BACHELOR'S THESIS 2022

Sentiment Analysis from ESG Point-of-View Using ML

Oscar Johansson, Alexander Möhle

ISSN 1651-2197
LU-CS/HBG-EX: 2022-10

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

HÖGSKOLEINGENJÖRSARBETE
Datavetenskap

LU-CS/HBG-EX: 2022-10

Sentiment Analysis from ESG Point-of-View Using ML

Sentimentanalys från ESG-perspektiv med hjälp av ML

Oscar Johansson, Alexander Möhle

Sentiment Analysis from ESG Point-of-View Using ML

Oscar Johansson
os3111jo-s@student.lu.se

Alexander Möhle
al3274mo-s@student.lu.se

June 22, 2022

Bachelor's thesis work carried out at Sanctify Financial Technologies AB.

Supervisors: Marcus Klang, marcus.klang@cs.lth.se
Gustav Johnsson Henningsson, gustav@sanctify.ai
Henrik Ljunger, henrik.ljunger@sanctify.ai

Examiner: Christin Lindholm, christin.lindholm@cs.lth.se

Abstract

Investing in sustainable companies can be difficult as there are *environmental, social and governance* (ESG) risks. Rating companies using the *ESG* scoring concept can allow investors to make informed choices that may result in better long-term investments. *Sanctify* is a company that provides AI-based financial analysis software that can calculate *ESG* scores. The company currently implements a pipeline which is based on a lexicon-based solution to provide sentiment analysis on *ESG*-related news articles. The goal of this thesis was to investigate how *ESG* sentiment classifications can be improved, by using modern deep machine learning architectures, such as *transformers*.

To produce a model adapted to an *ESG*-related corpus, several pre-trained *BERT*-based models were trained using transfer-learning. Hyperparameter tuning and fine-tuning were then performed on the best-performing models through iterations.

The results show that the best-performing model *DistilRoBERTa-finetuned-financial-news*, outperforms *VADER* and is comparable to other *BERT*-based models tuned for sentiment analysis. In conclusion, the model achieves a macro average F1-score of 80.4%.

Keywords: Sentiment Analysis, Machine Learning, ESG, BERT, Natural Language Processing

Sammanfattning

Investering i hållbara företag kan vara svårt eftersom det finns *miljömässiga, sociala och styrningsmässiga* risker (ESG). Genom att betygsätta företag med hjälp av *ESG* konceptet kan investerare göra val som kan resultera i bättre investeringar. *Sanctify* är ett företag som utvecklar AI-baserad mjukvara för analysering och räkna ihop ett *ESG*-rankning. Just nu använder företaget sig av en process som bygger på en lexikonbaserad lösning för att klassificera *ESG*-relaterade nyhetsartiklar. Målet med detta examensarbetet var att undersöka hur sentimentanalysen av *ESG*-artiklarna kan förbättras genom att använda moderna maskininlärningsarkitekturer som *transformers*.

För att ta fram en modell som är anpassad till en *ESG*-relaterad korpus tränades flera *BERT*-baserade modeller med hjälp av överföringslärning. Inställning av hyperparametrar och finjustering utfördes sedan på de bäst presterande modellerna genom iterationer.

Resultaten visar att den bäst presterande modellen *DistilRoBERTa-finetuned-financial-news* överträffar *VADER* och är jämförbar med andra *BERT*-baserade modeller som är anpassade för sentimentanalys. Sammanfattningsvis uppnår modellen en makrogenomsnittlig F1-poäng på 80,4%.

Nyckelord: Sentimentanalys, Maskininlärning, ESG, BERT, Språkteknologi

Acknowledgements

We would like to first and foremost thank Gustav Johnsson Henningsson and Henrik Ljunger for participating in our weekly meetings and providing valuable suggestions. We would extend this thanks to Patrik Elfborg for being incredibly reliable whenever we had issues with Linux and Git. Last but not least we would like to thank our supervisor Marcus Klang for providing us with lectures worth of knowledge about machine learning, and for his guiding feedback throughout this thesis.

Contents

Chapter 1

Introduction

This chapter will introduce the background of this thesis alongside purpose and goals. A set of research questions that formed the basis of this thesis will be presented. Our reasoning for undertaking this topic of research and implementation will be answered. Lastly, a section describing areas that were excluded from the thesis is included.

1.1 Background

ESG is a concept that is used to evaluate companies based on their long term environmental, social and governance risks, e.g. energy efficiency, worker safety, and board independence. Since most pollution comes from a minority of companies [Starr, 2016], those who are publicly traded may curb their pollution when investors leave from a deteriorating *ESG* score.

Sanctify is a fintech company based in Lund which focuses on the development of AI-based financial analysis software, with an emphasis on *ESG*. They also provide access to their data in the form of an API, with the target audience for their applications being mainly fund managers. A part of their pipeline is the processing of a large number of news articles to determine *ESG* scores of companies. One step in this processing is to measure the sentiment of the news articles through *sentiment analysis*.

In general, *sentiment analysis* means trying to determine whether a given text expresses itself as positive, neutral or negative. There exist different approaches to sentiment analysis such as *lexicographical* or different machine learning algorithms, e.g. *LSTM* [Hochreiter and Schmidhuber, 1997] or *Naive Bayes*, as well as a hybrid of the two approaches.

Sanctify's current solution is a lexicographical approach that is tuned with the addition of *ESG*-related terms. Modifications to the lexicon are needed to get an *ESG* perspective. For

example: "Company X has increased its greenhouse gas emissions". Without an *ESG* perspective, "greenhouse gas emission" would have no meaning and the text would in the best case be classified as neutral, and worst case positive. However, even with the modifications, the lexicon-based approach is not very generic and requires care when changing the lexicon.

This thesis will evaluate a machine learning approach and compare it to a lexicographical approach. Depending on the results, it would aid *Sanctify* in deciding which approach to further explore for improvements in their *ESG* sentiment analysis pipeline.

1.2 Purpose and Goals

The aim of this thesis is to explore, develop and test different machine learning models for sentiment analysis on an *ESG*-related dataset of news articles. In addition, it aims to develop a solution that can more accurately label news articles positive, neutral or negative from an *ESG* point-of-view.

1.3 Motivations

ESG is expanding as an crucial feature in many investors portfolios due to the current focus on climate and sustainability. If this thesis leads to an improvement in *Sanctify*'s products, investors will have more accurate information when investing in sustainable companies. It could also lead to an increase in interest for *ESG* by the financial market as a whole in the area of machine learning.

1.4 Research questions

This thesis will aim to answer the following questions:

- How is state-of-the-art sentimental analysis done currently?
- What tools can be used for a machine learning solution?
- How should different solutions be compared?
- How can a transformer model be optimized for text classification?
- What tools exist that can augment an *ESG*-based dataset for NLP?

1.5 Limitations

One limitation in this thesis is that the articles that will be used for testing will only be supplied by *Sanctify*. Another limitation is that implementing our prototypes in any *Sanctify* products is not in the scope of this thesis. This thesis is not an evaluation of different solutions for sentimental analysis other than from an *ESG* point-of-view. As this thesis is focused on machine learning solutions, only the lexicon-based solution used at *Sanctify* will be explored.

Instead of developing a new machine learning framework, existing frameworks will be used. The sentiment of the articles will not be divided into separate *ESG* categories and the articles will only be labelled as positive, neutral or negative but from an *ESG* point-of-view.

1.6 Division of Work

Since this thesis was done by two authors, the workload was divided equally to achieve a parallel workflow. Initially, both authors participated in the initial exploratory data analysis. Both authors developed the framework for the corpus and training of the models using *Flair* [Akbik et al., 2019]. Additionally, *Flair* was modified by Alexander Möhle to better accommodate the methods needed for training. Later, Oscar Johansson focused on the implementation of the pre-processing pipeline with augmentation and stratification, as well as a frontend for the framework. At the same time, Alexander Möhle focused on implementing a baseline with *VADER* and the *BERT*-based models, alongside their evaluation. Towards the end, Alexander Möhle did the practical work with the hyperparameter searching and fine-tuning, with assistance from Oscar Johansson. Throughout the thesis, both authors equally contributed to the writing of the report. Alexander Möhle was more responsible for the tables, while the graphs were delegated to Oscar Johansson.

1.7 Source Criticism

The following sections highlight an evaluation of the sources used in this thesis. They have been organized in order of credibility, with peer reviews and number of citations as metrics.

1.7.1 Peer Reviewed

These sources have been peer reviewed and published in a journal, making them highly credible.

[Bergstra et al., 2013]. [Connor et al., 2021], [Howard and Ruder, 2018], [Hutto and Gilbert, 2015], [Li et al., 2019], [Loshchilov and Hutter, 2017], [Malo et al., 2013], [Mehrabi et al., 2021], [O'Reilly and Chan [Mikolov et al., 2013], [Hochreiter and Schmidhuber, 1997], [Gautam and Yadav, 2014], [Hasan et al., 2018], [Shorten and Khoshgoftaar, 2019], [Pan and Yang, 2010]

1.7.2 Many citations

These sources have been highly cited, which gives them high credibility.

[Devlin et al., 2018], [Jianqiang and Xiaolin, 2017], [Kingma and Ba, 2014], [Liu et al., 2019], [Ruder, 2016], [Sanh et al., 2019], [Srivastava et al., 2014], [Vaswani et al., 2017]

1.7.3 Literature

The following literature were written by well credited professors or scientists in their respective field, making them highly credible.

[Buduma and Locascio, 2017], [Liddy, 2001], [Jurafsky and Martin, 2021], [Montgomery et al., 2021]

1.7.4 Frameworks

The following citations are for giving credit to the tools used during the thesis.

[Ma, 2019], [Akbik et al., 2019]

1.7.5 Fewer citations

These sources are less cited, which the reader should be aware of.

[Kumar et al., 2020], [Starr, 2016], [Yang et al., 2020], [Colón-Ruiz and Segura-Bedmar, 2020]

1.7.6 No citations

This source has no citations, which could make the information questionable. It should be cross-examined by other sources.

[Mehra et al., 2022]

Chapter 2

Technical Background

This chapter will introduce the various technologies used in this thesis and briefly go over them on a conceptual level. It will cover technologies and concepts like machine learning, model training, natural language processing, the transformer architecture, *Bidirectional Encoder Representations from Transformers* (BERT) and finally how to evaluate machine learning models. See **Section 7** for explanations for some abbreviations and terms.

2.1 Machine Learning

Machine learning is a subfield of *artificial intelligence* (AI). Machine learning models are trained to solve a task and can self-learn by finding patterns in data related to the task that the programmer wishes to use the model for [Buduma and Locascio, 2017]. In layman's terms, machine learning models can be seen as black boxes that, given certain inputs, return output predictions.

2.1.1 Linear Regression Example

To illustrate how machine learning models work, an example using *linear regression* will be described [Montgomery et al., 2021]. Linear regression in its simplest form can take one input variable (x) and predict one output value (y). To start linear regression, data is needed to map different data points in a two-dimensional space. See **Figure 2.1**.

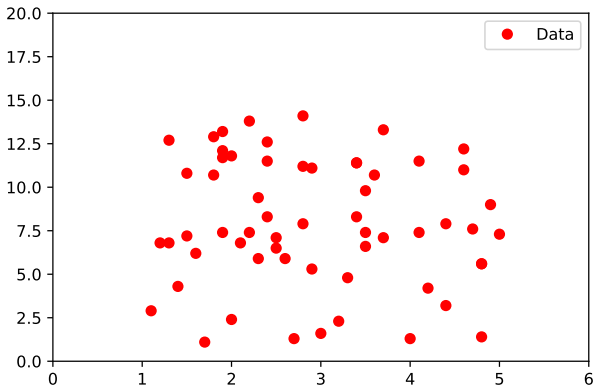

Figure 2.1: Two-dimensional graph with data points placed.

The data points are inserted by having a value on the *x-axis* and the *y-axis*. The *x-axis* value is the input to the model and the *y-axis* value is the expected output from the model. After the data points have been inserted, a straight line can be drawn in the space like seen in **Figure 2.2**.

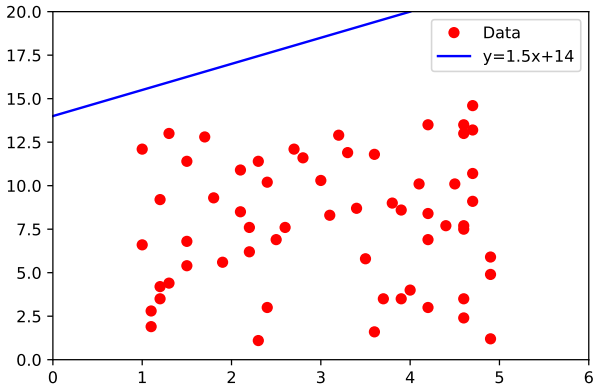

Figure 2.2: Two-dimensional graph with data points placed and a randomly initialized line.

The line equation is $y = ax + b$ which is the equation for a linear line, hence *linear regression*. At the start, the line is randomly initialized and therefore it will not always hit the data points in the way it is intended, as can be seen in **Figure 2.2**. To solve this, changes need to be made to the values of a and b so that the line will better fit and the y-values can be better predicted. This is done by calculating the mean sum of all errors. The mean sum of all errors is calculated by first measuring the vertical distance from one data point to the line and then squaring that distance. After this is done for all the data points, all the squared distances are summed together and divided by the number of data points. In machine learning terminology this is called *loss* and one formula for calculating it is called *mean squared error* (MSE) and can be seen in **Figure 2.1**. There also exists other formulas that can be used to calculate the loss.

The higher the loss, the less accurate the model will be at predicting what y-value corresponds to a given x-value. Should the loss be zero that would mean that the line passes through each data point and therefore can predict each y–value correctly for each x-value given.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{2.1}$$

A function called *loss function* can be created by calculating the different losses for each value of a and b, which in machine learning terms are called *weight* and *bias*. The loss function for a two-dimensional plot such as this example can be shown in a three-dimensional space. What values a and b should have for the loss to hit its minimum can sometimes be visualized in this space. However, for plots that are more than two dimensions, which most machine learning models are, the minimum values for a and b can not be visualized. To circumvent this problem, the loss and slope are calculated for a single point in the loss function and a step is taken in the direction where the slope is steepest. Note that since most machine learning models have more than two dimensions the slope is instead called *gradient descent* which will be cover in greater detail in **Section 2.2.7**.

The whole process can be seen as trying to find a way down a mountain in a blizzard, e.g. a person trying to find their way down, surveys the area to find the steepest point and then takes a step in that direction. After many steps, the bottom is eventually reached. To simulate a step in machine learning, the value of the steepest slope is multiplied by a value named *learning rate*. Learning rate in this scenario can be seen as the distance of the step. The whole process of minimizing the loss by updating a and b repeats until the minimum loss is reached. In machine learning terminology this process is called *fitting*, but can also be called *training*. After the fitting is done the line will *fit* the data points better as can be seen in **Figure 2.3**.

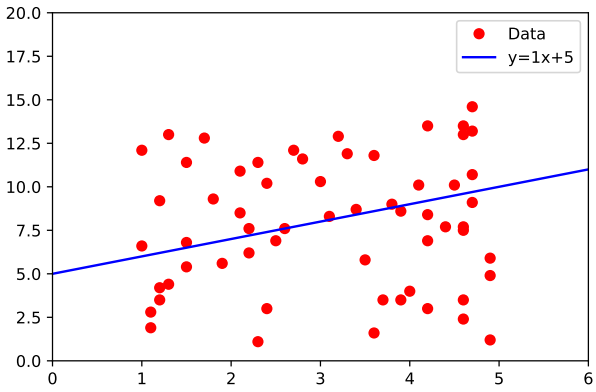

Figure 2.3: Two-dimensional graph with data points placed and a fitted line.

2.1.2 Neural Networks

The foundation of machine learning is the machine learning architecture called *artificial neural networks* (ANN), also referred to as *neural network*. Neural networks are constructed of *layers*, where each layer constraints multiple *nodes* that are *fully connected* to the nodes in front of them [Buduma and Locascio, 2017]. Each node has its own *weights*, *biases* and *activation function* that when all working together with the input, can produce an output that the model understands. The output from one node is then sent as input to all the other nodes in the next layer. The process is repeated until the last layer is reached and the output layer makes a prediction based on the input.

As mentioned, the first layer in a neural network is called input layer and the last is called output layer respectively. Between the input layer and output layer are *"hidden"* layers. The hidden layer has the ability to alter the output from the previous nodes into more complex inputs for the next layer. The effect of the hidden layers is what makes machine learning so powerful. An example of a neural network can be seen in **Figure 2.4**.

