DEGREE PROJECT IN TIDAB, SECOND LEVEL
STOCKHOLM, SWEDEN 2021

Python Has Better Text Classification Libraries Than Scala

AUGUST HENNING BRUCE
FILIP GARAMVÖLGYI

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Python Has Better Text Classification Libraries Than Scala

August Henning Bruce, Filip Garamvölgyi

2021-10-05

Bachelor’s Thesis

Examiner
Mira Kajko-Mattson

Academic adviser
Johan Montelius

KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science (EECS)
Department of Computer Science
SE-100 44 Stockholm, Sweden

Abstract

In today's internet era, more text than ever is being uploaded online. The text comes in many forms, such as social media posts, business reviews, and many more. For various reasons, there is an interest in analyzing the uploaded text. For instance, an airline business could ask their customers to review the service they have received. The feedback would be collected by asking the customer to leave a review and a score. A common scenario is a review with a good score that contains negative aspects. It is preferable to avoid a situation where the entirety of the review is regarded as positive because of the score, if there are negative aspects mentioned. A solution to this would be to analyze each sentence of a review and classify it by negative, neutral or positive depending on how the sentence is perceived.

With the amount of text uploaded today, it is not feasible to manually analyze text. To automatically classify text by a set of criteria is called text classification. The process of specifically classifying text by how it is perceived is a subcategory of text classification known as sentiment analysis. Positive, neutral and negative would be the sentiments to classify.

The most popular frameworks associated with the implementation of sentiment analyzers are developed in the programming language Python. However, over the years, text classification has had an increase in popularity. The increase in popularity has caused new frameworks being developed in new programming languages. Scala is one of the programming languages that have had new frameworks developed to work with sentiment analysis. However, in comparison to Python, it has less available resources. There are even fewer resources regarding sentiment analysis on a less common language such as Swedish. The problem is no one has compared a sentiment analyzer for Swedish text implemented using Scala and compared it to Python. The purpose of this thesis is to compare the quality of a sentiment analyzer implemented in Scala to Python. The goal of this thesis is to increase the knowledge regarding the state of text classification for less common natural languages in Scala.

To conduct the study, a qualitative approach with support of quantitative data was used. Two kinds of sentiment analyzers were implemented in Scala and Python. The first classified text as either positive or negative (binary sentiment analysis), the second sentiment analyzer would also classify text as neutral (multiclass sentiment analysis). To perform the comparative study, the implemented analyzers would perform classification on text with known sentiments. The quality of the classifications was measured using their F1-score.

The results showed that Python performed better for both tasks. In the binary task there was not as large of a difference between the two implementations. The resources from Python were more specialized for Swedish and did not seem to be as affected by the small dataset used as the resources in Scala. Scala had a F1-score of 0.78 for binary sentiment analysis and 0.65 for multiclass sentiment analysis. Python had a F1-score of 0.83 for binary sentiment analysis and 0.78 for multiclass sentiment analysis.

Keywords

LaBSE, Spark NLP, NLP, Text classification, Scala

Sammanfattning

I dagens internetera laddas mer text upp än någonsin online. Texten finns i många former, till exempel inlägg på sociala medier, företagsrecensioner och många fler. Av olika skäl finns det ett intresse av att analysera den uppladdade texten. Till exempel kan ett flygbolag be sina kunder att lämna omdömen om tjänsten de nyttjat. Feedbacken samlas in genom att be kunden lämna ett omdöme och ett betyg. Ett vanligt scenario är en recension med ett bra betyg som innehåller negativa aspekter. Det är att föredra att undvika en situation där hela recensionen anses vara positiv på grund av poängen, om det nämnts negativa aspekter. En lösning på detta skulle vara att analysera varje mening i en recension och klassificera den som negativ, neutral eller positiv beroende på hur meningen uppfattas.

Med den mängd text som laddas upp idag är det inte möjligt att manuellt analysera text. Att automatiskt klassificera text efter en uppsättning kriterier kallas textklassificering. Processen att specifikt klassificera text efter hur den uppfattas är en underkategori av textklassificering som kallas sentimentanalys. Positivt, neutralt och negativt skulle vara sentiment att klassificera.

De mest populära ramverken för implementering av sentimentanalysatorer utvecklas i programmeringsspråket Python. Men genom åren har textklassificering ökat i popularitet. Ökningen i popularitet har gjort att nya ramverk utvecklats för nya programmeringsspråk. Scala är ett av programmeringsspråken som har utvecklat nya ramverk för att arbeta med sentimentanalys. I jämförelse med Python har den dock mindre tillgängliga resurser. Det finns ännu färre resurser när det gäller sentimentanalyser på ett mindre vanligt språk som svenska. Problemet är att ingen har jämfört en sentimentanalysator för svensk text implementerad med Scala och jämfört den med Python. Syftet med denna avhandling är att jämföra kvaliteten på en sentimentanalysator implementerad i Scala med Python. Målet med denna avhandling är att öka kunskapen om tillståndet för textklassificering för mindre vanliga naturliga språk i Scala.

För att genomföra studien användes ett kvalitativt tillvägagångssätt med stöd av kvantitativa data. Två typer av sentimentanalysatorer implementerades i Scala och Python. Den första klassificerade texten som antingen positiv eller negativ (binär sentimentanalys), den andra sentimentanalysatorn skulle också klassificera text som neutral (sentimentanalys i flera klasser). För att utföra den jämförande studien skulle de implementerade analysatorerna utföra klassificering på text med kända sentiment. Klassificeringarnas kvalitet mättes med deras F1-poäng.

Resultaten visade att Python presterade bättre för båda uppgifterna. I den binära uppgiften var det inte lika stor skillnad mellan de två implementeringarna. Resurserna från Python var mer specialiserade för svenska och verkade inte påverkas lika mycket av den lilla dataset som används som resurserna i Scala. Scala hade ett F1-poäng på 0,78 för binär sentimentanalys och 0,65 för sentimentanalys i flera klasser. Python hade ett F1-poäng på 0,83 för binär sentimentanalys och 0,78 för sentimentanalys i flera klasser.

Nyckelord

LaBSE, Spark NLP, NLP, Textklassificering, Scala

Acknowledgments

We would like to thank Mr. Jakob Dobrzynski from Reco AB for providing us with the datasets used in this study as well as providing us the opportunity to conduct the study.

We would also like to thank our examiner Associate Professor Mira Kajko-Mattson for all feedback and help provided to write this paper.

Stockholm, 10 2021
August Henning Bruce, Filip Garamvölgyi

Table of contents

Abstract i
- Keywords i

Sammanfattning iii
- Nyckelord iii

Acknowledgments v
Table of contents vii
List of Figures ix
List of Tables xi
List of acronyms and abbreviations xiii

1 Introduction 1
- 1.1 Background 1
- 1.2 Problem 2
- 1.3 Purpose 2
- 1.4 Goal 2
- 1.5 Benefits, Ethics and Sustainability 2
- 1.6 Research Methodology 3
- 1.7 Delimitations 3
- 1.8 Structure of the thesis 4

2 Text Classification 5
- 2.1 Classifying Reviews 5
- 2.2 Scala and Python 6
- 2.3 Apache Spark 7
- 2.4 Mathematical Representation of Language 8
- 2.5 Language Models 9
- 2.6 Transfer Learning 10
- 2.7 Evaluating Performance 11
- 2.8 Related Work 12
- 2.9 Summary 12

3 Research methodology 13
- 3.1 Research Strategy 13
- 3.2 Research Phases 14
- 3.3 Research Method 16
- 3.4 Research Instruments 17
- 3.5 Comparison Criteria 17
- 3.6 Validity Threats 17

4 Design and Implementation 19
- 4.1 Phases of The Sentiment Analyzer 19
- 4.2 Processing data 20
- 4.3 Fine-tune LaBSE 20
- 4.4 Evaluate Quality 21
- 4.5 Dataset 22

5 Results and Analysis 25
- 5.1 Results using Spark NLP's SentimentDLApproach 25
- 5.2 Results using Spark NLP's ClassifierDLApproach 26
- 5.3 Python Results 28

5.4 Result Summary 29
5.5 Validity Analysis 30
5.6 Discussion 31
6 Conclusions and Future Work 33
6.1 Evaluation 33
6.2 Future work 33
6.3 Conclusion 34
References 35
Appendix A: Design and Implementation of Python Solution 37
Appendix B: Detailed results 41
Detailed results of SentimenDLApproach for binary sentiment analysis 41
Detailed results of SentimenDLApproach for multiclass sentiment analysis 46
Detailed results of ClassifierDLApproach for binary sentiment analysis 47
Detailed results of ClassifierDLApproach for multiclass sentiment analysis 52
Detailed results of binary sentiment analysis developed in Python 60
Detailed results of multiclass sentiment analysis developed in Python 62

List of Figures

Figure 2.1. Model of entities involved in the implementation of a sentiment analyzer 5
Figure 2.2. Text classification frameworks in Scala & Python 6
Figure 2.3. Vector space showcasing ten words close to music 8
Figure 2.4. An example of an artificial neural network 9
Figure 3.1. Research Strategy Model 13
Figure 3.2. Overview of the research phases 14
Figure 4.1. Phases of the sentiment analyzer 19
Figure 4.2. Workflow of pipeline 21
Figure 4.3. Workflow of the text classification pipeline model 21
Figure 4.4. Label split of the entire dataset 22
Figure 4.5. Label split of the training set excluding neutral 23
Figure 4.6. Label split of the training set including neutral 23
Figure 4.7. Label split of the testing set excluding neutral 24
Figure 4.8. Label split of the testing set including neutral 24
Figure 5.1. Plot of difference in F1 measure for binary sentiment analysis based on validation split and hyper-parameters using SentimentDLApproach ... 25
Figure 5.2. Plot of difference in F1 measure for binary sentiment analysis based on validation split and hyper-parameters using ClassifierDLApproach 27
Figure 5.3. Plot of difference in F1 measure for multiclass sentiment analysis based on validation split and hyper-parameters using ClassifierDLApproach Reliability Analysis 28

List of Tables

Table 5-1. Evaluation summary of best performing hyper-parameters with SentimentDLApproach when tested on binary sentiment dataset 25
Table 5-2. Evaluation summary of best performing hyper-parameters with SentimentDLApproach when tested on multiclass sentiment dataset 26
Table 5-3. Evaluation summary of best performing hyper-parameters fine-tuned for binary sentiment analysis using ClassifierDLApproach 26
Table 5-4. Evaluation summary of best performing hyper-parameters fine-tuned for multiclass sentiment analysis using ClassifierDLApproach 27
Table 5-5. Evaluation summary of binary sentiment analysis fine-tuned in Python ... 28
Table 5-6. Evaluation summary of fine-tuned binary sentiment analysis 29
Table 5-7. Evaluation summary of multiclass sentiment analysis fine-tuned in Python 29
Table 5-8. Evaluation summary of all implementations... 29

List of acronyms and abbreviations

BERT	Bidirectional Encoder Representations from Transformers
LaBSE	Language-agnostic BERT Sentence Embeddings
NLP	Natural language processing

1 Introduction

In today's internet era more text than ever is uploaded and made accessible online [1]. The text comes in many forms, such as social media posts, emails, business reviews and many more. For various reasons there is an interest in analyzing the uploaded text. However, with the amount that is being uploaded it is not feasible to manually analyze text anymore. Because of this, the field of Natural Language Processing (NLP) studies different ways of automatically analyzing large amounts of natural language [2].

As a consequence of Moore's law stating that the number of transistors in circuits doubles every two years, computers have increased tremendously in processing power during the 21st century. The increase in processing power has been beneficial for NLP and has led to many improvements in the techniques used to process text. For instance, there have been systems developed to automatically classify emails as harmful spam [3]. Or an airline business can now analyze their reviews to classify what perception their customers have of different aspects of their business [4].

To analyze text and based on its content give it a specific label, is a subfield in NLP referred to as text classification. The current frameworks with the most public resources available for NLP and text classification are developed in Python. With the field becoming more popular there has been an increase in support for text classification in other programming languages. But these new frameworks still lack many of the resources that are available in Python. Especially when the text classification is being performed in a less common natural language. Therefore, evaluating the state of one of the newer frameworks for text classification compared to Python is of interest.

1.1 Background

There are many industrial and non-industrial domains where text analysis is being used. In the case of many airlines and customer satisfaction, they ask their customers to leave feedback of the service they have received. The feedback is normally collected by asking customers to leave a review, containing text and a score of the service. One of the shortcomings with this type of review is that a review can have a good score, but parts of it point out negative aspects. Because of the good score the text in its entirety may be regarded as positive and the negative feedback will be disregarded. This problem can be solved using Natural Language Processing (NLP) and text classification. To solve the problem using text classification, each sentence of a review would be classified by its perception which is positive, neutral or negative [4].

There is a subcategory of text classification which seeks to classify the perception of text. This subcategory is known as sentiment analysis. The sentiment of text can be labeled as either positive, neutral or negative. By analyzing each sentence in the following text: “I am from Sweden. The meatballs here are great.”. If each sentence is classified by their sentiment the result would be: (“I am from Sweden.”, neutral), (“The meatballs here are great.”, positive).

To use NLP and sentiment analysis advanced technologies are needed. However, over the years frameworks have been released that have high abstraction levels. Which has allowed developers not as knowledgeable about NLP to apply it. To implement a sentiment analyzer, a model representing natural language in a mathematical form is needed, a dataset containing text and its sentiments, a framework to work with the language representation model, and finally a way to evaluate the quality of the classifications.

Different NLP frameworks allow the use of different language representation models. There exist several frameworks in different programming languages, such as *Python*, *R*, *Java*, *Scala* and more. Because the programming language Python is mostly associated with NLP, it has support for most of the available language models using its frameworks. Scala, a programming language that has recently received more resources to perform NLP, has less language representation models available.

Not only the programming language used for NLP will affect what resources are available, but also the natural language that NLP is performed on. English, Spanish and French are examples of natural languages that have been studied extensively and therefore have resources in multiple programming frameworks. For a less common language such as Swedish there currently exist no language representation models in Scala. The only way to perform NLP on Swedish text is using a multilingual representation model. Which can create mathematical representations of multiple languages at once, they have been proven efficient when working on a singular language but normally perform worse compared to a specialized language representation model [5].

1.2 Problem

Support for text classification has increased over the years. Scala is an example of a programming language benefiting of the gained support. However, there still is a lack of resources to perform text classification on a less common language such as Swedish. The problem is no one has compared a sentiment analyzer for Swedish text implemented using Scala compared to an implementation in Python.

1.3 Purpose

The purpose of this thesis is to compare the quality of a sentiment analyzer implemented in Scala to an implementation in Python. The quality comparison focuses on the correctness of the sentiment analysis by the Scala and Python implementations. The overall accuracy of the classifiers will be compared using their F1-score.

1.4 Goal

The goal of this thesis is to increase the knowledge regarding the state of text classification for less common natural languages in Scala.

1.5 Benefits, Ethics and Sustainability

Modern solutions for text classification involve machine learning and the use of artificial neural networks to achieve state-of-the-art performance. Because of this, high costs regarding energy consumption and processing power are associated with text classification. Text classification is the process of automated categorization of text, which means the need for manual work is eventually removed. Automation is also a way to make time consuming tasks faster and more efficient. As mentioned, there are benefits but also ethical and sustainability aspects that must be considered when working with text classification.

Section 1.5.1 discusses beneficial outcomes of this paper, Section 1.5.2 discusses the ethical aspects to consider and finally, Section 1.5.3 the environmental aspects.

1.5.1 Benefits

The result of the thesis could benefit any organization or project looking to implement a text classifier. The tools that were used for this study have a high abstraction level, which reduces the needed knowledge in the field of machine learning to take advantage of similar solutions. However, many of the concepts that must be understood to implement a text classifier are explained in this thesis.

This thesis study also explores the ability to implement advanced techniques such as machine learning and artificial neural network but with a lesser amount of prior knowledge required than to develop the solutions. Which makes the field more accessible using libraries and tools with a high abstraction level.

1.5.2 Ethics

Because this thesis study is about the topic of automation, the effects of the study were considered. As can be seen from a study in 2018 published by Frontier Issues it was estimated that over 80 per cent of current jobs would most likely be lost due to automation. It is also mentioned that the number is not realistic because even if jobs will be lost there will also be new once created [6]. In the case of this study, the dataset provided by the company Reco AB had specifically been put together for the use of automated categorization. They had previously not worked with any manual classifications. The results of this study would therefore not put anyone at the company providing the dataset at risk due to automation.

1.5.3 Sustainability

As for sustainability, it is known that applications of machine learning require many resources such as processing power and energy consumption. For instance, the text classifiers developed during this thesis paper used language representation models that were based on a technology known as Bidirectional Encoder Representations from Transformers (BERT). A study conducted to compare the costs and CO_2 emissions from developing different language representation models show that BERT emits up to 652 kg of CO_2. That amount is equivalent to roughly 33% of the CO_2 emitted from an individual passenger flying from New York to San Francisco [7]. Considering the carbon footprint caused by developing a BERT model from scratch it should be heavily considered if using a BERT model available online is sufficient. This study was possible to conduct using the BERT models that are available online.

When using a BERT based text classifier in a production environment an important topic is how to implement it in a way that would not use unnecessary resources. An unnecessary resource such as the energy consumption of an idle application or an application optimized for large workloads only being used for lesser workloads.

1.6 Research Methodology

The research was conducted in the form of a comparative study with the purpose of comparing the performance of a text classifier developed in Scala compared to a text classifier developed in Python. The used methods were a qualitative method with the support of quantitative data. The authors of the thesis do not have any prior knowledge about the area of interest. Therefore thorough research was required, the chosen research strategy was chosen with this in mind.

To summarize, being of qualitative nature, heavy focus was put on research, and a literary study was therefore conducted. The evaluation of the model was done by acquiring numerical data from the model and using evaluation frameworks which is described in Chapter 3 under research instruments.

