F1-score at multi-label classification on the Multi-Label Dataset.

The multi-label classification task is a more challenging problem. Figure 5.2 displays an overview of macro-average recall, precision, and F1-scores for each model. Compared to the results in Figure 5.1 for the multi-class classification, it is clear that the results are lower for most of the models, with an exception for the AdaBoost model, which seems to perform better at multi-label classification than multi-class classification at first glance.

Model Label	F1-score NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB	Count
Booking	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	1.000	3
Chance	0.000	0.000	0.000	**0.182**	0.000	0.000	0.000	0.182	8
Commentary	0.000	0.000	0.000	0.133	0.091	**0.160**	**0.160**	0.077	21
Description	0.000	0.000	0.000	**0.069**	0.000	0.047	0.000	0.047	42
Garbage	0.857	**0.867**	0.824	0.824	0.667	0.531	0.381	0.492	45
Goal	**0.656**	0.464	0.000	0.538	0.348	0.279	0.304	0.360	35
Injury	0.000	0.000	0.000	0.462	0.000	0.000	0.000	**0.714**	9
Link	**0.727**	0.667	0.600	0.600	0.600	0.600	0.545	0.600	6
Next game	0.000	0.000	0.000	0.143	0.000	0.000	0.154	**0.250**	11
Odds	0.667	0.000	0.667	**0.857**	**0.857**	**0.857**	0.800	0.400	4
Opinion	0.000	0.000	0.000	**0.333**	0.000	0.000	0.000	0.286	16
Quote	**1.000**	0.989	0.810	0.824	0.884	0.831	0.851	0.862	90
Rumour	0.000	0.000	0.000	**0.154**	0.000	0.000	0.000	0.000	6
Set-piece	0.000	0.000	0.000	0.300	0.000	0.154	0.000	**0.556**	12
Statement	0.267	0.372	0.000	0.428	0.443	0.333	**0.479**	0.278	78
Statistics	0.000	0.000	0.000	0.111	**0.154**	0.154	0.000	0.000	12
Storytelling	0.827	**0.840**	0.741	0.655	0.707	0.649	0.571	0.656	175
Substitution	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.400**	3
Summary	**0.682**	0.651	0.000	0.468	0.241	0.237	0.479	0.381	46
Table	**0.880**	0.000	0.000	0.444	0.696	0.400	0.667	0.519	14
Transfer	**0.533**	0.000	0.000	0.300	0.000	0.000	0.000	0.333	11
VAR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.500**	2
macro avg	0.323	0.220	0.166	0.401	0.258	0.238	0.245	**0.404**	649
micro avg	**0.667**	0.646	0.508	0.549	0.550	0.491	0.496	0.517	649
samples avg	**0.666**	0.663	0.504	0.521	0.506	0.433	0.435	0.448	649
weighted avg	**0.576**	0.546	0.379	0.529	0.480	0.432	0.439	0.483	649

Table 5.6: Classwise F1-scoring for each model on multi-label classification. Macro-average and weighted-average F1-scores for each model. The number of instances in each label.

Table 5.6 presents the F1-scores for the models trained for the multi-label classification task using the Multi-Label Dataset. The models being compared are, as in the case of the multi-class classification as well, the NB-BERT, NorBERT2, Random Forest (RF), AdaBoost (AB), LinearSVC (LSVC), Multilayer Perceptron (MLP), KNeighborsClassifier and GradientBoosting (GB). Table 5.6 includes the count of instances in the test data for each label, the classwise F1-scores, and the macro and weighted-average F1-scores as the F1-score table (Table 5.3) in Section 5.2.1. Table 5.6 also includes the micro-average F1-score and the samples average F1-score.

The first thing to note is that while the BERT models dominate at multi-class classification F1-scores, the spread of the highest classwise F1-scores is more extensive at the multi-label classification task. Another thing to be aware of is that the sample size of each label is smaller in the test data of the Multi-Label Dataset.

AdaBoost achieves the highest score in the same amount of labels as the NB-BERT model, but many of the labels have a low sample count except for the "Description" label. The AdaBoost resulting F1-score on the "Description" label is low, with a score of 0.069, but still the highest of all.

There are no labels that have no F1-score by all the models. Still, some are only scored by a few models such as "Booking", "Chance", "Commentary", "Description", "Injury", "Next game", "Opinion", "Rumour", "Set-piece", Substitution" and "VAR". These label adds up to being half the labels, but only 14 % of the labels in the test samples. The labels "Garbage" and "Quote" have relatively high F1-scores for all models, meaning all models can recognize these labels, but the BERT models still have the highest F1-scores for both.

Table 5.6 presents the first upset, with GradientBoosting having the best macro-average F1-score with a score of 0.404. This is compared to NB-BERT having a macro-average F1-score of 0.323 and NorBERT2 scoring of 0.220. The Random Forest model has the lowest macro-average F1-score of 0.166.

When comparing the micro-average, samples-average, and the weighted-average F1-scores, the NB-BERT model is back on top, with both BERT models scoring better than all the traditional machine learning models. The NB-BERT model scores 0.667 on the micro-average F1-score compared to GradientBoosting with a score of 0.517. The best micro-average F1-scoring traditional machine learning model is AdaBoost, with a score of 0.549. For the sample-average F1-score, NB-BERT scores 0.666 compared to the GradientBoosting score of 0.448. NB-BERT achieves a score of 0.576

To summarize, Table 5.6 indicates that the performance of the models depends on the label, and no single model consistently outperforms the other models across all labels. However, NB-BERT is still on top of the weighted-average that considers the label imbalance.

Model Label	Precision NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB	Count
Booking	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	**1.000**	3
Chance	0.000	0.000	0.000	**0.333**	0.000	0.000	0.000	**0.333**	8
Commentary	0.000	0.000	0.000	0.222	**1.000**	0.500	0.500	0.200	21
Description	0.000	0.000	0.000	0.125	0.000	**1.000**	0.000	**1.000**	42
Garbage	0.923	0.947	0.875	0.875	**0.958**	0.895	0.667	0.938	45
Goal	0.769	0.619	0.000	**0.824**	0.727	0.750	0.636	0.600	35
Injury	0.000	0.000	0.000	0.750	0.000	0.000	0.000	**1.000**	9
Link	0.800	**1.000**	0.750	0.750	0.750	0.750	0.600	0.750	6
Next game	0.000	0.000	0.000	0.333	0.000	0.000	**0.500**	0.400	11
Odds	**1.000**	0.000	**1.000**	**1.000**	**1.000**	**1.000**	0.667	0.333	4
Opinion	0.000	0.000	0.000	0.500	0.000	0.000	0.000	**0.600**	16
Quote	**1.000**	0.989	0.984	0.875	0.927	0.908	0.846	0.935	90
Rumour	0.000	0.000	0.000	**0.143**	0.000	0.000	0.000	0.000	6
Set-piece	0.000	0.000	0.000	0.375	0.000	**1.000**	0.000	0.833	12
Statement	0.519	**0.600**	0.000	0.463	0.500	0.476	0.531	0.432	78
Statistics	0.000	0.000	0.000	0.167	**1.000**	**1.000**	0.000	0.000	12
Storytelling	**0.795**	0.789	0.667	0.677	0.705	0.677	0.662	0.709	175
Substitution	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.286**	3
Summary	**0.714**	0.700	0.000	0.581	0.583	0.538	0.680	0.706	46
Table	**1.000**	0.000	0.000	0.462	0.889	0.667	0.800	0.538	14
Transfer	**1.000**	0.000	0.000	0.333	0.000	0.000	0.000	0.429	11
VAR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.500**	2
macro avg	0.387	0.257	0.194	0.490	0.411	0.462	0.322	**0.569**	649
micro avg	**0.830**	0.815	0.756	0.633	0.735	0.713	0.682	0.682	649
samples avg	0.779	**0.799**	0.608	0.572	0.585	0.511	0.516	0.517	649
weighted avg	0.623	0.580	0.390	0.598	0.609	0.652	0.540	**0.682**	649