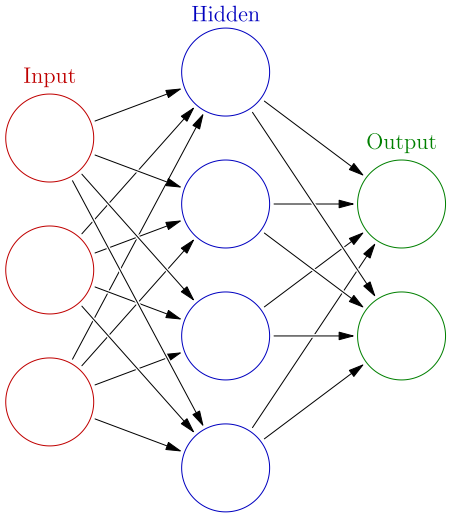

Figure 2.4: Overview of a neural network. *Glosser.ca, Colored Neural Network, CC BY-SA 3.0*

2.1.3 Recurrent Neural Network

Recurrent Neural Network (RNN) is a type of neural network that has an *internal memory* that remembers the information from previous calculations. It can remember this by feeding itself with its own outputs as inputs [Buduma and Locascio, 2017]. Another advantage of recurrent neural networks is that fewer hidden layers are required to compute the same output compared to standard neural networks. Also since there are fewer hidden layers there are fewer *weights* and *biases*, which makes the model less complex.

2.1.4 Backpropagation

The process of updating the weights and biases in machine learning is called *backpropagation*. After the gradient of the loss function has been calculated for the last output layer each weight in the preceding layer will be updated accordingly. This is then repeated iteratively until the input layer is reached and its weights updated, hence backpropagation, since we move “backwards” through the model from the output layer to the input layer.

2.1.5 AI vs Machine Learning

Being able to observe data and "learn" from it is what makes machine learning its own subfield within AI. AI *algorithms* do not have the capability to “learn" new data; instead, it makes decisions and predictions based on predefined rules set by the programmer.

2.2 Training Process

Training or *fitting* a machine learning model is done with a dataset that is split into three separate sets: *training*, *test*, and *validation*. The advantage of this approach is being able to evaluate the model on data that is not used in the learning process itself.

The training set can either be passed to the model as a *batch* or further split into *minibatches*. These are then fed to the model during training. The model predicts all the entries in a batch and then calculates the loss of these predictions. Each batch of samples from the set that is passed through the model is called an *iteration*. After an entire training set has been passed through the model one epoch has been completed. *Epoch* is a number that shows how many times the training set has been iterated during training.

After each epoch, the validation set is passed through the model and the loss of the set is calculated. Each epoch aims to *minimize the loss* of the set. The point of using the validation set is to evaluate if the model improved after the epoch. *A decrease in accuracy, or an increase in loss*, are good indicators that the model should stop training.

Otherwise, training stops after a set number of epochs or if a specified stopping criterion is triggered. After the training is stopped, the test set is passed through the model as the last evaluator. Since the entries in the test set have not been seen by the model during training, it is done to simulate the real-world use of the model.

2.2.1 Supervised Learning

There are different ways to train a model, one of which is called *supervised learning*. In supervised learning, the entire dataset has been *pre-labeled*. The labels are used during training but also when evaluating the model and are seen as the "correct" predictions. It can be said that the aim of the training is to predict *every label correctly*.

2.2.2 Dataset

A *dataset* in a machine learning context is a collection of data that is used when fitting a machine learning model. Datasets consist of different entries that relate to what the model is supposed to predict. For example, a model that predicts what type of clothing is seen in an image is trained using a dataset containing many images of different clothing items. In general, more data enables the training of more complex models, but additional data is also more expensive to process and label [Domingos, 2012]. Datasets can be called *corpora* when the data is composed of text.

Data Augmentation

When creating a dataset, the obstacles are two-fold. The first is to retrieve samples that will ensure that the dataset has enough *variance* in the data that the model is built upon. The second is the *manual* labelling of these samples, which is a time-consuming process. *Data augmentation* allows for increasing the size of the dataset by duplicating existing samples and then modifying them to create new samples [Shorten and Khoshgoftaar, 2019]. Data augmentation can be effective in preventing *overfitting* by forcing the model to adapt to the new patterns present in the augmented samples [Connor et al., 2021]. A common issue with datasets is the imbalance between labels. This can lead to the learning process being *biased* to the majority label when the weights are adjusted [Li et al., 2019]. A solution to this issue can be to increase the amount of minority labels through augmentation.

However, data augmentation in the *natural language processing* (NLP) domain is difficult because of the requirement of retaining the meaning of a text. For *sentiment analysis*, changing a sample to a sentiment that does not match the label could adversely affect the model. Data augmenting text samples can be done in many ways. Some of the common methods involve replacing a word with its synonym, inserting a random word in the sentence, swapping the position of two words, or randomly deleting a word. A powerful method of data augmentation is to use *contextualized word embeddings* to augment a given sample. This method uses a *transformer*-based model to find synonyms for words while preserving the context.

Stratification

As mentioned in **Section 2.2**, the dataset is split into three sets: training, validation, and test. This can be done by distributing the samples randomly. However, this risks introducing a *bias* to the sets, e.g. the training set could be biased in positive sentiment labels, thus affecting how the model is trained. With *stratified sampling*, the sets remain representative of the dataset by keeping the percentages of each label the same in each set. This is especially important if the dataset contains a large *disparity* between the minority and majority labels.

2.2.3 Amazon Elastic Compute Cloud

Instead of using local hardware to train a model, scalable cloud services can be used to substantially increase training speed and memory capacity. *Amazon Elastic Compute Cloud* (EC2) [Amazon, 2021] provides access to a virtual server where the training of a model can be performed.

2.2.4 Hyperparameter Searching

Hyperparameters controls how the *learning algorithm* behaves during training and affects the performance of the resulting model. Examples of hyperparameters are: *learning rate, optimization algorithm, anneal factor, and batch size*.

The goal of *hyperparameter searching* is to find the optimal values for the hyperparameters for a dataset that *maximizes the performance* of the model. While manual searching can be done for a small subset of hyperparameters, automated searching methods can be used instead to iterate over many combinations of hyperparameters.

Two common methods of hyperparameter searching are *random search* and *grid search*. Both use a grid with manually inputted values on hyperparameters for the selection process. Random search selects a random combination, while grid search selects a preset combination. After a method has been chosen, a model is trained on each combination, also called *hyperparameter tuning*. Lastly, the combination that performed the best is outputted. Another method is the *Tree-structured Parzen Estimator* (TPE) [Bergstra et al., 2013], which is based on *sequential model-based optimization* (SMBO). SMBO iterates between trained models and uses this information to estimate the performance of future configurations, which guides the hyperparameter selection process.

2.2.5 Transfer Learning and Fine-tuning

Training an NLP language model is time-consuming, requiring large amounts of data and powerful hardware. A less resource-intensive alternative is to use *transfer learning*, where instead of training a model from scratch, a *pre-trained* model is used [Pan and Yang, 2010]. These have already gained a deep understanding of language by being trained on a large corpus. They can then be adapted for specific NLP tasks, such as *sentiment analysis*, by only training a *classifier*.

Fine-tuning is a type of transfer learning where instead of only training a classifier, the weights of the pre-trained model are also adjusted. By tuning the weights, the model may be better adapted for the NLP task that the classifier was trained on. Fine-tuning is often done when the pre-trained model has been trained on a dataset that is much larger than the one used for fine-tuning. It is also common to *freeze* some layers when fine-tuning. By freezing a layer, the weights of that layer are not changed. Freezing can help when not wanting to adjust the weights in lower layers, which may affect the behaviour of the model negatively [Yang et al., 2020].

2.2.6 Overfitting and Underfitting

When a model is training, it will adjust its weights to minimize the *loss function*. While this is the normal process of improving the model, excess training may lead to *overfitting*. Overfitting occurs when a model trends towards memorizing all the samples in the training set. A consequence is that the model *generalizes worse* on future samples, such as the test set [Buduma and Locascio, 2017]. An example of this can be seen in **Figure 2.5**

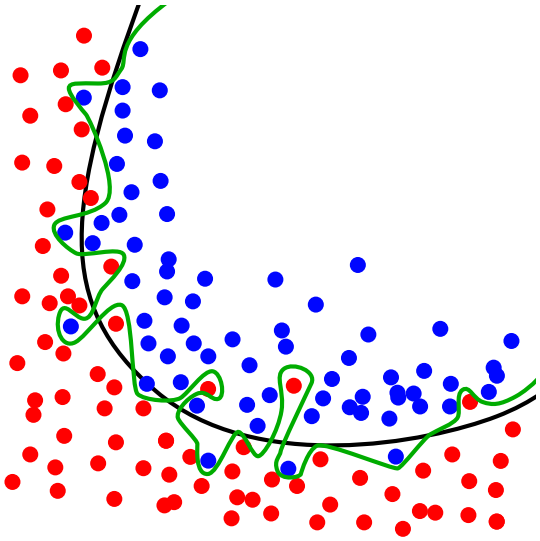

Figure 2.5: The green line represents an overfitted model that perfectly matches the data points. The black line on the other hand follows the trend, and is, therefore, more generalized. *Chabacano, Overfitting, CC BY-SA 4.0*

There are ways to combat overfitting. One approach is to stop training when the loss curve stops decreasing on the training set but increases on the validation set, also known as *early stopping*. Another approach is to use *data augmentation* as mentioned in **Section 2.2.2**. Other approaches include *regularization* and *dropout*, which are described in the following subsections.

The opposite of overfitting is *underfitting*, which occurs when the model has not learned enough patterns for it to become accurate and cannot follow the trend of the data points, see **Figure 2.6**. This occurs when there is *not enough training data* for the model or if training is ended *prematurely*.

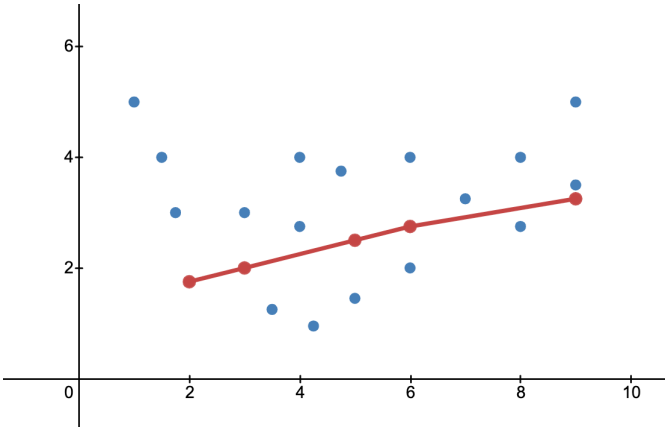

Figure 2.6: The red line represents an underfitted model that does not curve to more closely match the data points.
AAStein, Underfitting, CC BY-SA 4.0

Regularization

Regularization modifies the loss function by adding *penalties* to large weights. The goal is to prevent overfitting by converging to a prediction that is offset by the "true" prediction, which increases *generability*. A simplified equation for the loss function can be seen in **Equation 2.2**, where $Error(y, \hat{y})$ is the difference between the correct label y, and the predicted label $\hat{y}$.

$$Loss = Error(y, \hat{y}) \tag{2.2}$$

The penalty value λ is the hyperparameter that affects the strength of the regularization. Higher values have a bigger effect on reducing *overfitting*, but if raised too high it risks *underfitting* instead. The ideal value gives a model that generalizes well to unseen data. There are two types of regularization, *L1* and *L2* [Buduma and Locascio, 2017]. L1 assigns a penalty value based on the *absolute* values of the weights in the model. This can be seen in **Equation 2.3**, where $\sum_{i=1}^{N} |w_i|$ represents all the weights in the model.

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i| \tag{2.3}$$

Using L1 regularization can lead to some data points being assigned zero weight, effectively removing them from the model. An advantage of this approach is the elimination of values that are *insignificant*, which reduces the complexity of the model. A reduction of these values leads to a speedup in the calculation for the classifier, which can be beneficial depending on the task [O'Reilly and Chanmittakul, 2021].

L2, on the other hand, assigns a penalty value based on the *squared* value of the weights, which ensures that the weights will not reach zero, see **Equation 2.4**. A benefit of L2 is how it penalizes peaky weight vectors, e.g. [1,0,0,0] in favour of diffuse weight vectors, e.g [0.25, 0.25, 0.25, 0.25]. This encourages the model to use more of its inputs. The decision to use L1 relies mostly on if reduced *model complexity* is favoured, otherwise, L2 is used because of *superior performance* in most cases [Buduma and Locascio, 2017]. A visual example of the effect of using L2 regularization can be seen in **Figure 2.7**.

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{2.4}$$

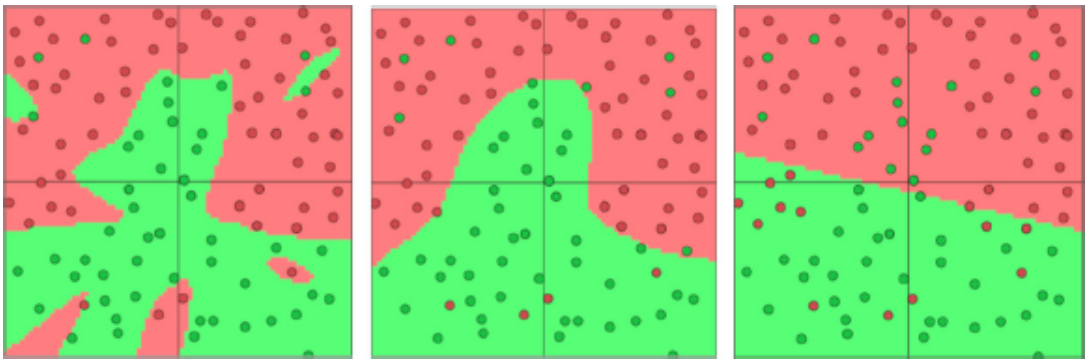

Figure 2.7: A visualization of neural networks trained with L2 regularization using strengths of 0.01, 0.1, and 1 (in that order), from [Buduma and Locascio, 2017].

Dropout

Dropout is used to prevent overfitting by reducing the number of active *nodes* in a model. This forces the model to make decisions even with missing information and reduces *dependencies* between combinations of nodes. It can be described as preventing overfitting by approximating the effect of training many different models in parallel [Buduma and Locascio, 2017]. A drawback to using dropout is the increase in training time due to the increase in noise that is inputted to the nodes [Srivastava et al., 2014]. Hence, a balancing act has to be made between speed and performance by tweaking the value of the *dropout hyperparameter*.

2.2.7 Learning Rate and Optimizers

The goal of the training process of a model is to minimize the *loss function*, which leads to a more accurate model, as seen in **Section 2.1.1**. The progression of the loss function is closely tied to the *learning rate*, which decides how far it will adjust at each iteration. When calibrating the learning rate hyperparameter, it will either go towards *converging* to the minimum point of the loss function or *overshoot* it. When the learning rate is too low, it requires frequent updates to converge, which costs time, see **Figure 2.8**. When it is too high it causes erratic updates which can diverge from the minimum point instead [Buduma and Locascio, 2017], see **Figure 2.9**. A balanced value for the learning rate enables quick convergence but requires trial and error to reach, see **Figure 2.10**.

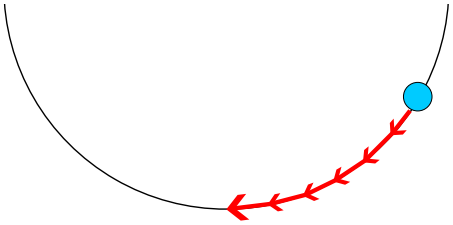

Figure 2.8: Learning rates that are too low cause frequent updates and slow convergence to the minimum point.

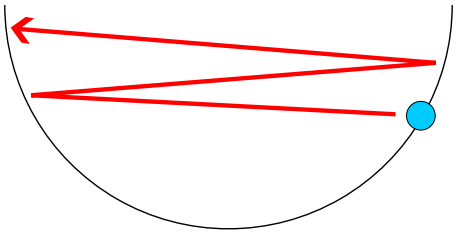

Figure 2.9: Learning rates that are too high cause erratic updates and divergence from the minimum point.

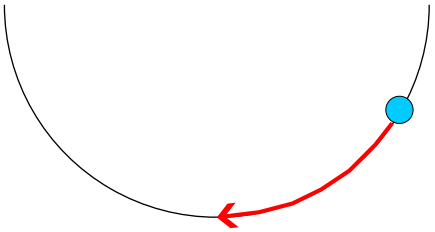

Figure 2.10: A balanced learning rate can quickly converge to the minimum point.

Optimization algorithms are responsible for adjusting the model's weights at each iteration to *minimize the loss function* [Buduma and Locascio, 2017]. As explained in **Section 2.1.1**, when wanting to minimize the loss, the gradient of the loss function is navigated downwards towards a (global) minimum. The gradient is calculated at each iteration by computing the loss and then reversing the resulting gradient to make it face downward.

The most widely used algorithm used for this purpose is *gradient descent*, see **Figure 2.11**. It is limited by the fact that the learning rate is unchangeable during runtime and that it risks becoming stuck in a local minimum point instead of the global minimum point. The two currently most popular optimization algorithms are *stochastic gradient descent* (SGD) and *Adam*, both of which will be described below.