1.7 Delimitations

This thesis is focused on evaluating a language-agnostic model, however, during this evaluation process there are limiting factors; alongside the conducted study, a system was developed. The goal of the system was to use the text classifier developed using the language-agnostic model. Because of the uncertainty whether a language-agnostic model would be able to classify text with a performance deemed satisfactory, the framework interacting with the model had to allow for the language model to easily be changed without requiring the entire architecture of the system to change.

It is known that a major factor when it comes to the performance of text classification is the amount of data used during the fine-tuning process. In this comparative study a relatively small dataset of 3000 rows was used.

This thesis paper does not intend to give an analysis of the inner workings of the language model. The intention is to create an implementation using the model and test the performance on the dataset given the task of sentiment analysis. There is also not an intent to focus on how the text classifier was implemented in Python but the performance of the developed application.

1.8 Structure of the thesis

Chapter 2 presents relevant information to understand the subject of text classification, NLP and how to evaluate the performance of a text classifier. Chapter 3 explains how the research was conducted as well as how the obtained data was used and collected. In Chapter 4 the development of the text classifier is explained and how the frameworks were used. Chapter 5 presents the evaluation results of the developed text classifier and Chapter 6 discusses and analyzes the results found in Chapter 5 and what can be done for future studies.

2 Text Classification

This chapter provides the needed information to understand text classification and how it can be used. Section 2.1 contains information of how text classification works and gives an example of how it can be used. Section 2.4 - 2.5 explains how languages are represented in computer science and linguistics, and what type of language models that create the representations are used today. Section 2.6 explains how language representation models can be repurposed for new language related tasks. Section 2.7 shows metrics that can evaluate the performance of a repurposed language representation model. Finally, Section 2.8 gives a brief overview of related work.

2.1 Classifying Reviews

Today there exist multiple websites with the purpose of collecting reviews. The websites collecting reviews can be about different topics, such as movies, hotels or various businesses at once. A common practice for these types of websites is to let a user write a review and leave a rating. However, there might be a situation of a review with a perfect score, alongside a text that contains a part that would be perceived as negative. To avoid having all parts of a review being considered either positive or negative because of its score, one method would be to label each sentence of a review as either positive, neutral or negative.

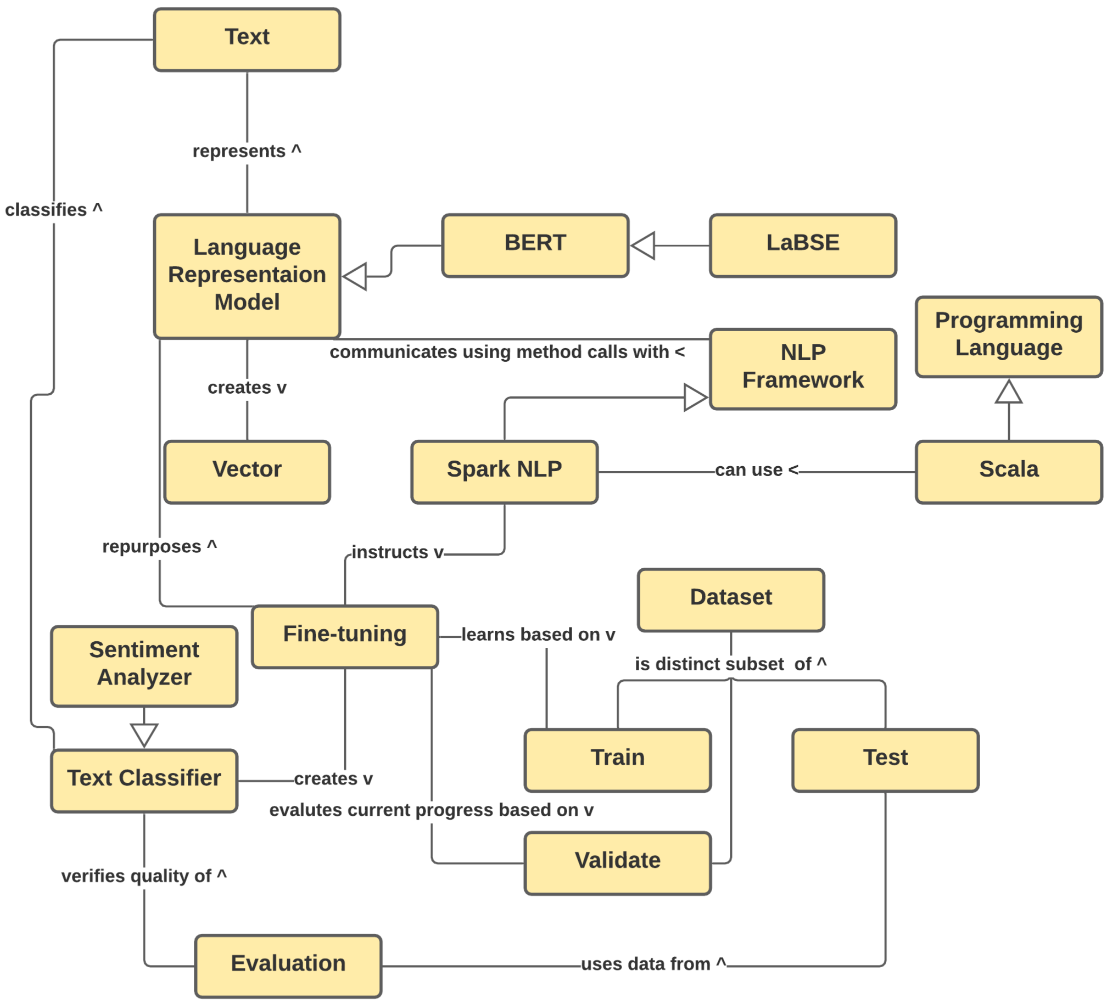

Figure 2.1. Model of entities involved in the implementation of a sentiment analyzer

The process of classifying text is referred to as text classification. The results of classifying each part of a review as either positive, neutral or negative can be beneficial to the entity reviewed as well as a reader of the review. For the entity, reviews would be a means of summarizing feedback, and for the reader it would be a way of getting an overview of positives and negatives associated with the entity. The sentences classified could then be worked further upon. The positive labeled sentences could be scanned for keywords as well as the negative sentences. If a few keywords are distinct it could entail a key characteristic that is either positive or negative. Once again, this information is important for both a reader of reviews and the receiver of the reviews.

Because of the time cost associated with manually classifying text, solutions have been developed to automatically label text. This field of study is known as natural language processing (NLP). NLP looks at how computers can analyze large amounts of ordinary text [8]. Ordinary text refers to text written without any thorough planning, as opposed to formal text. An example of ordinary text is a review submitted on a website. Text classification is one of the areas studied in NLP, text classification has a subcategory known as sentiment analysis. Sentiment analysis categorizes text by its sentiment, the sentiment of the text is normally positive, neutral or negative. In the case of only two sentiments, positive and negative, it is called binary sentiment analysis. In the case all three sentiments are used it is known as multiclass sentiment analysis.

To implement a sentiment analyzer using state-of-the-art technology, a dataset, language model, and programming framework is needed. Today's best performing language models are created using machine learning. Creating such language models require large amounts of time and processing power. To make this technology more accessible without having to perform the very time and resource intensive development, the language models can be downloaded online. The downloaded language models can then be repurposed to perform new tasks, such as sentiment analysis. This is made possible using different programing frameworks allowing interactions with the language models. However, to repurpose the language model it is normally done in combination with a dataset. The dataset contains text and the desired classification of the text. Figure 2.1 displays the key entities involved in the implementation of a sentiment analyzer.

2.2 Scala and Python

To implement a text classifier a programming language is required. Today most text classification solutions are developed in Python. Which has caused Python to be the language with most resources available for text classification. Python has several low-level frameworks that let the developer build a text classifier from scratch. There also exists frameworks that provide the developer with completed solutions.

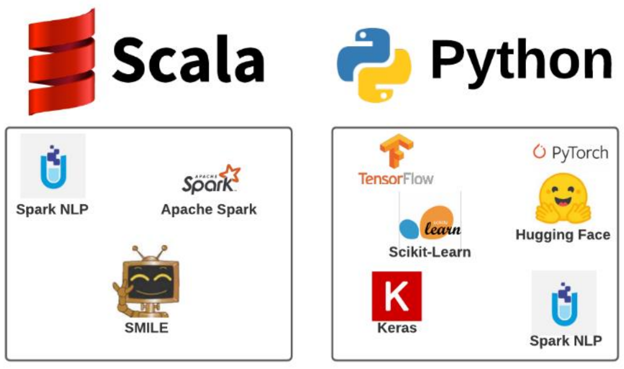

Figure 2.2. Text classification frameworks in Scala & Python

Scala as a programming language is not as common as many other languages [9]. Even so, Scala has received more support for text classification over the recent years. The frameworks that are available in Scala do not provide as many completed solutions for text classification as Python. Most frameworks provide support for developing a text classifier, but the developer is still required to do much of the work.

Scala and Python as languages to program in share a few similarities. Both languages have support for object oriented and functional programming. The major difference between the two is Scala being a statically typed while Python being dynamically typed. Which entails that Scala validates code during compile time while Python does so during runtime. The difference between implementing text classifiers in the two languages are the resources that are available. Python has multiple frameworks that provide models to encode text, while Scala has one popular framework that provides language representation models. The language models available in Scala mainly consist of the models made available by the Python frameworks refactored to work with Scala.

The two main ways of using existing language models in Python are through Hugging Face and TensorFlow [10, 11]. In Scala Spark NLP can be used to implement existing language models [12]. Figure 2.1 shows a set of the most popular frameworks used for text classification in Scala and Python.

2.3 Apache Spark

Apache Spark is a framework with the purpose of state-of-the-art large scale data processing developed in Scala. Spark combines multiple concepts used within data processing, such as streaming and machine learning [13]. Section 2.3.1 discusses data frames, a way to represent data in Apache Spark. Section 2.3.2 explains what a transformer in Apache Spark is. Section 2.3.3 explains what an estimator in Apache Spark is. Section 2.3.4 explains how pipelines are used in Apache Spark. Finally, Section 2.3.5 explains what the framework Spark NLP provides Apache Spark with.

2.3.1 Data frame

In Apache Spark, a data frame contains data. The data is represented in a table, where each object is represented in a row with the fields of data being defined by the columns [14]. The same way data is stored in a relational database, like SQL.

2.3.2 Transformer

In Apache Spark, the transforming process is defined as taking the values of each row in one or more columns and using that data to produce a new column [15]. In Apache Spark's machine learning library this concept is used during tasks such as text classifications. A data frame only containing text is transformed into a data frame containing both text and a predicted label.

2.3.3 Estimator

In Apache Spark an estimator uses a data frame like a transformer, however, it does not produce a new data frame. Instead, a transformer is produced by the estimator [15].

2.3.4 Pipelines

For machine learning Apache Spark uses Pipelines. A Pipeline consists of one or more Pipeline stages. In a Pipeline data is entered and goes through each stage, during each stage either the input data or output data from one of the stages is manipulated [15].

2.3.5 Spark NLP

Spark NLP is an open-source text processing library built on top of Apache Spark and SparkML [12]. It provides an API that can be used with Natural Language Processing pipelines. The library provides support for creating custom models as well as using trained models and pipelines.

The framework provides different language representation models. The supported models can be repurposed to perform various types of text classification tasks. Spark NLP makes it possible to implement sentiment analysis using two estimators: SentimentDLApproach and ClassifierDLApproach.

SentimentDLApproach learns to classify using text labeled as positive or negative. If there is too big of an uncertainty whether to classify positive or negative it will classify text as neutral. The level of certainty is defined as the probability of the text belonging to a classification as deemed by the classifier. Uncertainty is a probability below an adjustable threshold.

ClassifierDLApproach learns to classify using a dataset containing text with labels. All unique labels used by the dataset will represent the possible outcomes. To perform binary sentiment analysis text labeled as positive or negative would be used. For multiclass sentiment analysis text containing positive, neutral or negative must be used.

2.4 Mathematical Representation of Language

In natural language processing (NLP), text is usually not represented by the characters used when writing, instead, a mathematical representation of the text is used. The mathematical representation of the word is called a word embedding. Word embeddings consist of a real-valued vector instead of characters. The vector does not strive to represent the word's characters, more so the meaning of the word [16]. By encoding the meaning of the word, similar words are located close to one another in vector space.

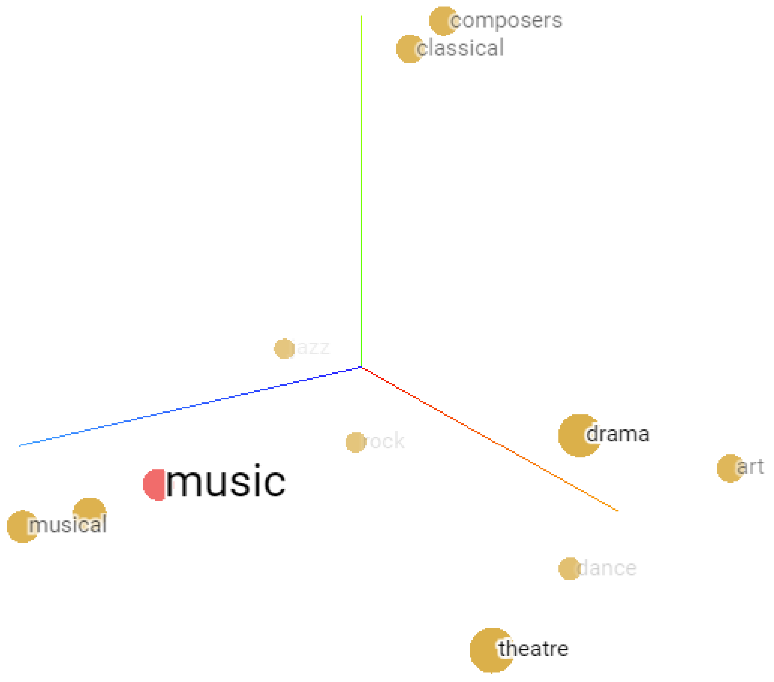

Figure 2.3. **Vector space showcasing ten words close to music**

Figure 2.1 visualizes how the word “music” is represented in vector space. The neighboring words indicate words close in meaning. The closer the word is the more similarities they should share. The size of the yellow circles corresponds to the distance between the words.

The use of vectors to represent words is not unique to NLP. The use of word vectors as a concept has also been studied in the scientific study of language known as linguistics [17]. The thought process is that a word's true meaning is defined by the words surrounding it, an idea popularized by Professor John Rubert Firth in 1957 [18].

By having an entire sentence consisting of vectors encoding the meaning of each word, the process of text classification can be performed by looking at the entire sentence as an encoded meaning, and label similar meanings in the same category.

2.5 Language Models

To take a word and represent its meaning as a vector, a language model performing the vectorization is needed. Today’s language models mainly use artificial neural network (ANN) approaches.

Section 2.3.1 gives an overview of how ANNs work. Section 2.3.2 showcases a state-of-the-art language model using an ANN approach. Finally, Section 2.3.3 showcases a multilingual language model.

2.5.1 Artificial Neural Network

ANNs are used as a way of teaching machines how to perform specific tasks. ANNs strive to mimic a brain. The neuron of an ANN sends signals to neighboring neurons, just like a brain. It does so by sending real numbers to the connected neurons. In computers this can be achieved by having a node system, each node represents a neuron. The edges in the node system that connect the neurons contain weights. Weights are used to adjust the strength of the signal a connected neuron is sending [19].

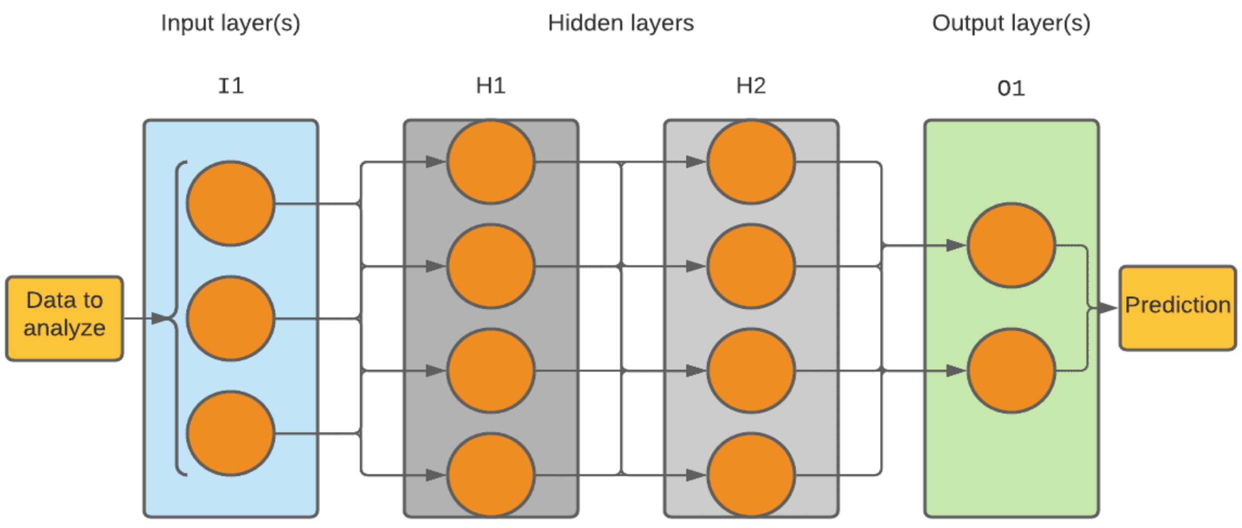

Figure 2.4. An example of an artificial neural network

In an ANN, the neurons exist in layers. All layers usually have specific functions that differ in each layer. The input layers of an ANN will receive the first signal, the signal will travel through each layer until it arrives at the last layer, the output layer.

Figure 2.2 shows an example of a potential architecture of an ANN. The data is entered and then goes through each layer and neuron, the output layer results in two outputs that are used to conclude a result.