Table 5.7: Classwise precision-scoring for each model on multi-label classification.

Table 5.7 displays the precision scores for the models trained on multi-label classification with the Multi-Label Dataset. It includes the same classwise scoring, averaged scores, and label counts as the previous table in this section. The trend with more spread in the best-achieving models, as seen in Table 5.6, also occurs in the precision scores.

The highest macro-average precision is again achieved by Gradient Boosting (0.569) as in the macro-average F1-score. Gradient Boosting also has the best weighted-average score (0.682), with NB-BERT having the second best (0.623). NB-BERT has the highest micro-average precision score of 0.830, and NorBERT2 has the best samples-average precision score of 0.799.

Model Label	Recall NB-BERT	NorBERT2	RF	AB	LSVC	MLP	KNN	GB	Count
Booking	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	**1.000**	3
Chance	0.000	0.000	0.000	**0.125**	0.000	0.000	0.000	**0.125**	8
Commentary	0.000	0.000	0.000	**0.095**	0.048	**0.095**	**0.095**	0.048	21
Description	0.000	0.000	0.000	**0.048**	0.000	0.024	0.000	0.024	42
Garbage	**0.800**	**0.800**	0.778	0.778	0.511	0.378	0.267	0.333	45
Goal	**0.571**	0.371	0.000	0.400	0.229	0.171	0.200	0.257	35
Injury	0.000	0.000	0.000	0.333	0.000	0.000	0.000	**0.556**	9
Link	**0.667**	0.500	0.500	0.500	0.500	0.500	0.500	0.500	6
Next game	0.000	0.000	0.000	0.091	0.000	0.000	0.091	**0.182**	11
Odds	0.500	0.000	0.500	0.750	0.750	0.750	**1.000**	0.500	4
Opinion	0.000	0.000	0.000	**0.250**	0.000	0.000	0.000	0.188	16
Quote	**1.000**	0.989	0.689	0.778	0.844	0.767	0.856	0.800	90
Rumour	0.000	0.000	0.000	**0.167**	0.000	0.000	0.000	0.000	6
Set-piece	0.000	0.000	0.000	0.250	0.000	0.083	0.000	**0.417**	12
Statement	0.179	0.269	0.000	0.397	0.397	0.256	**0.436**	0.205	78
Statistics	0.000	0.000	0.000	**0.083**	**0.083**	**0.083**	0.000	0.000	12
Storytelling	0.863	**0.897**	0.834	0.634	0.709	0.623	0.503	0.611	175
Substitution	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.667**	3
Summary	**0.652**	0.609	0.000	0.391	0.152	0.152	0.370	0.261	46
Table	**0.786**	0.000	0.000	0.429	0.571	0.286	0.571	0.500	14
Transfer	**0.364**	0.000	0.000	0.273	0.000	0.000	0.000	0.273	11
VAR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.500**	2
macro avg	0.290	0.202	0.150	0.353	0.218	0.189	0.222	**0.361**	649
micro avg	**0.558**	0.535	0.382	0.485	0.439	0.374	0.390	0.416	649
samples avg	**0.619**	0.604	0.456	0.522	0.485	0.408	0.409	0.433	649
weighted avg	**0.558**	0.535	0.382	0.485	0.439	0.374	0.390	0.416	649

Table 5.8: Classwise recall-scoring for each model on multi-label classification.

Table 5.8 presents the recall scores of various models in the multi-label classification task. It includes the same classwise scores and averages as the previous models of this section: Macro-average, micro-average, samples-average, and weighted-average recall score.

The macro-average recall score is highest for GradientBoosting (0.361), followed by AdaBoost (0.353). The NB-BERT model achieves a macro-average recall score of 0.290, and NorBERT2 has a score of 0.202. NB-BERT has the best results for the rest of the averaged recall scores, with 0.558 on the micro-average, 0.619 on the samples-average, and 0.558 on the weighted-average.

In summary, the best classwise scores vary across models. Still, NB-BERT achieves the best score on the metrics that consider the imbalance of the Multi-Label Dataset, such as micro-average, samples-average, and weighted-average recall.

Chapter 6

Discussion

This thesis explores the application of Norwegian pre-trained BERT models and other traditional machine learning models for text classification on Norwegian football articles. The effectiveness of Norwegian pre-trained BERT models' ability to process and understand natural language in a context with limited data availability has been showcased.

The discussion chapter will interpret the results obtained from the machine learning experiment, compare the results of the different approaches by examining the evaluation metrics, and further share views and thoughts about the data, the models, and the choices made in this thesis. The discussion will also delve into general questions related to NLP and the future use of large language models.

6.1 Analysis of the Results

The following sections delve deeper into the results obtained, analyzing them from various perspectives. This includes an examination of the performance and training runtime of the models, how the models dealt with dataset imbalance, and how different factors could have influenced the models' performance. Additionally, the ease of implementing these models is considered. It's noteworthy that understanding these elements is key to interpreting the results and their implications effectively.

6.1.1 Model Performance on Multi-Class Classification

Table 5.1 showcases the performance of the models in multi-class classification. The NB-BERT and NorBERT2 models outperform the other models, with accuracies of 0.875 and 0.863, respectively. However, these models also have longer runtimes than the traditional machine learning models. LSVC is the best traditional machine learning model, with an accuracy of 0.757 and a low runtime of 0.081 seconds. The accuracy difference between the best BERT model and the best traditional model is over 10 %, favoring BERT.

When dealing with an imbalanced dataset, it is crucial to use appropriate evaluation metrics to assess the model's performance and ensure the model is aligned with the goal of being a simplified representation of a real-world problem. Precision, recall, and F1-score are essential metrics in this context, as they provide a deeper understanding of the model's performance, giving more substance to a possible application of the model.