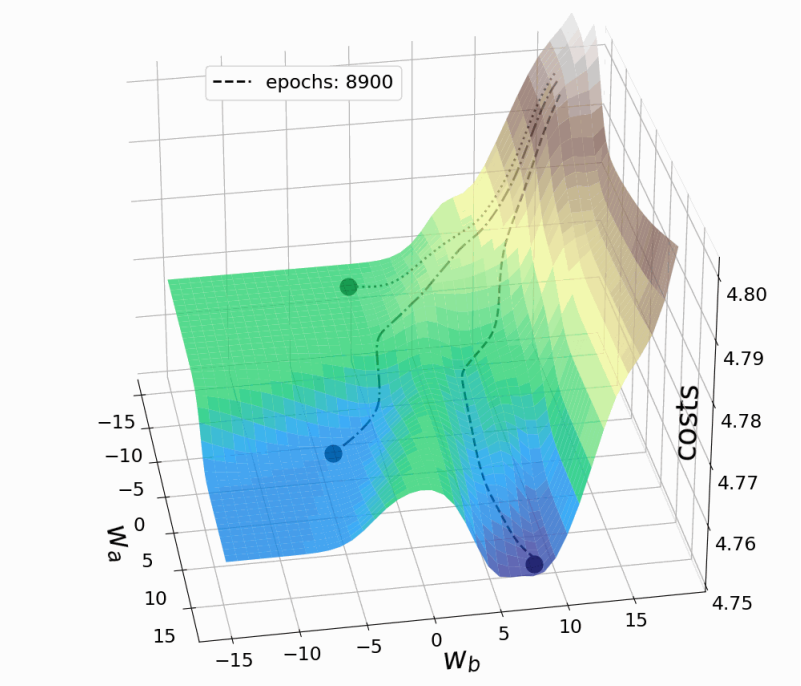

Figure 2.11: A visualization of gradient descent. The terrain represents the placement of data points, and the paths represent the loss function.

SGD

SGD is a variant of gradient descent. SGD calculates the gradient by performing an update for each training sample, or minibatch. These frequent updates cause *high fluctuations* as it tries to converge. A benefit of these fluctuations is that it allows SGD to escape local minimums, with the drawback of potentially overshooting the global minimum [Ruder, 2016].

An issue with how SGD fluctuates is that it causes our loss function to not follow an optimal path, which slows down our convergence. A technique that dampens the effect of this issue is called *momentum*. Momentum works by increasing the momentum term when the gradient points in the current direction, but decreases when the gradients change direction [Ruder, 2016]. Since this results in larger step sizes for the current direction, the addition of momentum may require lower learning rates [Buduma and Locascio, 2017]. See **Figure 2.12** for a visualized comparison.

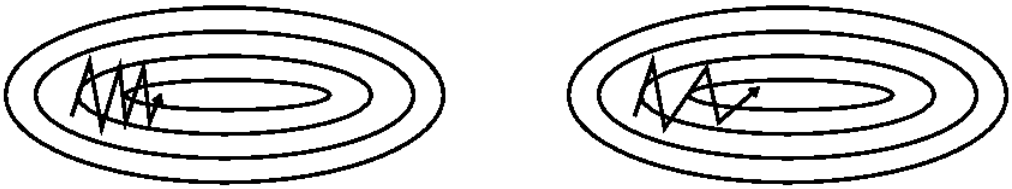

Figure 2.12: The figure on the left showcases SGD without momentum, making slow progress. The figure on the right shows SGD with momentum, converging faster to a minimum point. Source: Genevieve B. Orr.

Adam

Adaptive Moment Estimation (Adam) [Kingma and Ba, 2014] is an extension of gradient descent that allows for an *adaptive learning rate* instead of a static one, as in the case of SGD. It builds on the advantages of SGD's momentum in combination with previous adaptive gradient descent algorithms. By automatically decreasing the learning rate at each iteration, Adam can help the loss function quickly converge in the beginning and then slow down as it approaches the global minimum point.

A variant of Adam is *AdamW* [Loshchilov and Hutter, 2017], which improved the implementation of weight decay. Weight decay is the same as L2 regularization, but only when SGD is used. With AdamW, weight decay instead gets a different equation, which improved the generalization performance of Adam on image classification datasets [Loshchilov and Hutter, 2017].

2.2.8 Learning Rate Schedulers

Learning Rate Schedulers (LRS) are frameworks that adjust the learning rate during the training process of a model. While similar to adaptive optimization algorithms like Adam, they do not affect any weights in the model but instead only the learning rate itself. The main draw of using LRS is that they allow for the adjustment of the learning rate in combination with SGD. However, LRS can still be used with adaptive optimization algorithms. Examples of LRS are: *linear decay, exponential decay, step-based decay and reduce on plateau*.

The most popular LRS has been the reduce on plateau. It works by reducing the learning rate by a factor between 2-10 when a metric stops improving, most often loss. It checks for improvement at every epoch and uses a variable called *patience* to decide when to reduce the learning rate. Patience is simply the number of epochs with no improvement.

2.3 ESG

Environmental, social, and governance (ESG) are criteria that can be applied to companies to scout investment potential. By compiling data from a multitude of metrics relating to environmental, social and governing factors, these companies can be assigned a score relating to investment risks. There are multiple *ESG* standards published by different organizations, one of which is the *SASB* standards [Value Reporting Foundation, 2022]. *SASB* uses six categories to classify *ESG*: environment, social capital, human capital, business model & innovation, and leadership & governance. These categories are then further divided into subcategories. Furthermore, the *SASB* standards are industry-specific, meaning that they vary depending on the industry. Further details can be seen in **Appendix A.1**.

2.4 Natural Language Processing

Natural Language Processing (NLP) is a branch connected to linguistics, computer science and artificial intelligence. A principle is to aid computers in processing and understanding human language [Liddy, 2001]. Examples of NLP are translation of languages, recognizing and understanding speech, and answering questions.

2.4.1 Sentiment Analysis

Through the use of NLP, analytics that measures sentiment in texts can be produced. The most common sentiment takes the form of a polarity, e.g. positive, neutral, negative [Liu, 2020]. Sentiment analysis can be performed through either a *rule-based approach* or a *machine learning approach*. The two approaches can also be combined for a hybrid solution [Hasan et al., 2018].

2.4.2 Rule-based Sentiment Analysis

A rule-based approach uses a *pre-built lexicon* with assigned *polarity* to each word. By analyzing a corpus, a *sentiment score* can be assigned to each document, indicating the polarity of the text. However, most solutions using this approach only considers the binary polarity expressed in the text. By tuning the lexicon to also include the *intensity* of polarity, it can give a more accurate sentiment score. A solution that implements this method is *Valence Aware Dictionary and sEntiment Reasoner* (VADER). This is accomplished by incorporating five heuristics that affect the sentiment intensity [Hutto and Gilbert, 2015].

2.4.3 Machine Learning-based Sentiment Analysis

In contrast to defining the model based on a lexicon, a *machine learning model* can develop its own set of "rules" based on what is fed to the model during the training process. It can then provide predictions on the sentiment for unseen text. However, this approach requires pre-labelled data in the training phase [Gautam and Yadav, 2014]. This is a mostly manual process that is both time-consuming and subjective since texts can be interpreted differently depending on the person doing the labelling. There are different machine learning architectures that can perform sentiment analysis, such as *Long short-term memory* (LSTM) and *BERT*. *LSTM* is built on an RNN architecture while *BERT* is built on a *transformer* architecture.

2.4.4 Tokenization

Tokenization is the process of separating a text or sentence into tokens which can then be processed into word embeddings. For NLP this step is important since most state-of-the-art architectures process text at the *token level*. Tokenization can be performed with different techniques e.g white space or dictionary/rule-based tokenization. Below is an example of white space tokenization.

> Input: the fox jumps over the lazy brown dog.
> Output: (the), (fox), (jumps), (over), (the), (lazy), (brown), (dog)

BPE and *WordPiece* tokenization are other techniques to tokenize an sentence. They not only tokenize a sentence into words but can also split words into subwords, i.e. the root and the affix are separated. To not lose the context of the split words, the second subword resulting from the split starts of with two characters, *"@@"* for BPE and *"##"* for WordPiece, i.e. splitting the word "tokenizer" into "token" and "izer", would result in "izer" being represented as *"##izer"* if WordPiece is used.

2.4.5 Text Pre-processing

Pre-processing is used to prepare the data before it is used as input to the NLP model. This is accomplished through cleaning and manipulating the data to reduce the amount of noise in the corpus. While most NLP methods can handle a certain amount of noise, the reliability of the results can be highly affected if it exceeds a *threshold*. Noise present in the corpus can significantly impact the performance of *BERT* models during fine-tuning [Kumar et al., 2020]. There is a multitude of different approaches to pre-processing, some of which will be described below.

A common pre-processing step is the removal of *stop words*. These are words such as "the", "an", "so", "a". For most purposes regarding NLP, they do not have inherent meaning in a corpus and may be discarded. *Contextual machine learning models* such as *BERT* can by themselves learn that stop words are unimportant, thus removing them manually is unnecessary [Jianqiang and Xiaolin, 2017].

Other useful steps are the removal of punctuation, symbols, and numbers. In *sentiment analysis*, they carry no specific connotation, and similarly to stop words, they may be discarded. While these steps may not directly impact the resulting sentiment analysis [Jianqiang and Xiaolin, 2017], it is useful for decreasing the size of the corpus.

In a corpus, there will be occurrences of *inflected* words, e.g. playing, plays, and played. These words can be normalized to their common root form, in this case, play. This can be accomplished through two different methods, *stemming* and *lemmatization*.

Stemming reduces a word to its root form by removing the suffix, which reduces the inflexion. A drawback to this process is that the resulting root is not *guaranteed* to remain a valid word. Stemming can be performed through different stemmers such as *Porter* or *Lancaster*. Another way to stem a corpus is to use the *BPE* or *WordPiece* tokenizers explained in **Section 2.4.4**. Lemmatization, in contrast, will reduce inflection through the use of *vocabulary* and *morphological* analysis. This ensures that actual words are returned, in cases where this is important. The drawback to this approach is added complexity and cost of operation. To produce more accurate results with lemmatization, *part-of-speech* (POS) tags can be provided as parameters. This entails associating every word in the text with an appropriate tag e.g, verb, noun, adjective.

2.4.6 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is the process of calculating the relevancy of words in a given corpus. This is accomplished through two sub-processes: *term frequency* (TF) and *inverse document frequency* (IDF).

Term frequency is the prevalence of terms within a document. It can be defined in several ways, e.g *boolean* or *logarithmic* scaled *"frequencies"*. One way is to calculate the raw count of the terms, as seen in **Equation 2.5**.

$$\mathbf{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.5}$$

Document frequency is the number of documents that contain a given term. The inverse document frequency is calculated by the logarithm of the inverse document frequency, see **Equation 2.6**. The equation measures the importance of a term by giving less weight to frequent terms and the opposite for unique words. Combining the TF and IDF equations will result in the final equation of *TF-IDF*, see **Equation 2.7**.

$$\mathbf{IDF}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2.6}$$

$$\mathbf{TFIDF}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D) \tag{2.7}$$

2.4.7 Embeddings

The corpus is represented in a text format which is not suitable for a machine learning model. By converting the corpus to a numeric format, it can be used as input to the model. The standard method of accomplishing this is by representing each word in the corpus as a vector. This is one of the purposes of *embeddings*. The other purpose is to contain the *meaning* of the words in a sentence, and their *relationship* with each other.

Word Embeddings

Word Embeddings are composed of short, dense vectors that can capture *syntactic* and *semantic* connections between words. This means that words close in meaning will have similar vectors, e.g. "King - Man + Woman" results in a vector close in meaning to "Queen" [Mikolov et al., 2013]. They also perform better in NLP tasks compared to sparse vectors such as *Bag of Words* and *TF-IDF* [Jurafsky and Martin, 2021].

Word embeddings can either be *static* or *contextual*. With *static word embeddings*, a word will always have a single vector representation. The drawback of this approach is that words can have different meanings depending on the context, e.g. the bark of a dog, or the bark of a tree. Contextual word embeddings produce different vector representations depending on the sentence as a whole. These are produced by *transformer*-based models such as *BERT*.

2.5 Transformers

As mentioned before, there are different architectures for solving NLP problems. The *transformer* is one such architecture, and has been found to work well with NLP and also computer vision [Vaswani et al., 2017]. Since the transformer architecture is a complex structure, only the most important parts of the transformer architecture will be covered.

Encoder-Decoder

NLP models often take *word embeddings* as inputs for their predictions. However, there is only so much information available in a single word that has been converted to a word embedding. Sentences, however, vary in length and contain multiple word embeddings, which makes them difficult for the models to handle. A *fixed size vector* is much more convenient and easier for the models; instead of multiple word embeddings, only a single *sentence embedding* is wanted in some NLP tasks. A common way to create a sentence embedding is to calculate the mean value of all the word embeddings, also know as *mean pooling*. However, sentences that share the same words would get the same embeddings, while the structure of the sentence could be different in a way that affects the meaning of that sentence.

Man bites dog.
Dog bites man.

While the two sentences above have different meanings, they would still get the same embedding. This is where *encoders* and *decoders* come in. Encoders and decoders are both machine learning models that can be used separately or connected together to create one big *encoder-decoder* model.

Encoders are trained to take large *input vectors* and scale them down to smaller more compact vectors. Hopefully doing so without losing any meaningful information. Decoders do the opposite; they can take small *encoded vectors* and decompress them to generate new data. To be able to create new data from the encoded vector the decoders need to have a *"deep understanding"* of the encoded data.

It is the encoder's ability to keep meaningful information while scaling down an input vector that is favourable in the task of creating *sentence embeddings*. One way to create a sentence embedding with encoders is to have multiple encoders placed sequentially. To begin creating a sentence embedding, the first encoder is fed the first word of the sentence and a word embedding of that word is generated as output. The next word in the sentence is then fed to the next encoder together with the outputted word embedding from the first encoder. The pattern is repeated until the last word in the sentence has been encoded. The output from the last encoder can be seen as a sentence embedding.

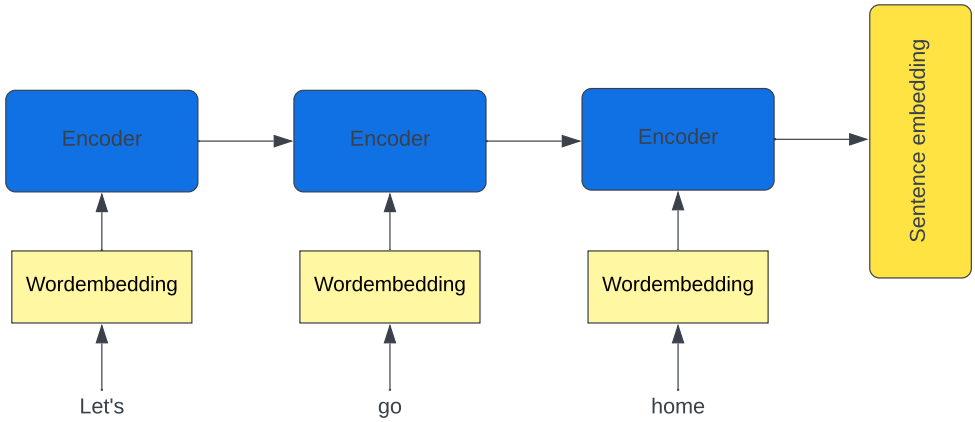

Figure 2.13: Example of how encoders can be used to create a sentence embedding.

Self-Attention

Self-attention [Vaswani et al., 2017] is also a valuable feature of the *transformer* architecture and is useful during *NLP* tasks. With the help of the self-attention mechanism and *matrix multiplication*, transformer models such as *BERT* can learn what words are more relevant to other words in a sentence. The self-attention mechanism functions on both the encoder and decoder and produces an *attention matrix* that can be used by both respectively. *Recurrent* layers used inside *encoders-decoders* have been replaced by *self-attention layers* in *BERT* [Vaswani et al., 2017]. Using self-attention layers has multiple benefits [Vaswani et al., 2017]. One benefit is reduced *complexity*. Another is that more of the computation can be *parallelized*, therefore minimizing the number of sequential operations required. A third benefit is the reduced *path length* between layers.

2.5.1 BERT

BERT [Devlin et al., 2018] is a transformer model that performs well on a variety of *NLP* tasks such as question answering and sentence continuation as demonstrated by being evaluated on the *SQuAD* v1.1, *SQuAD* v2.0 and *SWAG* benchmarks. By applying transfer learning, it can be adapted to other NLP tasks such as *text classification*. Because of this, there are many pre-trained models that can be used as described in **Section 2.2.5**. The architecture of *BERT* differs slightly from that of normal transformer models, see **Figure 2.14**.