To teach a machine to perform a specific task using an ANN, a large dataset is used. The dataset is commonly structured with a value in combination with the desired output based on the value. The value is entered and will be received in the ANN's input layer. It will signal through all layers until the output layer is reached. The ANN's output value, the prediction, is compared to the desired output. If the values do not match, an error is calculated, based on the error the weights attached to each connecting neuron will be adjusted. How the error and weight changes are calculated is decided by criteria specific to the ANN.

2.5.2 Bidirectional Encoder Representations from Transformers

A word's true meaning is understood when looking at the surrounding words [18]. The problem with various language models is that they do not follow this concept to its fullest extent. For instance, the word: "right", in the following sentences: "please move to the right" and "you are right", would by many language models be considered to have the same meaning. Bidirectional Transformers for Language Understanding (BERT) is a state-of-the-art ANN based language model with a new method of learning the meaning of words and encoding them [20]. BERT uses a new approach of encoding the meaning of words that make it possible to separate the meaning of a word such as "right" depending on the sentence it is used in.

Many language models prior to BERT would learn the meaning of words by looking at sentences left-to-right, a unidirectional approach. For BERT to achieve its state-of-the-art performance, it uses a bidirectional approach by taking a sentence as an input and randomly hiding words. BERT's ANN will then try to predict the hidden words based on the surrounding words. By using this technique BERT learns what different words mean given the context they are used in.

2.5.3 Language-agnostic BERT Sentence Embeddings

Natural language processing is also used for translations, when performing translations language models taught to understand various languages are normally used. These multilingual language models can be used for other tasks than translations. For instance, when performing a text classification task, it would be preferred to have a language model trained specifically for the language used by the text classification. However, it might be the case that the language is less common and therefore the only option is to use a multilingual model.

Language-agnostic BERT Sentence Embeddings (LaBSE) is a multilingual language model based on a version of BERT that vectorizes entire sentences instead of words [21]. Language-agnostic refers to the fact that the model does not need to distinguish what language is being vectorized. LaBSE has been taught to understand 109 different languages and can be used for NLP tasks such as text classification.

2.6 Transfer Learning

To perform a Natural Language Processing task such as text classification, a language model encoding the meaning of the text is normally used. Today it is common to use an Artificial Neural Network (ANN) based language model. Models that use ANNs, have been trained on large sets of data. LaBSE, as an example, uses more than one billion sentences for training. This process requires a large amount of time and computing power. To avoid the large number of resources needed to train an ANN based language model from scratch, many trained models are made available online. For instance, the researchers that created LaBSE, have made it possible to download a LaBSE model that creates embeddings without the need to do any training. Because the model does not need any training to perform its task, it is referred to as pre-trained.

The process of using a language model and adjusting it to perform a specific task such as text classification is referred to as transfer learning. Transfer learning can be described as taking a model made to perform task A and building upon that model to instead perform a related task B [22]. When a model is taught to perform a new task, it is referred to as fine-tuning. The fine-tuning process also requires a dataset. Today it is common to use ANNs to fine-tune a model. Section 2.5.1 further discusses the fine-tuning process.

2.6.1 Fine-tuning

When fine-tuning a language model to perform a new task such as text classification using an ANN, it is usually combined with hyper-parameters [23]. In an ANN data with a known result is entered, the ANN predicts what result it believes to be correct based on the data. If the ANN is wrong, the error is calculated, and the weights of the connected neurons are updated. Hyper-parameters are a way to set aspects of the update criteria.

Epoch is a hyper-parameter that tells the ANN how many times a dataset should be read. Batch size tells how many rows of data the ANN should go through before updating its weights based on the errors [24]. Learning rate sets how big the change to the weights should be and is normally a value between zero and one [25]. Dropout rate sets the rate at which randomly selected neurons should be ignored. The purpose of ignoring neurons is to avoid having a fine-tuned model that has had its weights adjusted to only be able to predict data like the dataset used during training [26].

Because there is not a limit to how many times an ANN can be trained, an ANN is normally evaluated during training and after the training. This is done by splitting the dataset used into three parts: train, validation and test. The train split is what the ANN uses to update its weights. Each time an epoch is over and the ANN has been updated, its performance is evaluated using the validation split. When all epochs are over the performance is tested on the test split. The train split is meant for learning, the validation split evaluates the learning progress and the test split ensures that the overall performance is good and has not been specialized to only classify the train and validation split.

2.7 Evaluating Performance

To evaluate how well a text classifier performs, the results of the text classifier need to be analyzed by a set of metrics. When analyzing the results of the classifier, a wrong classification would be considered a false positive (FP). The correct classification that was not predicted is considered a false negative (FN). If a classification is correctly predicted, it is a true positive (TP). Labels that are correctly not predicted are true negatives (TN).

To know how accurate a classifier is at predicting a label, precision (P) is used. Precision is the rate a classifier correctly predicts a label, as seen in formula 1.

$$P = \frac{TP}{TP+FP} \quad (1)$$

A classifier may be very accurate at predicting a certain label. However, that does not tell the whole story. If a classifier can predict two different subjects, A, B and C, and is given the task to predict ten observations that are of subject A. If the classifier only predicts A once, it will have a precision associated with the label A of 100%. In this example the Recall (R) metric of the classifier would be 10%. Recall shows at what rate a certain label is correctly predicted compared to the number of observations of the specific label, as seen in formula 2.

$$R = \frac{TP}{TP+FN} \, (2)$$

Because precision and recall combined gives a well-structured overview of a classifier’s overall performance, a harmonic average of both values is used to indicate the performance of a classifier. This value is known as F1-score and is harmonic in the sense that it is biased towards lower values, as seen in formula 3.

$$F1 = 2 \times \frac{P \times R}{P+R} \, (3)$$

Accuracy is a measurement of the quality of a classifier. Accuracy does not account for the distribution of a dataset and only considers the rate of correct classification out of all classifications made. Formula 4 shows how accuracy is calculated.

$$A = \frac{TF+TN}{TF+TN+FN+FP} \, (4)$$

2.8 Related Work

To understand what parts of the evaluation would be important for this study, similar studies were considered. What is worth mentioning is that the studies that will be discussed in this section did not consider framework and programming language. Instead, the focus was on the performance between different language models. While this study focuses on the programming language and framework used, the metrics evaluated in the other studies fulfill the same purpose in this one.

KBLab created a BERT model using a Swedish dataset [5]. This language model was then trained to do different NLP tasks, the performance of Swedish BERT (KB-BERT) was then compared to the results of Multilingual BERT (M-BERT) fine-tuned to do the same tasks. The main metric compared between the two models was F1 and the result of the study was KB-BERT outperforming M-BERT in all tasks. The difference in average F1 between the models was at most two percent units.

IEEE Access made changes to BERT in order to improve its performance [27]. In their comparison one of the tasks, they performed multiclass text classification. The metric used to make the comparison was the difference in F1-score between different language representation models.

2.9 Summary

Language-agnostic BERT Sentence Embeddings (LaBSE) is a language model that can be fine-tuned to perform different types of Natural Language Processing (NLP) tasks on 109 languages, two of those being binary and multiclass text classification. To fine-tune LaBSE in Scala Apache Spark and Spark NLP can be used. To verify fine-tuned LaBSE is performing as expected, precision, recall and accuracy can be taken into consideration. If precision and recall are not valued differently the F1 value can be considered instead to get an overall picture of the text classifier’s performance.

3 Research methodology

This chapter describes how the research of this thesis was carried out and which tools and strategies have been used to do so. In Section 3.1 the research strategy is presented. In Section 3.2 all the research phases are described in detail. In Section 3.3 the research method is motivated. In Section 3.4 the research instruments are explained. In Section 3.5 the comparison criteria are presented and in Section 3.6 the validity threats are described.

3.1 Research Strategy

The area of Natural Language Processing (NLP) that the thesis deals with was unexplored by the authors of the thesis, therefore, an immense amount of research was required. Because the thesis investigates the performance of sentiment analyzer implemented in the programming language Scala it further complicates things because Scala is not a very popular language and implementation options are therefore limited. To address the somewhat limited options a strategy was chosen to accommodate for this.

The research strategy is presented in Figure 3.1. As seen in the picture the following components are part of the research strategy: (1) research phases, (2) research method, (3) research instruments, (4) comparison criteria and (5) validity threats. Every component has a section of their own where they are discussed.

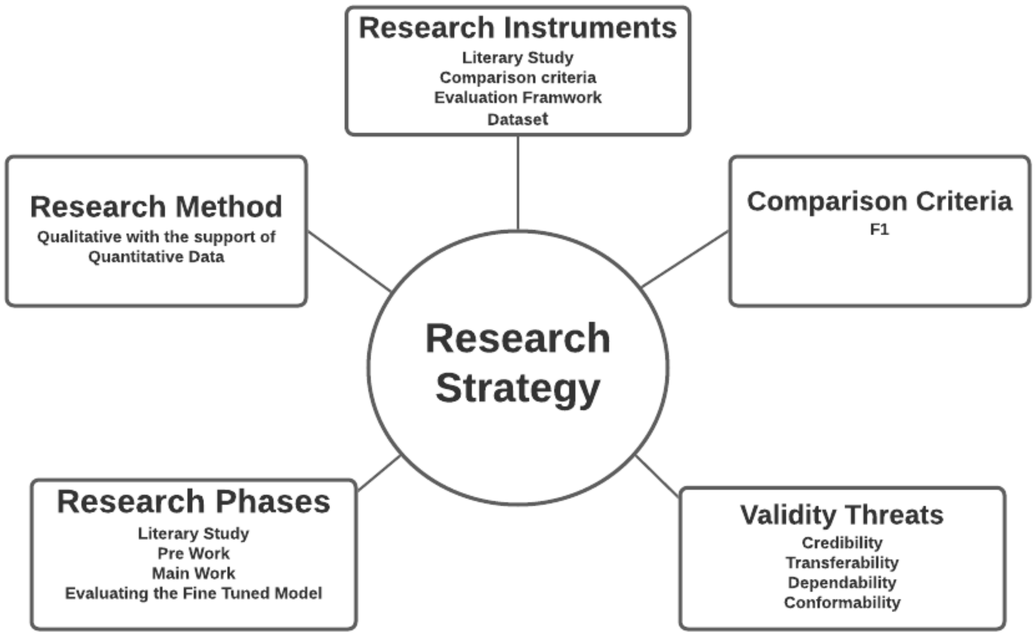

Figure 3.1. Research Strategy Model

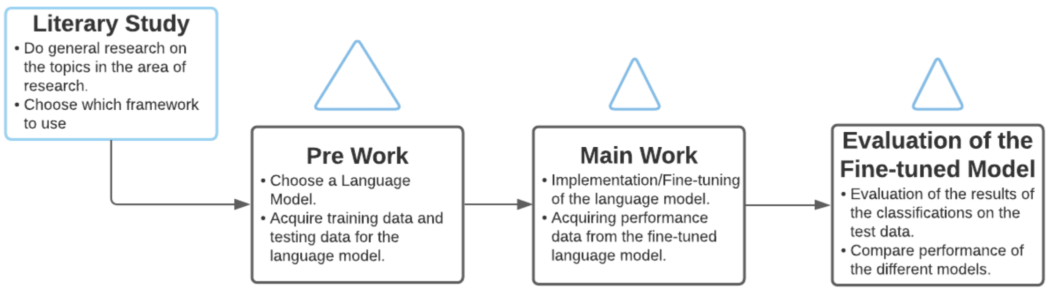
Literary Study
• Do general research on the topics in the area of research.
• Choose which framework to use
Pre Work
• Choose a Language Model.
• Acquire training data and testing data for the language model.
Main Work
• Implementation/Fine-tuning of the language model.
• Acquiring performance data from the fine-tuned language model.
Evaluation of the Fine-tuned Model
• Evaluation of the results of the classifications on the test data.
• Compare performance of the different models.

Figure 3.2. Overview of the research phases

3.2 Research Phases

This section describes the different research phases of the thesis, the phases are the major steps taken to go from no knowledge on the subject to a finished product. As can be seen in Figure 3.1 the four phases are Literary Study (1), Pre Work (2), Main Work (3) and Evaluating the fine-tuned Model (4).

1. The *Literary Study* sought to provide information in the field of text classification and provided the authors with information regarding a framework that would be able to perform Swedish text classification in Scala as well as relevant comparison criteria.
2. The *Pre Work* phase consisted of analyzing and manipulating the datasets that were acquired to test performance and to implement the text classifiers as well as picking a language model as a base before fine-tuning.
3. The *Main Work* phase was concerned with the implementation of the text classifiers using the language models and frameworks found during the literary study. During the Main Work phase, the text classifiers also performed classifications on the test data.
4. The Evaluation of the Fine-tuned Model phase had the purpose of evaluating the results of the classifications on the test data. The results of each implementation were recorded into a spreadsheet and the difference between implementations in Scala and Python could be compared.

As can be seen in Figure 3.2 the research phases are presented. It starts out with a (1) *Literary Study* followed by the (2) *Pre Work* phase followed by (3) *Main Work* phase and lastly the (4) *Evaluation* of the fine-tuned model phase. The literary study continues throughout every phase as can be indicated by the arrows. The size of the arrows corresponds to how much time was spent on the literary study of that phase.

3.2.1 Literary Study

The first step of the literary study was to determine which framework to use within the Scala programming language and get more familiar with the topics of Natural Language Processing (NLP), text classification and sentiment analysis. Finding articles and research papers on the topic of text classification and NLP was quite easy since the areas have been growing in the most recent years. Finding relevant articles within the realm of the Scala programming language on the topic of text classification and NLP was more limiting. The conclusion of the first iteration of the literary study concluded in the choice of Spark NLP, a framework that can be used for text classification in Scala.

Research was also conducted within the realm of the Python programming language, and it was found that Python is an established leader within the area of Natural Language Processing. In the Python case the options were plentiful, the chosen library was called Huggingface - transformers which is an NLP library [10]. Therefore, the choice was made to compare Python and Scala implementations in terms of performance in text classification.

The purpose of the literary study was to supply the authors with knowledge about the topic. As can be seen in Figure 3.2 the literary study was conducted multiple times throughout the project. The largest amount of time spent on literary study was before the *Pre Work* phase, but as one can see time is spent during every phase on literary study.

The literary study consisted of a data gathering phase and an analysis phase. During the data gathering phase the focus was on scientific articles regarding the subject and when enough data had been gathered analysis of the data began which consisted of an evaluation of its relevance toward answering the research question of the thesis. If the data was deemed relevant it was further processed and used. The used databases were Google Scholar, DiVa and Arxiv and because Scala was the already decided Programming Language to use, the initial key words used for finding an appropriate framework was, "scala nlp", "scala nlp library", "scala nlp framework", "scala natural language processing".

The database that yielded the most results when the previous stated keywords were used was Google Scholar. When the keywords "scala nlp library" were used 4 articles discussing Spark NLP on the first page were found and around 2000 results while using Google Scholar. When using Arxiv only 1-3 articles per keyword were found in total, and almost nothing was found on DiVa. There were mainly two frameworks that were considered during the initial phase of the literary study which were OpenNLP and Spark NLP, it quickly became evident that Spark NLP was the more appropriate option because of poor documentation of OpenNLP and because of the poor documentation it was very unclear whether one would be able to find a Swedish language model for fine-tuning.

3.2.2 Pre Work

In the *Pre Work* phase choice of which language model to use was made as well as acquiring training data and testing data for the language model.

The purpose of the *Pre Work* phase was to find a language model which could be used as a base for fine-tuning which is a way of specializing the model for a certain purpose, it is a very important step because the performance of the fine-tuned model is dependent on how much the language model knows about the language that it is going to be fine-tuned for. Acquiring training and test data is also a part of the *Pre Work* phase. The language model uses the training data for fine-tuning which is why this is also a very important step. The quality of the training data affects the finetuning of the language model, the testing data is used as evaluation of the model which is also important.

Finding options for a language model which supported Swedish in Scala was limiting. The only option was to use Language-agnostic BERT Sentence Embeddings (LaBSE) which is described in Chapter 2. In the case of Python, a language model specifically made for Swedish was used. The Swedish language model is based on BERT, but only trained on Swedish text.

Acquiring training data was done in two ways, the first solution was thanks to Reco AB, they provided a dataset with sentences manually labeled as positive, neutral or negative. The second way was also made possible thanks to Reco AB. Data of reviews combined with a grade, one being the lowest and five being the highest grade, were used to generate more negative labeled sentences. To do this an assumption was made, the assumption was that ⅕ star reviews with a low number of “sub-reviews” only contained negative classified sub-reviews which theoretically does not have to be the case but is probable. The new way of generating reviews was not as reliable but was better than having a low amount of training data.

3.2.3 Main Work

During the *Main Work* phase fine-tuning of the actual language models took place as well as acquiring performance data from the text classifiers.

The purpose of fine-tuning the language model was to perform sentiment analysis, that is to classify text as positive, negative or neutral. The fine-tuned models were then tasked to perform classifications on the test data, the results were then used for evaluation.

As stated in the *Pre Work* phase, LaBSE was the language model to be fine-tuned in Scala. To fine-tune LaBSE, research within the used framework was required, therefore Spark NLP documentation was studied, multiple fine-tuning approaches were tried and experimented with in order to yield the best result.

3.2.4 Evaluation of The Fine-tuned Model

During the evaluation phase of the research the results of each fine-tuning iteration was collected. The hyper-parameters used to achieve each result were also recorded. The data was collected by letting the implemented sentiment analyzers classify the test data. Which generated the data needed to compare performance between the implementation in Scala and Python.

Each time the fine-tuned models were tested, the evaluation libraries generated files containing the results of the classifications. The results were then inserted to a spreadsheet for comparisons. The spreadsheet was used for two reasons. The first reason being to record what hyper-parameters produced the most accurate classifier. The second reason being to compare results of the best performing classifiers between the Scala and Python implementation.