The results presented in Table 5.3 show the NB-BERT model achieved the highest weighted-average F1-score of 0.873, indicating that it is striking a good balance between the precision and recall. NorBERT2 closely follows with a weighted-average F1-score of 0.861. The LinearSVC model is a close competitor of the traditional machine learning models with a weighted-average F1-score of 0.747. The gap between the BERT models and the rest indicates a significant difference in their performance. The gap in F1-scores shows that the Norwegian-trained BERT models are more effective at handling the multi-class classification task on the Arx Dataset, with better balancing of precision and recall, resulting in better handling of the imbalanced dataset.

The NB-BERT and NorBERT2 also achieve the highest classwise F1-scores at all classes in 5.3, indicating that the traditional machine learning models can not keep up with results achieved by the high number of parameters, and performance boost, acquired by the pre-training of the BERT models.

Examining Table 5.4, which gives the precision score for the multi-class classification on the Arx Dataset, the NB-BERT and NorBERT2 models maintain the dominance, achieving the highest scores on most labels. However, some of the traditional machine learning models, such as Random Forest and Multilayer Perceptron, challenge the BERT models' dominance for some specific labels. For the "Red/Yellow Card" label, both Random Forest and Multilayer Perceptron models achieve a precision score of 1.0, which surpass both NB-BERT and NorBERT2, with precision scores of 0.879 and 0.887, respectively. Speculating on the reason, the better scores achieved by the traditional machine learning models might be rooted in the simpler feature representations they retain, which might reduce potential noise interfering with the model. Another reason might be randomness as a product of too few labels in the training or testing data. The difference can come down to only one or two misclassified samples.

The recall, presented in Table 5.5, is still led by NB-BERT, followed by NorBERT2. The most exciting score, apart from the BERT models still performing at a high level, is AdaBoost achieving the best recall score for the "Goal/Assist" label. A theory is that AdaBoost, an ensemble learning algorithm with a combination of multiple weaker learners, might be able to focus on a label that is especially hard to classify and concentrate its effort on a good performance for that specific label.

6.1.2 Model Performance on Multi-Label Classification

Table 5.2 demonstrates the performance of the models in multi-label classification. The BERT models, NB-BERT and NorBERT2, continue to lead in accuracy with 0.404 and 0.373, respectively. Their runtimes are also longer than most traditional models. Random Forest and AdaBoost have the best accuracies at 0.316 and 0.283 of the traditional machine learning models. However, AdaBoost has a longer runtime of 156.257 seconds, making Random Forest a better choice based on accuracy and runtime of 9.968. The difference in accuracy between the best BERT model and the best traditional model is 8.8 %, still favoring BERT.

The multi-label classification task presents a more challenging problem, as demonstrated by the results in Figure 5.2. The figure displays an overview of macro-average recall, precision, and F1-scores for each model. The results are generally lower compared to the multi-class classification results in Figure 5.1.

Table 5.6 shows the classwise F1-scores for each model on the multi-label classification task using the Multi-Label Dataset. Unlike the multi-class classification, there is a broader spread of the highest classwise F1-scores for the multi-label classification task. AdaBoost achieves the highest score in the same number of labels as the NB-BERT model, but many of these labels have a low sample count.

GradientBoosting has the best macro-average F1-score (0.404) among the models, while NB-BERT and NorBERT2 have macro-average F1-scores of 0.323 and 0.220, respectively. Nevertheless, NB-BERT returns to the top when comparing micro-average, samples-average, and weighted-average F1-scores. In this case, it could be argued that the weighted-average F1-score is more important as it considers the label performance and the label imbalance. This results in the model performance being evaluated on its ability to classify across all labels while also maintaining an account for the fact that some labels are more common than others. This is more realistic when considering the model's applicability in a real-world scenario.

The precision and recall, given in Table 5.7 and Table 5.8, provide further insight into the performance of the classification on the Multi-Label Dataset. The Gradient Boosting achieves the best macro-average precision (0.569) and the best weighted-average precision (0.682), showing that the model is best at correctly classifying without generating too many false positives. NB-BERT has the second-best score of 0.623 and the highest micro-average precision (0.830), while NorBERT2 achieves the best samples-average precision score of 0.799.

GradientBoosting has the best macro-average recall score of the models but has a more significant drop in weighted-average recall score with a score of 0.416. NB-BERT achieves the best weighted-average recall with a score of 0.558. Table 5.8 generally displays lower recall scores than previously seen precision scores, suggesting that the models are missing some true positive samples for most labels.

The scores presented show that, in most cases, the precision score is higher than the recall score for most methods. This suggests that the models are careful in their predictions, valuing correct classification at the cost of missing some instances. In other words, the models are quite conservative, avoiding false positives instead of false negatives. This behavior might be due to the class imbalance or just a result of the classes chosen for the datasets.

Achieving the correct balance between precision and recall is essential to ensure effective recognition and classification across all labels. The highest weighted-average F1-score achieved by NB-BERT indicates that the NB-BERT model strikes the best balance between these metrics for the multi-label classification task.

6.1.3 Runtime Considerations

KNeighborsClassifier consistently shows the shortest runtime across both classification tasks but does not offer competitive accuracies. The Multilayer Perceptron model has the longest runtime in the multi-class classification task but provides poor accuracies in both tasks.

BERT models, such as NB-BERT and NorBERT2, outperform traditional models in multi-class and multi-label text classification tasks but with longer runtimes. The longer runtimes can be attributed to the more substantial computational resources required for fine-tuning BERT models. Reducing the number of epochs to 2 in the fine-tuning of BERT models can lead to great results and significantly reduce the training time, with a small cost in accuracy.

The longer runtimes for the BERT models can be attributed to their model size, as the BERT models have more parameters than the traditional machine learning model. This is also consistent with the difference in runtimes for the different BERT models and their difference in parameters compared to each other. This makes them more resource-intensive during training by a long shot. The BERT models must optimize a more significant number of weights during training than traditional models, such as Random Forest and LSVC.

The BERT models are trained using GPUs, which speed up the training by parallelization. It is positive that they can be trained using GPUs, but it requires access to GPUs which may be costly or limited. The traditional machine learning models can typically only be trained using a CPU, which cannot achieve the same level of parallelization, which can be less resource-straining but also computationally slower.

The large number of parameters in the BERT models also demands higher memory resources during training and interference. This requirement can pose challenges when deploying the models on low-resource devices. The traditional machine learning models usually require little memory, making them more suitable in those restricted circumstances.

The duration of the runtime is also to be considered relative. In the results presented in Tables 5.1 and 5.2, the longest runtime for multi-class classification is observed for the Multilayer Perceptron (MLP) model, with a runtime of 1126.448 seconds (approx. 18.77 minutes), while the longest runtime for multi-label classification is found in the NB-BERT model, with a runtime of 194.396 seconds (approx. 3.24 minutes). For some tasks, this might be considered a long training time, but for creating a model that will have a greater purpose, such as a model being used as a tool for an analyst or as a feature on a website, it is not long at all. It would be very justifiable to say that the longest runtimes are relatively short to train a text classification model and that the only thing that should be considered is the accuracy and access to GPUs.