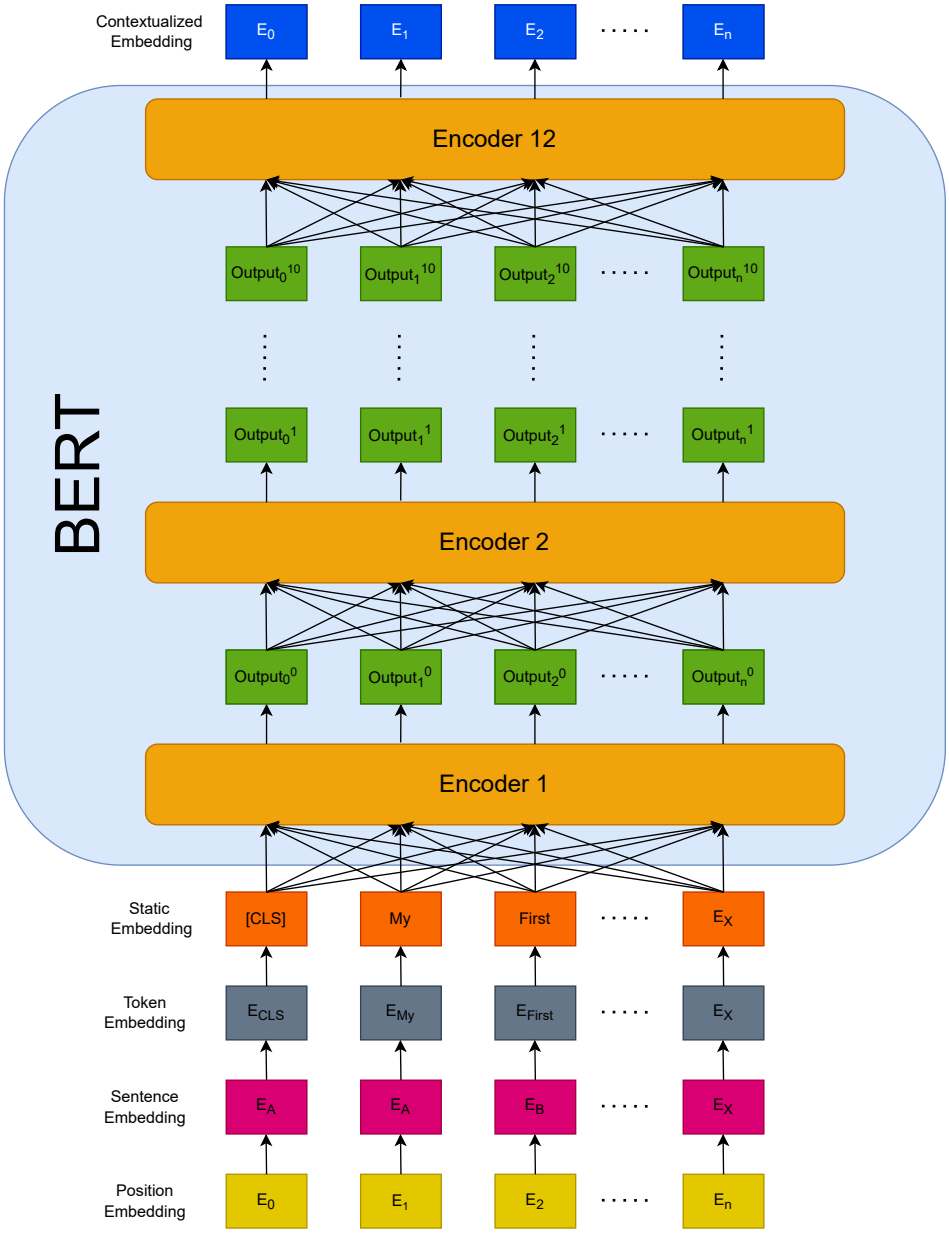

Figure 2.14: Overview of *BERT* architecture and its inputs.

As input, *BERT* takes *static word embeddings* that have been tokenized using a *WordPiece* tokenizer, see **Section 2.4.4**. Each static word embedding also contains a *position vector* that lets *BERT* understand a word's position in the sentence. These position vectors help with context, as it allows *BERT* to see which words are in close proximity to each other.

Each embedded word is then fed through 12 sequential *encoders* to create a *new* word embedding that has more context in them than the first static word embedding. Note that encoders in *BERT* are not used to scale sentences nor create sentence embeddings as per the example in **Section 2.5**. Instead, they are used to contextualize the word embeddings while keeping the word embeddings to a reasonable size. However, the problem with sentences being of different lengths still persists as described in **Section 2.5**. To counter this, *BERT* limits the number of tokens that can be inputted to 512. Should a sentence be longer than 512 tokens,

the rest is cut off. If instead, the sentence is too short, the rest is padded with embeddings that do not affect performance or the result. The limit also means that the amount of *contextual word embeddings BERT* can output is 512. The contextualized word embeddings from *BERT* can then be used for several NLP tasks.

2.5.2 BERT Pre-training

When *BERT* was *pre-trained*, *supervised learning* was not used. Instead, two other training methods were combined and used: *Masked Language Modelling* and *Next Sentence Prediction*. Worth noting is that none of these methods requires any human-labelled data, just a large corpus. The corpus *BERT* used for its training was the entire *English Wikipedia* library [Devlin et al., 2018].

Masked Language Modelling

When *BERT* was pre-trained with *Masked Language Modelling*, it was fed with complete sentences, where a percentage of words had been masked with a *[MASK]* token. *BERT* was then tasked to predict what words were removed by the mask. After the predictions were made, the predicted sentences were then compared to the original sentences. By doing this comparison with both sentences, no data needed to be manually labelled. As long as the sentences were grammatically correct, *BERT* would learn how to more accurately predict the masked words and gain a greater understanding of the English language. A sentence containing masked words can be seen in **Table 2.1**.

Next Sentence Prediction

The other method used during *BERT* pre-training was *Next Sentence Prediction*. The focus of this method is to let *BERT* score how well a sentence *"works"* and how grammatically correct it is. To train for this, *BERT* was fed with multiple sentences and tasked to predict if they followed each other or not. This method can be seen as a *supervised learning* method since there needs to be some sort of labelled data to tell *BERT* if it chose the correct sentence or not. The labelling for the dataset can, however, be automated.

To train *BERT* on this task, it needed some way of understanding that there was more than one sentence being presented to it. It also needed some way of understanding the context of the two sentences to be able to understand that they were related.

For *BERT* to understand that there were two different sentences, a *[SEP]* token is placed between the sentences. With the help of the *[SEP]* token, a *segment embedding* vector is added to each word embedding. Much like the *positional embedding* in each word, which tells *BERT* what position the word has in a sentence, the *segment embedding* tells *BERT* what sentence the word is in. The placing of the *[SEP]* token can be viewed in **Table 2.1**.

To understand the context of the two sentences, the *[CLS]* token was implemented. The *[CLS]* token is added at the start of the first sentence as seen in **Table 2.1** and then passed through all the encoders to get its own embedding. *BERT* then uses the *[CLS]* embedding to make its prediction of whether the second sentence connects with the first sentence. The *[CLS]* embedding was therefore trained to understand multiple sentence contexts and their relation to each other.

Table 2.1: Example from [Devlin et al., 2018] of how a sentence is constructed before it is forwarded to *BERT*. This allows *BERT* to train on both tasks in parallell. The *[CLS]* token allows the model to make a prediction of the *IsNext* boolean, indicating true or false.

Sentence	IsNext
[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]	True/False

2.5.3 Using BERT for Text Classification

As *BERT* was not trained specifically for text classification it might be difficult to understand what purpose *BERT* has for sentiment analysis. However, as mentioned in **Section 2.5.1**, *BERT* can perform well in a number of different NLP tasks. This is because *BERT* has a good understanding of language on a broad level. To use *BERT* for other NLP tasks; the inputs can be changed, additional output layers can be added, or both.

Since *BERT* is a large *machine learning network*, it has *weights* in each layer that can be adjusted. However, since the weights have been trained to understand such a broad concept as language they often do not need to be adjusted by much. Therefore, *BERT* can produce adequate results even without adjusting the weights inside *BERT*. However, for text classification, it is often beneficial to adjust the weights, since *BERT* is missing an output that can be seen as a sentence embedding. As covered in **Section 2.5**, sentence embeddings are good as inputs for models that perform text classification. In *BERT* the closest thing to a sentence embedding is the output from the *[CLS]* token since it was pre-trained on vaguely keeping the context of sentences in its embedding.

[Devlin et al., 2018] suggest that a decoder can be attached to the *[CLS]* output embedding to make *BERT* predict any sort of label. **Figure 2.15** illustrates what the architecture would look like following this proposal. The classifier for predicting the label can then be trained individually, either with or without adjusting the weights in *BERT*. If the internal weights of *BERT* are adjusted for the purpose of text classification it can be viewed as morphing the *[CLS]* embedding into a sentence embedding.

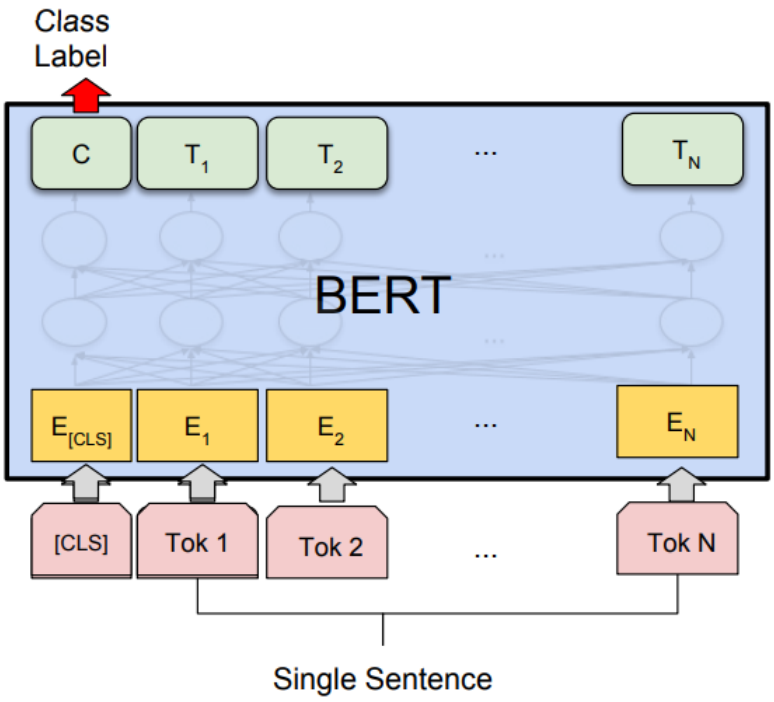

Figure 2.15: Suggested architecture for using *BERT* as a text classifier, from [Devlin et al., 2018].

2.5.4 Flair

Flair [Akbik et al., 2019] is an NLP framework that offers access to several pre-trained NLP transformer models to use for a different range of applications such as named entity recognition and text classification. *Flair* allows for such a different range by adding an untrained fully connected layer to the transformer model which becomes the classifier that is later trained. It also offers methods for simplifying the training and fine-tuning process, and an interface for using and combining different embeddings.

2.5.5 PyTorch

PyTorch [Paszke et al., 2019] is an open-source machine learning framework built to follow the design goals of the *Python* programming language. It is widely used in the computer science research community, providing APIs and methods for building complete machine learning models.

2.5.6 FinBERT

With a focus on the sentiment of financial text, *FinBERT* [Yang et al., 2020] is based on *BERT* and further fine-tuned with the Financial Phrasebank [Malo et al., 2013] which contains 4500 sentences with financial terms.

2.5.7 DistilBERT-base-uncased-finetuned-SST-2

Recent trends in the development of newer transformer models are increased amounts of parameters. While this often leads to better accuracy, they are also slower and harder to train. [Sanh et al., 2019] created *DistilBERT* to counteract this. DistilBERT is a smaller version of *BERT* but keeps almost all the understanding and performance [Sanh et al., 2019] of *BERT*. As with *FinBERT*, see **Section 2.5.6**, DistilBERT was further fine-tuned to perform sentiment analysis with a dataset that is smaller than the one used during DistilBERT's pre-training.

2.5.8 DistilRoBERTa-base

RoBERTa [Liu et al., 2019] is a *BERT* model that has had its training process slightly altered and is further pre-trained with a larger dataset than the original *BERT*[Devlin et al., 2018]. *RoBERTa* has been shown to be even more capable of understanding language than *BERT* however the additional training also makes the model larger which in turn can make the predictions slower. *DistilRoBERTa* fixes the slower prediction time by decreasing the size of the model using the same method as DistilBERT [Sanh et al., 2019].

2.5.9 DistilRoBERTa-finetuned-financial-news

Like *FinBERT* in **Section 2.5.6**, this model is a further fine-tuned of a pre-trained *BERT* model, however, the model that it is further fine-tuned on is *DistilRoBERTa-base* from **Section 2.5.8**. Also like *FinBERT* the dataset used for the fine-tuning is the Financial Pharsbank dataset [Malo et al., 2013].

2.6 Evaluating Machine Learning Models

There are different ways to evaluate machine learning models depending on what the user intends to use the models for. Some users value performance over accuracy and vice versa. The following subsections will describe ways to measure and evaluate machine learning models.

2.6.1 Accuracy

Accuracy is used as a base evaluation metric for a model. It is a measure of the amount of correct predictions out of all the predictions made, see **Equation 2.8**. A drawback to this method is that an imbalanced dataset can give a false certainty. It does not take into consideration false negatives and false positives, instead only focusing on the correct predictions, e.g. if 90% of people eat ice cream, a model would be evaluated as having 90% accuracy, even if it wrongly predicts that people who eat icecream does not, and vice versa.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.8}$$

2.6.2 Recall and Precision

Recall & precision are metrics for analyzing predictions from a model. They are based on the following terms of binary classification:

- True Positive (TP): Sentiment is correctly predicted as positive
- False Positive (FP): Sentiment is incorrectly predicted as positive
- False Negative (FN): Sentiment is incorrectly predicted as negative
- True Negative (TN): Sentiment is correctly predicted as negative

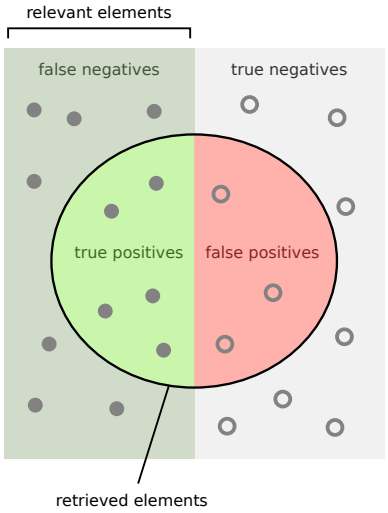

Figure 2.16: *Walber, Precision and recall, CC BY-SA 4.0*

Precision is the amount of correctly predicted positives out of all positive predictions that are made. This includes both true positives and false positives. In turn, *recall* is the amount of correctly predicted positives based on the actual amount of positive samples, see **Figure 2.16**. The equations for precision and recall can be seen in **Equation 2.9**.

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

For example, imagine throwing a fishing net into a lake, and catch 80 out of 100 total fish in that lake. That means the recall is 80%. However, if you also catch 80 rocks, that means the precision is 50%. Half of what was catched is junk.

To visualize these values, a *confusion matrix* can be used to compare predictions made by the model with the correct labels, see **Figure 2.17**. While binary classification with two labels uses a 2x2 matrix, the addition of the *"Neutral"* label requires a 3x3 matrix.

		Predicted	
		Positive	Negative
Actual	Positive	True positive	False positive
	Negative	False negative	True negative

Figure 2.17: Confusion matrix using binary sentiments, positive and negative.

2.6.3 F1-score

F1-score is a metric that can be used for assessing the performance of a model. It is calculated based on the results of both recall and precision. The F1-score can then be calculated by the equation seen in **Figure 2.10**. The score ranges between 0 and 1, with 1 meaning that every prediction is correct.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.10}$$

Important to note is that the F1-score is calculated separately for each class of labels, e.g positive, negative, neutral. To get a combined score across all classes, they can be averaged in three ways.

The first method is *micro averaging*, which gives a combined F1-score by counting the sum for each class. With micro averaging, only the total amount of each sample is considered. This means that it will favour the majority class since it contains the most amount of samples.

The second method is *weighted averaging*, which takes the mean of the F1-scores while including the support of each class. This method gives a larger weight to the majority class.

The third method is *macro averaging*, which on the other hand takes the arithmetic mean of the F1-scores. This means that each class is equally weighed. Given an imbalanced dataset, it will highlight the minority classes more.

2.7 Related Work

ESG in the area of sentiment analysis is largely unexplored. However, one such work was found, *ESGBERT*, which will be described below. Additionally, sentiment analysis for financial texts, in general, is closely related to *ESG*, which was explored by the authors of *FinBERT*.

2.7.1 Text classification for financial texts

[Yang et al., 2020] refers to [Howard and Ruder, 2018] when describing how further pre-training a language model on a target domain improves eventual classification performance. The authors further pre-trained *BERT* [Devlin et al., 2018] on the financial corpus TRC2-financial

which consists of 46143 documents. They then fine-tuned the model for sentiment analysis on the corpus Financial Phrasebank which consists of 4845 sentences. Their results showed a macro F1-score of 84% compared to 64% and 70% from *LSTM* and *LSTM* with *ELMo*, respectively.

2.7.2 Text classification for ESG

[Mehra et al., 2022] based their work on the results from previous research on domain-specific *BERT* models, one of which was *FinBERT*. They explored further pre-training *BERT* for their domain-specific task of *ESG* sentiment analysis. Masked language modelling was used on reports on financial performance (SEC Form 10-Q) from the Knowledge Hub of Accounting for Sustainability [Accounting for Sustainability, 2022] to update the weights of *BERT*. Their model was then fine-tuned on two classification tasks for sequence classification:

1. A change or no change in the environmental risk score of a company.
2. Positive or negative change in environmental risk scores of companies.