3.3 Research Method

In this section the used research methods are presented and motivated. The chosen research method was of comparative nature where the objective was to compare the performance of text classification in Scala with a text classification implementation in Python. The main methods used for evaluation were quantitative and the method used to study the needed background information for the thesis was qualitative in the form of a literary study.

3.3.1 Qualitative Research with Support of Quantitative Data

The world of natural language processing and text classification are completely unexplored by the authors therefore a large amount of research was required. Therefore, a qualitative research method was chosen for this purpose. A literary study was conducted with the purpose of finding a framework in Scala and a language model which supports Swedish. There was also a need to find information about what aspects to measure regarding the performance of the fine-tuned language model. By taking a qualitative approach insight into how to conduct the study was gained.

The thesis seeks to compare performance between an implementation of sentiment analysis in Scala compared to Python. Therefore, a quantitative approach in regard to analysis of the data was deemed appropriate. Because in a comparative study numerical data is needed to make such an analysis.

3.4 Research Instruments

The research instruments that were used in this study were a Literary Study, comparison criteria and evaluation frameworks for Scala and Python and a dataset used for training, validation and testing.

- *Literary Study*: Because the authors of the thesis had no previous knowledge of the natural language processing or text classification a literary study was therefore essential in order to gain the necessary knowledge in order to conduct the research.
- *Comparison criteria*: In order to measure the performance of the model a metric called F1 was derived by feeding the fine-tuned language model with test data. F1 is described in Section 2.
- *Evaluation Framework*: The evaluation framework used in the Scala implementation is retrieved from a machine learning library called MLLib [28]. The evaluation framework used in the Python implementation is retrieved from a machine learning library which is called scikit-learn [29].
- *Dataset*: To fine-tune the language model a dataset was constructed by manually classifying reviews as either positive, negative and neutral. Automatically generated reviews were also generated which is described more in the *Pre Work* phase.

3.5 Comparison Criteria

Because this study seeks to compare the performance of a sentiment analyzer implemented in Scala and Python comparison criteria needs to exist. The metric that will be used to compare performance is F1. F1 gives an overall indication of how accurate predictions are and were used in related studies comparing different language models. The focus of this study is the performance of the classifiers, there is not any emphasis put on other factors, such as time and processing resources. F1 is purely a measurement of classification accuracy and will indicate which implementation is most likely to predict the correct classification.

3.6 Validity Threats

This section discusses the validity threats of the thesis, the validity threats are: (1) Credibility, (2) Transferability, Dependability (3) and (4) Confirmability [30]. These terms have a corresponding term in quantitative research. Credibility corresponds to Internal Validity, transferability corresponds to External Validity, Dependability corresponds to Reliability and Conformability corresponds to objectivity. Validity Threats are presented in this section and discussed further in Chapter 5 during evaluation of the validity threats section.

3.6.1 Credibility

Credibility deals with the trustworthiness of the research findings. It seeks to determine how credible the findings are with reality. In order to make a claim about the quality of the findings the used frameworks for development of the model as well as the evaluation of the model needs to be credible.

3.6.2 Transferability

Transferability refers to the extent that the conclusions of a study might be applicable outside of the area in which the study was conducted. In order to have such a language model that does great at classifying reviews outside of the area where it was built, it needs to be great at classifying general reviews and not just reviews similar to what it was trained for.

3.6.3 Dependability

Dependability refers to the replicability of the research, that is if the same steps of the research were to be carried out, would that yield the same results? Because this thesis is of the comparative type in regards to the performance of text classification models the question becomes whether the used language models can be reproduced.

3.6.4 Conformability

Confirmability refers to what degree the researcher's objectivity might be a factor when conducting the research. Conformability refers to the researcher's comparable concern to objectivity. One threat is the correctness of the classified training and validation data which was manually classified by the researchers.

4 Design and Implementation

This chapter explains how the sentiment analyzer was implemented in Scala and how the dataset was handled. Because of the small amount of manually tested data, additional measures were taken to increase data. The measures taken are discussed in this chapter.

4.1 Phases of The Sentiment Analyzer

Process data consisted of reading the datasets and manipulating the text to make it more homogenous. The phase *Fine-tune LaBSE* was tasked to re-purpose LaBSE to a sentiment analyzer using the training and validation datasets. The *Evaluate quality* phase evaluated how successful the re-purposing of LaBSE was. The last section, Section 4.5, shows the analyzation process of the dataset and how it was modified.

All the above-mentioned phases are displayed in Figure 4.1. The figure also shows how the different parts of the dataset were distributed to the different phases.

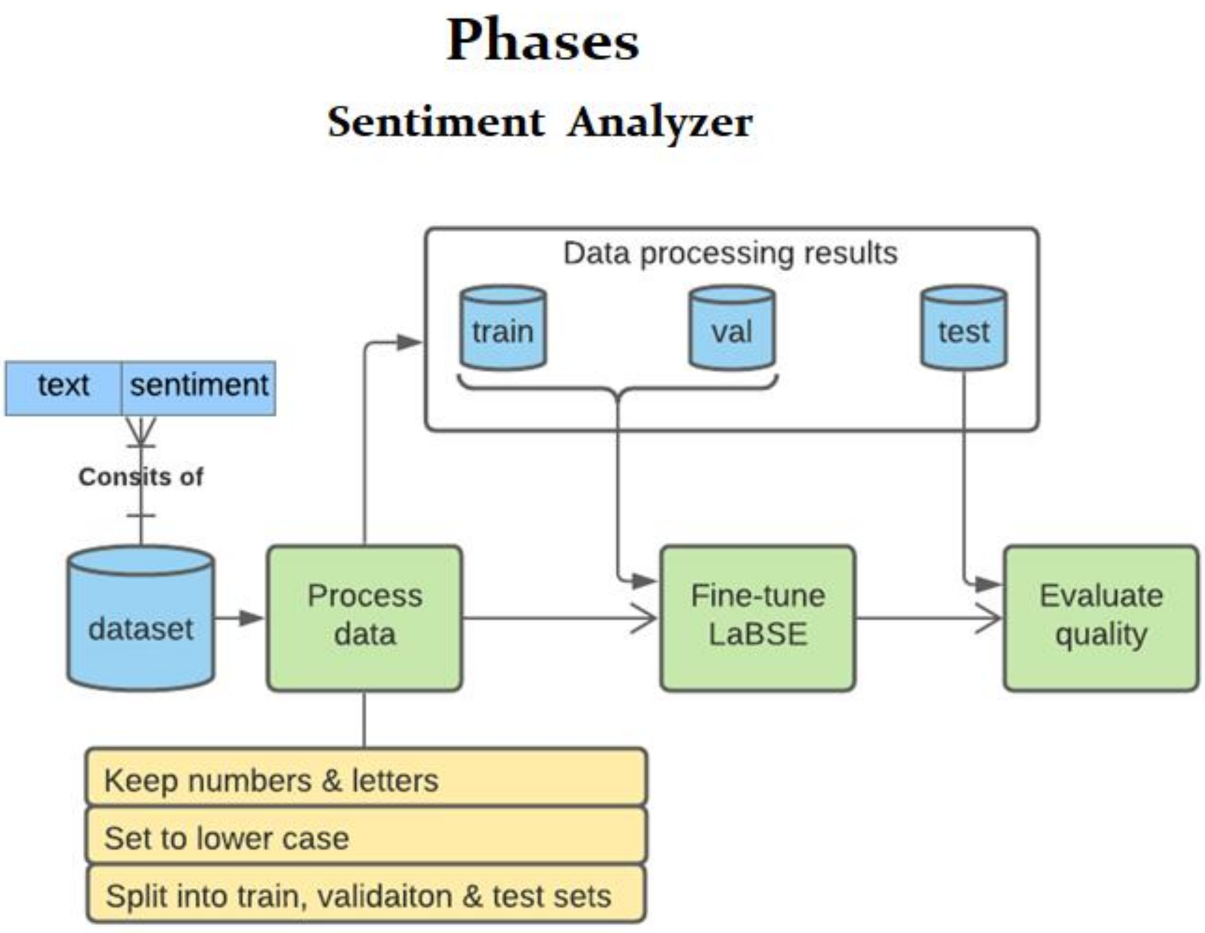

Figure 4.1. Phases of the sentiment analyzer

4.2 Processing data

To make the text more homogenous and to reduce variance in similar sentences, additional text processing was performed. The goal of the text processing was to remove non alphabetical characters except for numbers. This text processing was applied to the entire dataset; training, validation and test all included.

4.3 Fine-tune LaBSE

In this thesis work a framework was required that would make it possible in Scala to fine-tune Language-agnostic BERT Sentence Embeddings (LaBSE) to perform two types of text classifications. Both being Sentiment Analysis, however, one multiclass and the other binary in nature. The multiclass version predicted: positive, neutral and negative, while the binary predicted: positive and negative.

Following the literary study, it was decided to use Apache Spark and Spark NLP as frameworks. The reason for using Apache Spark and Spark NLP, was the quality of the documentation and ability to use LaBSE. The documentation provided by both frameworks were easy to follow and made it possible to decide whether the sought-after functionality was provided.

In Apache Spark when working with machine learning, Pipelines are used. The Pipeline is a representation of the learning algorithm and once a Pipeline's "fit" function has been called using the training data as the parameter value a Pipeline Model is created. A Pipeline Model represents a machine learning model and is in this case; LaBSE fine-tuned for sentiment analysis.

The model fine-tuned in this thesis work only used Pipeline stages provided by the Spark NLP library. For Spark NLP to transform a data frame the data frame needs to be adjusted to the Spark NLP ecosystem. As a starting point for Spark NLP a so-called "DocumentAssembler " is used. The DocumentAssembler takes the text data and represents it as a Spark SQL struct type instead of a String type. The rest of the Pipeline stages provided by Spark NLP need the data to be of this struct type to perform transformation and fit functions.

With the DocumentAssembler stage completed the next Pipeline stage is called "BertSentenceEmbeddings". There are multiple different types of BertSentenceEmbeddings provided by the library, in this thesis LaBSE is used.

The final Pipeline stage is not a transformer, but an estimator. An estimator is required for the Pipeline to be considered a real learning algorithm. When the fit function of an estimator is called a Pipeline Model is returned. In this case there are two different estimators used in two different Pipelines. Both estimators resulted in a model that did sentiment analysis, however, the training process differed between the two estimators.

The estimator "SentimentDLApproach" provided by Spark NLP, would only be trained using positive and negative labeled sentences. The output was the label with the highest probability of being correct. The transformer produced by the estimator also had the ability to predict neutral by entering a threshold value; if the probability was below the threshold value, neutral was used.

The second estimator used was also provided by Spark NLP and is called "ClassifierDLApproach" and produced a transformer able to predict labels used during training.

Figure 4.2 shows the workflow of the two different Pipelines to produce their corresponding Pipeline Models. Each rectangle represents a Pipeline stage, green is a transformer while yellow indicates an estimator, the blue tables are the fields of the data frames.

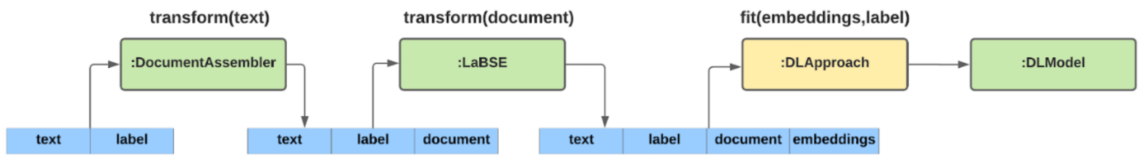

Figure 4.2. Workflow of pipeline

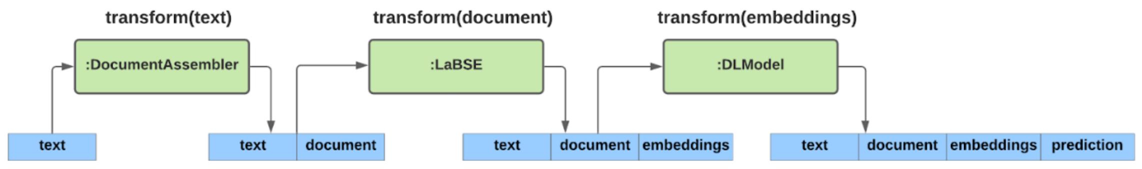

Figure 4.3. Workflow of the text classification pipeline model

Once the Pipeline had been fitted a Pipeline Model was created containing the same stages as the Pipeline, but instead of the two different estimators, two different transformers took their places. The two transformers being: SentimentDLModel and ClassifierDLModel. These Pipeline Models were then used to verify how well the predictions were performing.

Figure 4.3 illustrates the stages of the Pipeline Model. Green rectangles being transformers and blue rectangles representing the transformation of the data frames. Section 4.3.1 provides further details regarding the fine-tuning.

4.3.1 Training

Before conducting the analysis of the fine-tuned LaBSE model there were a few aspects that could be modified to affect the model's performance. During the phase of the unfitted Pipeline, as can be seen Figure 4.2, when creating the "DLApproach" Pipeline stages, hyper-parameters of the learning algorithm could be set. For the SentimentDLApproach stage the following parameters including hyper-parameters could be modified:

- Epochs
- Batch size
- Learning rate
- Dropout
- Threshold
- Validation split

These parameters were changed to create different fine-tuned versions of LaBSE. In the case of the ClassifierDLApproach there was no parameter to set a threshold.

The validation split is one of three splits that are made to the dataset used for the learning process. The first split is used to divide the data into training and testing datasets, the training dataset is split once more to have a validation dataset. For this thesis work, the first split was not changed, reason being it was deemed important that the test dataset would be large relative to the data used. However, the other datasets were modified to see how the performance would be affected.

4.4 Evaluate Quality

To verify how well the fine-tuned LaBSE model was performing, a set of metrics were analyzed, according to the criteria in Section 2.5. How often a predicted label was correct (precision) and how often relevant labels were predicted (recall). Both recall and precision were combined into one metric called F1-score. The F1-score was used as the measure of quality. The Apache Spark framework includes libraries to evaluate performance of classifiers. These libraries were used for the evaluation.

4.5 Dataset

During this research, a dataset was used containing 2.694 rows of sentences and their sentiments. The sentiments recorded were positive, neutral and negative. Figure 4.4 displays how the entire dataset is distributed, negative is 6.2%, neutral is 10.7% and positive is 83.1% of the entire dataset.

When fine-tuning Language-agnostic BERT Sentence Embeddings (LaBSE) it was done using two approaches: SentimentDLApproach and ClassifierDLApproach. With SentimentDLApproach neutral sentences were not required during fine-tuning. Instead, a threshold value was used, if the output was less than the threshold value it was translated to the neutral label. However, as can be seen in Figure 4.4, the distribution of labels was very uneven. To split the data into a train, validation and test dataset while having evenly distributed labels. It would lead to a very small amount of data being used. A small dataset for testing makes it difficult to reliably evaluate the performance of the fine-tuned LaBSE.

Besides the manually labeled data, there also existed data containing reviews and their grades. To make up for the small number of negative reviews, all reviews with the worst grade and had less than a hundred characters were split into sentences and labeled negative. This resulted in an additional 1.063 rows of negative sentences. The new rows of negative sentences were combined with 1.043 rows of positive sentences from the manually verified data. The combined data was then used as the dataset for the SentimentDLApproach. Figure 4.5 shows the distribution of the dataset used for fine-tuning LaBSE using SentimentDLApproach. Negative is 50.5% and positive is 49.5% of the dataset.