6.1.4 Recommendations Based on Model Performance

Based on the analysis of the model performance and training runtimes, the following recommendations can be derived:

For high-accuracy applications: If the primary objective is to achieve the highest accuracy in multi-class and multi-label text classification tasks, BERT models (NB-BERT and NorBERT2) are the only valid option. These models consistently outperform traditional machine learning models regarding the accuracy and weighted-average F1-score.

For heavily resource-constrained applications: If computational resources, such as processing power, memory, or budget, are limited, opt for the traditional machine learning model LSVC. While the accuracy and weighted-average F1-score of the LSVC model is lower than the BERT models, it provides shorter runtimes and lower resource consumption.

6.1.5 Possible Factors Affecting Model Results

There are multiple things to consider when evaluating a model and its results on a specific task. This section will reflect on different factors that could affect the models' performance.

The train/test split

Several challenges associated with the train/test split of the data can occur in the splits created for both the Arx Dataset and the Multi-Label Dataset. Some of these challenges will be pointed out and discussed.

Sections 3.1.1 and 3.2.1 showcase the imbalance of labels in the datasets the models are trained on, which can cause biased models. This occurs because the model fails to generalize to the underrepresented labels. The train/test split might be unfair for certain labels by including only poorly annotated samples in the training or testing data or just mismatching the proportions of labels in each entirely. Stratification can mitigate the problem to a certain extent but not remove it altogether. In future work, several techniques could be considered to further address this problem such as oversampling or data augmentation [38, 67].

As opposed to random sampling from the complete set of data samples, stratification is employed to maintain the proportion of labels in both the training and testing data [38]. Stratification can lead to too small sample sizes in specific labels, resulting in less reliable model evaluation, especially for the underrepresented labels. There is stratification in the split of the Arx Dataset and the Multi-Label Dataset. Some classes can be considered too small to be evaluated, such as "Club Drama" and "Personal Drama" in the Arx Dataset, with 1 sample each in the test split, and "Booking", "Substitution", and "VAR" in the Multi-Label Dataset. These can be considered removed from the data, as they do not occur often enough, but there is also a reason to keep them, as they are interesting subjects in football articles. Removing interesting subjects because they need to occur more would withdraw from the purpose of building a model that reflects texts from the real world.

There is a potential risk of overfitting the models to the specific data partition when only utilizing one train/test split. This would lead to poor generalization of new unseen data. Techniques such as cross-validation can be implemented to prevent this. In k-fold cross-validation, the model trains multiple times with different data splits (folds) to find good hyperparameters before testing the model on an unseen data partition, as presented in Section 2.2.1. Implementing k-fold cross-validation in the experiment would have increased the reliability of the results but was not included due to the datasets not being more balanced. Some labels in the Multi-Label Dataset needed more samples to be distributed evenly across multiple folds. Including k-fold cross-validation would have produced unreliable results for the Multi-Label Dataset. One train/test split was used for each dataset to keep the experiment the same for both models. The problem that can occur when using one split for all models is that the split is favorable for one of the models, making it achieve better results by chance.

Hyperparameter tuning

Hyperparameter tuning is one of the essential steps in the creation of machine learning models. The choice of hyperparameters usually dictates the performance of a model, and each model needs different hyperparameters for distinct tasks and different data. The tuning process experiments to find the best hyperparameters for the job.

The weighted-average F1-score was used as the metric to optimize during the hyperparameter tuning of the models. This metric was chosen as it effectively handles the class imbalance by weighting the classes by the number of instances. If the datasets had been less imbalanced, the accuracy could have been considered as the metric to be optimized.

Keeping in mind that the hyperparameter tuning can significantly impact the performance of the models in the thesis, the number of combinations tried for each model before choosing the best was set to the same for each task, at 30 combinations. This ensured that the time spent tuning each model was kept the same. The challenge still, seen in the tuning, is that some models are easier tuned than others, such as the BERT models, which means that more combinations of hyperparameters work well. The result can be that some models do not achieve optimal hyperparameters with the chosen hyperparameter span during the 30 different iterations of models. This challenge is hard to tackle, which is the reason for keeping the number of iterations of hyperparameter tuning the same, as it is possible to evaluate the models using the same effort at tuning each model. Based on the results, a suggestion is that the BERT models are easier to tune into good-performing models.

6.1.6 Ease of Implementation

When evaluating and comparing models, it is crucial to consider their performance and ease of implementation. If a model needs to be better documented and easier to integrate into a machine learning pipeline, it may deter practitioners from adopting it, regardless of its potential effectiveness. Therefore, the practical aspects of the deployment and maintenance of these models should be considered. Models impractical to deploy have a lower chance of being applied in real-world projects.

This thesis's models were well documented and easy to implement in a machine learning pipeline, as recorded in the Github repository of this thesis. The traditional machine learning models were all included in the `Scikit-Learn` machine learning library for Python, and the BERT models were all included in the `Transformers` package provided by Huggingface. Overall, the ease of implementation can be considered similar, and all differences in difficulty can be distinguished by the hyperparameter tuning.

6.2 The Annotation Process

Despite efforts to streamline the annotation process and maintain quality, several challenges are associated with the annotation of both the Arx and Multi-Label datasets. This section will mainly discuss the annotation process of the Multi-Label Dataset presented in this thesis.

The Multi-Label Dataset was annotated by a single annotator, which in all of reality, has introduced subjectivity and bias into the annotations. Other inconsistencies, such as errors caused by fatigue or misclassification, most likely occurred during the annotation. The ideal scenario would introduce multiple annotators, cross-verifying the labels, causing improved dataset reliability.

The Multi-Label Dataset comprises 22 non-mutually exclusive labels, which might not be sufficient to capture all the nuances in the dataset. Adding more would prove difficult, though, as the annotator requires extensive knowledge of each label to classify correctly. Non-mutual exclusiveness, or overlap in classes, is common in NLP, as language is inherently complex and ambiguous. One single text can touch upon multiple topics; for instance, a football article might present a summary of a game but also include an injury and a goal. Regarding the Multi-Label Dataset, the non-mutual exclusiveness of labels reflects the complexity of the texts, therefore presenting the challenge of defining the correct level of granularity for the annotations. Adding more specific label can capture even more nuances but requires more extensive knowledge to annotate and leads to more sparse and imbalanced data. More general labels can be easier to annotate but might not capture enough details to provide a relevant model. The balance between these levels of annotation requires domain knowledge and multiple iterations of annotations. The number of classes should also be affected by the intended application of the model, which in the case of this thesis, is to test out the models on a challenging classification task with few labeled instances relative for an NLP model.