While *ESG* is broken down into environmental, social, and governance risks, [Mehra et al., 2022] focused solely on environmental risk scores in their research. Furthermore, the risk scores were based on how exposed a company are to the risks as well as how well they are managing them.

Their results showed that *ESGBERT* achieved a 67.09% F1-score on the test set compared to 59.85% by *BERT* on task 1. *ESGBERT* additionally achieved a 79.3% F1-score on the test set compared to 43.17% by *BERT* on task 2.

Chapter 3

Method and Analysis

This chapter covers how different solutions were implemented and how different libraries/frameworks were chosen and used. It contains information about the dataset, how different models were trained and what was done to optimise them.

3.1 Phases

The work in this thesis was divided into 5 phases. Initially, an *Exploratory Data Analysis* was performed on the dataset, see **Section 3.3**, which was followed by implementing a preprocessing pipeline, see **Section 3.5**. Next, baseline sentiment analysis was established using *Zero-rule classification*, see **Section 3.6**. Then, *VADER* was implemented as a *lexicographic* solution, see **Section 3.7**. Lastly, the *transformer* approach was developed which included training, hyperparameter tuning and fine-tuning, see **Section 3.8**.

3.2 Development

The main development method used to test and produce results in this thesis was prototyping. Different architectures were developed and tested in order to understand how they worked and how they performed compared to each other. Initially, commonly used methods were utilised and then, as development went on, more modern and in-depth methods were tested. For prototyping, a more agile version of the *Cross-industry standard process for data mining* (CRISP-DM) [IBM, 2021], see **Figure 3.1** methodology was used. It was more agile in the sense that instead of testing a single model at a time and following the pattern of *CRISP-DM*, much of the work was done in parallel. Many models were tested at the same time and evaluated against each other to narrow down the prototypes. Therefore, the *CRISP-DM* cycle

was followed more on some models than others as some would get outperformed by others before the cycle could be repeated.

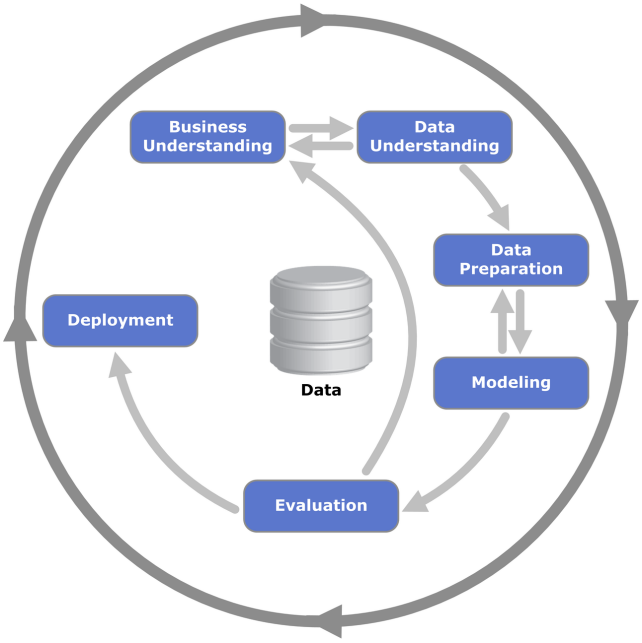

Figure 3.1: Diagram of the *CRISP-DM* method. *Kenneth Jensen, CRISP-DM Process Diagram, CC BY-SA 3.0*

To facilitate a pipeline of pre-processing and model training, an application was programmed using *Python* 3.8.13. The *Flair* framework v0.11 was integrated and setup for training, hyper-parameter tuning and fine-tuning. While *Flair* comes pre-packaged with the *PyTorch* framework, it does not come with *CUDA* included. *PyTorch* v1.11.0 was therefore reinstalled with *CUDA* v11.3.

An *EC2* instance was configured to enable faster training for the models and access to more memory for bigger batch sizes. The instance had the following hardware:

- **CPU:** AMD EPYC 7R32
- **GPU:** NVIDIA A10G
- **VRAM:** 22 GB
- **SYSTEM RAM:** 16GB

The following pre-trained transformer models were downloaded from the *Hugging Face* library that were current as of 31 May 2022:

- DistilBERT-base-uncased-finetuned-SST-2
- FinBERT
- DistilRoBERTa-base
- DistilRoBERTa-finetuned-financial-news

3.3 Data Exploration

The preliminary step before structuring a model is to first do an exploratory data analysis (EDA) on the given dataset. This process gives insight into how the data is structured, and if any corrections need to be made to ensure the model is trained correctly. These corrections can be in the form of pre-processing the corpus or augmenting the dataset by increasing the underrepresented labels with more examples. The following subsections will present the EDA on the dataset used in this thesis.

3.3.1 Dataset

The dataset provided by *Sanctify* contained a total of 1000 articles, which included the titles, summaries and texts for all the articles. The articles were classified by Sanctify as ESG-related by using the *SASB* standards. It was decided to focus on using the text samples for the sentiment analysis to maximize accuracy, resulting in a dataset of 1000 samples. The dataset was labelled using five labels:

- **Positive:** Positive sentiment.
- **Neutral:** Neutral sentiment.
- **Negative:** Negative sentiment.
- **Non-ESG:** Text that does not contain *ESG*-related terms
- **Trash:** Text that is faulty, or missing

After removing duplicates and samples labelled "trash", the resulting dataset was reduced to 851 samples. The "Non-ESG" label was then merged with the "Neutral" label to simplify the training process of the machine learning model. To get an understanding of which words influences the different sentiments, *TF-IDF* was used to get a statistical analysis. To reduce the scope of the analysis, it was used to produce the top 10 words for each sentiment, see **Table 3.1**. As can be seen, they share some common words such as "company" with each other. The negative sentiment contains more company names such as "facebook" or "apple".

Term	Weight
company	6.561626
energy	4.962647
statements	4.252362
business	3.545751
com	3.543007
forward	3.435693
year	3.412987
said	3.407047
looking	3.336220
new	3.285559

Term	Weight
company	17.697416
stock	15.845728
earnings	15.130869
debt	13.058095
market	12.743678
year	12.309421
dividend	10.793883
zacks	10.011297
growth	9.907130
shares	9.494976

Term	Weight
said	5.973998
company	5.315499
debt	4.059721
facebook	3.598327
apple	2.726536
year	2.668169
data	2.648217
reuters	2.625789
market	2.340214
stock	2.301205

Table 3.1: Top 10 common words for each sentiment. Left: Positive sentiment, Middle: Neutral sentiment, Right: Negative sentiment

3.3.2 Label Distribution

Figure 3.2 shows the label distribution of the original dataset and after cleaning. As can be seen, the distribution of the cleaned dataset is imbalanced in favour of the "Neutral" label.

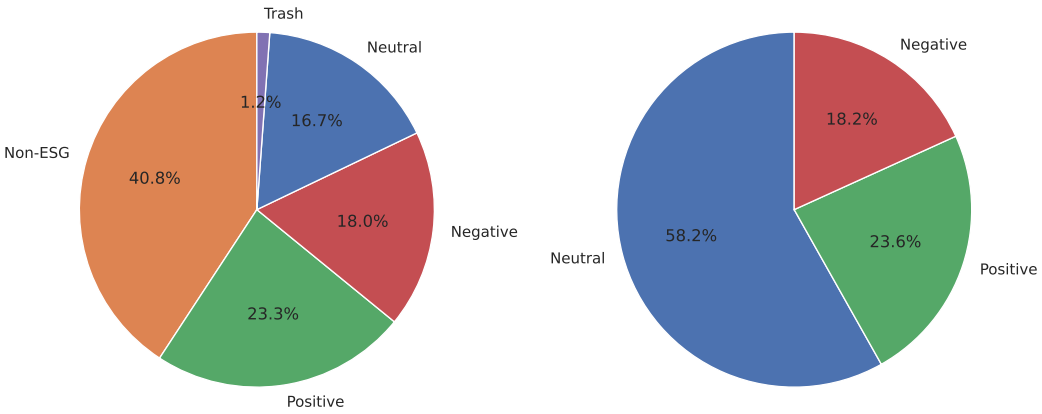

Figure 3.2: The left figure shows the distribution of the original dataset. The right figure shows the dataset after cleaning.

3.4 Pipeline

This section shows shows a general overview of the developed pipeline for sentiment analysis in this thesis. See **Figure 3.3**. Texts from the dataset are preprocessed and then the model makes its sentiment prediction on each.

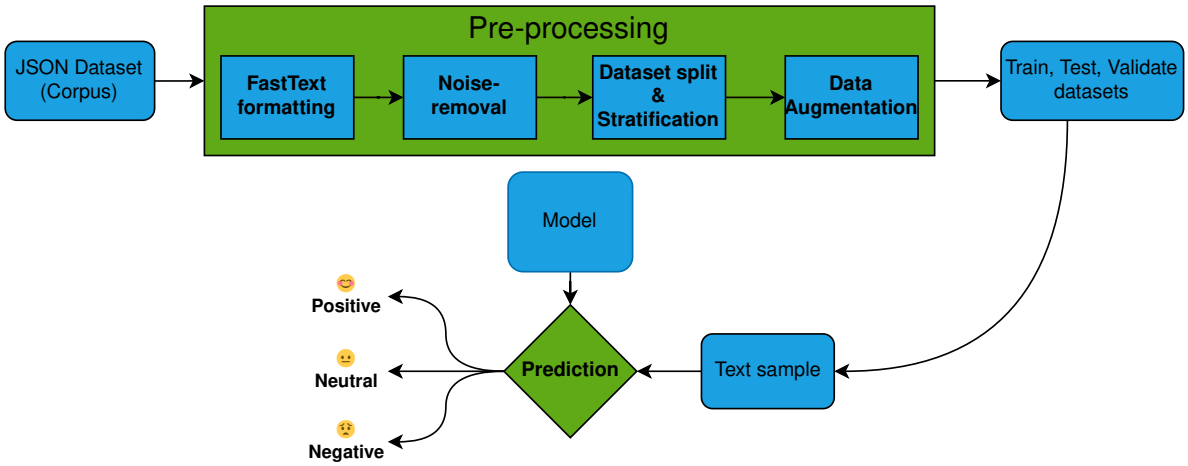

Figure 3.3: Overview of the sentiment analysis pipeline.

3.4.1 Overview

In total, this thesis developed five different pipeline variations. One for zero-rule approach, one for the lexicon approach and finally 3 for the transformer approaches. The first transformer approach used default parameters, the second used optimized hyperparameters by running hyperparameter searches, and the final used fine-tuning.

3.5 Pre-processing

Before the dataset could be used to train the initial models, it needed to be pre-processed into the correct corpus. Since the focus of the thesis was to only analyze the sentiment of the text of the articles, samples containing titles and summaries were filtered out. The next step was to process the actual texts. To minimize noise, non-alphabetic letters were removed, as well as parts that were only used for formatting. The dataset was provided in a *JavaScript Object Notation* (JSON) format, which was not compatible with our sentiment analysis approaches. The dataset was instead converted to the *FastText* format, in which each line contains an article's text and corresponding label, see the following example.

__label__Positive "articletext"

Next, the corpus was split into three sets: train, test and val. They were split according to a 60%, 20% and 20% ratio, which resulted in the train, test and val sets containing 451, 200, 200 samples respectively. By using stratification, see **Section 2.2.2**, these could be ensured to remain representative of their parent dataset.

As can be seen in **Figure 3.2**, the "Neutral" label outweighs the other labels by a factor of more than double. To counteract this imbalance, augmentation was used to increase the amount of samples for the minority labels. *NLPAug* [Ma, 2019] is a library that provides various methods for augmentation. Through *NLPAug* v1.1.10, a *BERT* model was used to augment enough samples for the minority labels that would result in equal amounts of samples for all three labels. The augmentation was only applied to the training set, and resulted in a sample size of 786.

3.6 Zero-rule Classification

The *Zero-rule* classifier was used as a comparator. Since the "Neutral" label was the majority label in the dataset, it was used to do a majority *Zero-rule* prediction on the test set, i.e. only classify texts as neutral.

3.7 Lexicon

The lexicon baseline in this thesis was made to be as close as possible to the already existing solution that *Sanctify* uses. It was, however, not a copy of their entire solution. Only the *Python* library *VADER*, which handles the classifying of the articles, was implemented. The *VADER* library comes as an out of the box sentiment analysis library that outputs a compound score ranging from -1 to 1, where a score of 1 is the maximum positive score and -1 is the most negative score a text can get. To be able to label articles as neutral, thresholds need to be set for the different labels. The thresholds can be manually tuned if needed but for this thesis, the threshold values were set to the recommended values as suggested on the *VADER GitHub* webpage. The thresholds can be seen in **Table 3.2**. The entire test set was then passed through *VADER* to get the predicted labels for each article in the set.

Table 3.2: Threshold for different labels as suggested on the *VADER GitHub* webpage.

Label	Threshold
Positive	>= 0.05
Neutral	> -0.05 and < 0.05
Negative	<= -0.05

3.7.1 Lexicon with Word List

To optimise *VADER* to the *ESG* use case, a word list was implemented. *Sanctify* aided with implementing the word list by suggesting some common *ESG*-terms to be added to the word list. Since the words in the word list are in standard form only, the corpus had to be stemmed to be able to use the word list. The test set was therefore stemmed and passed through *VADER* again.

3.8 Transformers

The transformer approach as described in **Section 2.5**, is the main machine learning solution explored in this thesis. Different transformer models were tested against each other to find which one performed the best on the test set. All of the models tested are under the *BERT* [Devlin et al., 2018] family tree and therefore have all the benefits of *BERT*, see **Section 2.5.1**. Two of the models are improved versions of *BERT* that have been distilled to a smaller size, see **Section 2.5.8**. One is also further fine-tuned on specific datasets, see **Section2.5.9**. The two other models are both *BERT* models. One is a distilled and fine-tuned version of *BERT*, see

Section 2.5.7 and the other is a standard *BERT* model fine-tuned to a dataset, see **Section 2.5.6**. The four models were chosen from the *Hugging Face* library based on how many downloads they had and what corpus they had previously been fine-tuned on. The *DistilRoBERTa-base* model was picked as a sort of benchmark for the other models. See **Table 3.3** for the name of every model chosen and what differs them.

Table 3.3: Different models tested in this thesis.

Name	Base model	Distilled	Fine-tuned
FinBERT	BERT	No	Yes
DistilBERT-base-uncased-finetuned-SST-2	BERT	Yes	Yes
DistilRoBERTa-base	RoBERTa	Yes	No
DistilRoBERTa-finetuned-financial-news	RoBERTa	Yes	Yes

To narrow down the amount of models to proceed with, all models were trained on *Flair*'s default hyperparameters, see **Table 3.4**, and with fine-tuning set to off. Fine-tuning was turned off to get a better understanding on how well the base models performed since fine-tuning only changes the weights inside a *BERT* model slightly. Having a good performing base model would likely increase the chances for a better result with fine-tuning turned on.

Table 3.4: Hyperparameters and their default values in *Flair*. Note that there are more hyperparameters available in flair but these are the ones that were used during the hyperparameter search in **Section 3.8.1**

Hyperparameter	Value
Anneal factor	0.5
Pooling	[CLS]
Dropout	0.1
Learning rate	0.1
Mini batch size	32
Optimizer	SGD
Patience	3
Scheduler	AnnealOnPlateau
Weight decay	0.01

Each model was trained with a maximum of 55 epochs ten times each, five times using the *[CLS]* pooling as input to the classifier and five additional times using a mean pooling of the contextualised word embeddings as input. The different inputs were tested because as mentioned in **Section 2.5.3**, text classification is suggested to be performed using the *[CLS]* token as input. However, this only holds true if the model is fine-tuned in order to adjust the *[CLS]* token embedding. The mean pooling of the word embeddings was therefore tested to see if the models would perform better with it since the models were not being fine-tuned. The idea was also to take the best performing model regardless of the input pooling method and then further train it with fine-tuning turned on to achieve the highest possible average macro average F1-score.

3.8.1 Optimizing

In hopes of getting the most out of the models that were chosen from the default hyperparameter testing, all the chosen models went through a hyperparameter search, see **Section 2.2.4**. For this purpose, the library Hyperopt v0.2.7 [Bergstra et al., 2013] was used.

The hyperparameter search was done before turning fine-tuning on and with the input pooling method that performed the best on the base models. The different hyperparameters and their ranges or values that were tested during the search can be viewed in **Table3.5**.

Table 3.5: Hyperparameters and their values or ranges that were used during the hyperparameter search.

Hyperparameter	Value/Range
Anneal factor	0.45 – 0.80
Pooling	Mean
Dropout	0.1 – 0.5
Learning rate	0.001 – 0.1
Mini batch size	8, 16, 32
Optimizer	AdamW
Patience	3
Scheduler	AnnealOnPlateau
Weight decay	0.001 - 0.01

Note that some hyperparameters only have one value, this is to limit the amount of search runs that have to be made in order to save time. For the sake of time, parallelism of two search runs was also implemented in *Flair*. This effectively halved the time it would take to complete an entire hyperparameter search. Some hyperparameters were initially not available to add to the search space and had to be manually added. The hyperparameters that had to be added can be seen in **Table 3.6**.

Table 3.6: Hyperparameters that had to be manually added to the search space.