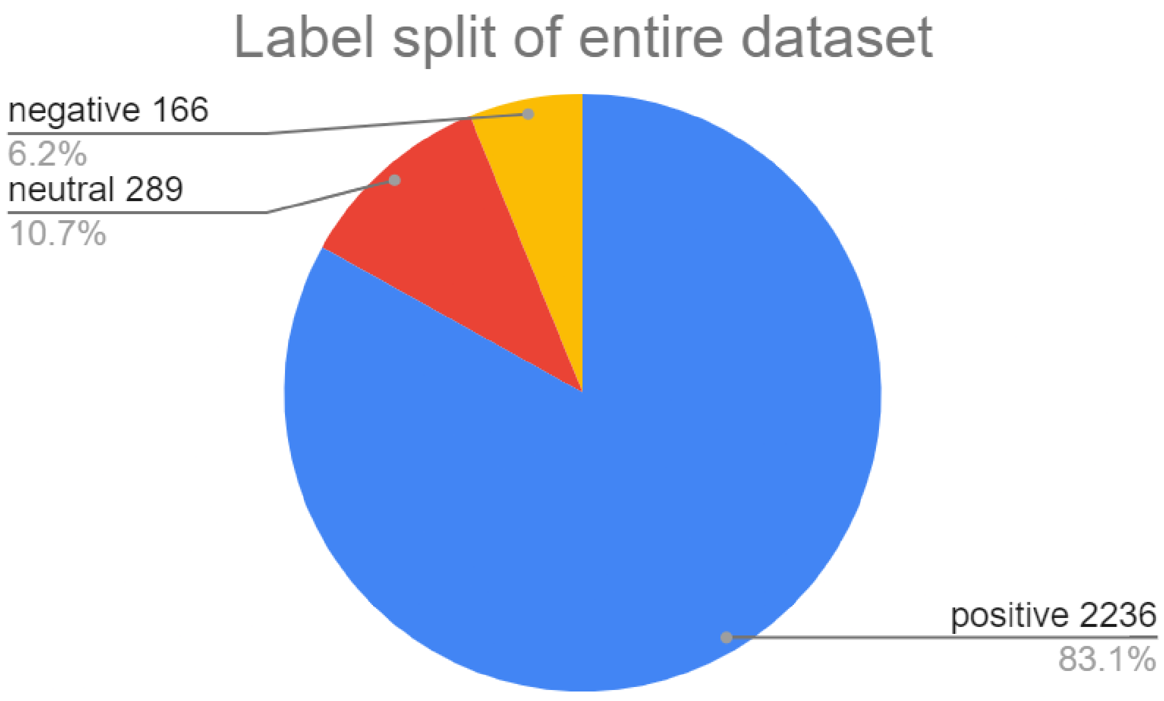

Figure 4.4. Label split of the entire dataset

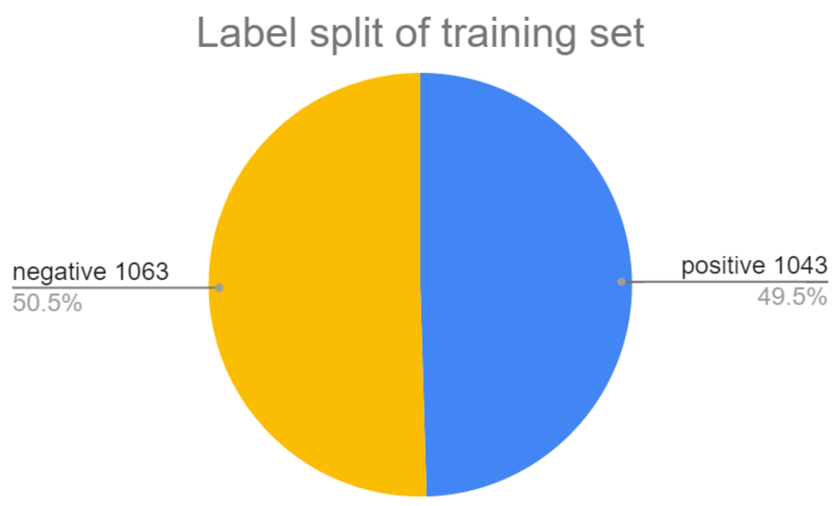

Figure 4.5. **Label split of the training set excluding neutral**

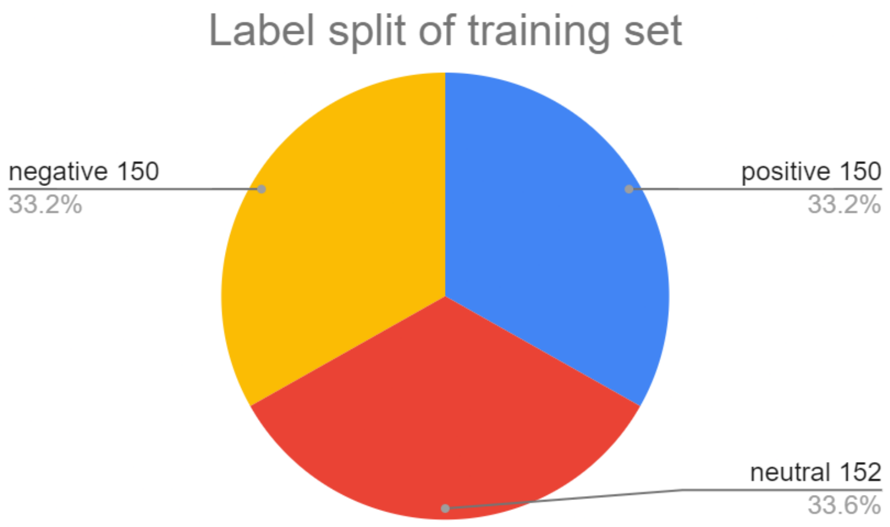

Figure 4.6. **Label split of the training set including neutral**

To use the ClassifierDLApproach for fine-tuning LaBSE, a dataset with all three labels is required, however, because of the unevenly distributed data, the amount of positive and negative sentences was limited to 150. The negative sentences did not come from the manually verified sentences. Figure 4.6 shows the distribution of the dataset used for fine-tuning LaBSE using ClassifierDLApproach. Negative is 33.2%, positive is 33.2% and neutral is 33.6% of the dataset.

To evaluate the performance of the fine-tuned LaBSE, two different test datasets were used. Both datasets contained the same positive and negative sentences, but one of the datasets also contained neutral sentences. Both test datasets used all negative sentences that had been manually verified. Because the negative sentences used for training had not been verified, it was deemed necessary to use the small number of negative reviews that had been verified for testing. See Figures 4.7 and 4.8 for the distributions.

Figure 4.7 shows the distribution of the dataset used for evaluating LaBSE when predicting if a sentence is positive or negative. Negative is 51.4% and positive is 48.6% of the dataset.

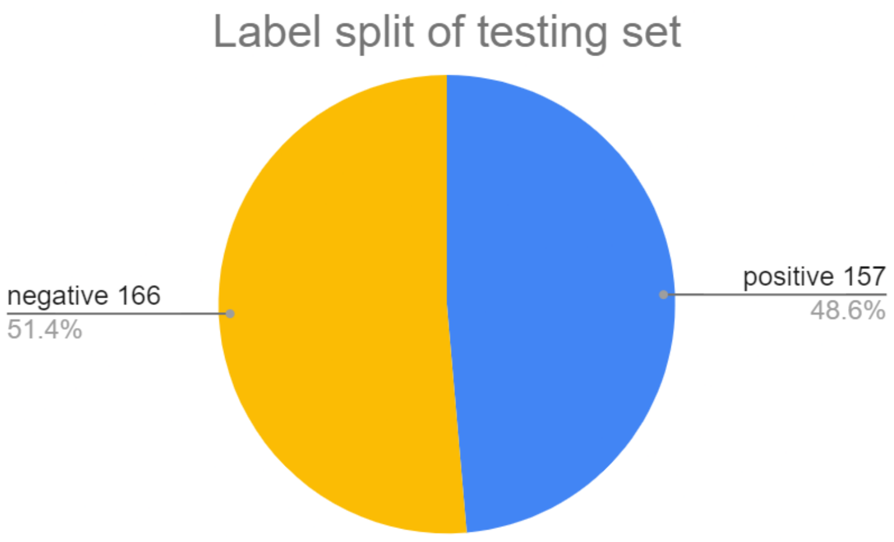

Figure 4.7. Label split of the testing set excluding neutral

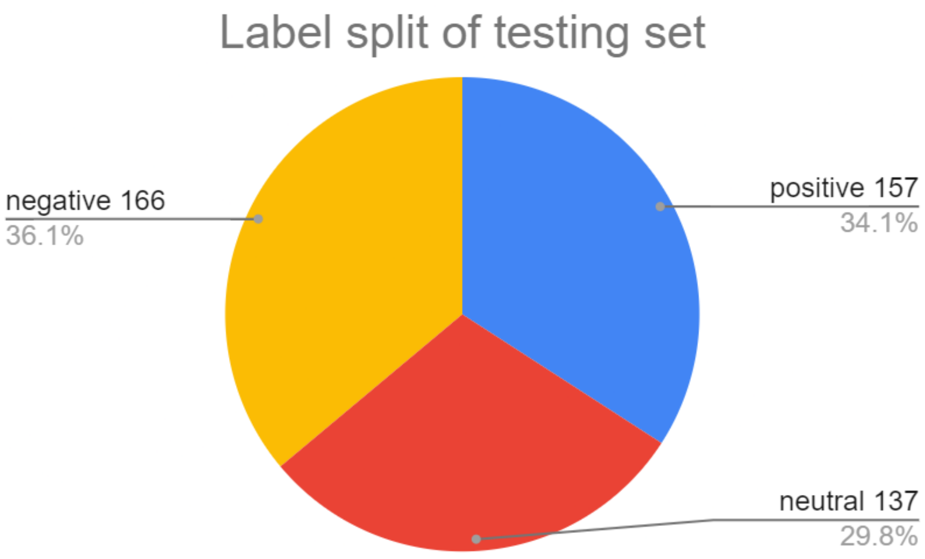

Figure 4.8. Label split of the testing set including neutral

Figure 4.8 shows the distribution of the dataset used for evaluating LaBSE when predicting if a sentence is positive, neutral or negative. Negative is 36.1%, neutral is 29.8% and positive is 34.1% of the dataset.

5 Results and Analysis

The following chapter presents how well the different sentiment analyzers implemented by the authors in Scala and Python performed. The level of validity and reliability of the results is also discussed. Section 5.1 – 5.2 visualize the performance results of the text classifiers implemented in Scala using two different approaches provided by the framework used for fine-tuning. Section 5.3 show the results of the text classifiers developed in Python. Section 5.4 gives a summary of the best results achieved in Scala and Python. Section 5.5 discusses the validity of the results. Finally, Section 5.6 contains a discussion of the results.

5.1 Results using Spark NLP's SentimentDLApproach

The results presented are all based on LaBSE fine-tuned using Spark NLP's SentimentDLApproach. The main hyper-parameter altered during each trial was the number of epochs. The size of the validation set was also altered.

Table 5-1 shows the result of fine-tuned LaBSE using SentimentDLApproach when classifying positive and negative sentences. The hyper-parameters used were the following: batch size: 64, epochs: 24, learning rate: 0,0005, dropout: 0.2 and a validation set split of 0.2.

Figure 5.1 plots the f1 measure of the fine-tuned LaBSE model based on the hyper-parameters and validation split used when classifying sentences as either positive or negative. The trendline is based on a polynomial of the second degree. The names of the lines, HP(tX, bY, lrZ, dA, vB), stands for: HP(threshold, batch size, learning rate, dropout, validation split).

	accuracy	**weighted F1**
overview	0.75	0.744
		F1
positive	-	0.778
negative	-	0.716

Table 5-1. Evaluation summary of best performing hyper-parameters with SentimentDLApproach when tested on binary sentiment dataset

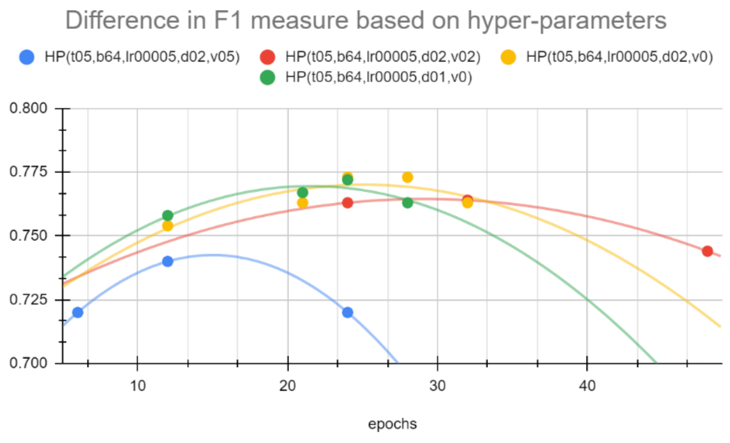

Figure 5.1. Plot of difference in F1 measure for binary sentiment analysis based on validation split and hyper-parameters using SentimentDLApproach

	accuracy	weighted F1
overview	0.5	0.471
		F1
positive	-	0.627
neutral	-	0.171
negative	-	0.572

Table 5-2. Evaluation summary of best performing hyper-parameters with SentimentDLApproach when tested on multiclass sentiment dataset

5.1.1 Including Neutral Sentences in the Test Set

Using the best performing hyper-parameters from table 5-1, the evaluation was changed to also include sentences with the neutral label. The initial threshold was set to 0.5 and 0.6, but the classifier did not start predicting “neutral” until the threshold was set to 0.7.

Table 5-2 shows the result of fine-tuned LaBSE using SentimentDLApproach when classifying positive, neutral and negative sentences. The hyper-parameters used were the following: batch size: 64, epochs: 24, learning rate: 0,0005, dropout: 0.2, threshold: 0.7 and a validation set split of 0.

5.2 Results using Spark NLP’s ClassifierDLApproach

The same method was used to evaluate LaBSE as mentioned in the section above when fine-tuning with Spark NLP’s ClassfierDLApproach. For this approach it was required to do training with sentences labeled as neutral, however, there were also tests conducted to see how well the fine-tuned LaBSE model would perform on only negative and positive sentences, hence the dataset with a larger number of negative sentences could be used.

Table 5-3 shows the result of fine-tuned LaBSE using ClassifierDLApproach when classifying positive and negative sentences. The hyper-parameters used were the following: batch size: 64, epochs: 64, learning rate: 0,0005, dropout: 0.2 and a validation set split of 0.

Figure 5.2 plots the f1 measure of the fine-tuned LaBSE model based on the hyper-parameters and validation split used when classifying sentences as either positive or negative. The trendline is based on a polynomial of the second degree. The names of the lines, HP(bX, lrY, dZ, vA), stand for: HP(batch size, learning rate, dropout, validation split).

	accuracy	weighted F1
overview	0.783	0.779
		F1
positive	-	0.808
negative	-	0.752

Table 5-3. Evaluation summary of best performing hyper-parameters fine-tuned for binary sentiment analysis using ClassifierDLApproach

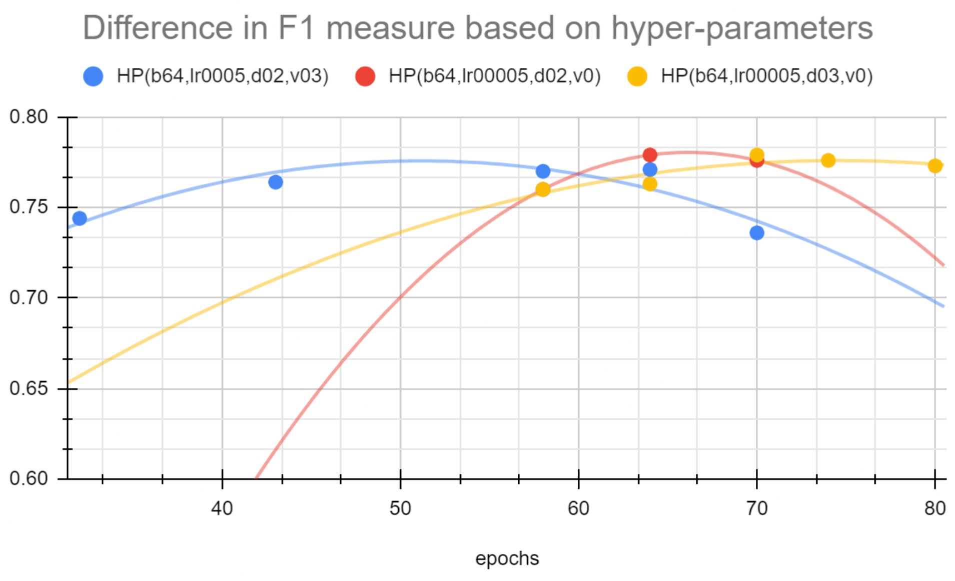

Figure 5.2. **Plot of difference in F1 measure for binary sentiment analysis based on validation split and hyper-parameters using ClassifierDLApproach**

5.2.1 Including Neutral Sentences in the Test and Training Sets

When training using the ClassifierDLApproch to be able to predict positive, neutral and negative, all three types of sentences were needed during training.

Table 5-4 shows the result of fine-tuned LaBSE using ClassifierDLApproach when classifying positive, neutral and negative sentences. The hyper-parameters used were the following: batch size: 64, epochs: 2048, learning rate: 0,0005, dropout: 0.2 and a validation set split of 0.

Figure 5.3 plots the f1 measure of the fine-tuned LaBSE model based on the hyper-parameters and validation split used when classifying sentences as either positive, neutral or negative. The trendline is based on a polynomial of the second degree. The names of the lines, HP(bX, lrY, dZ, vA), stand for: HP(batch size, learning rate, dropout, validation split).

	accuracy	**weighted F1**
overview	0.652	0.647
		F1
positive	-	0.71
neutral	-	0.665
negative	-	0.571

Table 5-4. **Evaluation summary of best performing hyper-parameters fine-tuned for multiclass sentiment analysis using ClassifierDLApproach**

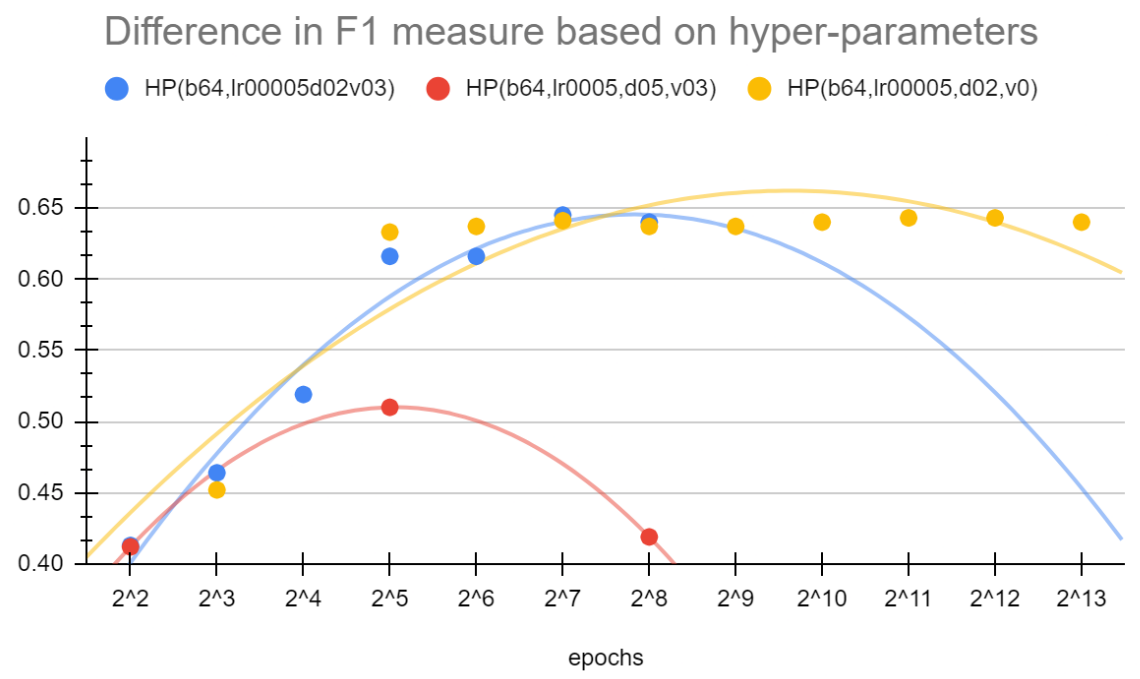

Figure 5.3. **Plot of difference in F1 measure for multiclass sentiment analysis based on validation split and hyper-parameters using ClassifierDLApproach Reliability Analysis**

5.3 Python Results

The following results are based on BERT models pre-trained on a Swedish dataset.