The aid of an active learning model was implemented in annotating the Multi-Label Dataset. The job of the active learning model is to assist the annotator by suggesting labels, but it is only as efficient as the training data. Some limitations may occur if the initial annotations follow a different standard than intended for the annotation process. This can happen if the annotator uses some time to familiarize with the labels available for the task. This can result in the active learning model suggesting poor labels, making the annotator unsure or biased when setting their label. Another constraint involves the specific algorithm employed for active learning. The BinaryRelevance model with a LinearSVC estimator is utilized with a predetermined set of hyperparameters. The model was chosen because of efficient and accurate results in multi-class classification on the Arx Dataset by Nordskog et al. [12]. However, that is no guarantee that the model will be optimal for the multi-label annotation of the Multi-Label Dataset. The only way to evaluate the aid provided by the active learning model at the annotation of the Multi-Label Dataset is to assess if the model provided more help than confusion, which is objectively hard to judge. Regarding the annotation of the Multi-Label Dataset, the annotator states that the model helped by pre-annotating the most manageable paragraphs to label, making them faster to confirm. The model also pre-annotated some paragraphs wrongly, providing the need for extra correction by the annotator. Overall, the active learning model provided motivation, making the annotation process more fun and interactive. Still, the inclusion of it did probably not affect the quality of annotations. To address the challenges that have come up during annotation, future annotation on the topic could include multiple annotators, multiple tests of annotation models, and revisiting the labels to ensure it adequately represents the data.

6.3 Reflective Analysis

This section will discuss the various aspects of the experiment that could have been changed or included, which might have led to more comprehensive results in comparing the models presented in this thesis. The reflection aims to provide valuable insight into future research on the topic.

The traditional machine learning algorithms used a TF-IDF representation of the datasets, as explained in Section 4.3.2. The TF-IDF, presented in Section 2.2.1, is a vectorization of the texts that primarily focuses on the frequency of the terms. Newer representations, such as Word2Vec, mentioned in Section 2.1, can capture semantic relationships between words, as well. The change to Word2Vec could possibly lead to noticeable improvements for the traditional machine learning models, but to determine the exact effect, a new experiment should be conducted with Word2Vec as the text vectorizer.

The hyperparameters of all models were tuned to get the best performance on the test split defined before the experiment. This setup could lead to the models becoming overfit to the test data if being tuned an excessive amount. The tuning was kept at 30 iterations to prevent this, but a better approach would be to exclude the test split from the hyperparameter tuning. This would have led to a more accurate evaluation of the performance of the models on unseen data. The positive thing is that most of the BERT models, trained during the hyperparameter tuning, achieved better results than all the traditional machine learning models. This indicates that the BERT model does not perform better just because it was more able to overfit the test data than the traditional machine learning algorithms.

The annotation process in this thesis could have been improved by having multiple annotators cross-validate each other's work to ensure the objectivity of the annotations. It would be worthwhile to explore multiple annotators in future research, in addition to incorporating a larger number of annotated samples. Increasing the sample size could potentially enhance the performance of the models by providing more instances of the infrequently occurring labels, thus enabling more efficient modeling.

To conclude, there are some changes to the text vectorization, applied in the preprocessing, the train/test split, and the annotation process that would have further improved the quality of the experiment in this thesis. However, the results presented in the thesis have contributed to the field of Norwegian pre-trained NLP models with a new comparison of the Norwegian pre-trained BERT models versus traditional machine learning models on Norwegian sports article classification.

6.4 Impact on Sports Article Classification

The top-performing models in this study achieve notable results for the multi-class classification of football articles. Most of the labels are also relevant to other sports, meaning that the models could be trained on a dataset containing paragraphs from multiple sports article domains with minor adjustments to the set of labels. This could improve the generalizability of the models and expand the use case. The best model trained for multi-class classification of football articles could be implemented today to assist search engines in filtering relevant paragraphs or personalizing content for people with specific reading tastes. The model could also provide the online news outlet with greater insights into what the composition of popular articles looks like. Information, such as the impact of having many non-informative ads for betting opportunities, as an example, can be valuable to evaluate. The potential repercussions of including them too often can be measured by tagging the paragraphs containing the odds information and collecting the amount of time the reader spends on the article.

The multi-label classification models achieve a lower accuracy, and a more thorough evaluation is recommended before applying them to real-life use cases. The positive of the multi-label model is that the lower accuracy is mostly caused by failure to predict at all, instead of predicting wrongly, which is not a critical mistake for a sports article classification model. Most use cases can still be implemented, but things that can decrease the user experience should be refrained from implementation, such as content personalization.

6.5 The Benefits and Limitations of BERT

Some aspects of the BERT models are beneficial for NLP tasks compared to the other models tested in this thesis and to other Transformer models in general. This section will reflect upon those aspects and explain why the BERT models are relevant for the sports article classification task. The possible drawbacks of the BERT models will also be discussed.

BERT has an unfair advantage against the traditional machine learning models as it is already pre-trained and contains 355 million parameters, in the case of NB-BERT$_{LARGE}$, as presented in Section 2.2.4. One could suggest that this makes the comparison between the traditional machine learning models and the BERT models unfair. A different take on it is that the experiment just showcases the new way of tackling NLP problems and that large pre-trained language models are seemingly the future. The number of parameters the pre-trained model contains, which is the inherent characteristic of Transformer-based models, is what allows the BERT model to learn more complex patterns, contributing to its superior performance. This is simply a benefit inherent to the BERT model. Another advantage related to the size of the pre-trained BERT model is that it makes the need for large amounts of task-specific data much smaller. Fine-tuned BERT models manage well with small amounts of training data compared to other NLP models, as showcased in this thesis. The drawback of the BERT models containing millions of parameters before being fine-tuned is, as mentioned earlier, that the models are computationally expensive. They require powerful GPUs and significant time to train, especially if fine-tuned with a large dataset.

The BERT model is one of several recent Transformer-based architectures introduced. The other includes GPT and T5, as mentioned in Section 1.2. These models have also achieved state-of-art performance on various NLP tasks, but the models have some key differences. One of the primary benefits of the BERT model is that it is open-source. Everyone can see the source code and the data used to create it. This is true for the T5 model as well. This is not the case with the latest GPT models, as they must be accessed through an API (application programming interface) due to concerns of misuse and the size of the model. Being open-source is a benefit as it provides accessibility and customization and ensures continuous improvement of the model. The other benefit of the BERT model and the T5 model is that they are trained to understand context bidirectionally, as explained in Section 2.2.4, unlike GPT, which is unidirectional. Being a bidirectional model can be more effective for some NLP tasks as it can capture context from both the left and the right side of a relevant part of an input. The last key difference is that BERT models are specifically trained for tasks requiring deep language understanding, which allows them to capture language patterns and relationships, making them a suitable choice for tasks like sports article classification. T5 uses both the Encoder and Decoder, shown in Figure 2.6, which makes it built for solving text-to-text problems. This can be versatile but requires some specific formulation to solve a text classification task. The GPT model focuses on language generation tasks. Overall, the choice of BERT is the most natural for the text classification task.