Hyperparameter
Pooling
Dropout
Scheduler

After the hyperparameter search was concluded the suggested hyperparameters for each respective model were used to train them five times each to establish a confidence interval. The two models that had the highest average macro average F1-scores were chosen to be further fine-tuned.

3.8.2 Fine-tuning

The two models that had the best average macro average F1-score after hyperparameter tuning was then chosen to be further fine-tuned. The models that resulted from the training were

loaded in and fine-tuning was turned on. The input pooling method to the classifier which previously had been mean was also changed to *[CLS]* because of reasons covered in **Section 2.5.3**. For the sake of curiosity mean pooling was also used and therefore the two models that were chosen were fine-tuned ten times, five times with *[CLS]* pooling and five times with mean pooling. The other hyperparameters used for fine-tuning were *Flair*s [Akbik et al., 2019] default hyperparameters for fine-tuning and can be viewed in **Table 3.7**.

After the fine-tuning was complete, the models that had the highest macro average F1-score were evaluated by passing through the test set again. By using the resulting vector with all the predictions along with the vector of all the true labels, a confusion matrix, see **Figure 2.17**, was made for the best version of both models. The confusion matrix enabled an easier visualization of which labels that were classified incorrectly.

Table 3.7: Hyperparameters and their default values in *Flair* when fine-tuning.

Hyperparameter	Value
Anneal factor	0.5
Dropout	0.1
Learning rate	0.00005
Max epochs	10
Mini batch size	4
Optimizer	AdamW
Patience	3
Scheduler	LinearSchedulerWithWarmup
Warmup fraction	0.1
Weight decay	0.01

3.9 Evaluation

After each model had finished training the test set was passed through the model to get its predictions on the classifications. The predictions were saved as a vector that was later used when computing the performance of each model. The test set had a support of 36 negative, 117 neutral and 47 positive labeled texts from articles for a total of 200 texts. None of the texts in the test set were augmented, all were original texts that went through pre-processing to remove noise.

Since most of the texts in the test set had the neutral label followed by the positive and lastly negative label, it meant that the test set was imbalanced. The most fair method to compare the different models would therefore be the macro average F1-score. As mentioned in **Section 2.6.3**, the F1-score is a sort of summarization of both recall and precision into a single score. The averaged macro average F1-score of the five training runs during each phase was used when deciding what models to proceed with and further develop.

Chapter 4

Results

This chapter covers the various results achieved when training and testing different solutions and models. First, the main result is presented which is then followed by a deeper look at the result from the different phases mentioned in **Section 3** such as baseline, hyperparameter search and fine-tuning. The results were evaluated based on the test set, of which the support was 200 samples. The main result, which is the best macro average F1-scores achieved by each approach in this thesis can be seen in **Table 4.1**, where the bold result is the absolute highest average F1-score achieved. Worth noting is that all the results presented from the *transformer* approach in **Table 4.1** were achieved with *Flair*'s default hyperparameters.

Table 4.1: The highest macro average F1-score achieved by the different approaches.

Name	Type	Macro avg F1-score
Zero-rule classification	Baseline	0.246
VADER with Word list	Lexicon-based	0.309
VADER	Lexicon-based	0.312
DistilBERT-base-uncased-finetuned-SST-2	Transformer	0.727
DistilRoBERTa-base	Transformer	0.780
FinBERT	Transformer	0.802
DistilRoBERTa-finetuned-financial-news	Transformer	**0.804**

Figure 4.1 shows an confusion matrix from the model that had the highest macro average F1-score as can be seen in **Table 4.1**.

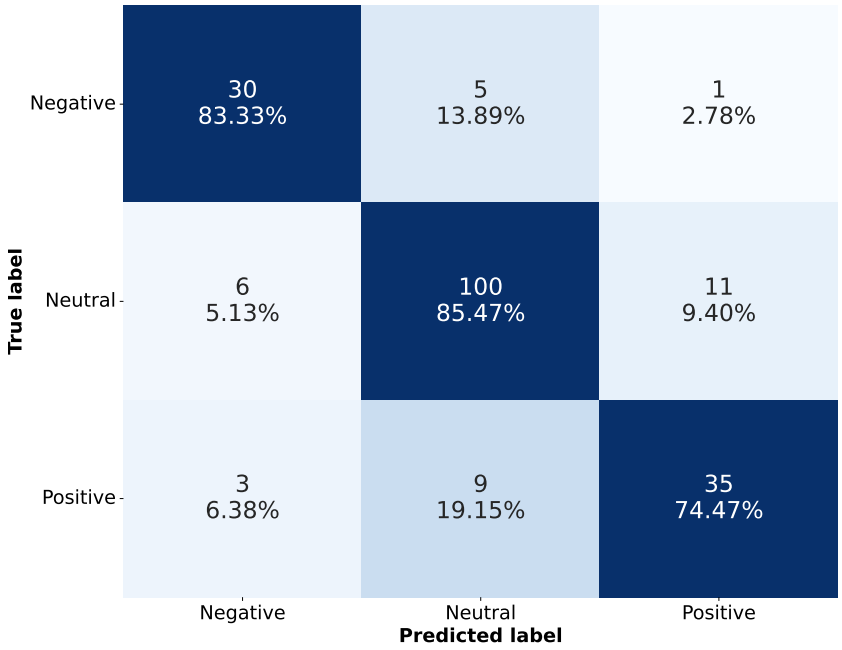

Figure 4.1: Confusion matrix from *DistilRoBERTa-finetuned-financial-news* that achieved the highest macro average F1-score of 0.804. The percentages sum up to 100% on the horizontal axis for each row. The dark diagonal cells highlight the amount of correct predictions.

4.1 Zero-rule Classification

Table 4.2 shows the classification report from the *Zero-rule* classification that was explained in **Section 3.6**. As can be seen, by the recall for the neutral label is 1 and 0 for the other two labels, it only classifies texts as neutral. Note that since the macro average emphasises all three labels equally in the test set, the macro average recall is 0.333, which means that it correctly classified a third of all the labels in the set.

Table 4.2: *Zero-rule* classification report.

	Precision	Recall	F1-score	Support
Negative	0.000	0.000	0.000	36
Neutral	0.585	1.000	0.738	117
Positive	0.000	0.000	0.000	47
Macro avg	0.195	0.333	**0.246**	200

4.2 Lexicon

The lexicon approach was made with *VADER* as covered in **Section 3.7**. **Table 4.3** shows a classification report that was generated after the test set had been iterated through. The

classification report shows precision and recall for the different labels as well as the support for each label. It also shows the overall accuracy of the solution as well as the macro average F1-score. Note that *VADER* is not a machine learning model but since it can also generate classifications it can be evaluated with F1-score. The low recall score for the neutral label is worth noting since it means that *VADER* has a hard time classifying something as neutral. On the other hand, the precision for neutral is high at 0.750 meaning when it classifies it is often correct.

Table 4.3: *VADER* classification report.

	Precision	Recall	F1-score	Support
Negative	0.536	0.417	**0.469**	36
Neutral	**0.750**	0.026	0.050	**117**
Positive	0.268	**0.957**	0.419	47
Macro avg	0.518	0.467	**0.312**	200

4.2.1 Lexicon with Word List

Table 4.4 shows the resulting macro average F1-scores, recall and precision just like in **Table 4.3**, however, the results are now from classifications from *VADER* with an added word list. *VADER* with the word list has even lower neutral recall than *VADER* without the word list.

Table 4.4: *VADER* with word list classification report.

	Precision	Recall	F1-score	Support
Negative	0.471	0.444	**0.457**	36
Neutral	**1.000**	0.017	0.034	**117**
Positive	0.280	**0.979**	0.436	47
Macro avg	0.584	0.480	**0.309**	200

Table 4.5 shows comparison between *VADER*, *VADER* with a word list and *Zero-rule* classification. The precision and recall in **Table 4.5** are the macro averages for all the labels in the test set. The values that are highlighted in bold are the highest number in that respective column. Worth noting is that the accuracy for *Zero-rule* classification is only that high because accuracy does not take into account classifications that were wrong. The precision is a much more accurate measure of how well the approach could classify texts but still, the macro average F1-score is the best indicator of how good an approach is and here *VADER* is the best.

Table 4.5: Comparison between results of *VADER* and *VADER* with word list and *Zero-rule* classification.

Model	Accuracy	F1-score	Precision	Recall
VADER	0.315	**0.312**	0.518	0.467
VADER + word list	0.320	0.309	**0.584**	**0.480**
Zero-rule classification	**0.585**	0.246	0.195	0.333

4.3 Transformers - Default

The **Table 4.6** shows the four first models that were chosen and what their average accuracy and average macro average F1-score resulted in after being trained with *Flair*'s default hyperparameters. The table is split into a section where the models were trained with *[CLS]* input pooling and another where they were trained on mean input pooling. The bold results are the best result of average macro average F1-score and accuracy witch each pooling method. In all tables and figures a confidence value of 95% was used to calculate the margin of error. Worth noting is that all the models except *DistilRoBERTa-base*, performed better with mean pooling than CLS. **Figure 4.2** is a visual representation of the average macro average F1-scores presented in **Table 4.6** for a better understanding of the difference in scores.

Table 4.6: Average macro average F1-score and average accuracy of different transformer models that were trained with *Flair*'s default, see **Table 3.4** hyperparameters, once with *[CLS]* pooling and once with mean pooling.

	CLS		MEAN	
Model	**Accuracy**	**F1-score**	**Accuracy**	**F1-score**
FinBERT	0.759 ± 0.017	0.723 ± 0.014	0.801 ± 0.008	0.767 ± 0.012
DistilBERT-base-uncased-finetuned-SST-2	0.718 ± 0.004	0.678 ± 0.018	0.746 ± 0.009	0.712 ± 0.013
DistilRoBERTa-base	0.773 ± 0.012	**0.747 ± 0.016**	0.766 ± 0.020	0.744 ± 0.016
DistilRoBERTa-finetuned-financial-news	**0.774 ± 0.015**	0.731 ± 0.004	**0.818 ± 0.010**	**0.789 ± 0.011**

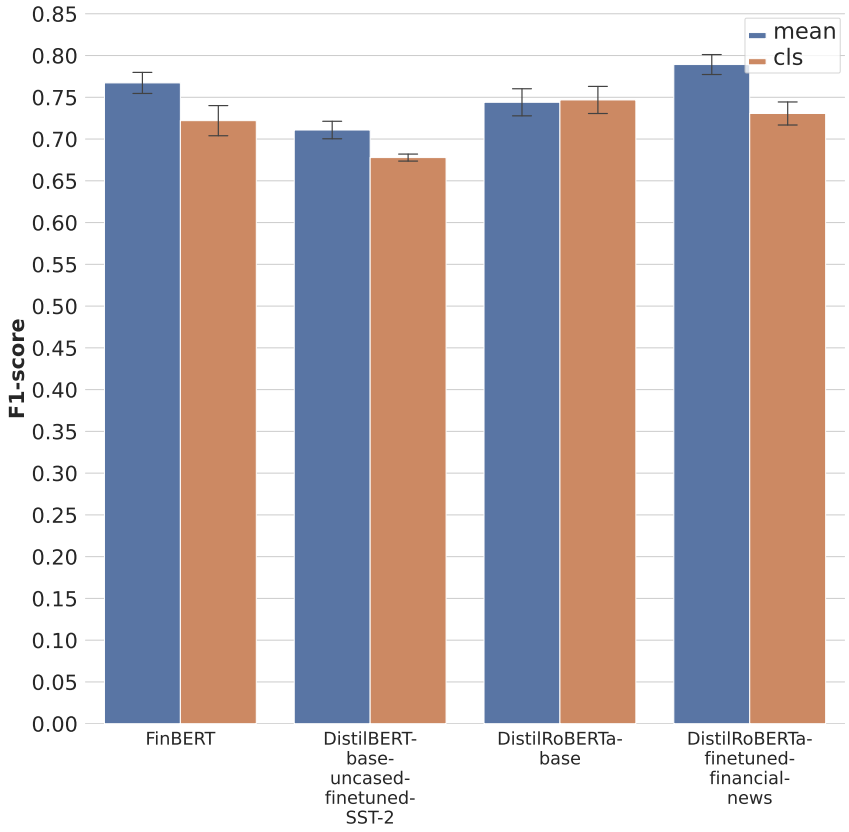

Figure 4.2: Comparison of the average macro average F1-scores between the models trained with *Flair*'s default hyperparameters and with different pooling methods.

4.3.1 Transformers - Optimized

Tables 4.7, **4.8**, and **4.9** shows the suggested hyperparameters from the hyperparameter search.

Table 4.7: The suggested hyperparameters for *FinBERT* after the hyperparameter search.

Hyperparameter	Value
Anneal factor	0.7
Pooling	Mean
Dropout	0.4
Learning rate	0.00107
Mini batch size	32
Optimizer	AdamW
Patience	3
Scheduler	AnnealOnPlateau
Weight decay	0.00255

Table 4.8: The suggested hyperparameters for *DistilRoBERTa-base* after the hyperparameter search.

Hyperparameter	Value
Anneal factor	0.5
Pooling	Mean
Dropout	0.4
Learning rate	0.01670
Mini batch size	32
Optimizer	AdamW
Patience	3
Scheduler	AnnealOnPlateau
Weight decay	0.00255

Table 4.9: The suggested hyperparameters for *DistilRoBERTa-finetuned-financial-news* after the hyperparameter search.

Hyperparameter	Value
Anneal factor	0.55
Pooling	Mean
Dropout	0.1
Learning rate	0.00524
Mini batch size	32
Optimizer	AdamW
Patience	3
Scheduler	AnnealOnPlateau
Weight decay	0.00648

Table 4.10 shows the results from the models trained with their respective hyperparameters from the hyperparameter search. The numbers in bold are the highest number in their respective column. All models were trained five times using mean pooling. All tables and figures used a confidence value of 95% was to calculate the margin of error. **Figure 4.3** shows a comparison of average macro average F1-scores between these models. *DistilRoBERTa-finetuned-financial-news* again outperforms the other models although the average macro average F1-score is lower than what it was for the same model with default hyperparameters, as can be seen in **Table 4.6**.

Table 4.10: Average accuracy and average macro average F1-scores of the models trained with the hyperparameters that resulted from the hyperparameter search.

Model	Accuracy	F1-score
FinBERT	0.788 ± 0.006	0.761 ± 0.008
DistilRoBERTa-base	0.787 ± 0.018	0.744 ± 0.011
DistilRoBERTa-finetuned-financial-news	**0.811 ± 0.013**	**0.781 ± 0.012**

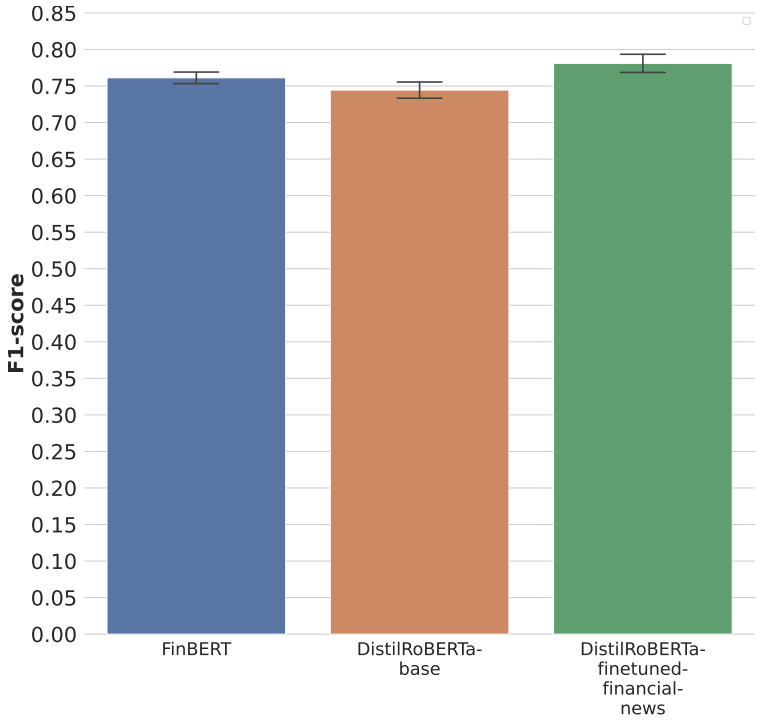

Figure 4.3: Comparison of the average macro average F1-scores between the models with tuned hyperparameters.

4.3.2 Transformers - Fine-tuned

Tables 4.11 and 4.12 show the results after fine-tuning *FinBERT* and *DistilRoBERTa-finetuned-financial-news*. The values highlighted in bold shows the highest value in that column. **Figure 4.4** shows a graphical representation of these tables. **Figures 4.5** and **4.6** show the confusion matrices for the two best fine-tuned models. **Tables 4.11**, **4.12**, and **Figure 4.4** used a confidence value of 95% was to calculate the margin of error. With *[CLS]* pooling, *FinBERT* is superior, however with mean pooling the results are mixed but ultimately *FinBERT* outperforms *DistilRoBERTa-finetuned-financial-news* in terms average macro average F1-score.

Table 4.11: Comparison of average accuracy, macro F1-scores, macro recall, and macro precision between the fine-tuned models *FinBERT* and *DistilRoBERTa-finetuned-financial-news*, with *[CLS]* pooling.