5.3.1 Binary Sentiment Analysis

Table 5-5 is the evaluation result from a Swedish BERT model fine-tuned for binary sentiment analysis. Table 5-6 is an already fine-tuned Swedish BERT model for binary sentiment analysis evaluated with the dataset used during this thesis work.

Table 5-5 shows the evaluation results of Swedish BERT fine-tuned in Python to do binary sentiment analysis.

Table 5-6 shows the evaluation of an already fine-tuned Swedish BERT model implemented for evaluation testing of the dataset used during the thesis.

	accuracy	weighted F1
overview	0.83	0.83
		F1
negative	-	0.83
positive	-	0.82

Table 5-5. **Evaluation summary of binary sentiment analysis fine-tuned in Python**

	accuracy	weighted F1
overview	0.85	0.85
		F1
negative	-	0.83
positive	-	0.87

Table 5-6. Evaluation summary of fine-tuned binary sentiment analysis

	accuracy	weighted F1
overview	0.79	0.78
		F1
negative	-	0.66
positive	-	0.84
neutral	-	0.84

Table 5-7. Evaluation summary of multiclass sentiment analysis fine-tuned in Python

5.3.2 Multiclass Sentiment Analysis

Table 5-7 shows the evaluation result of a Swedish BERT model fine-tuned for multiclass sentiment analysis implemented in Python.

5.4 Result Summary

Table 5-8 shows the best performing F1 result of each implementation, the bolded number is the highest F1. SDLA stands for SentimentDLApproach and CDLA stands for ClassifierDLApproach. KB-BERT, is the BERT model trained on a Swedish dataset, and SWE-SENT-BERT is the Swedish BERT model already fine-tuned to perform binary sentiment analysis.

	binary	multiclass
LaBSE (SDLA)	0.74	0.47
LaBSE (CDLA)	0.78	0.65
KB-BERT	0.83	**0.78**
SWE-SENT-BERT	**0.85**	-

Table 5-8. Evaluation summary of all implementations

5.5 Validity Analysis

In this section the validity threats are further discussed and how they were dealt with during the thesis. The discussed validity threats are (1) credibility, (2) transferability, (3) dependability and (4) conformability.

5.5.1 Credibility

To make such a claim about the quality of the findings, established evaluation frameworks were used which are listed under Section 1.4. To evaluate the model, test data is needed and in order to ensure that the true quality of the model, test data which the model has not seen before were chosen and used for acquiring performance data.

The training data used in sentiment training of the model consists of negative, neutral and positive classifications of sentences, one threat that might affect the quality of the model is that when constructing the training data, the authors of the study noticed that most of the classified training data had a positive sentiment. Which would mean that when training the model unbalanced results could occur. The solution to this problem was to introduce a way to generate more negative reviews. In order to do this an assumption was made, the assumption was that ⅕ star reviews with a low number of “sub-reviews” only contained negative classified sub-reviews which theoretically does not have to be the case but is probable. This means that when training the model “sub-reviews” which are not classified right could exist, for example classified negative reviews which should be neutral or even positive. This would mean that the model might use training data where the examples they are going to use for training are classified wrong which would affect the fine-tuned model.

Training and validation sets consisted of artificial data, by categorizing reviews based on an assumption. However, the test data only consisted of data that had been manually verified.

5.5.2 Transferability

In order for the model to be applicable to classify reviews outside of the limited research area it needs to be trained in such a way that the model is good at classifying general reviews and not only reviews that are similar to the ones that it has trained on. In order to minimize this threat the reviews in which model was tested on were reviews the model had never seen before, by doing this the researchers could be sure that the results of the model were credible as well as producing a model that is good at classifying general reviews which promotes transferability.

5.5.3 Dependability

Given that this thesis seeks to evaluate two implementations of text classification and then compare them, the replicability of the research is highly dependent on the ability to build a model producing the same results. Given that the implementations that were used are built from frameworks which arguably have high reliability, the dependability of the research should therefore be high.

5.5.4 Conformability

The largest confirmability threat in the research is connected to the training and validation data used while fine-tuning the language model. The training and validation data by manually classifying reviews as either positive, negative or neutral. The process of manually classifying the reviews is a confirmability threat because the interpretation of the reviews by the person classifying affects the outcome of the trained model. The training data is used by the language model for fine-tuning. This was combatted by the nature of how the text was classified. The company providing the classifications had given the people responsible for the classifications guidelines to follow.

5.6 Discussion

The results of this thesis work indicate that given the dataset and LaBSE in combination with Spark NLP performs worse in all tasks compared to the Swedish BERT model fine-tuned in Python as well as the available fine-tuned binary sentiment analyzer.

When fine-tuning LaBSE and Swedish BERT to do binary sentiment analysis the difference in results were not as significant as the difference when performing multiclass sentiment analysis. This may be a result of the fact that it is more probable to get a classification right when there are only two possible labels.

In Scala there are very limited options in terms of available frameworks and libraries for implementing NLP models while the availability of frameworks and libraries in Python are much greater. Many of the frameworks in Python are also used in the different areas of research in machine learning and text classification. This means that the probability of Python having a framework or library that performs better than Scala is high.

One interesting observation is when comparing the difference in F1 score between the Scala implementations of Binary and Multiclass classification with the difference of performance in the Python implementation of Binary and Multiclass classifications. We observe that in the Scala case the F1 score went from 0.779 in the Binary case to 0.647 in the Multiclass case. We observe that in the Python case the F1 score went from 0.83 in the Binary case to 0.78 in the Multiclass case. The difference between the Scala implementations was 0.132 and in the Python case 0.05. One hypothesis is that the quality of the dataset would explain the big difference in the F1 score between the Binary and Multiclass implementation in the Scala case. Because a situation where the data used for training is not representing all the possible outcomes. Due to not having trained on similar data. The classifications on the test data would be poor. However, because the python implementation does not have the same drop in quality this hypothesis can be disregarded.

What can be seen in Figure 5.1, 5.2 and 5.3 is that LaBSE performs better when reducing the amount of data used for the validation split. Which may indicate that given a larger dataset a higher F1 rating could be achieved.

6 Conclusions and Future Work

Scala is one of the programming languages that have had new frameworks developed to work with sentiment analysis. However, in comparison to Python, it has less available resources. There are even fewer resources regarding sentiment analysis on a less common language such as Swedish. The problem is no one has compared a sentiment analyzer for Swedish text implemented using Scala and compared it to Python. The purpose of this thesis is to compare the quality of a sentiment analyzer implemented in Scala to Python. The goal of this thesis is to increase the knowledge regarding the state of text classification for less common natural languages in Scala.

To conduct the study, a qualitative approach with support of quantitative data was used. Two kinds of sentiment analyzers were implemented in Scala and Python. The first classified text as either positive or negative (binary sentiment analysis), the second sentiment analyzer would also classify text as neutral (multiclass sentiment analysis). To perform the comparative study, the implemented analyzers would perform classification on text with known sentiments. The quality of the classifications was measured using their F1-score.

The results showed that Python performed better for both tasks. In the binary task there was not as large of a difference between the two implementations. The resources from Python were more specialized for Swedish and did not seem to be as affected by the small dataset used as the resources in Scala. Scala had a F1-score of 0.78 for binary sentiment analysis and 0.65 for multiclass sentiment analysis. Python had a F1-score of 0.83 for binary sentiment analysis and 0.78 for multiclass sentiment analysis.

6.1 Evaluation

The results that were achieved with Scala, Apache Spark and Spark NLP using the given dataset were poorer compared to the results from the language model fine-tuned in Python. For the binary sentiment analysis, the difference was not as significant as the multiclass sentiment analysis.

One apparent factor regarding the binary sentiment analysis implemented in Scala, was the increase in performance when the rate of data used for validation was reduced.

Fine-tuning in Python required less resources regarding the number of epochs. For the best performing multiclass sentiment analysis using LaBSE, 2048 epochs were used, while the implementation in Python only required two.

6.2 Future work

In this thesis a closer look at public resources for NLP were researched. There are other languages and frameworks that also have increased their options for machine learning over the years. These areas would also be interesting to research and present the current performance that can be achieved.

Sentiment analysis is one area of text classification, there are other areas, such as finding subjects of texts where LaBSE might prove more efficient than sentiment analysis. Text classification is only a subcategory of NLP, there are multiple other applications of NLP that can be studied, how do the opportunities look to implement them using LaBSE, etc.

Changes are made very rapidly in this area of study, during the time of this thesis work, Spark NLP released new language models. The released models also make it possible to perform NLP tasks on multiple languages. It would be interesting to make a new comparison using the newly released models to see if there have been any significant improvements.

In this study it was concluded that the main reason for the worse performance in Scala was because the language models in Python were better optimized to work with Swedish text. However, it might be the case that the framework used in Scala played a large role. Another comparative study that could be made is the implementation of a language model available to use in both Spark NLP and a framework in Python. The results of both implementations could be compared to see the effect of the frameworks.

6.3 Conclusion

Given the increase in performance when the dataset was increased, there is reason to believe that tasks with similar complexity would benefit from having more data than used in this study.

When implementing the different text classifiers in Python and Scala there were small differences regarding the programming that was required. The major difference between the implementations are the language models available in each programming language. Because of Pythons popularity in the field of Natural Language Processing (NLP) and text classification it had more available language models that worked with Swedish text. Most crucially it had a language model specifically developed to create word embeddings of Swedish text. During the literary study one of the papers that were read, compared NLP tasks performed using a multilingual language model and a specialized language model. The results of the paper showed that the specialized language model was better in every regard at the NLP task for the language used. This was also the case in this study.

At this moment in time the lack of support of a language model that works well on Swedish text is holding the performance of text classification back in Scala compared to the options in Python.

References

[1] Francesca Cassidy, ‘A Day in Data’, *Raconteur*. [Online]. Available: https://www.raconteur.net/infographics/a-day-in-data/. [Accessed: 21-Jun-2021]

[2] Steven Bird, Ewan Klein, and Edward Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. 2009.

[3] Sumant Sharma and Amit Arora, ‘Adaptive approach for spam detection’, *Int. J. Comput. Sci. Issues IJCSI*, vol. 10, no. 4, p. 23, 2013.

[4] Wei Xue and Tao Li, ‘Aspect Based Sentiment Analysis with Gated Convolutional Networks’, *ArXiv180507043 Cs*, May 2018 [Online]. Available: http://arxiv.org/abs/1805.07043. [Accessed: 15-Jun-2021]

[5] Martin Malmsten, Love Börjeson, and Chris Haffenden, ‘Playing with Words at the National Library of Sweden -- Making a Swedish BERT’, *ArXiv200701658 Cs*, Jul. 2020 [Online]. Available: http://arxiv.org/abs/2007.01658. [Accessed: 04-Jun-2021]

[6] Matthias Bruckner, Marcelo LaFleur, and Ingo Pitterle, ‘The impact of the technological revolution on labour markets and income distribution’, *Front. Issues*, p. 51, Jul. 2017.

[7] Emma Strubell, Ananya Ganesh, and Andrew McCallum, ‘Energy and Policy Considerations for Deep Learning in NLP’, *ArXiv190602243 Cs*, Jun. 2019 [Online]. Available: http://arxiv.org/abs/1906.02243. [Accessed: 14-Jun-2021]

[8] ‘Natural language processing’, *Wikipedia*. 21-May-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Natural_language_processing&oldid=1024275775 . [Accessed: 03-Jun-2021]

[9] ‘Stack Overflow Developer Survey 2020’, *Stack Overflow*. [Online]. Available: https://insights.stackoverflow.com/survey/2020/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2020. [Accessed: 17-Jun-2021]

[10] ‘Hugging Face – The AI community building the future.’ [Online]. Available: https://huggingface.co/. [Accessed: 15-Jun-2021]

[11] ‘TensorFlow’. [Online]. Available: https://www.tensorflow.org/. [Accessed: 17-Jun-2021]

[12] ‘Quick Start - Spark NLP’. [Online]. Available: https://nlp.johnsnowlabs.com/docs/en/quickstart. [Accessed: 03-Jun-2021]

[13] ‘Apache Spark™ - Unified Analytics Engine for Big Data’. [Online]. Available: https://spark.apache.org/. [Accessed: 03-Jun-2021]

[14] ‘Spark SQL and DataFrames - Spark 3.1.2 Documentation’. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes. [Accessed: 03-Jun-2021]

[15] ‘ML Pipelines - Spark 3.1.2 Documentation’. [Online]. Available: https://spark.apache.org/docs/latest/ml-pipeline.html#transformers. [Accessed: 03-Jun-2021]

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

[17] ‘Linguistics’, *Wikipedia*. 13-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Linguistics&oldid=1028432773. [Accessed: 15-Jun-2021]

[18] J.R. Firth, *A Synopsis of Linguistic Theory, 1930-1955*. 1957 [Online]. Available: https://books.google.se/books?id=T8LDtgAACAAJ

[19] Jure Zupan, ‘Introduction to Artificial Neural Network (ANN) Methods: What They Are and How to Use Them’, *Acta Chim. Slov.*, vol. 41, Jan. 1994.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, ‘BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding’, *ArXiv181004805 Cs*, May 2019 [Online]. Available: http://arxiv.org/abs/1810.04805. [Accessed: 03-Jun-2021]

[21] Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang, ‘Language-agnostic BERT Sentence Embedding’, *ArXiv200701852 Cs*, Jul. 2020 [Online]. Available: http://arxiv.org/abs/2007.01852. [Accessed: 01-Jun-2021]

[22] Emilio Soria Olivas, Jose David Martin Guerrero, Marcelino Martinez Sober, Jose Rafael Magdalena Benedito, and Antonio Jose Serrano Lopez, *Handbook Of Research On Machine*

Learning Applications and Trends: Algorithms, Methods and Techniques, Illustrated edition. Hershey, PA: Information Science Reference, 2009, ISBN: 978-1-60566-766-9.

[23]'Amazon Machine Learning - Developer Guide', *Amaz. Web Serv.*, vol. 2016, p. 146.

[24]Jason Brownlee, 'Difference Between a Batch and an Epoch in a Neural Network', *Machine Learning Mastery*. 19-Jul-2018 [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. [Accessed: 03-Jun-2021]

[25]Jason Brownlee, 'How to Configure the Learning Rate When Training Deep Learning Neural Networks', *Machine Learning Mastery*. 22-Jan-2019 [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/. [Accessed: 03-Jun-2021]

[26]Jason Brownlee, 'Dropout Regularization in Deep Learning Models With Keras', *Machine Learning Mastery*. 19-Jun-2016 [Online]. Available: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/. [Accessed: 03-Jun-2021]

[27]Shanshan Yu, Jindian Su, and Da Luo, 'Improving BERT-Based Text Classification With Auxiliary Sentence and Domain Knowledge', *IEEE Access*, vol. 7, pp. 176600–176612, 2019. DOI: 10.1109/ACCESS.2019.2953990

[28]'Evaluation Metrics - RDD-based API - Spark 3.1.1 Documentation'. [Online]. Available: https://spark.apache.org/docs/3.1.1/mllib-evaluation-metrics.html. [Accessed: 07-Jun-2021]

[29]'sklearn.metrics.classification_report — scikit-learn 0.24.2 documentation'. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. [Accessed: 07-Jun-2021]

[30]Andrew K. Shenton, 'Strategies for ensuring trustworthiness in qualitative research projects', *Educ. Inf.*, vol. 22, no. 2, pp. 63–75, 2004. DOI: 10.3233/EFI-2004-22201

[31]'KB/bert-base-swedish-cased · Hugging Face'. [Online]. Available: https://huggingface.co/KB/bert-base-swedish-cased. [Accessed: 07-Jun-2021]

[32]'Hugging Face: State-of-the-Art Natural Language Processing in ten lines of TensorFlow 2.0'. [Online]. Available: https://blog.tensorflow.org/2019/11/hugging-face-state-of-art-natural.html. [Accessed: 15-Sep-2021]

[33]'BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick'. [Online]. Available: http://mccormickml.com/2019/07/22/BERT-fine-tuning/#41-bertforsequenceclassification. [Accessed: 15-Sep-2021]

[34]'Fine-tuning with custom datasets'. [Online]. Available: https://huggingface.co/transformers/custom_datasets.html. [Accessed: 15-Sep-2021]

[35]'marma/bert-base-swedish-cased-sentiment · Hugging Face'. [Online]. Available: https://huggingface.co/marma/bert-base-swedish-cased-sentiment. [Accessed: 07-Jun-2021]

[36]'Who is using scikit-learn? — scikit-learn 0.24.2 documentation'. [Online]. Available: https://scikit-learn.org/stable/testimonials/testimonials.html. [Accessed: 15-Sep-2021]

Appendix A: Design and Implementation of Python Solution

The following appendix explains how the python classifier was implemented and which libraries were used for implementation and evaluation.

The Model and chosen NLP library

To develop a model designed to perform a specific task, and in this case the task is to develop a model which can classify text as either neutral, negative or positive one of the most common strategies is to take a model which has already been taught the language to be used and then use that model as a base to specialize it for the specific task. The specialization is called fine-tuning and is the process of teaching a model to perform a new task. The chosen language model which was used as a base before fine-tuning is a BERT based model designed by The National Library of Sweden which was trained on 15-20 GB of Swedish text [31].

Huggingface

Huggingface is an open-source natural language processing focused startup. One of the most popular libraries within Huggingface is called Transformers and is a python-based library that provides support for loading existing pretrained models and fine-tuning them for a specific task [32]. The chosen nlp-library therefore became transformers because of how easily the pre-trained Swedish BERT model could be loaded with only one line of code. The transformers library provides a set of types of classes that can be used for a variety of NLP tasks. These classes are all built on top of a trained BERT model and have different top layers and output types designed to accommodate their specific NLP task. Therefore, the Swedish BERT model could be used within one of these classes and in this implementation, it would be a class used for text classification. The one used in this implementation is called BertForSequenceClassification and can be used for sentence classification [33].