Transformer models, such as BERT, are achieving state-of-art performance in NLP, as showcased in Chapter 5 of this thesis, but there are some drawbacks to these large-scale language models. They are computationally expensive to pre-train, resulting in a larger negative impact on the environment than traditional machine learning algorithms. The defense for the continued training and development of Transformer models, excluding the presented improvement in performance, is that these pre-trained models are reusable. One large BERT model, for instance, can be fine-tuned for a wide range of tasks, reducing the need to train separate models for every problem out there. The fine-tuning of these models is relatively inexpensive computational-wise, and the resulting models achieve excellent performance.

6.6 Future Work

This thesis has demonstrated the potential of Norwegian pre-trained BERT models for text classification, specifically on multi-class and multi-label classification of football article paragraphs. The results have demonstrated the potential of Transformer models for low-resource languages, but there are still several areas in which future research could expand upon the findings of this thesis:

- **More Transformer architectures:** Evaluating the performance of other Transformer models, such as GPT and T5, on the same datasets. This would supply a more comprehensive dive into the utilization of Transformer models and their efficiency in the context of NLP in low-resource languages.
- **Additional NLP tasks:** The current work has focused on text classification tasks. However, Transformer models have shown promise in various NLP tasks such as question answering, text summarization, and machine translation. Future research could investigate the application of Norwegian pre-trained BERT models to these tasks, which would further explore their versatility in handling different language challenges.
- **Other low-resource languages:** This thesis examined the potential of Transformer models for the Norwegian language, which is considered a low-resource language. It would be interesting to replicate this research for other low-resource languages, to further demonstrate the performance of Transformer models at NLP tasks across languages with more challenging initial conditions.

With the research conducted in this thesis and further pursuit into these avenues of research, it is hoped that the field of NLP for lower-resource languages will continue to advance. Further research will hopefully lead to more utilization and development of large language models, providing better language understanding and more chances to process the vast array of different human languages.

Chapter 7

Conclusion

The primary goal of this thesis was to explore the potential of Norwegian pre-trained BERT models for text classification tasks in the context of lower-resource languages. The goal was to be achieved by doing paragraph classification of Norwegian sports articles using both Norwegian pre-trained BERT models and traditional machine learning models. The thesis demonstrated the effectiveness of Norwegian pre-trained BERT models in accurately and efficiently classifying multi-class paragraphs from football articles, showcasing their potential.

The thesis also introduced a new multi-label dataset of paragraphs from football articles, which served as a valuable resource in evaluating the BERT models. Including a multi-label classification task allowed the Norwegian pre-trained BERT models to prove their versatility and provide a more comprehensive exhibition of their ability at text classification.

In the multi-class and the multi-label classification tasks, the Norwegian pre-trained BERT models achieved notable results, with accuracies of ~ 0.88 and ~ 0.40 and weighted-average F1-scores of ~ 0.87 and ~ 0.58, respectively. These results significantly outperformed the traditional machine learning models such as; Random Forest, AdaBoost, Linear Support Vector Machine, Multilayer Perceptron, KNeighbors Classifier, and Gradient Boosting. The outperformance highlighted the advantages of using Transformer-based models for lower-resource languages.

The research question posed at the beginning of this thesis was: "How can Norwegian pre-trained BERT models be utilized for effective text classification on paragraphs from Norwegian football articles, and how does their performance compare to traditional machine learning models?". In response, the findings of this thesis suggest that Norwegian pre-trained BERT models can be effectively utilized for text classification on paragraphs from Norwegian football articles by employing them in both multi-class and multi-label classification tasks. The BERT models demonstrated superior performance in accuracy and F1-score over traditional machine learning models, providing compelling evidence of the viability of Transformer-based models, particularly BERT, for lower-resource languages like Norwegian. However, for real-world applications, especially for more challenging tasks such as multi-label classification, further improvements, and better dataset annotation may be required. The precise influence of each factor, namely the model capabilities and the dataset quality, on the current performance levels is still unclear and needs more research.

In conclusion, this thesis contributes to the growing body of research on Transformer models for lower-resource languages by demonstrating the potential of Norwegian pre-trained BERT models in text classification tasks. The promising results achieved in this study indicate that NLP is reaching a satisfactory level for providing sustainable solutions to some comprehensive text processing problems, such as multi-class classification. For other tasks, such as multi-label classification, Norwegian pre-trained BERT models have achieved the best results of the models tested. However, there is still some way to go before it can be concluded that the solutions are adequate for real-life multi-label problems. It remains inconclusive whether the results are due to the model not handling the more challenging classification task or the Multi-Label Dataset needing better annotations.

Overall, the progress in Natural Language Processing holds great promise for the future of the field. The improvements posed by utilizing Transformer models, particularly for lower-resource languages, are actively driving further progress in the ability to process text. The possibility of performing text processing with limited data resources is now very much real. Hopefully, the future will hold even more development in NLP, pushing for more effective solutions and widespread availability.

Bibliography

[1] Pinkesh Badjatiya et al. “Deep Learning for Hate Speech Detection in Tweets”. In: *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*. ACM Press, 2017. DOI: 10.1145/3041021.3054223. URL: https://doi.org/10.1145%2F3041021.3054223.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *CoRR* abs/1409.0473 (2014).

[3] Dongfang Lou et al. “MLBiNet: A Cross-Sentence Collective Event Detection Network”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4829–4839. DOI: 10.18653/v1/2021.acl-long.373. URL: https://aclanthology.org/2021.acl-long.373.

[4] Statistics Norway. *Population in Norway*. Retrieved from Statistics Norway (SSB) on May 6, 2023. 2023. URL: https://www.ssb.no/en/befolkning/statistikker/folkemengde.

[5] W3Techs. *Usage of content languages for websites*. Retrieved on May 6, 2023. 2023. URL: https://w3techs.com/technologies/overview/content_language.

[6] Krystal Hu. *CHATGPT sets record for fastest-growing user base - analyst note*. Feb. 2023. URL: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/.

[7] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[8] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: (2020). arXiv: 2005.14165 [cs.CL].

[9] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

[10] Per E Kummervold et al. “Operationalizing a National Digital Library: The Case for a Norwegian Transformer Model”. In: *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*. Reykjavik, Iceland (Online): Linköping University Electronic Press, Sweden, June 2021, pp. 20–29. URL: https://aclanthology.org/2021.nodalida-main.3.

[11] Andrey Kutuzov et al. "Large-Scale Contextualised Language Modelling for Norwegian". In: *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*. Reykjavik, Iceland (Online): Linköping University Electronic Press, Sweden, June 2021, pp. 30–40. URL: https://aclanthology.org/2021.nodalida-main.4.

[12] Aanund Jupskås Nordskog et al. "Semantic Analysis of Soccer News for AutomaticGame Event Classification". In: CBMI'19. Dublin, Ireland, 2019.

[13] TV 2. *TV 2 - Nyheter, sport, vær og underholdning.* 2023. URL: https://www.tv2.no/.

[14] Verdens Gang. *VG - Verdens Gang.* 2023. URL: https://www.vg.no/.