[CLS] Pooling				
Model	**Accuracy**	**F1-score**	**Precision**	**Recall**
FinBERT	**0.795 ± 0.016**	**0.763 ± 0.023**	**0.757 ± 0.020**	**0.781 ± 0.032**
DistilRoBERTa-finetuned-financial-news	0.793 ± 0.019	0.744 ± 0.024	0.753 ± 0.027	0.744 ± 0.025

Table 4.12: Comparison of average accuracy, macro F1-scores, macro recall, and macro precision between the fine-tuned models *FinBERT* and *DistilRoBERTa-finetuned-financial-news*, with mean pooling.

Mean Pooling				
Model	**Accuracy**	**F1-score**	**Precision**	**Recall**
FinBERT	0.794 ± 0.015	**0.759 ± 0.022**	0.755 ± 0.021	**0.768 ± 0.019**
DistilRoBERTa-finetuned-financial-news	**0.795 ± 0.013**	0.757 ± 0.016	**0.769 ± 0.021**	0.757 ± 0.015

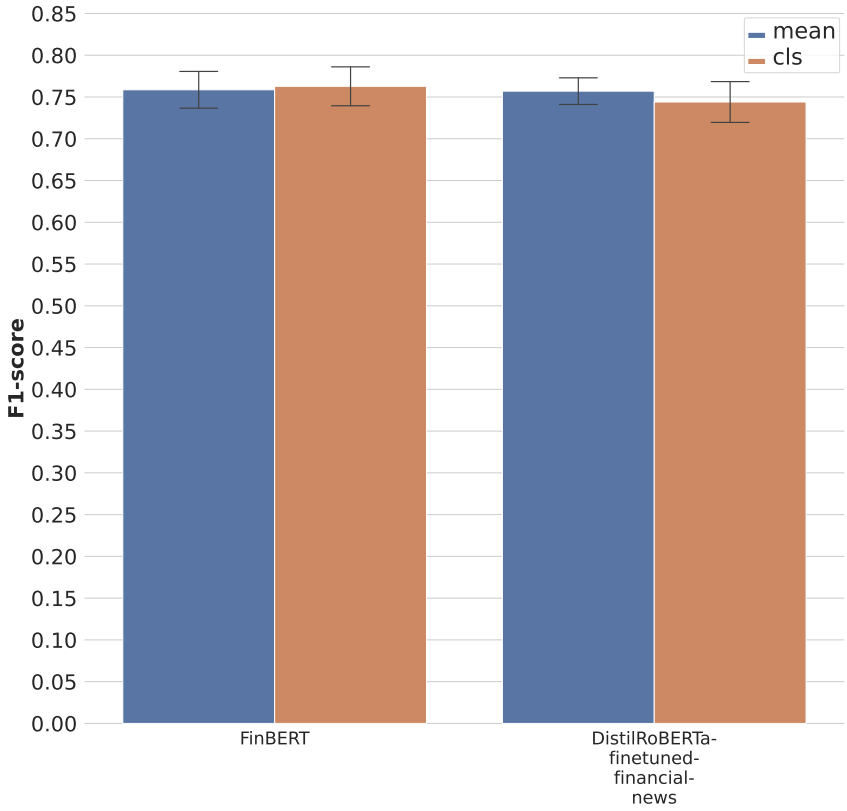

Figure 4.4: Comparison between the average macro average F1-scores of the finetuned models *FinBERT* and *DistilRoBERTa-finetuned-financial-news* with different pooling methods.

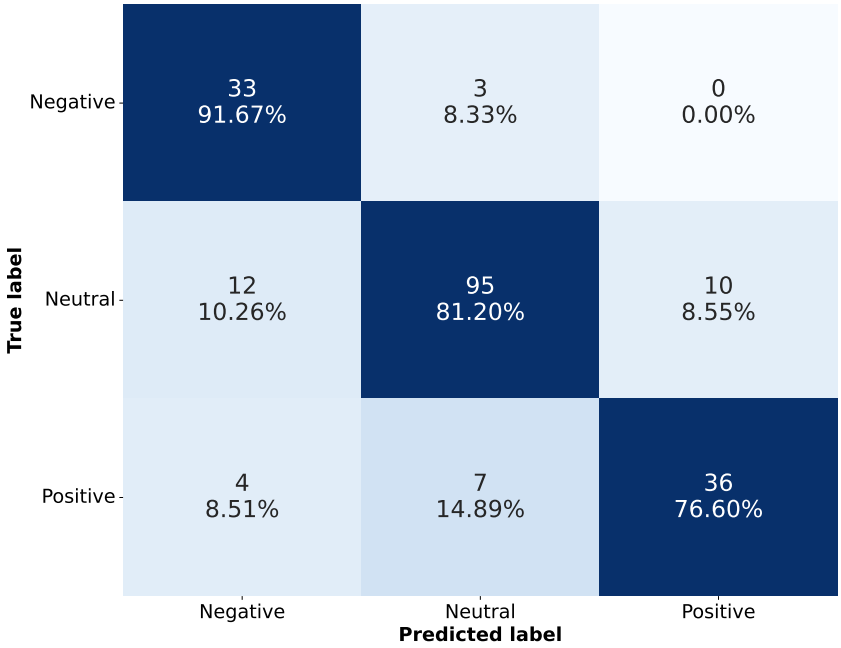

Figure 4.5: Confusion matrix for the best performing fine-tuned *FinBERT* model. The percentages sum up to 100% on the horizontal axis for each row. The dark diagonal cells highlight the amount of correct predictions.

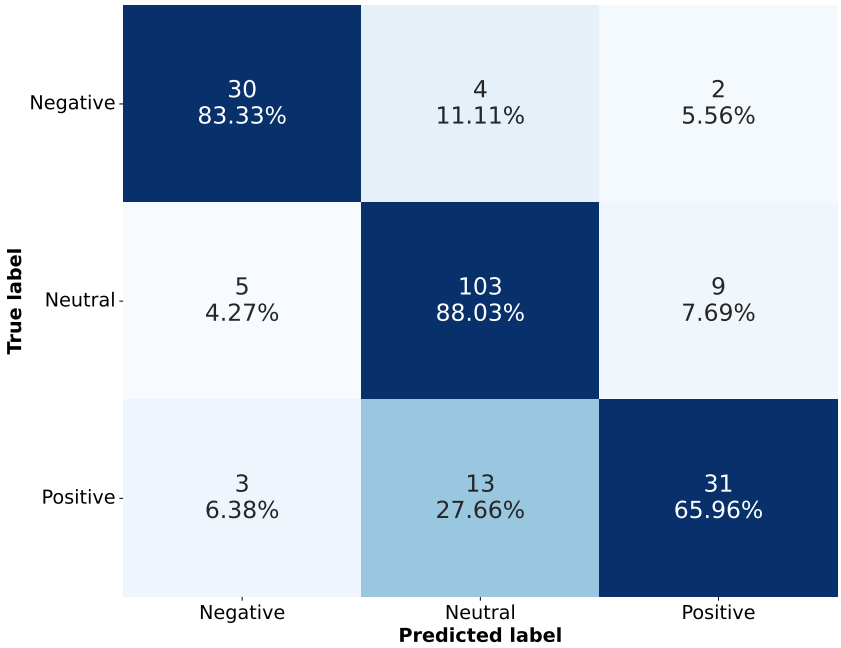

Figure 4.6: Confusion matrix for the best performing fine-tuned *DistilRoBERTa-finetuned-financial-news* model. The percentages sum up to 100% on the horizontal axis for each row. The dark diagonal cells highlight the amount of correct predictions.

Chapter 5

Discussion

This chapter will discuss the various findings and results from **Section 4**. It will discuss what the results mean, and delve into some findings that stand out or may be of interest and hypotheses as to why they affect the results.

5.1 Comparison of Different Approaches

As can be seen in **Table 4.1** the transformer solution outperforms the lexicon solution. Even the worst transformer model, *DistilBERT-base-uncased-finetuned-SST-2*, outperformed *VADER* in terms of macro average F1-score. Overall the performance of the lexicon approach was poor in comparison to the machine learning approach. With that said, *VADER* is still better than only guessing a single label, as the *Zero-rule* classification was the worst in terms of macro average F1-score.

5.1.1 Where VADER Fails

The biggest reason for *VADER*'s low macro average F1-score can be attributed to its inability to classify a text as neutral. As seen in both **Table 4.3** and **4.4** the recall for neutral is low. While the macro average F1-score equalizes the evaluation of the labels in the test set, the low amount of recall of the neutral label will still bring the macro average F1-score down significantly. The precision for the neutral labels is however high, indicating that when *VADER* classifies something as neutral it is most often correct. Taking both these things into consideration it can be concluded that the threshold for classifying a text as neutral may be too narrow.

The word list that was implemented into *VADER* in **Section 3.7.1** did not help *VADER* achieve

any higher scores, instead, it had the opposite effect and lowered the macro average F1-score. As can be seen in**Table 4.4** the accuracy goes up, meaning it is more often classified correctly however the recall of neutral labels is even worse.

VADER seems to be biased towards the positive label since the recall in both **Table 4.3** and **4.4** is high, almost being 1, but the precision is the lowest out of any label precision. The positive bias coupled together with the low recall on the neutral label could mean that neutral text often gets classified as positive texts.

5.2 Observations Using Transformers

All models tested in the transformer approach performed somewhat equally with the exception of the *DistilBERT-base-uncased-finetuned-SST-2* model. As covered in **Section 3.8**, the models that had the highest average macro average F1-score were selected to be further improved with hyperparameter searching and tuning. However, as can be seen in **Table 4.10**, the models that were trained with the found hyperparameters did not improve any of the models. Similar can be said about fine-tuning. Neither fine-tuning with *[CLS]* pooling nor mean pooling improved any model over using mean pooling with default hyperparameters. The following subsections will be covering potential reasons.

5.2.1 Hyperparameter Tuning

When training the models with the hyperparameters found in the hyperparameter search the hope was to see an improvement in the macro average F1-score. As covered in **Section 5.2** this was not the case, instead, the hyperparameters barely affected the macro F1-score. There are a number of reasons why this might be.

The most plausible explanation is the dataset. Whether the amount of text samples were too few or contained too little variation in the examples is hard to say. What can be said is that even with the found hyperparameters the loss of the validation set did not go any further down than what it did when training the models with the default hyperparameters. The found hyperparameters did however make the validation loss go down faster. The models, therefore, needed fewer epochs to reach the same loss as with the default hyperparameters.

Because of time limitations, the hyperparameter search was only run once for each model. This may not have given the models enough trials to zero in on the optimal values for the hyperparameters. Additionally, the search space might have been too large for the number of trials that were run. It might be beneficial to narrow down the search space and run the hyperparameter search more than once.

5.2.2 Fine-tuning

Since *BERT* is a model primarily trained for transfer learning, the use of fine-tuning is recommended to have a model perform well in the programmers' specific use case, however in this thesis fine-tuning did not amount to any improvement over the models that were trained

with the default hyperparameters as can be seen in **Figure 4.4**. A possible reason for this is that the fine-tuned models were only trained with *Flair*'s default fine-tuning parameters.

5.2.3 CLS Vs MEAN

In **Table 4.6** it can be seen that using *[CLS]* as an input pooling produces worse results than using mean pooling, which was recurring in this thesis. *[CLS]* pooling only beats mean pooling once and can be viewed in **Figure 4.4**. The most obvious reason for this is as mentioned in **Section 2.2.5**, *[CLS]* pooling really only becomes a valid input to the classifier after the model has been fine-tuned. However even with the one exception where *[CLS]* beat mean pooling all the top three models used mean pooling for their classifier input.

Again the possibility that the dataset is too small may play a role in this or it may be because of the hyperparameters used when fine-tuning. It can be said however that since mean pooling performs just as well or even better than *[CLS]* pooling, mean pooling might be preferred if the programmer does not intend to fine-tune the model to a specific use case.

5.2.4 Confusion Matrices

As can be seen in **Figure 4.5**, **Figure 4.6**, and **Figure 4.1**the different models perform differently on the test set. If the numbers in the top right and bottom left corner are high, it would mean that the model is unstable and often classify polar opposite of the actual labels which is not wanted.

As can be seen in **Figure 4.5** the top right and bottom left are almost empty. Instead, the most incorrect classifications are to the middle left and middle right of the neutral label, followed by the bottom middle and top middle. While this is not desirable, it is better than guessing positive texts as negative and vice versa.

In **Figure 4.6** the model produces a more equal spread. This means that there is not a specific label the model has trouble with more than that it is slightly unsure on all of them. As long as the unsure numbers are small this might be acceptable. The bottom middle and middle right are in comparison larger than the other wrongful classifications, however as discussed earlier, it is not ideal but it is better than the corners. It shows that the model has some trouble classifying positive texts as neutra and vice versal.

The model in **Figure 4.2**, which had the highest macro average F1-score is also balanced in the predictions, with the left and right corner being low numbers. Since both the model in **Figure 4.2** and **4.6** has *DistilRoBERTa-finetuned-financial-news* as the base model they are the closest that can be compared. The model in **Figure 4.2** performs better on the "Positive" label as can be seen when looking in the bottom middle which is lower and the bottom right which is higher than the model in **Figure 4.6**. So in comparison the model that was not fine-tuned performed better on the positive labels but worse on the neutral one, on the negative label they are about the same however the non fine-tuned model classified one less negative text as a positive one. The differences are small but since the dataset is not that large either the small differences still affect the macro average F1-score.

In conclusion, the model in **Figure 4.5** has a hard time classifying neutral label texts, often

classifying them as either negative or positive. The model in **Figure 4.6** has a hard time classifying neutral and positive texts, often switching them around, i.e. classifying neutral texts as positive and positive texts as neutral. The final and best model in **Figure 4.2** has the same problem but is slightly better towards the positive sentiment.

What causes the difference is unknown. Since the models were trained on the same dataset it can be ruled out that it has anything to do with the difference. The most obvious difference is that the models do not share the same base model, one being *BERT* and the other two being *RoBERTa*. *BERT* seems to have a harder time correctly classifying something as neutral, while *RoBERTa* makes more misclassifications on the positive and negative labels. With this said, a model which classified everything correctly is remarkable and hard to get.

5.2.5 Analysis on Misclassified Texts

In Figure 4.1 the top right and bottom left corner numbers are relatively low in comparison to the other numbers which is good. When looking at the texts that were wrongly classified in the left corner, i.e. the texts that were classified as negative when they were pre-labeled as positive, no apparent pattern could be detected if compared to other texts that were correctly classified. Nonetheless, if not compared to the other texts, then some observations can be made.

The first observation is that all the texts had many mentions of company names and places in them. The second observation is that two of the texts had what could be considered as noise since non-english words were present in the texts. The third and final observation is that all the texts generally had very negative sentiment when looking at them objectively. Many negative words were present and in general the texts often brought up examples of negative events such as controversies or how something badly affects the environment. Only a small portion of the texts could be seen as positive for a certain company from an *ESG* point-of-view.

Since there was only one text that was misclassified as positive when it was pre-labeled as negative, no comparison can be made with other texts. However, by viewing the text it is very clear that the text is negative and that the pre-label is correct. As to why it was classified as a positive text is unknown and would require further investigations into each word's individual weight. To be noted is that only the misclassified texts from the very best model were looked at and therefore the hypothesis discussed above only apply to that specific model.

Chapter 6

Conclusion

In this thesis, a corpus of *ESG*-related news articles were used to train four transformer-based machine learning models. The objective was to predict the sentiment of the articles positive, neutral or negative with a higher accuracy than the lexicographical solutions.

Compared to the lexicographical solutions explored in this thesis, the results show that *BERT*-based models perform markedly better in the sentiment classification task for the *ESG* dataset used in this thesis. The average macro average F1-score of each model was evaluated after 10 runs, to establish a confidence interval for each pooling method. From a selection of four initial models, the best performing was selected in each iteration for hyperparameter tuning and fine-tuning. The final models selected for fine-tuning were the models *DistilRoBERTa-finetuned-financial-news* and *FinBERT*. *DistilRoBERTa-finetuned-financial-news* achieved the highest macro average F1-scores of 0.804 from being trained with *Flair*'s default hyperparameters.

A learning experience from the duration of this thesis was the realization of the minor impact hyperparameter tuning and fine-tuning had on the final results. In hindsight, it would have been more promising to explore more models in their default state, with a focus on more pre-training.

6.1 Reflection of Ethical Aspects

A possible social benefit from this thesis is the increased insight and visibility from the use of sentiment analysis in the area of *ESG*. As mentioned in **Section 1.1**, most pollution comes from a minority of companies. Reduced investment from a poor ESG-score may lead to positive changes for the climate.

The ML model used in this thesis used publicly available news articles as the basis of training,

which may lead to issues if more ESG-related models are released on the market. Companies who suspect a poor ESG-rating may attempt to increase the amount of positive news about their company, which could introduce biases in future ML models.

6.2 Answers to Research Questions

At the beginning of this thesis, the aim was to be able to answer the research questions stated in **Section 1.4**, the answers to the questions are as follows:

- *How is state-of-the-art sentimental analysis done currently?*
  Current state-of-the-art machine learning models for sentiment analysis are evaluated using common benchmarks. For most benchmarks, transformer-based models achieved the highest performance [PapersWithCode, 2022].