Dataset

The dataset consists of training, validation and a test dataset, before fine-tuning the model the training, validation and test data is loaded and preprocessed. The training data is the data that the model is going to be using for learning, the validation data is going to be used after every iteration of training to see how well the model is performing so that the model can be updated accordingly. The test dataset is used when the training is completely done and is used for evaluation of how well the model performs.

In order to fine-tune the Swedish BERT language model, training data which the model can use to learn from is needed and the training data needs to be formatted in the right way. Therefore, preprocessing of the dataset is performed. The preprocessing consists of removing unnecessary characters like punctuation, comma, exclamation point and so on as well as making the text lowercase, removing URLs and formatting it to a desirable format which is to add whitespace so that every sample has the same size. All samples need to have the same size when using BERT as well as not being more than 512 tokens [33]. The next step of the preprocessing is to tokenize the text which is the process of splitting a sequence of characters into tokens, it can for example be splitting text into words or words into single characters. In the text into words example the words would then be called tokens to the text, and in the word into character example the characters would be the tokens to the word. As previously mentioned, the dataset consists of training, validation and test examples, the preprocessing is done for all of these sets of sentences. The tokenizer can be loaded like this:

tokenizer = AutoTokenizer.from_pretrained("KB/bert-base-swedish-cased")

And tokenization is then done for the whole dataset like so:

train_dataset = Review(train_encodings, train_labels)

```
val_dataset = Review(val_encodings, val_labels)

test_dataset = Review(test_encodings, test_labels)
```

When the dataset has been preprocessed it needs to be turned into a Dataset Object, a dataset object just stores the samples and their corresponding labels, this is to make it easier to fine-tune the model because many of the fine-tuning techniques use Dataset Objects. The code below shows the constructor to the Review class takes encodings which are the tokenized reviews and all labels which are either “Positive”, “Negative” or “Neutral”.

```
class Review(torch.utils.data.Dataset):

  def __init__(self, encodings, labels):

    self.encodings = encodings

    self.labels = labels

  def __getitem__(self, idx):

    item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

    item['labels'] = torch.tensor(self.labels[idx])

    return item

  def __len__(self):

    return len(self.labels)
```

Training

In order to fine-tune the bert model certain hyper-parameters need to be set, hyper-parameters are values which control the learning process, these are set before learning. Hyper-parameters include:

Batch-size, which determines how many training examples should be processed before the model updates its parameters.

Learning-rate, which determines to what degree the model should update itself when update occurs.

Epochs, which determines how many times the whole batch of training examples should be processed.

To fine-tune the model a Dataloader object is needed and a Dataloader object just wraps an iterable around the dataset to enable easy access to the samples. An optimizer is also chosen in order to optimize the weights of the model. The model is then fine-tuned like so:

```
#Create Dataloader

train_loader = DataLoader(train_dataset, batch_size=1, shuffle=True)

#optimizer

optim = AdamW(model.parameters(), lr=2e-5)

for epoch in range(3):

  for batch in train_loader:

    optim.zero_grad()
```

```
input_ids = batch['input_ids'].to(device)
attention_mask = batch['attention_mask'].to(device)
labels = batch['labels'].to(device)
outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
loss = outputs[0]
loss.backward()
optim.step()
```

The fine-tuning code was retrieved from the transformers documentation [34].

The next step is to create a prediction function which takes a fine-tuned model and a test_set. The function returns a list of probabilities where each element is connected to a certain sample from the test_set where the value of that element is the probability of the element being of a certain class. When the fine-tuning of the model is done the test_set is fed to the prediction function which uses the fine-tuned model and the test_set and produces probabilities. The test-set are samples which the model has never seen before and has therefore not been used in training, therefore the test_set should be representative of how well the model would perform in a real-life situation. The probabilities are the confidence of a sample being of a certain class, the used threshold for predicting that a sample is of a certain class is picked to be a certain value. A function which takes a list of probabilities, and a threshold value is used and produces a list of predictions, this list is then compared to a list with the actual predictions. In order to make the comparison an evaluation framework called sklearn is used. In the framework a function exists which calculates the accuracy, recall and f1 score. In order to use the function, it needs two lists where one contains the predicted values, and one is the actual values.

The results of the python implementation were the accuracy, recall and f1 scores of a binary and multiclass classification. The thesis also uses a model that already was fine-tuned and ready to make predictions. The already fine-tuned model only has the ability to make predictions whether the sample is positive or negative, which is binary classification. The model that is ready to make predictions is called marma and is based on the same BERT model which is used for the authors python implementation and is trained on 20 000 app store reviews [35].

The already fine-tuned binary classification model can be loaded like so:

```
sa = pipeline('sentiment-analysis', model='marma/bert-base-swedish-cased-sentiment')
```

The used evaluation framework is retrieved from a machine learning library which is called scikit-learn. The used function from the scikit is called classification report which takes a couple of parameters, the most important parameters are y_true and y_pred, t_true is an array which contain the correct values of the samples and y_pred contains the predicted values of the samples, by supplying the function with these two arrays the function can calculate precision, recall and f1_score. Scikit-learn is used by many different companies like spotify, Huggingface and many others [36].

Appendix B: Detailed results

Detailed results of SentimenDLApproach for binary sentiment analysis

label	precision	recall	F1	
neutral	0	0	0	
negative	0.825	0.5963855422	0.6923076923	
positive	0.6666666667	0.8662420382	0.7534626039	
accuracy	wprecision	wrecall	wf-measure	
0.725308642	0.7457304527	0.725308642	0.7198046473	
	accuracy	**weighted F1**		
overview	0.725	0.72		
		F1		
positive	-	0.753		
negative	-	0.692		
batch size	epochs	learning rate	dropout	validation split
64	6	0.0005	0.2	0.5

label	precision	recall	F1	
positive	0.6777251185	0.9108280255	0.777173913	
neutral	0	0	0	
negative	0.8761061947	0.5963855422	0.7096774194	
accuracy	wprecision	wrecall	wf-measure	
0.7469135802	0.7772730615	0.7469135802	0.7401936912	
	accuracy	**weighted F1**		
overview	0.747	0.74		
		F1		
positive	-	0.777		
negative	-	0.71		
batch size	epochs	learning rate	dropout	validation split
64	12	0.0005	0.2	0.5

label	precision	recall	F1	
positive	0.6666666667	0.8662420382	0.7534626039	
neutral	0	0	0	
negative	0.825	0.5963855422	0.6923076923	

accuracy	wprecision	wrecall	wf-measure	
0.725308642	0.7457304527	0.725308642	0.7198046473	
	accuracy	**weighted F1**		
overview	0.725	0.72		
		F1		
positive	-	0.753		
negative	-	0.692		
batch size	epochs	learning rate	dropout	validation split
64	24	0.0005	0.2	0.5

label	precision	recall	F1	
positive	0.6971153846	0.923566879	0.7945205479	
neutral	0	0	0	
negative	0.8965517241	0.6265060241	0.7375886525	
accuracy	wprecision	wrecall	wf-measure	
0.7685185185	0.7971441407	0.7685185185	0.7628995134	
	accuracy	**weighted F1**		
overview	0.77	0.763		
		F1		
positive	-	0.795		
negative	-	0.738		
batch size	epochs	learning rate	dropout	validation split
64	24	0.0005	0.2	0.2

label	precision	recall	F1	
positive	0.6826923077	0.9044585987	0.7780821918	
neutral	0	0	0	
negative	0.8706896552	0.6084337349	0.7163120567	
accuracy	wprecision	wrecall	wf-measure	
0.75	0.7769048613	0.75	0.7440330418	
	accuracy	**weighted F1**		
overview	0.75	0.744		
		F1		
positive	-	0.778		
negative	-	0.716		

batch size	epochs	learning rate	dropout	validation split
64	48	0.0005	0.2	0.2

label	precision	recall	F1	
positive	0.7009803922	0.9108280255	0.7922437673	
neutral	0	0	0	
negative	0.8833333333	0.6385542169	0.7412587413	
accuracy	wprecision	wrecall	wf-measure	
0.7685185185	0.7922446139	0.7685185185	0.7636766127	
	accuracy	**weighted F1**		
overview	0.769	0.764		
		F1		
positive	-	0.792		
negative	-	0.741		
batch size	epochs	learning rate	dropout	validation split
64	32	0.0005	0.2	0.2

label	precision	recall	F1	
positive	0.6971153846	0.923566879	0.7945205479	
neutral	0	0	0	
negative	0.8965517241	0.6265060241	0.7375886525	
accuracy	wprecision	wrecall	wf-measure	
0.7685185185	0.7971441407	0.7685185185	0.7628995134	
	accuracy	**weighted F1**		
overview	0.769	0.763		
		F1		
positive	-	0.795		
negative	-	0.738		
batch size	epochs	learning rate	dropout	validation split
64	32	0.0005	0.2	0

label	precision	recall	F1	
positive	0.7053140097	0.9299363057	0.8021978022	
negative	0.9051724138	0.6325301205	0.7446808511	
accuracy	wprecision	wrecall	wf-measure	

0.7770897833	0.8080276167	0.7770897833	0.7726380069	
overview	**accuracy**	**weighted F1**		
	0.777	0.773		
		F1		
positive	-	0.802		
negative	-	0.745		
batch size	epochs	learning rate	dropout	validation split
64	24	0.0005	0.2	0

label	precision	recall	F1	
positive	0.7093596059	0.9171974522	0.8	
negative	0.8916666667	0.6445783133	0.7482517483	
accuracy	wprecision	wrecall	wf-measure	
0.7770897833	0.8030530179	0.7770897833	0.7734049233	
overview	**accuracy**	**weighted F1**		
	0.777	0.773		
		F1		
positive	-	0.8		
negative	-	0.748		
batch size	epochs	learning rate	dropout	validation split
64	28	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6971153846	0.923566879	0.7945205479	
negative	0.8956521739	0.6204819277	0.7330960854	
accuracy	wprecision	wrecall	wf-measure	
0.7678018576	0.7991497717	0.7678018576	0.7629525579	
overview	**accuracy**	**weighted F1**		
	0.768	0.763		
		F1		
positive	-	0.795		
negative	-	0.733		
batch size	epochs	learning rate	dropout	validation split
64	20	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6926829268	0.9044585987	0.7845303867	
negative	0.8728813559	0.6204819277	0.7253521127	
accuracy	wprecision	wrecall	wf-measure	
0.7585139319	0.7852926458	0.7585139319	0.7541167846	
	accuracy	**weighted F1**		
overview	0.759	0.754		
		F1		
positive	-	0.785		
negative	-	0.725		
batch size	epochs	learning rate	dropout	validation split
64	12	0.0005	0.2	0

label	precision	recall	F1	
positive	0.7033492823	0.9363057325	0.8032786885	
negative	0.9122807018	0.6265060241	0.7428571429	
accuracy	wprecision	wrecall	wf-measure	
0.7770897833	0.8107258013	0.7770897833	0.7722261295	
	accuracy	**weighted F1**		
overview	0.777	0.772		
		F1		
positive	-	0.803		
negative	-	0.743		
batch size	epochs	learning rate	dropout	validation split
64	24	0.0005	0.1	0

label	precision	recall	F1	
positive	0.7024390244	0.9171974522	0.7955801105	
negative	0.8898305085	0.6325301205	0.7394366197	
accuracy	wprecision	wrecall	wf-measure	
0.7708978328	0.7987454837	0.7708978328	0.7667261803	
	accuracy	**weighted F1**		
overview	0.771	0.767		
		F1		
positive	-	0.796		

negative	-	0.739		
batch size	epochs	learning rate	dropout	validation split
64	20	0.0005	0.1	0

label	precision	recall	F1	
positive	0.698019802	0.898089172	0.7855153203	
negative	0.867768595	0.6325301205	0.7317073171	
accuracy	wprecision	wrecall	wf-measure	
0.7616099071	0.7852591198	0.7616099071	0.7578616716	
overview	**accuracy**	**weighted F1**		
	0.761	0.758		
		F1		
positive	-	0.786		
negative	-	0.732		
batch size	epochs	learning rate	dropout	validation split
64	12	0.0005	0.1	0

label	precision	recall	F1	
positive	0.6971153846	0.923566879	0.7945205479	
negative	0.8956521739	0.6204819277	0.7330960854	
accuracy	wprecision	wrecall	wf-measure	
0.7678018576	0.7991497717	0.7678018576	0.7629525579	
overview	**accuracy**	**weighted F1**		
	0.768	0.763		
		F1		
positive	-	0.795		
negative	-	0.733		
batch size	epochs	learning rate	dropout	validation split
64	28	0.0005	0.1	0

Detailed results of SentimenDLApproach for multiclass sentiment analysis

label	precision	recall	F1		
positive	0.503875969	0.8280254777	0.6265060241		
neutral	0.2235294118	0.1386861314	0.1711711712		

negative	0.6923076923	0.4879518072	0.5724381625		
accuracy	wprecision	wrecall	wf-measure		
0.5	0.4883807249	0.5	0.4713839809		
	accuracy	**weighted F1**			
overview	0.5	0.471			
		F1			
positive	-	0.627			
neutral	-	0.171			
negative	-	0.572			
batch size	epochs	learning rate	dropout	validation split	threshold
64	12	0.0005	0.1	0	0.7

Detailed results of ClassifierDLApproach for binary sentiment analysis

label	precision	recall	F1	
positive	0.680952381	0.9108280255	0.7792915531	
negative	0.8761061947	0.5963855422	0.7096774194	
accuracy	wprecision	wrecall	wf-measure	
0.7492260062	0.781248149	0.7492260062	0.7435146299	
	accuracy	**weighted F1**		
overview	0.749	0.744		
		F1		
positive	-	0.779		
negative	-	0.71		
batch size	epochs	learning rate	dropout	validation split
64	32	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.7121212121	0.898089172	0.7943661972	
negative	0.872	0.656626506	0.7491408935	
accuracy	wprecision	wrecall	wf-measure	
0.773993808	0.7942880195	0.773993808	0.7711234714	
	accuracy	**weighted F1**		
overview	0.774	0.771		
		F1		
positive	-	0.794		

negative	-	0.749		
batch size	epochs	learning rate	dropout	validation split
64	64	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.712195122	0.9299363057	0.8066298343	
negative	0.906779661	0.6445783133	0.7535211268	
accuracy	wprecision	wrecall	wf-measure	
0.7832817337	0.8121983216	0.7832817337	0.7793355759	
overview	**accuracy**	**weighted F1**		
	0.783	0.779		
		F1		
positive	-	0.807		
negative	-	0.754		
batch size	epochs	learning rate	dropout	validation split
64	64	0.0005	0.2	0

label	precision	recall	F1	
positive	0.7067307692	0.9363057325	0.8054794521	
negative	0.9130434783	0.6325301205	0.7473309609	
accuracy	wprecision	wrecall	wf-measure	
0.7801857585	0.8127614494	0.7801857585	0.7755950882	
overview	**accuracy**	**weighted F1**		
	0.78	0.776		
		F1		
positive	-	0.805		
negative	-	0.747		
batch size	epochs	learning rate	dropout	validation split
64	70	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6956521739	0.9171974522	0.7912087912	
negative	0.8879310345	0.6204819277	0.7304964539	
accuracy	wprecision	wrecall	wf-measure	
0.7647058824	0.7944704119	0.7647058824	0.760006785	

	accuracy	weighted F1		
overview	0.765	0.76		
		F1		
positive	-	0.791		
negative	-	0.73		
batch size	epochs	learning rate	dropout	validation split
64	58	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6975609756	0.9108280255	0.7900552486	
negative	0.8813559322	0.6265060241	0.7323943662	
accuracy	wprecision	wrecall	wf-measure	
0.7647058824	0.7920190648	0.7647058824	0.7604214824	
	accuracy	**weighted F1**		
overview	0.765	0.76		
		F1		
positive	-	0.79		
negative	-	0.732		
batch size	epochs	learning rate	dropout	validation split
64	58	0.0005	0.3	0

label	precision	recall	F1	
positive	0.7101449275	0.9363057325	0.8076923077	
negative	0.9137931034	0.6385542169	0.7517730496	
accuracy	wprecision	wrecall	wf-measure	
0.7832817337	0.8148062192	0.7832817337	0.7789536178	
	accuracy	**weighted F1**		
overview	0.783	0.779		
		F1		
positive	-	0.808		
negative	-	0.752		
batch size	epochs	learning rate	dropout	validation split
64	70	0.0005	0.3	0

label	precision	recall	F1	

positive	0.7067307692	0.9363057325	0.8054794521	
negative	0.9130434783	0.6325301205	0.7473309609	
accuracy	wprecision	wrecall	wf-measure	
0.7801857585	0.8127614494	0.7801857585	0.7755950882	
	accuracy	**weighted F1**		
overview	0.78	0.776		
		F1		
positive	-	0.805		
negative	-	0.776		
batch size	epochs	learning rate	dropout	validation split
64	74	0.0005	0.3	0

label	precision	recall	F1	
positive	0.6990291262	0.9171974522	0.7933884298	
negative	0.8888888889	0.6265060241	0.7349823322	
accuracy	wprecision	wrecall	wf-measure	
0.7678018576	0.7966041126	0.7678018576	0.7633716737	
	accuracy	**weighted F1**		
overview	0.768	0.763		
		F1		
positive	-	0.793		
negative	-	0.735		
batch size	epochs	learning rate	dropout	validation split
64	64	0.0005	0.3	0

label	precision	recall	F1	
positive	0.6712962963	0.923566879	0.7774798928	
negative	0.8878504673	0.5722891566	0.695970696	
accuracy	wprecision	wrecall	wf-measure	
0.7430340557	0.7825903904	0.7430340557	0.7355897173	
	accuracy	**weighted F1**		
overview	0.783	0.736		
		F1		
positive	-	0.777		
negative	-	0.696		

batch size	epochs	learning rate	dropout	validation split
64	70	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.7079207921	0.9108280255	0.7966573816	
negative	0.8842975207	0.6445783133	0.7456445993	
accuracy	wprecision	wrecall	wf-measure	
0.773993808	0.7985664173	0.773993808	0.7704402861	
	accuracy	**weighted F1**		
overview	0.774	0.77		
		F1		
positive	-	0.797		
negative	-	0.756		
batch size	epochs	learning rate	dropout	validation split
64	58	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.702970297	0.9044585987	0.791086351	
negative	0.8760330579	0.6385542169	0.7386759582	
accuracy	wprecision	wrecall	wf-measure	
0.7678018576	0.7919127685	0.7678018576	0.7641509788	
	accuracy	**weighted F1**		
overview	0.768	0.764		
		F1		
positive	-	0.791		
negative	-	0.739		
batch size	epochs	learning rate	dropout	validation split
64	43	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.7073170732	0.923566879	0.8011049724	
negative	0.8983050847	0.6385542169	0.7464788732	
accuracy	wprecision	wrecall	wf-measure	
0.7770897833	0.8054719026	0.7770897833	0.7730308781	
	accuracy	**weighted F1**		

overview	0.777	0.773		
		F1		
positive	-	0.801		
negative	-	0.746		
batch size	epochs	learning rate	dropout	validation split
64	80	0.0005	0.3	0