[15] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[16] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[17] Chi Sun, Luyao Huang, and Xipeng Qiu. "Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence". In: *CoRR* abs/1903.09588 (2019). arXiv: 1903.09588. URL: http://arxiv.org/abs/1903.09588.

[18] Murat Tezgider, Beytullah Yildiz, and Galip Aydin. "Text classification using improved bidirectional transformer". In: *Concurrency and Computation: Practice and Experience* 34.9 (2022), e6486. DOI: https://doi.org/10.1002/cpe.6486. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6486. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6486.

[19] Yang Liu and Mirella Lapata. "Text Summarization with Pretrained Encoders". In: *CoRR* abs/1908.08345 (2019). arXiv: 1908.08345. URL: http://arxiv.org/abs/1908.08345.

[20] Myle Ott et al. "fairseq: A Fast, Extensible Toolkit for Sequence Modeling". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 48–53. DOI: 10.18653/v1/N19-4009. URL: https://aclanthology.org/N19-4009.

[21] Antti Virtanen et al. "Multilingual is not enough: BERT for Finnish". In: *ArXiv* abs/1912.07076 (2019).

[22] Wilson Wongso, Henry Lucky, and Derwin Suhartono. *Pre-Trained Transformer-Based Language Models for Sundanese.* Sept. 2021. DOI: 10.21203/rs.3.rs-907893/v1.

[23] Martin Malmsten, Love Börjeson, and Chris Haffenden. *Playing with Words at the National Library of Sweden – Making a Swedish BERT.* 2020. arXiv: 2007.01658 [cs.CL].

[24] Daniel Holmer and Arne Jönsson. "Comparing the performance of various Swedish BERT models for classification". In: *:* 2020.

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

[26] I M Rabbimov and S S Kobilov. "Multi-Class Text Classification of Uzbek News Articles using Machine Learning". In: *Journal of Physics: Conference Series* 1546.1 (May 2020), p. 012097. DOI: 10.1088/1742-6596/1546/1/012097. URL: https://dx.doi.org/10.1088/1742-6596/1546/1/012097.

[27] Adrita Barua, Omar Sharif, and Mohammed Moshiul Hoque. "Multi-class Sports News Categorization using Machine Learning Techniques: Resource Creation and Evaluation". In: *Procedia Computer Science* 193 (2021). 10th International Young Scientists Conference in Computational Science, YSC2021, 28 June – 2 July, 2021, pp. 112–121. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.11.002. URL: https://www.sciencedirect.com/science/article/pii/S1877050921021268.

[28] Mita K. Dalal and Mukesh A. Zaveri. "Automatic Text Classification of sports blog data". In: *2012 Computing, Communications and Applications Conference*. 2012, pp. 219–222. DOI: 10.1109/ComComAp.2012.6154802.

[29] Young Joon Park et al. "A deep learning-based sports player evaluation model based on game statistics and news articles". In: *Knowledge-Based Systems* 138 (2017), pp. 15–26. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2017.09.028. URL: https://www.sciencedirect.com/science/article/pii/S095070511730446X.

[30] Julia Hirschberg and Christopher D. Manning. "Advances in natural language processing". In: *Science* 349.6245 (2015), pp. 261–266. DOI: 10.1126/science.aaa8685. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8685. URL: https://www.science.org/doi/abs/10.1126/science.aaa8685.

[31] Siwei Lai et al. "Recurrent Convolutional Neural Networks for Text Classification". In: *AAAI Conference on Artificial Intelligence*. 2015.

[32] Tarun Lalwani et al. "Implementation of a Chat Bot System using AI and NLP". In: *International Journal of Innovative Research in Computer Science & Technology* 6 (May 2018), pp. 26–30. DOI: 10.21276/ijircst.2018.6.3.2.

[33] Kai Jiang and Xi Lu. "Natural Language Processing and Its Applications in Machine Translation: A Diachronic Review". In: *2020 IEEE 3rd International Conference of Safe Production and Informatization (IICSPI)*. 2020, pp. 210–214. DOI: 10.1109/IICSPI51290.2020.9332458.

[34] Uday Kamath, John Liu, and James Whitaker. "Introduction". In: *Deep Learning for NLP and Speech Recognition*. Cham: Springer International Publishing, 2019, pp. 3–38. ISBN: 978-3-030-14596-5. DOI: 10.1007/978-3-030-14596-5_1. URL: https://doi.org/10.1007/978-3-030-14596-5_1.

[35] Ming Zhou et al. "Progress in Neural NLP: Modeling, Learning, and Reasoning". In: *Engineering* 6.3 (2020), pp. 275–290. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2019.12.014. URL: https://www.sciencedirect.com/science/article/pii/S2095809919304928.

[36] Joseph Weizenbaum. "ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine". In: *Commun. ACM* 9.1 (Jan. 1966), pp. 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168. URL: https://doi.org/10.1145/365153.365168.

[37] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999. ISBN: 0262133601.

[38] Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Birmingham, UK: Packt Publishing, 2022. ISBN: 978-1801819312.

[39] M. E. Maron. "Automatic Indexing: An Experimental Inquiry". In: *J. ACM* 8.3 (July 1961), pp. 404–417. ISSN: 0004-5411. DOI: 10.1145/321075.321084. URL: https://doi.org/10.1145/321075.321084.

[40] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *International Conference on Learning Representations*. 2013.

[41] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *ArXiv* abs/1310.4546 (2013).

[42] Daniel Jurafsky and James H. Martin. “Chapter 6.8 - Word2Vec”. In: *Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2000, pp. 119–125. ISBN: 978-0-13-095069-7.

[43] *Chat.openai.com*. URL: `https://chat.openai.com/`.

[44] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: *ArXiv* abs/1604.07316 (2016).

[45] Yiheng Wei et al. “Fraud Detection by Machine Learning”. In: *2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)* (2020), pp. 101–115.

[46] Taekeun Hong et al. “Enhancing Personalized Ads Using Interest Category Classification of SNS Users Based on Deep Neural Networks”. In: *Sensors (Basel, Switzerland)* 21 (2020).

[47] Christina Christakou, Spyros Vrettos, and Andreas Stafylopatis. “A hybrid movie recommender system based on neural networks”. In: *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)* (2005), pp. 500–505.

[48] Tom M Mitchell. “Introduction”. In: *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[49] Daniel Jurafsky and James H. Martin. “Chapter 4.1 - Naive Bayes Classifiers”. In: *Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2000, pp. 59–62. ISBN: 978-0-13-095069-7.

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 1476-4687. DOI: `10.1038/nature14539`. URL: `https://doi.org/10.1038/nature14539`.

[51] Laith Alzubaidi et al. “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”. In: *Journal of Big Data* 8 (2021).

[52] Ivo D. Dinov. “Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines”. In: *Data Science and Predictive Analytics: Biomedical and Health Applications using R*. Cham: Springer International Publishing, 2018, pp. 383–422. ISBN: 978-3-319-72347-1. DOI: `10.1007/978-3-319-72347-1_11`. URL: `https://doi.org/10.1007/978-3-319-72347-1_11`.