- *What tools can be used for a machine learning solution?*
  Machine learning solutions for sentiment analysis are mainly done through two different approaches, *LSTM* or transformers. In this thesis, it was decided to focus on transformers since research showed increased performance compared to solutions using *LSTM* [Colón-Ruiz and Segura-Bedmar, 2020]. The *Flair* framework simplified the process of fine-tuning a selection of pre-trained models from *Hugging Face*.

- *How should different solutions be compared?*
  In **Section 2.6**, several methods of evaluating the results from a model were described. Accuracy can give a good indication of how the model is performing when the test set is balanced and missed classifications are not that important. Should the test set be imbalanced and/or the classifications be of importance then the macro average F1-score is a much more accurate and telling metric to evaluate a model. The dataset used in this thesis was imbalanced in favour of the neutral label as described in **Section 3.5**. As it was evaluated that all three labels were to be equally weighed, the macro average F1-score was used as the main evaluation metric.

- *How can a transformer model be optimized for text classification?*
  With the initial step of transfer learning, a pre-trained model can be used, saving time and resources compared to developing a custom model from scratch. Furthermore, the pre-training can be extended by using the dataset intended for the NLP task as done in **Section 2.7**. By then tuning the hyperparameters, the model may achieve faster or better convergence to a minimum point in the loss curve. Finally, with fine-tuning, the model's embeddings may be better adapted to the dataset it is trained on.

- *What tools exist that can augment an* ESG*-based dataset for NLP?*
  To augment a dataset there are two options, to modify it, or increase the quantity. Modification may be in the form of pre-processing, which can involve removing noise that adversely impacts the performance of a model. Quantity may be changed in the form of text augmentation, increasing the amount of data that is fed to the model, which may increase performance.

6.3 Future Work

This section will describe potential future improvements to the results in this thesis. The following subsections will be topics that were either out of scope for this thesis or dismissed because of a lack of time.

6.3.1 Dataset Size

For this thesis, a limitation was the size of the labelled dataset. Since most research shows increased performance when a model is trained with large datasets [Mutuvi et al., 2020] [Adadi, 2021], further research with larger datasets with *ESG*-related news articles is warranted. Alternatively, further experimentation with the augmentation of the dataset may be explored. In this thesis, augmentation was mainly used to achieve a balanced dataset for all the labels, however, it could also be utilized to expand the dataset further.

6.3.2 Further Pre-training

As shown in **Section 2.7**, further pre-training *BERT* with the dataset produced superior performance. It would be interesting to follow the same methodology with the dataset used in this thesis.

6.3.3 Fine-tuning During Hyperparameter Tuning

During the hyperparameter search phase, fine-tuning was disabled, which meant that the embedding layers were frozen. It could be the case that enabling fine-tuning during searching could result in a hyperparameter combination that yields a higher F1-score than those presented. Additionally, when the ranges for the hyperparameters were selected, values that could be more beneficial during fine-tuning were not considered. A future improvement could be to perform more hyperparameter searching for the fine-tuning step, as it could lead to an increase in performance.

6.3.4 VADER Thresholds

As seen in **Table 4.3**, *VADER* performed poorly on the neutral sentiment. Since the sentiment classification depends on the thresholds, further tweaking could result in better performance.

6.3.5 VADER pre-processing

To improve *VADER*'s classifications further text pre-processing might be required. For example company names might be very loaded with either positive of negative sentiment and therefore might affect the compound score given by *VADER*. To remove the company names and names of places and people might also be beneficial the *transformer* approach.

6.3.6 Piecewise Predictions

When analysing the dataset, it was observed that most articles contained hundreds of words, with some exceeding *BERT*'s token limit of 512. It could be the case that performance suffered because the true sentiments of the articles were not conveyed. Future exploration could examine splitting the articles into paragraphs or sentences, and then making predictions on each. By then combining the predictions, the final result may be more accurate.

6.3.7 LSTM

To reduce the scope of this thesis, only transformers were explored as the machine learning approach. However, it could be the case that an *LSTM*-based model could offer better performance on the dataset. According to [Colón-Ruiz and Segura-Bedmar, 2020], an *LSTM*-based model achieved acceptable results while requiring less training time.

6.3.8 Machine Learning Bias

When a dataset used for training a model contains biases, these biases may affect the predictions made by a model [Mehrabi et al., 2021]. For the dataset used in this thesis, this bias could be company names. If a company name is more often occurring under articles labelled negative, the model may associate that company name with a negative sentiment. A future improvement can be to remove the company names from the training set to remove this potential source of bias.

Chapter 7

Terminology

The following list contains terms that either are not fully explained in this thesis or provide a summarized explanation for the terms.

- **Corpus:** A collection of large amounts of texts, and can be used in a machine learning context for NLP tasks. The corpus can be annotated, which means a tag is added for each document, e.g. annotating labels for the use in sentiment analysis.
- **Support:** Number of occurrences of labels in the specified dataset.
- **Classification layer:** A classification layer (or classifier) is added on top of *BERT*'s own layers if the NLP task involves classification, such as sentiment analysis. The most common method is to use a linear layer, which transforms X dimensions matrices to Y dimensions matrices. For sentiment analysis this means that *BERT*'s 512 input dimensions will be output as the number of labels in the dataset, e.g. 3 output dimensions for the labels "Positive", "Neutral", and "Negative".
- **Pooling:** Affects how the sentence embedding is created for *BERT* before it is sent to the classifier. Can either be "CLS" or an aggregation of the contextualised word embeddings like "mean" or "max".
- **Batch size:** Number of training samples that are utilized during an iteration. Higher values require more GPU memory during training.
- **NLP:** Natural Language Processing are sets of methods that are used for aiding software in understanding the human language.
- **ML:** ML, or Machine Learning, is a field of study branching from AI that enables computers to "self-learn" by using different algorithms that can detect patterns in data.

- **AI:** Artificial intelligence is when a machine displays intelligence otherwise found in animals or humans. This intelligence may be expressed as the ability to rationalize and take decisions that achieve specific goals.
- **VADER:** *Python* library for processing textual data. Features rule-based sentiment analysis.
- **BERT:** Short for Bidirectional Encoder Representations from Transformers, *BERT* is a transformer-based machine learning model that specializes in NLP.
- **RoBERTa:** A model based on *BERT* that modified the pre-training phase with additional data and techniques that improved performance for several datasets.
- **Sentiment Analysis:** Analytics based on the use of natural language processing (NLP) in order to gauge user sentiment towards a given subject. This sentiment can either be positive, negative or neutral.
- **TF-IDF:** Short for Term Frequency-Inverse Document Frequency is the process of calculating the relevancy of words in a given corpus.

References

[Accounting for Sustainability, 2022] Accounting for Sustainability (2022). Knowledge hub. (accessed May 25, 2022) `https://www.accountingforsustainability.org/en/index.html`.

[Adadi, 2021] Adadi, A. (2021). A survey on data-efficient algorithms in big data era. *Journal of Big Data*, 8:1–54. (accessed June 6, 2022).

[Akbik et al., 2019] Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019). Flair: An easy-to-use framework for state-of-the-art nlp. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59.

[Amazon, 2021] Amazon (2021). *Amazon EC2*. (accessed May 25, 2022) `https://aws.amazon.com/ec2/`.

[Bergstra et al., 2013] Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA. PMLR.

[Buduma and Locascio, 2017] Buduma, N. and Locascio, N. (2017). *Fundamentals of deep learning : designing next-generation machine intelligence algorithms, 1st Ed.* O'Reilly Media.

[Colón-Ruiz and Segura-Bedmar, 2020] Colón-Ruiz, C. and Segura-Bedmar, I. (2020). Comparing deep learning architectures for sentiment analysis on drug reviews. *Journal of Biomedical Informatics*, 110:103539.

[Connor et al., 2021] Connor, S., Taghi M., K., and Borko, F. (2021). Text data augmentation for deep learning. *Journal of Big Data*, 8(1):1 – 34.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. (accessed May 5, 2022) arXiv:1810.04805.

[Domingos, 2012] Domingos, P. M. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55:78 – 87.

[Gautam and Yadav, 2014] Gautam, G. and Yadav, D. (2014). Sentiment analysis of twitter data using machine learning approaches and semantic analysis. *2014 Seventh International Conference on Contemporary Computing (IC3), Contemporary Computing (IC3), 2014 Seventh International Conference on*, pages 437 – 442. (accessed May 20, 2022).

[Hasan et al., 2018] Hasan, A., Moin, S., Karim, A., and Shamshirband, S. (2018). Machine learning-based sentiment analysis for twitter accounts. *Mathematical and Computational Applications*, 23(1). (accessed May 4, 2022).

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780. (accessed April 28, 2022).

[Howard and Ruder, 2018] Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. (accessed May 6, 2022) arXiv:1801.06146.

[Hutto and Gilbert, 2015] Hutto, C. and Gilbert, E. (2015). Vader: A parsimonious rule-based model for sentiment analysis of social media text. (accessed February 25, 2022).

[IBM, 2021] IBM (2021). *CRISP-DM Help Overview*. (accessed May 25, 2022) `https://www.ibm.com/docs/en/spss-modeler/SaaS?topic=dm-crisp-help-overview`.

[Jianqiang and Xiaolin, 2017] Jianqiang, Z. and Xiaolin, G. (2017). Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, 5:2870–2879. (accessed April 3, 2022).

[Jurafsky and Martin, 2021] Jurafsky, D. and Martin, J. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, draft*, volume 3.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. (accessed May 10, 2022) arXiv:1412.6980.

[Kumar et al., 2020] Kumar, A., Makhija, P., and Gupta, A. (2020). Noisy text data: Achilles' heel of BERT. (accessed May 16, 2022) arXiv:2003.12932.

[Li et al., 2019] Li, X., Sun, X., Meng, Y., Liang, J., Wu, F., and Li, J. (2019). Dice loss for data-imbalanced nlp tasks. (accessed May 15, 2022) arXiv:1911.02855.

[Liddy, 2001] Liddy, E. (2001). Natural language processing. in encyclopedia of library and information science, 2nd ed.

[Liu, 2020] Liu, B. (2020). *Introduction*, page 1–17. Studies in Natural Language Processing. Cambridge University Press, 2 edition. (accessed April 25, 2022).

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. (accessed May 2, 2022) arXiv:1907.11692.

[Loshchilov and Hutter, 2017] Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. (accessed May 14, 2022) arXiv:1711.05101.

[Ma, 2019] Ma, E. (2019). NLP augmentation. (accessed May 15, 2022) `https://github.com/makcedward/nlpaug`.

[Malo et al., 2013] Malo, P., Sinha, A., Takala, P., Korhonen, P., and Wallenius, J. (2013). Good debt or bad debt: Detecting semantic orientations in economic texts. (accessed April 18, 2022) arXiv:1307.5336.

[Mehra et al., 2022] Mehra, S., Louka, R., and Zhang, Y. (2022). ESGBERT: Language model to help with classification tasks related to companies environmental, social, and governance practices. (accessed April 16, 2022) arXiv:2203.16788.

[Mehrabi et al., 2021] Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., and Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys*, 54(6):1 – 35. arXiv:1908.09635.

[Mikolov et al., 2013] Mikolov, T., tau Yih, W., and Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *NAACL*.

[Montgomery et al., 2021] Montgomery, D. C., Peck, E. A., and Vining, G. G. (2021). *Introduction to linear regression analysis, 5th Ed.* John Wiley & Sons.

[Mutuvi et al., 2020] Mutuvi, S., Boros, E., Doucet, A., Jatowt, A., Lejeune, G., and Odeo, M. (2020). Multilingual epidemiological text classification: A comparative study. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6172–6183, Barcelona, Spain (Online). International Committee on Computational Linguistics. (accessed June 6, 2022).

[O'Reilly and Chanmittakul, 2021] O'Reilly, J. A. and Chanmittakul, W. (2021). L1 regularization-based selection of eeg spectral power and ecg features for classification of cognitive state. *2021 9th International Electrical Engineering Congress (iEECON), International Electrical Engineering Congress (iEECON), 2021 9th*, pages 365 – 368.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359. (accessed May 16, 2022).

[PapersWithCode, 2022] PapersWithCode (2022). Sentiment analysis. (accessed May 25, 2022) `https://paperswithcode.com/task/sentiment-analysis`.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc. arXiv:1912.01703.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. (accessed May 8, 2022) arXiv:1609.04747.

[Sanh et al., 2019] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. (accessed May 4, 2022) arXiv:1910.01108.

[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48. (accessed May 5, 2022).

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958. (accessed May 10, 2022).

[Starr, 2016] Starr, D. (2016). Just 90 companies are to blame for most climate change, this 'carbon accountant' says. *Science*. (accessed May 25, 2022).

[Value Reporting Foundation, 2022] Value Reporting Foundation (2022). Sasb standards. (accessed April 25, 2022) `https://www.sasb.org/`.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. (accessed April 25, 2022) arXiv:1706.03762.

[Yang et al., 2020] Yang, Y., UY, M. C. S., and Huang, A. (2020). FinBERT: A pretrained language model for financial communications. (accessed April 18, 2022) arXiv:1908.10063.

Appendices

Appendix A

SASB Materiality Map

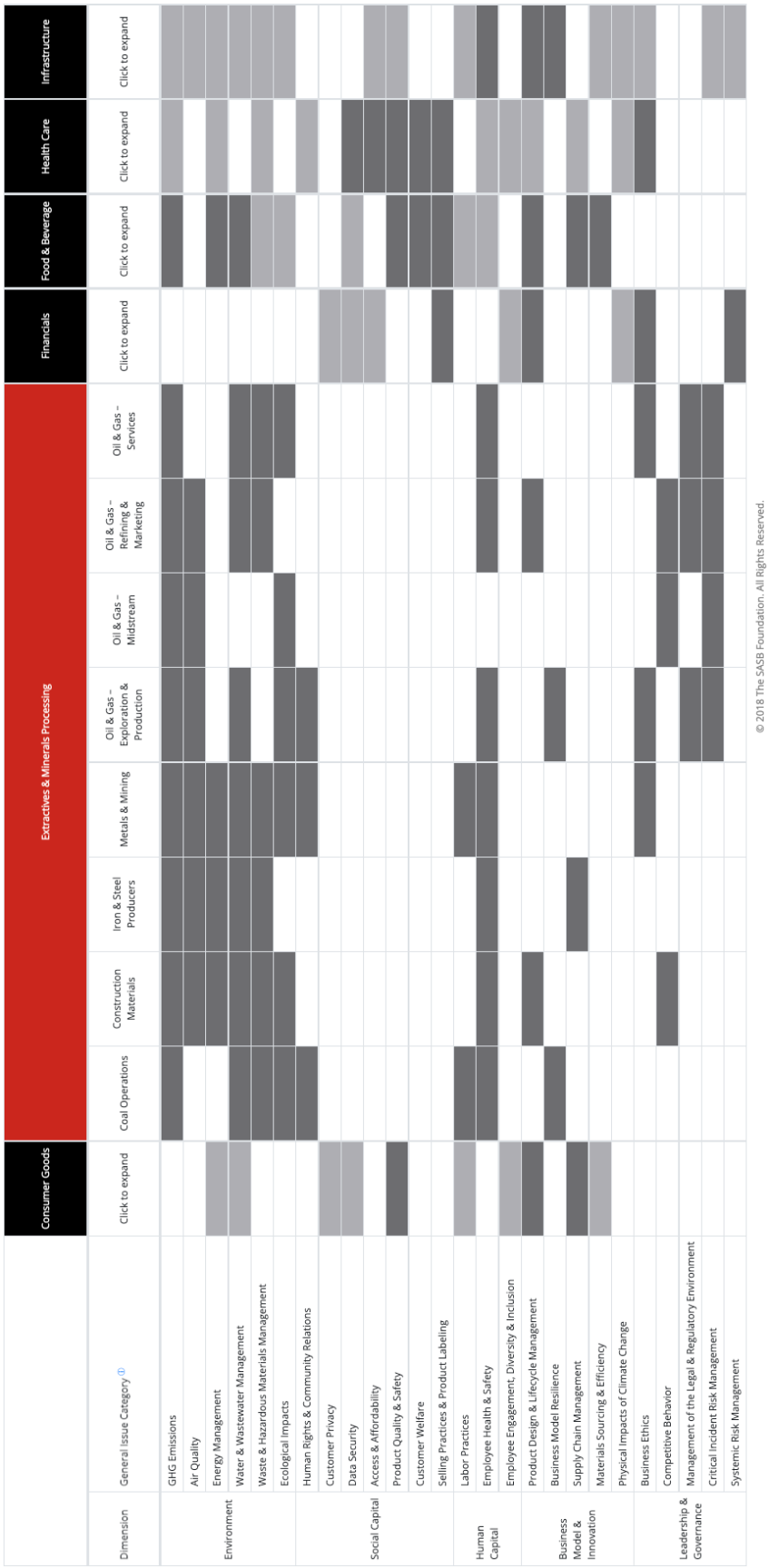

Figure A.1: SASB Materiality Map, © 2022. Reprinted with permission from *Value Reporting Foundation*. All rights reserved.