Detailed results of ClassifierDLApproach for multiclass sentiment analysis

label	precision	recall	F1	
positive	0.5502183406	0.8025477707	0.6528497409	
neutral	0.4978165939	0.8321167883	0.6229508197	
negative	0.5	0.006024096386	0.0119047619	
accuracy	wprecision	wrecall	wf-measure	
0.5239130435	0.5164894627	0.5239130435	0.4126475263	
	accuracy	**weighted F1**		
overview	0.524	0.413		
		F1		
positive	-	0.653		
neutral	-	0.623		
negative	-	0.012		
batch size	epochs	learning rate	dropout	validation split
64	4	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.5407725322	0.8025477707	0.6461538462	
neutral	0.5308056872	0.8175182482	0.6436781609	
negative	0.8125	0.07831325301	0.1428571429	
accuracy	wprecision	wrecall	wf-measure	
0.5456521739	0.6358623189	0.5456521739	0.46379206	
	accuracy	**weighted F1**		
overview	0.546	0.464		
		F1		
positive	-	0.646		
neutral	-	0.644		
negative	-	0.143		

batch size	epochs	learning rate	dropout	validation split
64	8	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.6473988439	0.7133757962	0.6787878788	
neutral	0.4919354839	0.8905109489	0.6337662338	
negative	0.7179487179	0.1686746988	0.2731707317	
accuracy	wprecision	wrecall	wf-measure	
0.5695652174	0.6265571021	0.5695652174	0.5190043749	
	accuracy	**weighted F1**		
overview	0.57	0.519		
		F1		
positive	-	0.679		
neutral	-	0.634		
negative	-	0.273		
batch size	epochs	learning rate	dropout	validation split
64	16	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.6827586207	0.6305732484	0.6556291391	
neutral	0.5555555556	0.802919708	0.6567164179	
negative	0.6581196581	0.4638554217	0.5441696113	
accuracy	wprecision	wrecall	wf-measure	
0.6217391304	0.6359827778	0.6217391304	0.6157306078	
	accuracy	**weighted F1**		
overview	0.622	0.616		
		F1		
positive	-	0.656		
neutral	-	0.657		
negative	-	0.544		
batch size	epochs	learning rate	dropout	validation split
64	32	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.6473988439	0.7133757962	0.6787878788	

neutral	0.572972973	0.7737226277	0.6583850932	
negative	0.6862745098	0.421686747	0.5223880597	
accuracy	wprecision	wrecall	wf-measure	
0.6260869565	0.6392619227	0.6260869565	0.6162714623	
	accuracy	**weighted F1**		
overview	0.626	0.616		
		F1		
positive	-	0.679		
neutral	-	0.658		
negative	-	0.522		
batch size	epochs	learning rate	dropout	validation split
64	64	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.6987179487	0.6942675159	0.696485623	
neutral	0.579787234	0.795620438	0.6707692308	
negative	0.6982758621	0.4879518072	0.5744680851	
accuracy	wprecision	wrecall	wf-measure	
0.65	0.6631377437	0.65	0.6447941947	
	accuracy	**weighted F1**		
overview	0.65	0.645		
		F1		
positive	-	0.696		
neutral	-	0.671		
negative	-	0.574		
batch size	epochs	learning rate	dropout	validation split
64	128	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.6705882353	0.7261146497	0.6972477064	
neutral	0.5930232558	0.7445255474	0.6601941748	
negative	0.686440678	0.4879518072	0.5704225352	
accuracy	wprecision	wrecall	wf-measure	
0.6456521739	0.6532080251	0.6456521739	0.6404448537	
	accuracy	**weighted F1**		

overview	0.646	0.64		
		F1		
positive	-	0.697		
neutral	-	0.66		
negative	-	0.57		
batch size	epochs	learning rate	dropout	validation split
64	256	0.0005	0.2	0.3

label	precision	recall	F1	
positive	0.5336134454	0.8089171975	0.6430379747	
neutral	0.5205479452	0.8321167883	0.6404494382	
negative	0.6666666667	0.01204819277	0.02366863905	
accuracy	wprecision	wrecall	wf-measure	
0.5282608696	0.5777370567	0.5282608696	0.4187554981	
	accuracy	**weighted F1**		
overview	0.528	0.419		
		F1		
positive	-	0.643		
neutral	-	0.64		
negative	-	0.024		
batch size	epochs	learning rate	dropout	validation split
64	256	0.0005	0.5	0.3

label	precision	recall	F1	
positive	0.6932515337	0.7197452229	0.70625	
neutral	0.5786516854	0.7518248175	0.653968254	
negative	0.6806722689	0.4879518072	0.5684210526	
accuracy	wprecision	wrecall	wf-measure	
0.6456521739	0.6545812355	0.6456521739	0.6409408598	
	accuracy	**weighted F1**		
overview	0.646	0.641		
		F1		
positive	-	0.706		
neutral	-	0.654		
negative	-	0.568		

batch size	epochs	learning rate	dropout	validation split
64	128	0.0005	0.5	0

label	precision	recall	F1	
positive	0.6826347305	0.7261146497	0.7037037037	
neutral	0.5842696629	0.7591240876	0.6603174603	
negative	0.6782608696	0.4698795181	0.5551601423	
accuracy	wprecision	wrecall	wf-measure	
0.6434782609	0.651760654	0.6434782609	0.6371772982	
	accuracy	**weighted F1**		
overview	0.643	0.637		
		F1		
positive	-	0.704		
neutral	-	0.66		
negative	-	0.555		
batch size	epochs	learning rate	dropout	validation split
64	64	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6766467066	0.7197452229	0.6975308642	
neutral	0.5842696629	0.7591240876	0.6603174603	
negative	0.6695652174	0.4638554217	0.5480427046	
accuracy	wprecision	wrecall	wf-measure	
0.6391304348	0.6465789192	0.6391304348	0.6325020146	
	accuracy	**weighted F1**		
overview	0.639	0.633		
		F1		
positive	-	0.698		
neutral	-	0.66		
negative	-	0.548		
batch size	epochs	learning rate	dropout	validation split
64	32	0.0005	0.2	0

label	precision	recall	F1	
positive	0.69375	0.7070063694	0.7003154574	

neutral	0.5762711864	0.7445255474	0.6496815287	
negative	0.6666666667	0.4939759036	0.5674740484	
accuracy	wprecision	wrecall	wf-measure	
0.6413043478	0.6489881939	0.6413043478	0.637296931	
overview	**accuracy**	**weighted F1**		
	0.641	0.637		
		F1		
positive	-	0.7		
neutral	-	0.65		
negative	-	0.567		
batch size	epochs	learning rate	dropout	validation split
64	256	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6390532544	0.6878980892	0.6625766871	
neutral	0.4785992218	0.897810219	0.6243654822	
negative	0.7941176471	0.1626506024	0.27	
accuracy	wprecision	wrecall	wf-measure	
0.5608695652	0.6472238777	0.5608695652	0.5095274151	
overview	**accuracy**	**weighted F1**		
	0.561	0.51		
		F1		
positive	-	0.663		
neutral	-	0.624		
negative	-	0.27		
batch size	epochs	learning rate	dropout	validation split
64	32	0.0005	0.5	0

label	precision	recall	F1	
positive	0.5161290323	0.8152866242	0.6320987654	
neutral	0.5333333333	0.8175182482	0.6455331412	
negative	0.5	0.006024096386	0.0119047619	
accuracy	wprecision	wrecall	wf-measure	
0.5239130435	0.5154324451	0.5239130435	0.4122907326	
	accuracy	**weighted F1**		

overview	0.524	0.412		
		F1		
positive	-	0.632		
neutral	-	0.646		
negative	-	0.012		
batch size	epochs	learning rate	dropout	validation split
64	8	0.0005	0.5	0

label	precision	recall	F1	
positive	0.6807228916	0.7197452229	0.6996904025	
neutral	0.5801104972	0.7664233577	0.6603773585	
negative	0.6902654867	0.4698795181	0.5591397849	
accuracy	wprecision	wrecall	wf-measure	
0.6434782609	0.654201528	0.6434782609	0.6372615122	
	accuracy	**weighted F1**		
overview	0.643	0.637		
		F1		
positive	-	0.7		
neutral	-	0.66		
negative	-	0.56		
batch size	epochs	learning rate	dropout	validation split
64	512	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6848484848	0.7197452229	0.701863354	
neutral	0.5801104972	0.7664233577	0.6603773585	
negative	0.6929824561	0.4759036145	0.5642857143	
accuracy	wprecision	wrecall	wf-measure	
0.6456521739	0.6565900825	0.6456521739	0.6398601593	
	accuracy	**weighted F1**		
overview	0.647	0.64		
		F1		
positive	-	0.702		
neutral	-	0.66		
negative	-	0.564		

batch size	epochs	learning rate	dropout	validation split
64	1024	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6951219512	0.7261146497	0.7102803738	
neutral	0.5824175824	0.7737226277	0.6645768025	
negative	0.701754386	0.4819277108	0.5714285714	
accuracy	wprecision	wrecall	wf-measure	
0.652173913	0.6639490939	0.652173913	0.6465612685	
	accuracy	**weighted F1**		
overview	0.652	0.647		
		F1		
positive	-	0.71		
neutral	-	0.665		
negative	-	0.571		
batch size	epochs	learning rate	dropout	validation split
64	2048	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6829268293	0.7133757962	0.6978193146	
neutral	0.5842696629	0.7591240876	0.6603174603	
negative	0.6949152542	0.4939759036	0.5774647887	
accuracy	wprecision	wrecall	wf-measure	
0.647826087	0.6578704092	0.647826087	0.6432179987	
	accuracy	**weighted F1**		
overview	0.648	0.643		
		F1		
positive	-	0.7		
neutral	-	0.66		
negative	-	0.577		
batch size	epochs	learning rate	dropout	validation split
64	4096	0.0005	0.2	0

label	precision	recall	F1	
positive	0.5740740741	0.7898089172	0.6648793566	

neutral	0.5	0.8540145985	0.6307277628	
negative	0.9	0.05421686747	0.1022727273	
accuracy	wprecision	wrecall	wf-measure	
0.5434782609	0.6696296296	0.5434782609	0.4516805113	
overview	**accuracy**	**weighted F1**		
	0.543	0.452		
		F1		
positive	-	0.665		
neutral	-	0.631		
negative	-	0.102		
batch size	epochs	learning rate	dropout	validation split
64	8	0.0005	0.2	0

label	precision	recall	F1	
positive	0.6909090909	0.7261146497	0.7080745342	
neutral	0.5745856354	0.7591240876	0.6540880503	
negative	0.6929824561	0.4759036145	0.5642857143	
accuracy	wprecision	wrecall	wf-measure	
0.6456521739	0.6570131457	0.6456521739	0.640106942	
overview	**accuracy**	**weighted F1**		
	0.646	0.64		
		F1		
positive	-	0.708		
neutral	-	0.654		
negative	-	0.564		
batch size	epochs	learning rate	dropout	validation split
64	8192	0.0005	0.2	0

Detailed results of binary sentiment analysis developed in Python

Label	Precision	Recall	F1
Negative	0.92	0.49	0.64
Positive	0.65	0.96	0.78
accuracy	wprecision	wrecall	wf1
0.72	0.79	0.72	0.71
	accuracy	**weighted F1**	
overview	0.72	0.71	

		F1	
negative	-	0.64	
positive	-	0.78	
batch size	epochs	learning rate	
16	8	0.00005	
Label	Precision	Recall	F1
Negative	0.92	0.53	0.67
Positive	0.67	0.96	0.79
accuracy	wprecision	wrecall	wf1
0.74	0.8	0.74	0.73
	accuracy	**weighted F1**	
overview	0.74	0.73	
		F1	
negative	-	0.67	
positive	-	0.79	
batch size	epochs	learning rate	
16	1	0.00004	
Label	Precision	Recall	F1
Negative	0.89	0.49	0.63
Positive	0.65	0.94	0.77
accuracy	wprecision	wrecall	wf1
0.71	0.77	0.71	0.7
	accuracy	**weighted F1**	
overview	0.71	0.7	
		F1	
negative	-	0.63	
positive	-	0.77	
batch size	epochs	learning rate	
16	1	0.00003	
Label	Precision	Recall	F1
Negative	0.88	0.56	0.68
Positive	0.68	0.92	0.78

accuracy	wprecision	wrecall	wf1
0.74	0.78	0.74	0.73
	accuracy	**weighted F1**	
overview	0.74	0.73	
		F1	
negative	-	0.68	
positive	-	0.78	
batch size	epochs	learning rate	
16	1	0.00002	

Label	Precision	Recall	F1
Negative	0.8	0.87	0.83
Positive	0.86	0.78	0.82
accuracy	wprecision	wrecall	wf1
0.83	0.83	0.83	0.83
	accuracy	**weighted F1**	
overview	0.83	0.83	
		F1	
negative	-	0.83	
positive	-	0.82	
batch size	epochs	learning rate	
16	1	0.00001	

Detailed results of multiclass sentiment analysis developed in Python

Label	Precision	Recall	F1
Negative	0.95	0.36	0.52
Positive	0.8	0.92	0.86
Neutral	0.64	0.94	0.76
accuracy	wprecision	wrecall	wf1
0.74	0.8	0.74	0.71
	accuracy	**weighted F1**	
overview	0.74	0.71	
		F1	
negative	-	0.52	
positive	-	0.86	
neutral	-	0.76	

batch size	epochs	learning rate	
16	8	0.00005	
Label	Precision	Recall	F1
Negative	0.94	0.4	0.56
Positive	0.81	0.94	0.87
Neutral	0.67	0.95	0.79
accuracy	wprecision	wrecall	wf1
0.76	0.81	0.76	0.74
	accuracy	**weighted F1**	
overview	0.76	0.74	
		F1	
negative	-	0.56	
positive	-	0.87	
neutral	-	0.79	
batch size	epochs	learning rate	
16	7	0.00005	
Label	Precision	Recall	F1
Negative	0.95	0.38	0.55
Positive	0.83	0.88	0.85
Neutral	0.62	0.95	0.75
accuracy	wprecision	wrecall	wf1
0.74	0.8	0.74	0.72
	accuracy	**weighted F1**	
overview	0.74	0.72	
		F1	
negative	-	0.55	
positive	-	0.85	
neutral	-	0.75	
batch size	epochs	learning rate	
16	6	0.00005	
Label	Precision	Recall	F1
Negative	0.9	0.42	0.57

Positive	0.86	0.9	0.88
Neutral	0.64	0.95	0.76
accuracy	wprecision	wrecall	wf1
0.76	0.8	0.76	0.74
	accuracy	**weighted F1**	
overview	0.76	0.74	
		F1	
negative	-	0.57	
positive	-	0.88	
neutral	-	0.76	
batch size	epochs	learning rate	
16	5	0.00005	
Label	Precision	Recall	F1
Negative	0.93	0.34	0.5
Positive	0.83	0.92	0.88
Neutral	0.63	0.95	0.76
accuracy	wprecision	wrecall	wf1
0.74	0.8	0.74	0.71
	accuracy	**weighted F1**	
overview	0.74	0.71	
		F1	
negative	-	0.5	
positive	-	0.88	
neutral	-	0.76	
batch size	epochs	learning rate	
16	4	0.00005	
Label	Precision	Recall	F1
Negative	0.95	0.41	0.57
Positive	0.83	0.9	0.87
Neutral	0.64	0.95	0.76
accuracy	wprecision	wrecall	wf1
0.75	0.81	0.75	0.73
	accuracy	**weighted F1**	

overview	0.75	0.73	
		F1	
negative	-	0.57	
positive	-	0.87	
neutral	-	0.76	
batch size	epochs	learning rate	
16	3	0.00005	
Label	Precision	Recall	F1
Negative	0.96	0.51	0.66
Positive	0.75	0.95	0.84
Neutral	0.77	0.93	0.84
accuracy	wprecision	wrecall	wf1
0.79	0.83	0.79	0.78
	accuracy	**weighted F1**	
overview	0.79	0.78	
		F1	
negative	-	0.66	
positive	-	0.84	
neutral	-	0.84	
batch size	epochs	learning rate	
16	2	0.00005	
Label	Precision	Recall	F1
Negative	0.78	0.56	0.65
Positive	0.71	0.71	0.71
Neutral	0.71	0.92	0.8
accuracy	wprecision	wrecall	wf1
0.73	0.73	0.73	0.72
	accuracy	**weighted F1**	
overview	0.73	0.72	
		F1	
negative	-	0.65	
positive	-	0.71	
neutral	-	0.8	

batch size	epochs	learning rate	
16	1	0.00005	

TRITA-EECS-EX-2019:XX