[53] Frank Emmert-Streib et al. “An Introductory Review of Deep Learning for Prediction Models With Big Data”. In: *Frontiers in Artificial Intelligence* 3 (2020).

[54] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. “ACTIVATION FUNCTIONS IN NEURAL NETWORKS”. In: *International Journal of Engineering Applied Sciences and Technology* 04 (May 2020), pp. 310–316. DOI: `10.33564/IJEAST.2020.v04i12.054`.

[55] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014).

[56] Jindong Wang and Yiqiang Chen. “Pre-Training and Fine-Tuning”. In: *Introduction to Transfer Learning: Algorithms and Practice*. Singapore: Springer Nature Singapore, 2023, pp. 125–140. ISBN: 978-981-19-7584-4. DOI: `10.1007/978-981-19-7584-4_8`. URL: `https://doi.org/10.1007/978-981-19-7584-4_8`.

[57] Jindong Wang and Yiqiang Chen. “Introduction”. In: *Introduction to Transfer Learning: Algorithms and Practice*. Singapore: Springer Nature Singapore, 2023, pp. 3–38. ISBN: 978-981-19-7584-4. DOI: 10.1007/978-981-19-7584-4_1. URL: https://doi.org/10.1007/978-981-19-7584-4_1.

[58] Y. Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 5 (Feb. 1994), pp. 157–66. DOI: 10.1109/72.279181.

[59] Jinmian Ye et al. “Learning Compact Recurrent Neural Networks with Block-Term Tensor Decomposition”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2017), pp. 9378–9387.

[60] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[61] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[62] Albert Zeyer et al. “A Comparison of Transformer and LSTM Encoder Decoder Models for ASR”. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. 2019, pp. 8–15. DOI: 10.1109/ASRU46091.2019.9004025.

[63] *Wikipedia*. URL: https://en.wikipedia.org/.

[64] Per Kummervold, Freddy Wetjen, and Javier de la Rosa. “The Norwegian Colossal Corpus: A Text Corpus for Training Large Norwegian Language Models”. In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, June 2022, pp. 3852–3860. URL: https://aclanthology.org/2022.lrec-1.410.

[65] Yang You et al. “Reducing BERT Pre-Training Time from 3 Days to 76 Minutes”. In: *CoRR* abs/1904.00962 (2019). arXiv: 1904.00962. URL: http://arxiv.org/abs/1904.00962.

[66] *Norsk aviskorpus*. https://www.nb.no/sprakbanken/ressurskatalog/oai-nb-no-sbr-4/. Dataset. 2020.

[67] Peter Harrington. *Machine Learning in Action*. USA: Manning Publications Co., 2012. ISBN: 1617290181.

[68] Kenneth Reitz. *Requests: HTTP for Humans*. Accessed: 2023-03-12. 2023. URL: https://requests.readthedocs.io.

[69] Leonard Richardson. “Beautiful soup documentation”. In: (Apr. 2007). URL: https://beautiful-soup-4.readthedocs.io/en/latest/.

[70] *ARX*. URL: https://datasets.simula.no/arx/.

[71] Hiroki Nakayama et al. *doccano: Text Annotation Tool for Human*. Software available from https://github.com/doccano/doccano. 2018. URL: https://github.com/doccano/doccano.

[72] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[73] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. arXiv: 1910.03771 [cs.CL].

[74] Adam Paszke et al. “Automatic Differentiation in PyTorch”. In: *NIPS 2017 Workshop on Autodiff*. Long Beach, California, USA, 2017.

[75] T. Hastie, R. Tibshirani, and J.H. Friedman. "Chapter 15 - Random Forests". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009, pp. 587–604. ISBN: 9780387848846.

[76] T. Hastie, R. Tibshirani, and J.H. Friedman. "Chapter 10 - Boosting and Additive Trees". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009, pp. 337–389. ISBN: 9780387848846.

[77] Thorsten Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features". In: *Machine Learning: ECML-98*. Ed. by Claire Nédellec and Céline Rouveirol. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142. ISBN: 978-3-540-69781-7.

[78] Andreas Wichert and Luis Sa-Couto. "Chapter 6 - Multilayer Perceptron". In: *Machine Learning: A Journey to Deep Learning : with Exercises and Answers*. World Scientific, 2021, pp. 262–291. ISBN: 9789811234057.

[79] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[80] P. Szymański and T. Kajdanowicz. "A scikit-based Python environment for performing multi-label classification". In: *ArXiv e-prints* (Feb. 2017). arXiv: 1702.01460 [cs.LG].

[81] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. "On the stratification of multi-label data". In: *Machine Learning and Knowledge Discovery in Databases* (2011), pp. 145–158.

[82] Piotr Szymański and Tomasz Kajdanowicz. "A Network Perspective on Stratification of Multi-Label Data". In: *Proceedings of the First International Workshop on Learning with Imbalanced Domains: Theory and Applications*. Ed. by Luís Torgo et al. Vol. 74. Proceedings of Machine Learning Research. ECML-PKDD, Skopje, Macedonia: PMLR, 2017, pp. 22–35.

[83] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[84] Ryan Turner et al. "Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020". In: *CoRR* abs/2104.10201 (2021). arXiv: 2104.10201. URL: https://arxiv.org/abs/2104.10201.

[85] Ekaba Bisong. "Google Colaboratory". In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7. URL: https://doi.org/10.1007/978-1-4842-4470-8_7.

[86] Daniel Jurafsky and James H. Martin. *Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2000. ISBN: 978-0-13-095069-7.

[87] T. Hastie, R. Tibshirani, and J.H. Friedman. Springer series in statistics. Springer, 2009. ISBN: 9780387848846.

Appendix A

Table of Python-packages

Package name	Version	Purpose of use
Beautiful Soup	4.10.0	Web scraping tools for pulling data out of HTML
Datasets	2.10.1	A lightweight and extensible library to easily share and access datasets
Doccano	1.8.3	Open source text annotation tool
Doccano Client	1.2.7	Python client for Doccano's API
Flask	2.2.2	A micro web framework written in Python
HuggingFace Hub	0.13.2	A repository of trained machine learning models, contributed by the wide AI community
NumPy	1.24.2	A package for scientific computing with Python
Pandas	1.5.3	Data exploration and manipulation
Plotly	5.13.0	A Python graphing library
PyTorch	1.13.1	An open source machine learning framework
Requests	2.28.2	A library for making HTTP requests in Python
Scikit-Learn	1.2.2	Tools for predictive data analysis
Scikit-Multilearn	0.2.0	A Python library for performing multi-label classification
Transformers	4.26.0	State-of-the-art Natural Language Processing tools from HuggingFace for PyTorch
Wandb	0.14.0	Weights and Biases, a tool for machine learning experiment tracking and model management.

Table A.1: The Python-packages used during various stages of the data science workflow, including data exploration, data processing and machine learning, along with their respective versions and their purpose of use.

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway