Distant Supervision and Sentiment Embeddings for Ternary Twitter Sentiment Analysis

Mats Byrkjeland
Frederik Gørvell de Lichtenberg

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Björn Gambäck, IDI

Norwegian University of Science and Technology
Department of Computer Science

Mats Byrkjeland and Frederik Gørvell de Lichtenberg

Distant Supervision and Sentiment Embeddings for Ternary Twitter Sentiment Analysis

Master's Thesis in Computer Science, Spring 2017

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

Tang et al. (2014) acknowledged the context-based word embeddings' inability to discriminate between words with opposite sentiments that appear in similar contexts. An example is the words "good" and "bad"—two opposites that appear in the same contexts. Context-based word embedding methods like word2vec would likely treat these as similar words. Tang et al. proposed a promising method for incorporating sentiment information in word embeddings. These embeddings are called Sentiment-Specific Word Embeddings or *Sentiment Embeddings*.

To train sentiment embeddings, large amounts of sentiment-annotated data are needed. Manual annotation is too expensive for this purpose. Fast, automatical annotation is used to set a low-quality (weak) label on large corpora of tweets. This procedure is often referred to as *distant supervision*. The traditional approach is to use the occurrences of emoticons to guess binary sentiment (positive/negative).

In this thesis, we compare various lexicon-based sentiment classifiers against each other on manually annotated Twitter data from the International Workshop on Semantic Evaluation (SemEval). Their performance as distant supervision methods are tested as part of a complete Twitter Sentiment Analysis system. Instead of only looking at the positive and negative sentiment classes, the neutral class is included. Both prediction performance and speed of the distant supervision methods are evaluated.

We propose the Ternary Sentiment Embedding Model—a new model for creating sentiment embeddings for the ternary sentiment classification task. It is based on the Hybrid Ranking Model of Tang et al. (2016), but trains on ternary-labeled distant-supervised data instead of binary-labeled. The model trains sentiment embeddings from datasets made with different distant supervision methods. The model is used as part of a complete Twitter Sentiment Analysis system and is compared to existing systems.

The experiments of Chapter 8 show that the Ternary Sentiment Embedding Model performs better than the Hybrid Ranking Model of Tang et al. (2016) in most cases. Our results show that the quality of the distant-supervised dataset has a great impact on the quality of the produced sentiment embeddings, and hence the entire Twitter Sentiment Analysis system.

Sammendrag

Tang mfl. (2014) anerkjente kontekstbaserte ordvektorers manglende evne til å diskriminere mellom ord med motsatte sentiment som forekommer i like kontekster. Et eksempel er ordene *good* og *bad* – to motsatte ord som ofte deler kontekst. Kontekstbaserte ordvektormetoder som word2vec ville sannsynligvis behandlet disse som like ord. Tang mfl. foreslo en lovende metode for å inkorporere sentimentinformasjon i ordvektorer. Disse vektorene kalles Sentiment-Specific Word Embeddings eller *sentimentvektorer* (*Sentiment Embeddings*).

For å trene sentimentvektorer trengs store mengder sentimentmerkede data. Manuell merking er for dyrt for dette formålet. Rask, automatisk merking blir brukt til å sette et lavkvalitets (svakt) merke på store samlinger av tweets. Denne prosedyren blir ofte referert til som *fjernveiledning* (*distant supervision*). Go mfl. (2009) introduserte en av de første tilnærmingene til fjernveiledning av tweets, som brukte forekomster av uttrykksikon for å gjette sentimentpolaritet (positiv/negativ).

I denne avhandlingen sammenligner vi flere leksikonbaserte sentimentklassifiserere mot hverandre på manuelt annoterte Twitter-data fra International Workshop on Semantic Evaluation (SemEval). Deres ytelse som fjernveiledningsmethoder er testet som del av et komplett Twitter-sentimentanalysesystem. I stedet for å bare se på de positive og negative sentimentklassene, er den nøytrale klassen inkludert. Både prediksjonsytelse og fart til fjernveiledningsmetodene evalueres.

Vi foreslår den Tredelte sentimentvektormodellen (Ternary Sentiment Embedding Model) – en ny modell for å trene sentimentvektorer for den tredelte sentimentklassifiseringsoppgaven. Den er basert på Hybridrangeringsmodellen (Hybrid Ranking Model) til Tang mfl. (2016), men trenes på treveismerkede, fjernveiledede data i stedet for toveismerkede. Modellen trener sentimentvektorer fra datamengder laget med ulike fjernveiledningsmetoder. Modellen er brukt som del av et komplett Twitter-sentimentanalysesystem og sammenlignes med eksisterende systemer.

Eksperimentene i Kapittel 8 viser at den Tredelte sentimentvektormodellen yter bedre enn Tang mfl. (2016) sin Hybridrangeringsmodell (Hybrid Ranking Model) i de fleste tilfeller. Resultatene våre viser at kvaliteten på de fjernveiledede datamengdene har en stor innvirkning på kvaliteten til de resulterende sentimentvektorene og dermed hele sentimentanalysesystemet.

Preface

This Master's Thesis has been carried out at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. The thesis concludes our Master of Computer Science degrees at the Department of Computer Science (IDI) at NTNU and was supervised by Professor Björn Gambäck.

Mats Byrkjeland and Frederik Gørvell de Lichtenberg
Trondheim, June 11, 2017

Contents

1. Introduction **1**
1.1. Twitter . . . 1
1.2. Motivation . . . 2
1.3. Goals . . . 2
1.4. Contributions . . . 3
1.5. Thesis Structure . . . 3

2. Background Theory **5**
2.1. Textual Preprocessing . . . 5
2.1.1. Part-of-Speech Tagging . . . 5
2.1.2. Bag-of-Words . . . 5
2.1.3. Stop Word Removal . . . 5
2.1.4. Stemming . . . 6
2.1.5. Reduce Elongated Words . . . 6
2.2. Machine Learning Algorithms . . . 6
2.2.1. Support Vector Machines . . . 6
2.2.2. Maximum Entropy . . . 6
2.3. Evaluation . . . 7
2.3.1. Techniques . . . 7
2.3.2. Metrics . . . 7
2.4. Word Embeddings . . . 9
2.4.1. word2vec . . . 11
2.4.2. GloVe . . . 12
2.5. Tools . . . 13
2.5.1. scikit-learn . . . 13
2.5.2. MultiVec . . . 13
2.5.3. glove-python . . . 13
2.5.4. Twokenize . . . 13
2.5.5. Preprocessor . . . 14
2.5.6. Twitter API . . . 14
2.5.7. twit . . . 15
2.5.8. AFINN . . . 15
2.5.9. VADER . . . 15
2.5.10. TextBlob . . . 15
2.5.11. Fredriksen–Jahren Lexicon Classifier . . . 15
2.5.12. Theano . . . 16

2.5.13. TensorFlow 16
2.5.14. Keras 16

3. Related Work **17**
3.1. Twitter Sentiment Analysis 17
3.1.1. Preprocessing 17
3.1.2. Feature Extraction 17
3.1.3. Machine Learning Classifiers 18
3.2. The Collobert and Weston Model 18
3.3. Sentiment-Specific Word Embeddings 20
3.3.1. Basic Model 1 (SSWE$_h$) 21
3.3.2. Basic Model 2 (SSWE$_r$) 21
3.3.3. Unified Model (SSWE$_u$) 21
3.3.4. Hybrid Ranking Model 22
3.4. The International Workshop on Semantic Evaluation 23

4. Ternary Sentiment Embedding Model **27**
4.1. Core Layers 27
4.2. Ternary Sentiment Linear Layer 28
4.3. Combined Loss Function 28
4.4. Model Training 29

5. Architecture **31**
5.1. Tweet Collector 31
5.2. CATTS 32
5.3. Fredriksen–Jahren Lexicon Classifier Python Port 32
5.4. Twitty 34
5.4.1. Processing Raw Files 34
5.4.2. Filtering 34
5.4.3. Preprocessing 35
5.4.4. Train Word Embeddings 35
5.4.5. Train Classifier 35
5.4.6. Test Classifier 35
5.5. Distant Supervision Program 35
5.6. TSABL 36
5.6.1. Fetching Twitter Data 36
5.6.2. Preprocessing 36
5.6.3. Training Word Embeddings 36
5.6.4. Training and Testing Classifiers 38

6. Distant Supervision of Tweets **39**
6.1. Filtering 40
6.2. Methods 41
6.2.1. Emoticons 41

6.2.2. Emoticons Extended . . . 42
6.2.3. AFINN . . . 42
6.2.4. TextBlob . . . 42
6.2.5. VADER Sentiment Analysis . . . 43
6.2.6. Combo Average . . . 44
6.2.7. Fredriksen–Jahren Lexicon Classifier (FJLC) . . . 44
6.3. Grid Parameter Searches . . . 45
6.3.1. VADER . . . 45
6.3.2. TextBlob . . . 45
6.3.3. Combo Average Method . . . 46
6.4. Comparisons . . . 46
6.4.1. Runtime . . . 48
6.4.2. Prediction Quality . . . 48

7. Optimizing System **49**
7.1. Optimization Plan . . . 49
7.2. Experimental Setup . . . 49
7.2.1. Datasets for Training Word Embeddings . . . 49
7.2.2. Datasets for Testing and Training Classifiers . . . 50
7.2.3. Preprocessing . . . 50
7.2.4. Classifier . . . 51
7.2.5. Hyperparameters . . . 51
7.3. Hyperparameter Search Results . . . 52
7.3.1. Alpha . . . 53
7.3.2. Context Window Size . . . 54
7.3.3. Embedding Length . . . 55
7.3.4. Hidden Layer Size . . . 56
7.3.5. Learning Rate . . . 57
7.3.6. Margin . . . 58
7.3.7. Summary of Best Parameters . . . 59
7.4. Comparing Distant Supervision Methods . . . 59
7.5. SVM Hyperparameter Search Results . . . 61

8. Evaluating the Final System **63**
8.1. Experimental Setup . . . 63
8.1.1. Final System . . . 63
8.1.2. Datasets for Testing and Training Classifiers . . . 64
8.2. Comparison of Distant Supervision Methods . . . 65
8.3. Comparison with Hybrid Ranking Model . . . 65
8.4. Comparison with Baselines . . . 66
8.4.1. Description of Baselines . . . 66
8.4.2. Results . . . 68
8.5. Comparison with Published Results . . . 68
8.5.1. Comparison with Tang et al. (2016) . . . 68

8.5.2. Comparison with SemEval 69

9. Discussion 71

9.1. Distant Supervision 71
9.1.1. Combination Methods 71
9.1.2. Speed versus Quality 72
9.2. Optimizing System 72
9.2.1. Limitations of the Hyperparameter Search 72
9.2.2. Data for the Hyperparameter Search Classifier 73
9.2.3. Evaluation of Hyperparameter Search Results 73
9.2.4. Distant Supervision Method 73
9.3. Comparison of Distant Supervision Methods 74
9.4. Comparison with Hybrid Ranking Model 75
9.5. Comparison with Baselines 76
9.6. Comparison with Published Results 77
9.6.1. Comparison with Tang et al. (2016) 77
9.6.2. Comparison with SemEval 77

10. Conclusion and Future Work 79

10.1. Distant Supervision 79
10.2. Ternary Sentiment Embedding Model 79
10.3. Future Work 80
10.3.1. Distant Supervision 80
10.3.2. Exhaustive Optimization 80
10.3.3. Investigate Impact of Using More Data 81
10.3.4. Word-Sense Disambiguation 81
10.3.5. Word-Specific Sentiment 81

Bibliography 82

A. Hyperparameter Search 89

A.1. Alpha 90
A.2. Context Window Size 91
A.3. Embedding Length 92
A.4. Hidden Layer Size 93
A.5. Learning Rate 94
A.6. Margin 96

B. SemEval 2017 Results 99

List of Figures

2.1. The word2vec models. 11

3.1. The Collobert and Weston model. 20
3.2. The Hybrid Ranking Model. 22

4.1. The Ternary Sentiment Embedding Model. 28

5.1. Architecture of the Fredriksen–Jahren Lexicon Classifier (FJLC) system. . 33
5.2. Overview of the Twitty Pipeline. 34

7.1. α comparison. 53
7.2. Context window size comparison. 54
7.3. Embedding length comparison. 55
7.4. Hidden length comparison. 56
7.5. Learning rate comparison. 57
7.6. Margin comparison. 58
7.7. Macro F_1 scores for the Ternary Sentiment Embedding Model on different datasets. 60

A.1. Macro F_1 scores for the Ternary Sentiment Embedding Model with different values for α. 90
A.2. Macro F_1 scores for the Ternary Sentiment Embedding Model with different context window sizes. 91
A.3. Macro F_1 scores for the Ternary Sentiment Embedding Model with different embedding lengths. 92
A.4. Macro F_1 scores for the Ternary Sentiment Embedding Model with different hidden layer sizes. 93
A.5. Macro F_1 scores for the Ternary Sentiment Embedding Model with different learning rates. 94
A.6. Macro F_1 scores for the Ternary Sentiment Embedding Model with different margins. 96

List of Tables

2.1. Preprocessor elements and tokens. 14

3.1. SemEval 2016 Task 4A Results 24

6.1. Statistics for SemEval training and test sets. 40
6.2. Sentiment distribution for the SemEval datasets. 40
6.3. Emoticons mapped to positive and negative by the Twitter API. 41
6.4. Emoticons and emojis mapped to positive and negative by the Emoticons Extended method. 43
6.5. Results for the VADER method. 45
6.6. Results for the TextBlob method. 45
6.7. The best results for the Combo Average method. 46
6.8. Comparison of F_1 scores for distant supervision methods. 47
6.9. Comparison of precision and recall for distant supervision methods. 47
6.10. Comparison of Macro F_1 scores for distant supervision methods per dataset. 47

7.1. Distribution of sentiment for distant supervised datasets. 50
7.2. Datasets for optimization of the Ternary Sentiment Embedding Model . . 50
7.3. Hyperparameter search values. 52
7.4. α comparison values. 53
7.5. Context window size comparison values. 54
7.6. Embedding length comparison values. 55
7.7. Hidden length comparison values. 56
7.8. Learning rate comparison values. 57
7.9. Margin comparison values. 58
7.10. Best and selected hyperparameters. 59
7.11. Macro F_1 scores for the Ternary Sentiment Embedding Model using different distant supervision methods. 60
7.12. Coarse parameter search for C values between 0.001 and 1000. 61
7.13. Parameter search for C values between 0.001 and 0.009. 61
7.14. Parameter search for C values between 0.01 and 0.09. 61

8.1. Hyperparameter values of final model. 63
8.2. Datasets for training and testing classifier. 64
8.3. Combined datasets for training and testing classifier. 64
8.4. Comparison of distant supervision methods. 65

8.5. Comparison between the Ternary Sentiment Embedding Model and the Hybrid Ranking Model using different distant supervision methods. 66
8.6. Comparison between the Ternary Sentiment Embedding Model and baselines. 68
8.7. Comparison with published results from Tang et al. (2016). 69
8.8. Comparison with different SemEval years. 69

A.1. Macro F_1 scores for the Ternary Sentiment Embedding Model with different values for α. 90
A.2. Macro F_1 scores for the Ternary Sentiment Embedding Model with different context window sizes. 91
A.3. Macro F_1 scores for the Ternary Sentiment Embedding Model with different embedding lengths. 92
A.4. Macro F_1 scores for the Ternary Sentiment Embedding Model with different hidden layer sizes. 93
A.5. Macro F_1 scores for the Ternary Sentiment Embedding Model with different learning rates. 95
A.6. Macro F_1 scores for the Ternary Sentiment Embedding Model with different margins. 97

B.1. SemEval 2017 Task 4A Results 99

1. Introduction

The effort needed for a person to publish his or her written opinion and reach an audience of thousands has become minuscule. Traditionally, people who wanted their meaning or work published and distributed had to go through certain media like newspapers and broadcast radio or TV. Today, Internet has opened the opportunity for everyone to publish whatever they desire. For instance, video creators can easily publish their work on YouTube, artists can publish their songs on SoundCloud. And ordinary people can publish their thoughts and opinions in social media. The phenomenon of *blogs*, where people often write about their personal lives, has long been tremendously popular.

More relevant to this thesis is the concept of *microblogs*. Microblogs are small documents containing the author's opinion or thoughts. Microblogging has grown popular due to social media like Twitter and Facebook. The abundance of microblogs being published every second makes it possible to get an insight into the public opinion on and reactions to events in the world in real-time.

1.1. Twitter

Twitter is a social medium where the users post microblogs. It is one of the most popular websites in the world, having approximately 313 million monthly active users as of June 2016, according to their home page[1].

Twitter is quite well known for the strict 140 character limit on tweets (Twitter posts). This limitation of length forces users to write posts in an informal language, often to-the-point, grammatically incorrect, with slang, typos, emoticons and abbreviations. It is also common to use Twitter-specific elements like user mentions (*@username*) to target another Twitter user and hashtags (*#tag*) to symbolize topics. Usually, all content in a tweet has counted toward this limit, but now Twitter is changing which elements count[2]. For instance, user mentions, URLs and media like videos and images, will no longer count toward the limit. This will enable the users to write longer tweets. It will be interesting to see if and how this changes the way people compose their tweets.

The fact that tweets are so to-the-point, with the challenges posed by their informal language, make them an interesting field of study within natural language modeling and sentiment analysis.

[1] `https://about.twitter.com/company`
[2] `https://blog.twitter.com/express-even-more-in-140-characters`

1.2. Motivation

The popularity of *word embeddings* have been boosted in recent years. Since the introduction of *word2vec* (Mikolov et al., 2013a), which is much faster to train than its predecessors, word embeddings have become ubiquitous in the field of Natural Language Processing. This can be seen in the International Workshop on Semantic Evaluation (SemEval) meetings of recent years. In 2016, eight out of the top ten performing systems used word embeddings as part of their architecture (Table 3.1, p. 24). This motivated us to further investigate the use of word embeddings as part of a Twitter Sentiment Analysis system.

Word embeddings learn word representations by looking at the contexts in which words appear, where the *context* of a word is defined by which other words are its neighbors in a text. This falls short for sentiment analysis, as words that appear in similar contexts might have entirely different sentiments associated with them. An example is the words “good” and “bad”. These words often appear in similar contexts, although they are of opposite sentiment. Traditional word embedding methods would likely create similar vectors for these words. Tang et al. (2014) presented a new model that employs both context and sentiment information in word embeddings. These embeddings are called Sentiment-Specific Word Embeddings (SSWE) or *Sentiment Embeddings*.

The models for training SSWE only consider positive and negative sentiment of tweets. The data on which Tang et al. train SSWE are tweets weakly labeled as either positive or negative using a simple distant supervision method. Tang et al. (2014) point out that SSWE performs better when used for the binary classification task, i.e. classifying tweets as “positive” or “negative”, compared to the ternary task where tweets can also be classified as “neutral”. This motivated us to look at a ternary variation of the model for training sentiment embeddings. In order to train a ternary model, a distant supervision method that can weakly label tweets for all three classes is needed. Distant supervision of neutral tweets is considered an open problem (Tang et al., 2016). The possible performance gain for the ternary task motivated us to compare and consider different distant supervision methods for a large corpus of tweets that can be used to train sentiment embeddings with a ternary model.

1.3. Goals

G.1: Investigate Approaches to Distant Supervision of Tweets

The idea of distant supervision is to automatically classify large amounts of data without the need for human intervention. Traditional methods of doing distant supervision of sentiment of tweets have used the occurrence of simple emoticons to guess negative or positive sentiment. We aim to improve distant supervised datasets for Twitter Sentiment Analysis on the ternary sentiment classification task by investigating methods of automatically annotating both positive, negative and neutral tweets from a large corpus of Twitter data.

G.2: Improve Sentiment Embeddings for Ternary Sentiment Classification

The Hybrid Ranking Model for training Sentiment-Specific Word Embeddings (SSWE) of Tang et al. (2014) shows promising results for binary sentiment classification. We aim to improve performance on the ternary sentiment classification task by both developing a new model architecture with a new loss function and by training on three-way classified distant supervised data.

1.4. Contributions

C.1 Comparison of Distant Supervision Methods

C.2 The Ternary Sentiment Embedding Model

C.3 Produced Tools and Programs

CATTS (Section 5.2). A web application for manually annotating tweets.

`https://catts.byrkje.land`

`https://github.com/draperunner/catts`

Distant Supervision Program (Section 5.5). A program for saving and comparing distant supervision methods.

`https://github.com/draperunner/distant-supervised-tweets`

FJLC (Section 5.3). A Python port of the Lexicon Classifier of Fredriksen and Jahren (2016).

`https://github.com/draperunner/fjlc`

TSABL (Section 5.6) A full Twitter Sentiment Analysis program that can fetch tweets, preprocess tweets, train word embeddings and classify tweets.

Tweet Collector (Section 5.1). A program for fetching and saving tweets.

`https://github.com/draperunner/tweet-collector`

1.5. Thesis Structure

Chapter 2 introduces important concepts that are used throughout the thesis to provide a knowledge basis.

Chapter 3 explains what work has been done in the fields of Twitter Sentiment Analysis, word embeddings, and sentiment embeddings.

Chapter 4 describes the proposed Ternary Sentiment Embedding Model for training ternary sentiment embeddings.

Chapter 5 describes the programs developed as part of this thesis.

Chapter 6 introduces a set of distant supervision methods and a comparison between them.

Chapter 7 comprises hyperparameter searches and dataset comparisons for finding the optimal setup for the Ternary Sentiment Embedding Model.

Chapter 8 is where the Ternary Sentiment Embedding Model is compared against baselines and other methods to establish its performance.

Chapter 9 contains discussion of the results achieved, the challenges met, and possible improvements.

Chapter 10 provides a conclusion of the thesis, as well as description of future work.

2. Background Theory

In this chapter some important concepts that are used throughout the thesis are explained. First, some common stages of preprocessing of text corpora are covered, followed by machine learning classifiers. Evaluation techniques and metrics are then introduced. In Section 2.4, word embeddings and the common word embedding methods *word2vec* and *GloVe* are explained. Lastly, tools and libraries employed in the project implementations are described in Section 2.5.

2.1. Textual Preprocessing

Natural Language Processing (NLP) is the field concerned with analyzing and generating natural language — the language you and I speak, read and write — to extract sentiment and meaning automatically using machines. Words in their raw form are not always best suited for natural language processing directly. Often some preprocessing is necessary. In this section some common methods are explained.

2.1.1. Part-of-Speech Tagging

Part-of-speech (POS) tagging is the procedure of classifying words to their appropriate part-of-speech. Examples of common parts-of-speech are nouns, verbs and adjectives. Usually taggers can give more detailed tags like for instance conjugation for verbs, and forms for nouns. There are POS taggers specialized for tweets (Owoputi et al., 2013).

2.1.2. Bag-of-Words

The bag-of-words model is a popular object categorization method (Zhang and Mayo, 2010) that takes words and counts the occurrences of each distinct word found in a set of text documents to generate a histogram. As an example, consider the simple text document "Paul likes movies. Anne likes music". The set of distinct words are {Paul, likes, movies, Anne, music}. The resulting vector from counting the occurrences of words will be: [1, 2, 1, 1, 1].

2.1.3. Stop Word Removal

The most common words in English appear in the majority of documents, and therefore rarely help in discriminating between them. Thus, it is common to remove these words, called *stop words*. Examples of stop words may be "the", "a", "is", and "to".

2.1.4. Stemming

Stemming is the procedure of reducing words to their *stem*. This makes words that are variations of the same base form equal, so that they can be treated as the same word. For example, words like “walk”, “walker”, and “walked” would be reduced to the stem “walk”.

2.1.5. Reduce Elongated Words

In informal contexts, such as Twitter, users sometimes write elongated words with repeating letters to emphasize their meaning. For instance, the word “sweeeeeeet” is likely to carry a more positive sentiment than just the word “sweet”. Such words can be reduced in order to recognize and standardize the words. The repeating letters are typically reduced to two or three occurrences. For instance, “sweeeeeeet” can be reduced to “sweet” or “sweeet”. By reducing to three letters, the reduced word is still different from the correctly spelled word, so that its emphasis is preserved.

2.2. Machine Learning Algorithms

Machine learning classifiers are computer programs that try to create a classification function from some training data in order to classify different categories in the data. There are many different kinds of machine learning algorithms, and some are more suited to text classification tasks than others. In the following subsections, a brief overview of some relevant machine learning algorithms is provided.

2.2.1. Support Vector Machines

Cortes and Vapnik (1995) introduced the *support-vector network* learning machine for binary classification problems. Support Vector Machines (SVMs) work by mapping the input vectors to a high-dimensional space, in which a linear decision surface is generated. To efficiently find a decision surface that generalizes well in a high-dimensional space, the idea of *optimal hyperplanes* is used. An optimal hyperplane in the context of SVMs is defined as the linear decision function that maximizes the margin between the vectors of the two classes. Only a few vectors, the *support vectors*, need to be considered to find this margin.

The SVM has a parameter C, which is used to tell the classifier how to balance finding a hyperplane with a large margin that separates classified samples and the amount of misclassified samples. Having a smaller C value means the classifier allows more misclassified samples in favor of a larger margin.

2.2.2. Maximum Entropy

In the context of machine learning methods, a *Maximum Entropy* (MaxEnt) classifier is a Logistic Regression classifier that generalizes to multiclass problems (Nigam et al., 1999). The classifier is also called *multinomial logistic regression.* The MaxEnt classifier is a probabilistic classifier that, compared to other probabilistic classifier such as Naïve Bayes, makes minimum assumptions. MaxEnt is based on the Principle of Maximum Entropy, and selects the model that fits the training data with the largest entropy. The classifier does not assume conditional independence of features, an assumption made by Naïve Bayes, and is as such well suited for text classification tasks, since the words in a sentence typically are not independent.

2.3. Evaluation

This section describes how a machine learning model can be evaluated and some metrics that are often used in evaluation of classification systems.

2.3.1. Techniques

The standard evaluation procedure of machine learning models splits data into three categories: a *training* set, a *validation* set and a *test* set. The training set is used for adjusting the weights of the model, in order to improve the accuracy on the training set. The validation set is used to avoid overfitting to the training data. The validation is not part of the training data. By regularly evaluating against the validation data, one can validate that the model is improving its scores on both the training data and the independent validation data. This is useful when testing different parameters of the model. The test set is used to evaluate the finalized model.

Unwanted bias can be introduced to the model depending on how the data has been split into the three categories. Optimally one would want all sets to represent the general case as closely as possible. Consider the extreme case where the training set contains only samples of class A, the validation set only class B, and the test set class C. The trained model would only know about the class A and be unable to classify B and C. To avoid such bias, certain statistical methods have been developed, called *cross-validation.*

A common cross-validation method is k-fold cross-validation. In this method the data is split into k mutually exclusive sets (*folds*). Each of these sets are used as the test set once, while the rest are used as training sets. The test scores are then averaged over the k runs.

2.3.2. Metrics

A set of evaluation measures is needed to compare different classification models. For sentiment analysis systems it is interesting to score a system based on how good it is at classifying documents correctly. The measures *precision*, *recall* and *F-score* are often used for such evaluation tasks. These measures use the numbers of true positives, true negatives, false positives and false negatives to calculate their value. True positives (tp) is the number of correctly classified positive samples, while true negatives (tn) is the same for negative samples. False positives (fp) is the number of falsely classified positive samples, while false negatives (fn) is the same for negative samples.

Precision

Precision measures how many of the samples classified as positive (or negative for the opposite case) that are correctly classified. If every sample that returns positive actually is correctly classified, the precision is 1. The formula for precision is given as:

$$precision = \frac{tp}{tp + fp} \tag{2.1}$$

Recall

Recall measures how many of the true positive (or negative) samples that are classified correctly. If the classifier returns all the true positive samples as positive, the recall is 1. Recall is defined as:

$$recall = \frac{tp}{tp + fn} \tag{2.2}$$

F-score

Precision and recall are combined in a weighted average to give the F-score. The general definition of F-score is given as:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.3}$$

where β is the weight. The traditional balanced F-score uses $\beta = 1$, as this gives the harmonic mean of precision and recall.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.4}$$

F_1^{PN}-score

The F_1^{PN}-score is a modified version of the traditional F-score that is used by the International Workshop on Semantic Evaluation (SemEval) to score and rank the participating systems on the three-way Twitter Sentiment Analysis task. The approach is to calculate F-scores for positive samples and negative samples and then average the two. Since each sample must be classified as belonging to one of three classes, precision and recall are also slightly altered. Precision for the positive class is now given as:

$$precision^P = \frac{PP}{PP + PU + PN} \tag{2.5}$$

where PP is true positive samples classified as positive (the same as true positives for normal precision), PU is true neutral samples classified as positive and PN is true negative samples classified as positive. Recall is now given as:

$$recall^P = \frac{PP}{PP + UP + NP} \tag{2.6}$$

where PP is as for precision above, while UP is true positive samples classified as neutral and NP is true positive samples classified as negative. The F_1-score for the positive class is then calculated by using Equation 2.4 with the recall and precision for the positive class. The F_1-score for the negative class is defined analogously. F_1-scores for positive and negative samples are averaged to get the F_1^{PN}-score.

$$F_1^{PN} = \frac{F_1^{POS} + F_1^{NEG}}{2} \tag{2.7}$$

Macro F_1-score

While the F_1^{PN}-score is the official metric of SemEval for both the binary and ternary classification tasks, one can argue how representative it is for the latter. Another metric used by Tang et al. (2016) is the *Macro* F_1-score, which extends F_1^{PN} to average over both the positive, negative and neutral F_1-scores.

$$\mathit{Macro}\ F_1 = \frac{F_1^{POS} + F_1^{NEG} + F_1^{NEU}}{3} \tag{2.8}$$

Here F_1^{NEU} is analogous to the F_1^{POS} and F_1^{NEG} scores above, but for the neutral class.

AvgRec

The SemEval workshop changed its primary scoring metric in 2017 to a new score called AvgRec for average recall (Rosenthal et al., 2017).

$$\mathit{AvgRec} = \frac{\mathit{recall}^{POS} + \mathit{recall}^{NEG} + \mathit{recall}^{NEU}}{3} \tag{2.9}$$

where recall^{POS}, recall^{NEG} and recall^{NEU} are the recall scores for the positive, negative and neutral classes, respectively.

2.4. Word Embeddings

Word embeddings are techniques for representing words as low-dimensional real-valued vectors that capture semantic and lexical properties of the words. The vectors themselves can be used for computing similarities between words or phrases, and as input features for other NLP tasks such as text classification, part-of-speech tagging and sentiment analysis. Word embeddings have been found to be good at capturing syntactic and semantic regularities in language (Mikolov et al., 2013c). The regularities are constant vector offsets between pairs of words sharing a particular relationship, and can be found by applying simple algebraic operations to the word vectors. A typical example of this is that $\mathit{vector}(\text{“}\mathit{King}\text{”}) - \mathit{vector}(\text{“}\mathit{Man}\text{”}) + \mathit{vector}(\text{“}\mathit{Woman}\text{”})$ results in a vector which is close to $\mathit{vector}(\text{“}\mathit{Queen}\text{”})$. Closeness between word vectors is measured using cosine similarity.

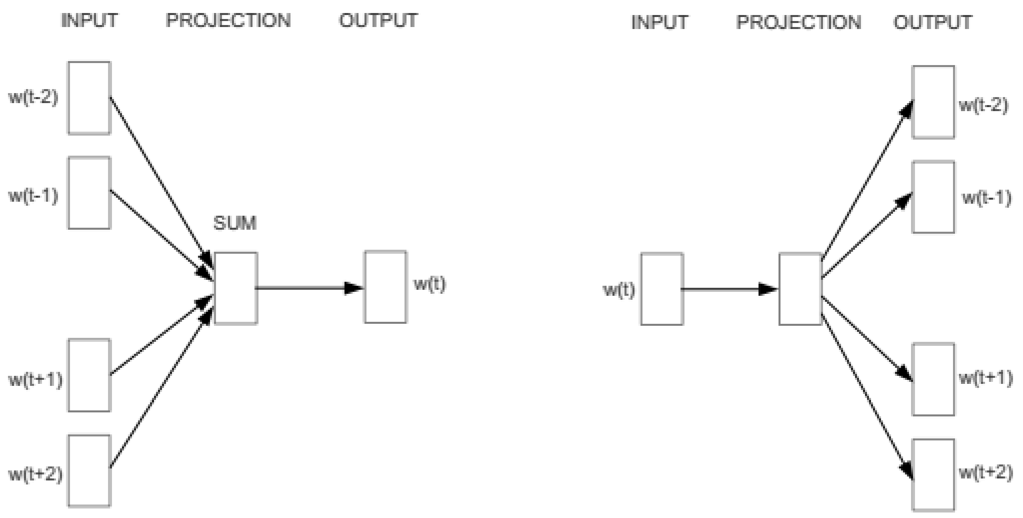
The idea of representing words as vectors has a long history (Firth, 1957; Hinton, 1986), and in the 1990s methods for using automatically generated contextual features were developed. One such method is Latent Semantic Analysis (LSA) (Deerwester et al., 1990) where feature vectors for words are learned on the basis of their probability of co-occurring in the same documents. These methods are also called Matrix Factorization Methods.

The word embedding technique was introduced by Bengio et al. (2003). They trained what they called *learned distributed feature vectors* as part of a neural language model. The model consisted of an embedding layer, hidden layers and a softmax layer. The embedding layer in their model was a matrix consisting of free variables that was shared for all the input words. This matrix was multiplied with index vectors to create the word embeddings for each word. The matrix was learned jointly with the rest of the neural language model. The main bottleneck in this neural language model was identified to be the use of the softmax layer, which is proportional to the number of words in the vocabulary.

Collobert and Weston (2008) showed the utility of using pre-trained word embeddings. They presented a single Convolutional Neural Network architecture that was trained jointly on multiple NLP tasks. The model did away with the expensive softmax layer by using a different objective function. The network was trained to output a higher score for correct word sequences than incorrect ones. Correct word sequences can be found by looking at all possible sequences (or context windows) in the corpus, while incorrect ones are created by replacing the focus word of a sequence with another word from the vocabulary. The objective was to maximize distance between the scores output for the correct and incorrect sequences. The word embeddings were used and trained jointly with the rest of the model, in the same manner as for Bengio et al. (2003). By using a less expensive objective function, the model can be trained on a much larger dataset. Collobert et al. (2011) proposed a unified multilayer neural network that can be applied to various natural language processing tasks. This model architecture is described in detail in Section 3.2 (p. 18).

Mikolov et al. (2013a) and Mikolov et al. (2013b) introduced the word2vec model. The word2vec model is presented in greater detail in Section 2.4.1. They observed that most of the complexity from previous models was caused by the use of a non-linear hidden layer in the model. The new architectures presented aimed to do away with these computationally expensive layers.

Pennington et al. (2014) argued that the models for learning word representations using a limited context window poorly utilized the statistics of the corpus. Global matrix factorization methods, such as LSA, are better at using statistics from the corpus, but they do worse on word analogy tasks and are more expensive to train. Pennington et al. proposed a low rank approximation of co-occurrence statistics in their system called GloVe which they claimed outperformed other models on word analogy, word similarity, and named entity recognition tasks. This claim was challenged by Levy et al. (2015) who stated that the Skip-gram model from word2vec trained with negative-sampling outperformed GloVe on every task.



(a) The CBOW model. (b) The Skip-gram model.

Figure 2.1.: The word2vec models. Figure from (Mikolov et al., 2013a).

2.4.1. word2vec

The word2vec model presented by Mikolov et al. (2013a) introduces two new log-linear architectures for learning word embeddings: the Continuous Bag-of-Words model and the Continuous Skip-gram model. The models can be seen as simple neural networks with a single hidden layer. The training is unsupervised, meaning that there is no label to use in the objective function. Instead, the target word functions as the label. This form of unsupervised training is similar to unsupervised feature learning with auto-encoders. The similarity is that a neural network with hidden layers is trained in an unsupervised manner, where the goal is to extract the weights of the resulting hidden layers. Each input word is encoded with a 1-of-N (also called “one-hot”) encoding. Each input vector has length equal to the size of the vocabulary, with a *1* at the position that represents the word and with *0*s for the rest of the vector. When these vectors are multiplied by a matrix, the resulting vector is the row from the matrix at the position of the 1 in the vector. Therefore, for the word2vec models, the resulting weights in the hidden layer are the word embeddings that are pursued. Mikolov et al. (2013b) also presented two approximations to the softmax function for calculating the posterior distribution of words that is more computationally efficient, called Hierarchical Softmax and Negative Sampling.

Continuous Bag-of-Words Model

The Continuous Bag-of-Words (CBOW) model uses a context window around a focus word in order to predict the word. The context window consists of words both before and after the focus word. All the context words are projected in a shared projection layer where the word embeddings are simply averaged. As for other bag-of-words models, the ordering of the context words does not matter. However, unlike other bag-of-words models, the architecture uses continuous distributed representation of the context. The model used by Mikolov et al. (2013a) is trained by using the four words before and the four words after the focus word. The training criterion is to correctly classify the focus word given its context. The model architecture is shown in Figure 2.1a.

Continuous Skip-gram Model

The continuous Skip-gram model is similar to the CBOW model, but here the objective function is opposite. While the CBOW model uses the context of a word to try to predict the word, the Skip-gram model uses the focus word to predict the surrounding words. The training objective is to minimize the summed predictions error across the context words. The model architecture can be seen in Figure 2.1b. Each word is used as input to the classifier that predicts the words within a range before and after the word. The size of the range can be increased for better quality of the resulting word embeddings, but it increases the computational complexity (Mikolov et al., 2013a). The context words are weighted by their distance from the focus word in the context window. According to Mikolov[1], the Skip-gram model works well with small amounts of training data and gives better representations even for rare words and phrases, compared to the CBOW model. However, the Skip-gram model is not as fast to train and it gives slightly worse accuracy for frequent words.

2.4.2. GloVe

Pennington et al. (2014) present a *Global Vectors* model called GloVe. The model means to combine the benefits from global matrix factorization models like LSA and local context window methods like the word2vec models. Matrix factorization methods create large matrices that capture statistical information about a corpus, which are then decomposed to low-rank approximations. In some models, the large matrices capture when words appear together in a context, such as term-term co-occurrence matrices. These methods are good at capturing global statistical properties from a corpus, while local context window methods are better at word similarity tasks.

The GloVe model creates a word-word co-occurrence matrix X where each entry in the matrix X_{ij} is the number of times word j occurs in the context of word i. The probability that a word k appears in the context of the word w is given as $P(k|w)$ and can be calculated by using the word-word co-occurrence matrix. Pennington et al. show that ratios of the co-occurrence probabilities with various probe words give better performance than by using only the probabilities. The training objective of the GloVe model is then to learn word vectors such that their dot products equal the logarithm of the words' probability of co-occurrence. This constraint can be written as:

[1]`https://groups.google.com/forum/#!searchin/word2vec-toolkit/c-bow/word2vec-toolkit/NLvYXU99cAM/E51d8LcDxlAJ`

$$w_i^T w_j + b_i + b_j = log(X_{ij}) \tag{2.10}$$

where w_i is the vector for the focus word, w_j is the vector for the context word, b_i and b_j are scalar biases for the focus word and the context word. The constraint is weighted by a weighting function that helps preventing learning from extremely common word pairs.

2.5. Tools

This section provides a description of the tools and libraries that are used for creating the programs in this thesis.

2.5.1. scikit-learn

scikit-learn is a machine learning library for the Python programming language[2]. The library contains a wide range of state-of-the-art machine learning classifiers for both supervised and unsupervised learning (Pedregosa et al., 2011). Among the classifiers are SVMs and MaxEnt (described in Section 2.2). The LinearSVC classifier is an SVM classifier with a linear kernel that uses the LIBLINEAR[3] library for large linear classification.

2.5.2. MultiVec

MultiVec (Bérard et al., 2016) is an implementation of word2vec, bivec and paragraph vector in C++, with a Python wrapper. It supports training both mono- and bilingual word embedding models. MultiVec supports most of word2vec's features, and is easy to use in conjunction with other Python modules. Both the CBOW and Skip-gram models are available for word2vec. MultiVec is open-source under the Apache 2.0 License and is available on GitHub[4].

2.5.3. glove-python

glove-python is a Python library for training GloVe word embedding models, which is available on GitHub[5]. It uses asynchronous stochastic gradient descent (SGD), and is implemented in Cython. It is published as a Python package on the Python Package Index, making it easy to install and incorporate in other Python programs.

2.5.4. Twokenize

Twokenize is a tokenizer designed for Twitter text and is written in Java. It is part of the TweetNLP[6] (Owoputi et al., 2013) set of tools for natural language processing of tweets.

[2]`http://scikit-learn.org/stable/`
[3]`https://www.csie.ntu.edu.tw/~cjlin/liblinear/`
[4]`https://github.com/eske/multivec`
[5]`https://github.com/maciejkula/glove-python`
[6]`http://www.cs.cmu.edu/~ark/TweetNLP/`

The tokenizer is ported to Python 2 under the name *ark-twokenize-py* and is available on GitHub[7]. The port has been slightly altered to work with Python 3 for this thesis. Twokenize splits a tweet into a vector with its constituent parts as well as separating punctuation from words. The tokenizer can also split contractions like “shouldn’t” to “should” and “n’t”.

2.5.5. Preprocessor

Preprocessor is a simple library that can tokenize and/or clean (remove) Twitter specific elements. It supports URLs, Hashtags, Mentions, Reserved words (“RT” and “FAV”), Emojis, and Smileys. With *tokenizing*, instances of these will be turned into defined tokens. The specific tokens along with examples of the specific elements can be seen in Table 2.1. By *cleaning*, instances will simply be deleted from the text. Configuration can be done to customize which elements to tokenize and which to clean. The Preprocessor library is available as a package on the Python Package Index, which makes it convenient to install. Its code is available on GitHub[8].

Instance	Example	Token
URL	http://example.com	URL
Mention	@username	$MENTION$
Hashtag	#topic	$HASHTAG$
Reserved Word	RT	$RESERVED$
Emoji	☺	$EMOJI$
Smiley	:)	$SMILEY$
Number	123	$NUMBER$

Table 2.1.: Twitter specific elements that can be removed or tokenized with Preprocessor.

2.5.6. Twitter API

Twitter provides various Application Programming Interfaces (APIs) for developers to use. The REST APIs let the developer fetch specific data of interest by sending HTTP requests. Representational State Transfer (REST) is an architecture for distributing data between hosts on the World Wide Web. REST was defined by Fielding (2000), and describes a set of stateless operations for data communication. The requirement of statelessness means that every request from client to server must contain all information necessary for the server to process the request. The Twitter REST APIs provide access to write and read Twitter data, from user profiles to tweets, including searching for certain tweets based on user defined queries.

For fetching tweets in real-time, Twitter provides the Streaming APIs. When connected to a stream, Twitter provides samples of tweets, without having to poll the REST APIs.

[7] `https://github.com/myleott/ark-twokenize-py`

[8] `https://github.com/s/preprocessor`

2.5.7. twit

twit[9] is a popular package for Node.js that can handle communication with both the Twitter REST and Streaming APIs. The most valuable part of twit is its interface to the Streaming API. Behind the scenes, it handles reconnections if the connection to the stream is lost, and it implements Twitter's guidelines to avoid breaking the API's rate limits.

2.5.8. AFINN

AFINN (Nielsen, 2011) is a sentiment lexicon that assigns each word a value between -5 (very negative) to 5 (very positive). It includes slang, smileys and annotations that are common in microblogs. AFINN is implemented as a Python library, which is available on GitHub[10]. It rates a whole tweet by evaluating each word and summing the scores. This means that a tweet might get a sentiment score greater than 5 or less than -5. Out-of-vocabulary words are assigned a score of 0, which is the same as for neutral words.

2.5.9. VADER

Valence Aware Dictionary and sEntiment Reasoner (VADER) by Hutto and Gilbert (2014) is a rule-based model for general sentiment analysis, tuned for microblogs. VADER returns a normalized 3D vector where each element represents a score for each of the sentiment classes positive, negative and neutral, respectively. VADER also gives a *compound* score, which is a single sentiment score from -1 (very negative) to 1 (most positive).

2.5.10. TextBlob

TextBlob is a Python library that supports various text processing tasks, like common NLP tasks and sentiment analysis. It is available on GitHub[11]. It has two sentiment analysis models implemented. The default is the *PatternAnalyzer*, which is based on the *pattern* Python library. Another implementation is the *NaiveBayesAnalyzer* which is an NLTK classifier trained on movie reviews. The PatternAnalyzer model produces a positive and negative sentiment score between -1 and 1 and a subjectivity score between 0 and 1.

2.5.11. Fredriksen–Jahren Lexicon Classifier

Fredriksen and Jahren (2016) wrote an automatic sentiment lexicon creator and a lexicon sentiment analysis system as part of their Master's thesis. Fredriksen and Jahren found that their best performing automatically created lexicon outperformed manually

[9] `https://github.com/ttezel/twit`
[10] `https://github.com/fnielsen/afinn`
[11] `https://github.com/sloria/textblob`

annotated lexica, and their sentiment analysis system produced good results comparable to sophisticated machine learning based systems. The runtime performance is significantly better than the compared systems, making it promising for real-time and distant-supervised classification. Fredriksen and Jahren open-sourced their work, which is available on GitHub[12]. To better be able to integrate the program with the programs of this thesis, the Java implementation was ported to Python. This is explained further in Section 5.3 (p. 32).

2.5.12. Theano

Theano (Alain et al., 2016) is a Python library for efficiently defining, optimizing and evaluating mathematical expressions involving multidimensional arrays. Theano allows the user to define calculations in a symbolic way, which are optimized and compiled to C code for fast and efficient evaluation. Because of this, Theano is a popular library used when defining and evaluating artificial neural networks. Theano is open-source and available on GitHub[13].

2.5.13. TensorFlow

TensorFlow is a software library developed by the Google Brain team at Google for expressing and implementing machine learning algorithms (Abadi et al., 2015). Calculations expressed using TensorFlow can be executed on a wide variety of systems, making it easier for the user to execute machine learning algorithms on for instance GPUs. TensorFlow was open-sourced under the Apache 2.0 open-source license on November 9, 2015 and is available on GitHub[14].

2.5.14. Keras

Keras is a high-level neural networks library that focuses on enabling fast experimentation for the Python programming language. With Keras, one can easily define layers, objective functions and training optimizers to easily build and test neural network architectures. Keras supports using either Theano or TensorFlow as backend for handling computation of multidimensional arrays. Switching between Theano and Tensorflow is possible through simple configuration. Likewise for switching between GPU and CPU processing. Keras is open-source under the MIT license and it available on GitHub[15].

[12]`https://github.com/freva/Masteroppgave`
[13]`https://github.com/Theano/Theano`
[14]`https://github.com/tensorflow/tensorflow`
[15]`https://github.com/fchollet/keras`

3. Related Work

Some of the recent work done in the fields of Twitter Sentiment Analysis and word embeddings will be presented in this chapter. Emphasis will be made on some of the historical arguments made in the development of the fields.

3.1. Twitter Sentiment Analysis

While sentiment analysis in news articles and blogs has been explored for a quite a while, the field of sentiment analysis in microblogs is fairly young. Go et al. (2009) claimed to be the first to analyze machine learning techniques in the specific field of microblogs, and pointed out that this might be because of Twitter's recent rise to popularity. Today, Twitter has been one of the largest social media for many years, and the field of Twitter Sentiment Analysis (TSA) has grown to become a popular topic in sentiment analysis.

3.1.1. Preprocessing

The overall procedure of doing TSA has been roughly the same since the start. First, the tweets are *preprocessed*. See Section 2.1 (p. 5) for descriptions of common textual preprocessing techniques. Go et al. (2009), for their unigram feature extractor, converted all URLs to the symbol "URL" and created an equivalence class for all user mentions (@username). Others remove the URL altogether (Gomez-Adorno and Sidorov, 2016; Steinskog and Therkelsen, 2016). To cope with elongated words ("loooong", "huuu-ungry", etc.), some translate these to their ordinary form ("long", "hungry") (Jiang et al., 2011; Deriu et al., 2016), while others reduce the repeated characters to two ("loong", "huungry") (Boag et al., 2015), so that elongated words can be separated from non-elongated words. The count of elongated words can be used as a feature in itself (Balikas and Amini, 2016).

Traditionally, minimal preprocessing has been done to tweets when training word embeddings. It has been assumed that the word embeddings would learn the similarities between words anyways, rendering the preprocessing steps redundant. But Gomez-Adorno and Sidorov (2016) showed that preprocessing improves word embedding vectors, making it relevant to explore various preprocessing methods in combination with word embeddings.

3.1.2. Feature Extraction

After textual preprocessing comes the extraction and selection of *features*. Go et al. (2009) compared unigram and bigram models and found that a combination performed better than the two individually. Agarwal et al. (2011) compared the unigram model with two other models: a feature based model and a tree kernel based model. The tree kernel based model outperformed the other two by a clear margin. The use of Parts-of-Speech (POS) tags as a feature is common. Go et al. (2009) reported that the tags increased the classification accuracy by almost six percent on the Stanford Classifier, but decreased the accuracy in the case of their Naïve Bayes classifier.

Emoticons have played a great role in TSA. Go et al. (2009) used them to annotate training data. Gomez-Adorno and Sidorov (2016) converted them to the words they represent – for instance, ":)" becomes "smile". Ræder (2016) and Steinskog and Therkelsen (2016) translated unicode emoticons to a smaller set of ASCII equivalents. Agarwal et al. (2011) reported that emoticons and hashtags help in classifying sentiment, but only marginally. Zhou et al. (2016), who had the top performing system of SemEval 2016 task 4A (Nakov et al. 2016; see Section 3.4, p. 23), also reported contributions from emoticons in all classifiers.

3.1.3. Machine Learning Classifiers

Various machine learning algorithms have been tested over the years. Support Vector Machines (SVM), Naïve Bayes and Maximum Entropy (MaxEnt) are common classifiers. Go et al. (2009) tried all of these three, with unigram and bigram feature models. Their unigram model with a multinomial Naïve Bayes classifier performed the best. Steinskog and Therkelsen (2016) compared seven different classifiers: a MaxEnt classifier, a Bernoulli and a Multinomial Naïve Bayes based classifier, SVMs with sigmoid, linear, radial basis function kernels, and a Stochastic Gradient Descent (SGD) classifier. Of these, the top performer on recall and F_1-scores was the MaxEnt classifier, the SGD being the only one to beat it on precision.

3.2. The Collobert and Weston Model

Collobert et al. (2011) proposed a task-general neural network architecture for natural language processing. To achieve versatility, they refrained from using man-made task-specific input features, and instead let their system learn an internal representation from vast amounts of unlabeled data. They do as little preprocessing as possible, and train their multilayer neural network architecture in an end-to-end fashion. The model architecture is illustrated in Figure 3.1. Henceforth this architecture will be referred to as the Collobert and Weston (C&W) architecture.

The architecture's first layer extracts features for each word. The second layer extracts features from a window of words, treating it as a sequence with local and global structure (i.e., it is not treated like a bag of words). The following layers are standard neural network layers. As the architecture is highly relevant for the works of Tang et al. (2014) and us, the layers will be described in detail. The same notations as in Collobert et al. (2011) are used.

The first layer is called the *Lookup Layer*. Words are fed into the architecture as their respective indices in a fixed-size vocabulary D. The indices are then mapped to vectors of a fixed size by indexing a lookup table $LT \in \mathbb{R}^{d_{wrd} \times |D|}$. Here d_{wrd} is the size of the desired vectors, or word embeddings, and is a hyperparameter that must be specified by the user. The parameters, or weights, of the lookup table are randomly initialized, and are, after the network has been trained, what constitutes the word embeddings for the words in the vocabulary D. Collobert et al. use the window approach to tag one word at the time. The window approach is based on the assumption that a word's tag is mainly dependent on its surrounding words. This window is called the *context window* of a focus word. When considering a word, its window of neighboring words is passed through the lookup layer, producing a feature vector for each word. The column vectors are concatenated to a matrix ${f_\theta}^1$ of size $d_{wrd} \times k_{sz}$, where k_{sz} is the size of the context window.

The next layer is the *Linear Layer*. The input vector ${f_\theta}^1$ can be passed through one or several layers that perform affine transformations over their inputs:

$$ {f_\theta}^1 = W^l {f_\theta}^{l-1} + b^l \tag{3.1}$$

where $W^l \in \mathrm{I\!R}^{n^l_{hu} \times n^{l-1}_{hu}}$ and $b^l \in \mathrm{I\!R}^{n^l_{hu}}$ are the parameters to be trained. The hyperparameter n^l_{hu} is usually called the number of hidden units of the l^{th} layer. The simplest C&W model uses only a single linear layer before the next, non-linear layer.

The next layer is the *HardTanh Layer*, which introduces some non-linearity to the model. A "hard" version of the hyperbolic tangent is chosen as it is slightly cheaper to compute compared to the real hyperbolic tangent, while it does not lose any generalization performance (Collobert and Weston, 2008). The HardTanh function is as follows:

$$ HardTanh(x) = \begin{cases} -1, & \text{if } x < -1 \\ 1, & \text{if } x > 1 \\ x, & \text{otherwise} \end{cases} \tag{3.2}$$

Finally, a *Linear Layer* is used to get an output vector with dimension equal to the available number of tags or classes for the given task. Each element of this vector is interpreted as a score for the corresponding tag. For the task of learning word embeddings from context information, the output vector of the final layer has size 1.

During training, the objective of the C&W model is to obtain a higher model score for a given context window than a corresponding corrupted context window. For each context window that is used to train the model, a corrupted context window is created by replacing the central focus word with a randomly drawn word from the vocabulary. Both the correct and the corrupted context windows are passed through the model, and the training objective is that the original context window is expected to obtain a higher model score than the corrupted by a margin of 1. The objective can be formulated as a hinge loss function written as:

$$ loss_{cw}(t, t^r) = max(0, 1 - f^{cw}(t) + f^{cw}(t^r)) \tag{3.3}$$

where t is the original context window, t^r is the corrupted context window and $f^{cw}(\cdot)$ is the calculated model score.

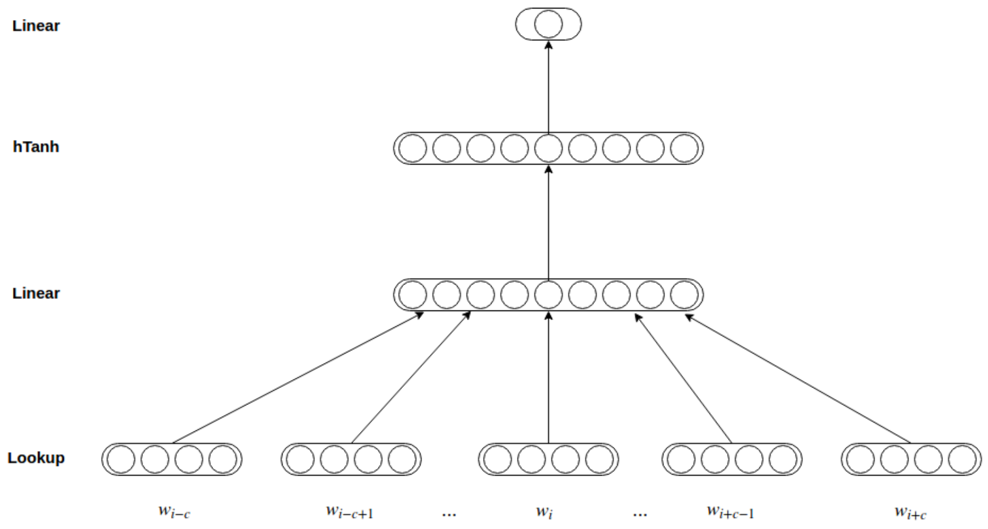


Figure 3.1.: The Collobert and Weston model. The feature vectors of the words in the context window are concatenated and fed to the linear layer. w_i is the focus word, and the context window size $k_{sz} = 2c + 1$.

3.3. Sentiment-Specific Word Embeddings

The word embedding methods described in Section 2.4 (p. 9) only consider the contexts, or collocation, of words to determine similarity. This works well for tasks like determining POS tags, as the grammatical role of words are heavily dependent on the surrounding words. For sentiment analysis, on the other hand, context does not necessarily pinpoint the sentiment of a word. Tang et al. (2014) described this problem and pointed out that words with opposite sentiment, for instance “good” and “bad”, often occurred in the same contexts, resulting in similar word embedding vectors.

To improve word embeddings for sentiment analysis, Tang et al. proposed Sentiment-Specific Word Embeddings (SSWE). They enhanced the Collobert and Weston (C&W) word embedding model (Section 3.2) by employing massive amounts of distant-supervised tweets. Instead of manually annotating tweets, they assigned tweets containing positive emoticons a positive label, and tweets containing negative emoticons a negative label. This way they were able to collect large amounts of tweets and assign a sentiment annotation automatically. Tang et al. proposed three different strategies of incorporating the sentiment information of the tweets into their embeddings, namely *Basic Model 1 (SSWE$_h$)*, *Basic Model 2 (SSWE$_r$)* and *Unified Model (SSWE$_u$)*. The Basic Models only look at sentiment polarity of sentences, leaving out the contexts of words, when training sentiment-specific word embeddings. The Unified Model combines both the contexts of words as well as the sentiment polarity of sentences to build its embeddings. Tang et al. (2016) later improved on the Unified Model by introducing the *Hybrid Ranking Model*.

3.3.1. Basic Model 1 (SSWE$_h$)

The Basic Model 1 (SSWE$_h$) follows the window-based C&W approach. The top layer of C&W is modified so that its output vector's dimension equals the number of possible labels K. Each element in the output vector defines a probability of a word's label. For instance, for the positive/negative sentiment classification task, [1, 0] would represent the positive label and [0, 1] would represent the negative. A softmax activation layer is put on top of the C&W top linear layer. According to Tang et al., the softmax layer is suitable for this scenario because its outputs are interpreted as conditional probabilities. The SSWE$_h$ model is trained by predicting the positive n-gram, or context window, as [1, 0] and the negative as [0, 1].

3.3.2. Basic Model 2 (SSWE$_r$)

The Basic Model 2 (SSWE$_r$) aims to improve the performance of SSWE$_h$ by relaxing its strict constraint. While SSWE$_h$ looks only at [1, 0] and [0, 1], SSWE$_r$ handles more fuzzy distributions. Say [0.7, 0.3] would be interpreted as a positive label, and [0.2, 0.8] would be interpreted as a negative. If the sentiment polarity of a tweet is positive, the predicted positive score is expected to be greater than the predicted negative score, and vice versa.

The SSWE$_r$ model borrows the bottom four layers from the SSWE$_h$ model, but has no need for the top softmax layer. It is removed because the SSWE$_r$ does not require probabilistic interpretation. The model's relaxed constraint is modeled with a ranking objective function. The function is given as:

$$loss_r(t) = max(0, 1 - \delta_s(t) f_0^r(t) + \delta_s(t) f_1^r(t)) \tag{3.4}$$

where f_0^r is the predicted positive score, f_0^r is the predicted negative score and $\delta_s(t)$ is a function that reflects the gold sentiment polarity of the context window t:

$$\delta_s(t) = \begin{cases} 1, & \text{if } f^g(t) = [1, 0] \\ -1, & \text{if } f^g(t) = [0, 1] \end{cases} \tag{3.5}$$

Here $f^g(t)$ is the correct sentiment label for the context window t, and $f^g(t) = [1, 0]$ means that the context window is labeled “positive”.

3.3.3. Unified Model (SSWE$_u$)

The Unified Model (SSWE$_u$) combines the context of words and the polarity of sentences. While the Basic Models did not use the *corrupted context window* training strategy of C&W, the Unified Model does. The objective function consists of two training objectives. The first one is that the original context window should obtain a higher language model score than the corrupted context window. The second is that the sentiment score of the original context window should be more consistent with the gold polarity annotation of the sentence than the corrupted context window. The loss function of the Unified Model is a weighted combination of two loss functions:

$$loss_u(t, t^r) = \alpha \cdot loss_{cw}(t, t^r) + (1 - \alpha) \cdot loss_{us}(t, t^r) \tag{3.6}$$

where $0 \leq \alpha \leq 1$ weights the two parts and $loss_{cw}$ is the loss function from the C&W model given by Equation 3.3. The function $loss_{us}$ is given by:

$$loss_{us}(t, t^r) = max(0, 1 - \delta_s(t) f_1^u(t) + \delta_s(t) f_1^u(t^r)) \tag{3.7}$$

where $\delta_s(t)$ is as given in Equation 3.5. The function is similar to Equation 3.4, but now compares the predicted sentiment scores for the original and the corrupted context window.

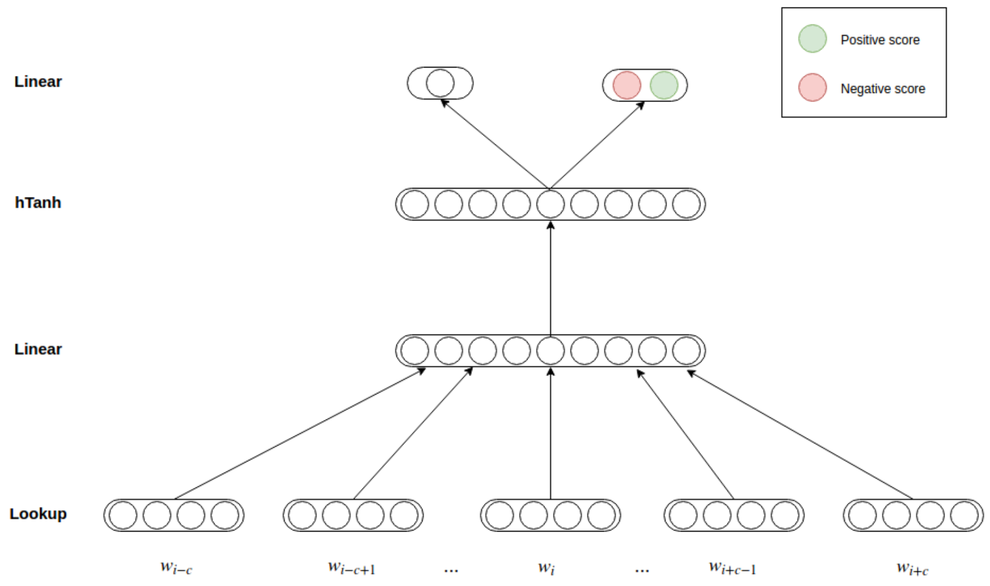


Figure 3.2.: The Hybrid Ranking Model.

3.3.4. Hybrid Ranking Model

Tang et al. (2016) present a model similar to the Unified Model, but with a slightly altered objective function. The sentiment loss function is now as in Equation 3.4 (p. 21), meaning that the model no longer looks at the predicted sentiment score for the corrupted context window, it only compares the predicted positive and negative score for the correct context window when calculating the loss. The new loss function is given as:

$$loss_{hyRank} = \alpha_{rank} \cdot loss_{sRank} + (1 - \alpha_{rank}) \cdot loss_{cRank} \tag{3.8}$$

where $0 \leq \alpha_{rank} \leq 1$ weights the two functions, $loss_{sRank}$ is equivalent to $loss_r$ given by Equation 3.4 and $loss_{cRank}$ is equivalent to $loss_{cw}$ given by Equation 3.3 (p. 19). The Hybrid Ranking Model is illustrated in Figure 3.2. The figure shows the feature vectors for each word in a context window being concatenated and fed to the linear layer, similarly to the C&W model in Section 3.2 (p. 18). The top linear layer consists of the context-aware layer on the left and the sentiment-aware layer on the right. The context-aware layer calculates the context score f^{cw}, while the sentiment-aware layer calculates the sentiment scores f^{r} for the input context window.

3.4. The International Workshop on Semantic Evaluation

The International Workshop on Semantic Evaluation (SemEval) is an ongoing series of evaluations of computational semantic analysis systems. The workshop has since 2012 been hosted annually, and has a growing number of tasks and subtasks. The SemEval workshop has provided a task for Twitter Sentiment Analysis (TSA) since 2013. The workshop has provided training data and a platform to test and compare different TSA systems, which has led to significant advancements to the state-of-art in the field.

The SemEval workshop was most recently held in 2017. At the time of writing this thesis, the proceedings from the workshop are yet unreleased and a further analysis of the submitted systems is not provided. The task paper for the TSA task (Rosenthal et al., 2017) provides a summary of the results, and the top ten performing systems with their score are shown in Appendix B.

The SemEval workshop held in 2016 provided a task for TSA systems called "Task 4: Sentiment Analysis in Twitter". The task consisted of five subtask, where Subtask A is of most interest to us. Task 4A is "Given a tweet, predict whether it is of positive, negative, or neutral sentiment." (Nakov et al., 2016). The top ranking systems for Task 4A in SemEval 2016 are presented in Table 3.1. The systems were scored using a variation of the F_1-score, called F_1^{PN}-score, presented in Section 2.3.2 (p. 7).

Nakov et al. (2016) reported that there was a dominance of methods based on deep learning, while kernel machines (see Section 2.2.1, p. 6) were less frequently used compared to previous years. Five out of the ten top-ranked systems used deep neural networks of some sort. They further reported that the use of *distant supervision* was common in the submitted systems, as there is an abundance of freely available tweets with, for instance, smileys or emojis that can be used as imprecise labels. Eight out of the ten top-ranked systems used word embeddings in some capacity.

The top-performing systems from SemEval 2016 are also top-ranked for test datasets from previous years. A general pattern was identified: the top-ranked system of each year outperforms the top-ranked system from the previous year on the dataset from the previous year.

The top-scoring system for Task 4A was called *SwissCheese* (Deriu et al., 2016) and used an ensemble of 2-layer convolutional neural networks (CNNs). Predictions were combined using a random forest classifier. In addition to small amounts of labeled data, the system was trained on large amounts of unlabeled data using distant supervision.

Rank	Name	F_1^{PN}-score	Classifier	Features
1	SwissCheese	0.633	Ensemble of CNN	Word embeddings (word2vec + GloVe)
2	SENSEI-LIF	0.630	Fusion of CNN and MLP (multilayer perceptron)	Word embeddings (3 kinds) + sentiment polarity lexicon
3	UNIMELB	0.617	Ensemble of CNN, LSTM and NB	Word embeddings (word2vec)
4	INESC-ID	0.610	Non-Linear Sub-space Embedding (NLSE) model	Word embeddings
5	aueb.twitter.sentiment	0.605	Ensemble of SVMs	Word embeddings (GloVe) + traditional NLP features
6	SentiSys	0.598	Logistic Regression	Traditional NLP features
7	I2RNTU	0.596	Asymmetric SIMPLS (ASIMPLS)	Word embeddings
8	INSIGHT-1	0.593	CNN	Word embeddings
9	TwiSE	0.586	Stacked generalization with SVM and Logistic Regression	Traditional NLP features
10	ECNU	0.585	Logistic Regression	Word embeddings + traditional NLP features

Table 3.1.: The F_1^{PN} scores, classification algorithms and input features used by the top ranking systems for SemEval 2016 Task 4A.

The system used the Skip-gram model from word2vec (see Section 2.4.1, p. 11) to train word embeddings. The word embeddings were used to initialize the first layer in their system.

Similarly to the SwissCheese system, the *SENSEI-LIF* system presented by Rouvier and Favre (2016) used CNNs and word embeddings in its classifier. However, the system did not utilize an ensemble, but instead combined different CNNs trained with different input embeddings in a fusion neural network. The system used three different kinds of word embeddings as input to the CNNs called lexical embeddings, part-of-speech embeddings and sentiment embeddings. The lexical embeddings were the word embeddings from training with the Skip-gram model, while the other two were variations created to give additional semantic information.

The system *aueb.twitter.sentiment* (Giorgis et al., 2016) did not utilize deep neural networks for its classifier. The system used an ensemble of two SVM classifiers, where one used traditional features such as POS based, sentiment lexicon based and negation based features, while the other used pre-trained word embeddings produced by GloVe (see Section 2.4.2, p. 12). Each classifier was split in two parts. The first part, called subjectivity detection, tried to predict if a tweet is objective or subjective. These scores were appended to the inputs as they were passed to the second part, called sentiment polarity detection. Here the system tried to predict which sentiment a tweet has based on the original input and the score from the objectivity detection part. The scores from the two classifiers were then averaged to get the final prediction.

4. Ternary Sentiment Embedding Model

A new neural network model for training word embeddings called the *Ternary Sentiment Embedding Model* is proposed. The model extends the Hybrid Ranking Model by Tang et al. (2016) for training Sentiment-Specific Word Embeddings (SSWE) which in turn extends the Collobert and Weston (C&W) model by Collobert et al. (2011). The C&W model is presented in Section 3.2 (p. 18), and the Hybrid Ranking Model is presented in Section 3.3.4 (p. 22). The Hybrid Ranking Model utilizes tweets labeled as “positive” or “negative” to train word embeddings. The proposed model extends this by also looking at tweets labeled as “neutral”. The model is a multilayer neural network that uses context and sentiment information of tweets in order to learn vector representations of words. It consists of three bottom (core) layers and two top layers that work in parallel. The model architecture is shown in Figure 4.1.

4.1. Core Layers

The first three layers of the Ternary Sentiment Embedding Model are identical to the three first layers of both the C&W model and the Hybrid Ranking Model. The model uses a lookup layer to map word indices in a fixed sized vocabulary to feature vectors. The vectors representing the words in a context window are concatenated and fed to a linear layer. The size of the context window, i.e. the total number of words that are considered, is a model hyperparameter. The output from the linear layer is subsequently passed to a HardTanh layer. This process is explained in further detail in Section 3.2 (p. 18). The C&W model has a final linear layer that outputs a vector of size 1 giving a context score for a given focus word and context window. This layer is dubbed the *Context Linear Layer*. As with the C&W model, the objective of the context part of the model is to assign a higher context score to a correct context window than a corrupted context window by a given margin. This objective is formulated as a hinge loss function:

$$loss_c(t, t^r) = max(0, m - f^c(t) + f^c(t^r)) \tag{4.1}$$

where m is the margin, t is the correct context window, t^r is the corrupted context window and $f^c(\cdot)$ is the score given by the context linear layer. With $m = 1$, this is identical to the hinge loss function of the C&W model seen in Equation 3.3 (p. 19).

4.2. Ternary Sentiment Linear Layer

The new model architecture introduces a new top layer for calculating sentiment scores. The layer is a simple linear layer that outputs a vector of size 3 where the values represent positive, negative and neutral scores for a given context window. Each context window is labeled with a sentiment of either "positive", "negative" or "neutral". The objective of the model is to give a higher score for the value corresponding to the context window's label than each of the other possible labels. A new margin hinge loss function is proposed to represent this loss and used to train the model. The new hinge loss function is formulated as:

$$loss_s(t) = max(0, m - f_c^s(t) + f_{i1}^s(t)) + max(0, m - f_c^s(t) + f_{i2}^s(t)) \tag{4.2}$$

where t is a context window, m is the margin, $f_c^s(\cdot)$ is the sentiment score for the currently labeled sentiment of the input context window, and $f_{i1}^s(\cdot)$ and $f_{i2}^s(\cdot)$ are the sentiment scores for the other two classes. This function increases the loss if the difference between the score for the target label and either of the other scores is less than the margin.

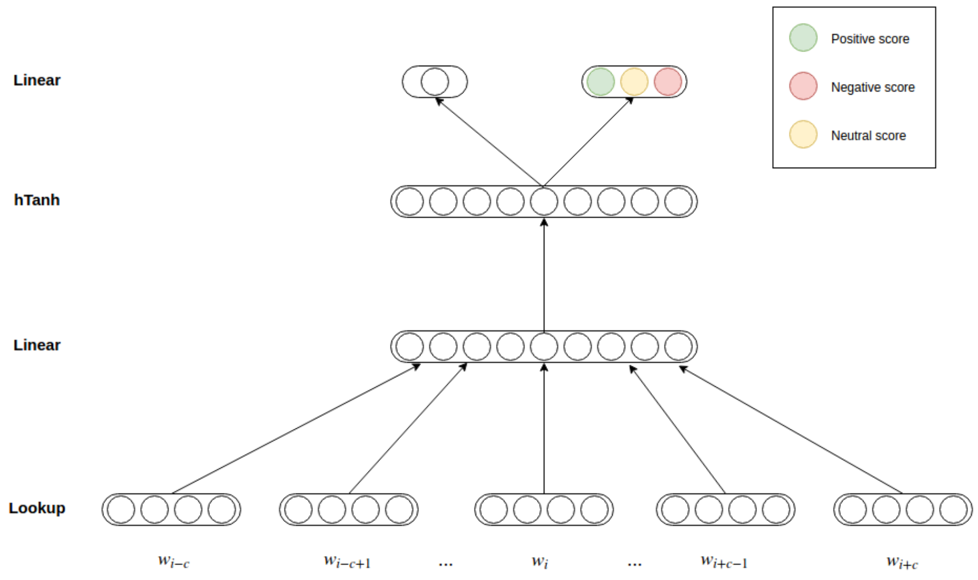


Figure 4.1.: The Ternary Sentiment Embedding Model. The two top layers are the Context Linear Layer on the left and the new Ternary Sentiment Linear Layer on the right.

4.3. Combined Loss Function

The hinge losses for the Context Linear Layer, $loss_c$ (shown in Equation 4.1), and the Sentiment Linear Layer, $loss_s$ (shown in Equation 4.2), are combined to give the total loss for the proposed model. The combined loss function is a weighted linear combination by a hyperparameter α.

$$loss(t, t^r) = \alpha \cdot loss_s(t) + (1 - \alpha) \cdot loss_c(t, t^r) \tag{4.3}$$

4.4. Model Training

The parameters of the neural network model are trained by taking the derivative of the loss through backpropagation. This is the approach used by Collobert et al. (2011) for training the C&W model. Stochastic Gradient Descent (SGD) is used to update the model parameters. This means that samples, in this case context windows created from tweets, are randomly drawn from the training corpus, and the model parameters are updated for each sample that is passed through the model. The update is done according to the formula:

$$\theta_t = \theta_{t-1} - l_r \cdot g_t \tag{4.4}$$

where θ_t is the value of the parameter θ at time t, g_t is the gradient of the parameter at time t and l_r is the learning rate.

Initialization of the model parameters is done as in Tang et al. (2016). The parameters of the lookup layer are initialized with values from the uniform distribution $U(-0.01, 0.01)$, while the parameters of the hidden layers are initialized using the *fan-in* technique described by Collobert et al. (2011). The fan-in is the number of inputs used by a layer, and the technique is to draw the initial parameters from a centered distribution with variance equal to the inverse of the square-root of the fan-in. This means that the initial parameters for the hidden layers are drawn from the uniform distribution $U(\frac{-0.01}{InputLength}, \frac{0.01}{InputLength})$. This fan-in technique is used for the learning rate as well, meaning the learning rates from Equation 4.4 for the hidden layers are divided by the fan-in.

5. Architecture

In this chapter the implementation, architecture and functionality of the applications that were developed as part of this thesis are described.

5.1. Tweet Collector

A simple application was built to collect tweets from Twitter. The application connects to the Twitter's Streaming API `statuses/sample` endpoint[1] to fetch tweets and save them in a MongoDB[2] database, in order to build a tweet corpus. The application was built using Node.js, relying heavily on the *twit* library which is explained in Section 2.5.7 (p. 15) and handles the connection to the stream. The Tweet Collector collects about 50 tweets per second.

As the tweets received from Twitter's Streaming API come in JSON format, using MongoDB as database system is a sensible choice, because it saves its data in JSON-like documents. This means that tweets can be directly inserted in the database as they are. Also, MongoDB (from hu**mongo**us) is built with large data collections in mind and scales well. Though, to save disk space, tweets are pruned by removing some excessive data fields. The pruning steps are explained in the following list. The `user`, `retweeted_status` and `quoted_status` objects mentioned are properties of a single tweet's JSON document.

Reducing `user` object A tweet contains a lot of data about its author, for instance the profile picture, colors of profile page, etc. This is reduced to only contain the user's ID[3].

Reducing `retweeted_status` object Retweets contain a lot of metadata about the original tweet, contained in an object called `retweeted_status`. The content is reduced to only comprise the original tweet's ID and its author's ID.

Reducing `quoted_status` object A tweet might refer to or *quote* another tweet. Similarly to `retweeted_status`, the metadata that is in the `quoted_status` object is reduced to only contain the quoted tweet's ID and its author's ID.

Removing falsy fields A tweet might contain falsy[4] fields. *Falsy/truthy* is a concept in JavaScript, where a falsy value is a value which is either `false`, `undefined`, `null`,

[1]`https://dev.twitter.com/streaming/reference/get/statuses/sample`
[2]`https://www.mongodb.com/`
[3]The ID of a Twitter user is not the username, but a unique integer .
[4]`https://developer.mozilla.org/en-US/docs/Glossary/Falsy`

`0` or the empty string `""`. Truthy values are all values that are not falsy. The fields containing falsy values are removed, as their non-existence in itself can be interpreted as a falsy value.

5.2. CATTS

The *Crowdsourced Annotation Tool for Twitter Sentiment (CATTS)* is a web application for manual annotation of tweets. It supports annotating both sentiment and sarcasm. The annotated tweets define a dataset that is dynamically built as users annotate. This dataset is available for download in both JSON and text formats, through the CATTS API[5]. CATTS is open-source and available at GitHub[6].

CATTS was implemented using Meteor[7], a full-stack JavaScript framework, combined with React[8] on the frontend.

We ended up not needing to manually annotate tweets for this thesis, as we got a sufficient amount of labeled data from SemEval. It is still included here, as it is open-source and available for others to use.

5.3. Fredriksen–Jahren Lexicon Classifier Python Port

Fredriksen and Jahren (2016) open-sourced their lexicon classifier (see Section 2.5.11, p. 15), which is available on GitHub[9]. To better be able to integrate the program with the programs of this thesis, the Java implementation was ported to Python. The program can take various options, as Fredriksen and Jahren's original program does, like which sentiment lexicon to use and sets of negation, intensifiers and stop words. The port uses by default the optimal parameters found in Fredriksen and Jahren (2016). The port is open source[10] and published on the Python Package Index (PyPI)[11]. In this section the process of the Fredriksen–Jahren Lexicon Classifier (FJLC) is explained. Note that this is the work of Fredriksen and Jahren (2016) and that our port is merely a translation of the program from Java to Python.

An overview of the architecture of the FJLC system is shown in Figure 5.1. A tweet that is being processed goes through three main stages: *preprocessing*, *analysis* and *classification*.

The preprocessing steps done include, but are not limited to, lowercasing, normalizing letters to to the Latin alphabet[12], removing Twitter-specific elements[13], and removing non-alphanumerical characters.

[5]CATTS API – `https://catts.byrkje.land/download`
[6]CATTS Code – `https://github.com/draperunner/catts`
[7]`https://www.meteor.com/`
[8]`https://facebook.github.io/react/`
[9]`https://github.com/freva/Masteroppgave`
[10]`https://github.com/draperunner/fjlc`
[11]`https://pypi.python.org/pypi/fjlc`
[12]For instance, "Déjà vu" would be translated to "Deja vu"
[13]URLs, user mentions, hashtags, usernames, RT-tags

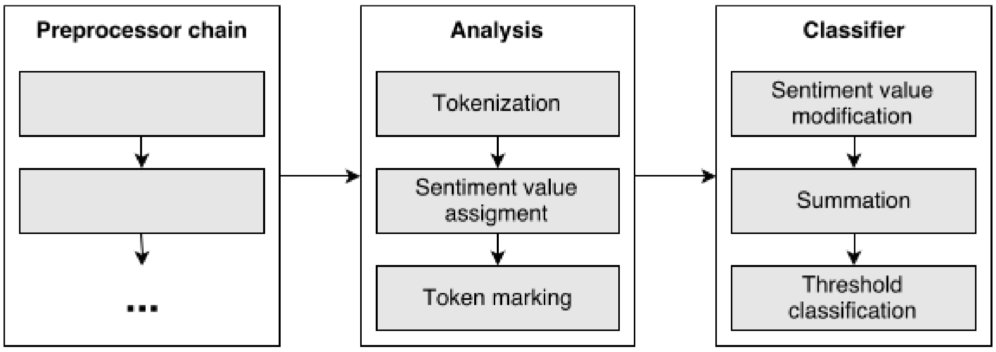


Figure 5.1.: Architecture of the Fredriksen–Jahren Lexicon Classifier (FJLC) system. Figure from Fredriksen and Jahren (2016).

After preprocessing, the tweet enters the analysis stage. First, it gets split into what Fredriksen and Jahren call *optimal* tokens. Optimal tokens are the longest non-overlapping n-grams of a sentence that are also found in a given vocabulary. Out-of-vocabulary tokens are tokenized as unigrams.

Each token is next assigned a sentiment value by looking up the token's value in the provided sentiment lexicon. Out-of-vocabulary words are assigned a value of 0. The tokens are then checked for *intensification* or *negation* cues. Intensification can happen by *intensifier* occurrence. Intensifiers can be words like "very", "extremely" or "slightly". Each such intensifier is assigned an intensification value. The sentiment value of a token following an intensifier is multiplied by the intensification value of that word. Note that intensifiers with value less than 1 will work as dampeners. If a *negation* is observed, like "not", "can't" or "ain't", the tokens in the rest of the sentence are negated. Negation happens by subtracting or adding a chosen *negation value* to the token sentiment value depending on whether the token sentiment value is positive or negative, respectively. Finally, if a sentence ends with "!" or "?", all tokens of that sentence are intensified by the exclamation mark or the question mark intensifier value.

The final stage calculates the sentiment score for the entire tweet by summing the sentiment values for each of its tokens. The resulting score is compared against a lower threshold T_L and a greater threshold T_G. Tweets with score above the greater threshold are classified as positive, tweets with score below the lower threshold are classified as negative. Tweets with score between the two thresholds are classified as neutral. The thresholds used in this thesis are those used by Fredriksen and Jahren in their optimal setup: $T_L = -0.04$ and $T_G = 0.00$.

$$sentiment(tweet) = \begin{cases} \text{negative}: & sentimentScore(tweet) < -0.04 \\ \text{positive}: & sentimentScore(tweet) > 0.00 \\ \text{neutral}: & \text{otherwise} \end{cases} \tag{5.1}$$

5.4. Twitty

To train different word embeddings and test these with a couple of simple classifiers as part of our specialization project (Byrkjeland and de Lichtenberg, 2016), the *Twitty* program was created. In this thesis, it is used for training word2vec and GloVe word embeddings in Section 8.4 (p. 66), and the Twitter Sentiment Analysis Byrkjeland–de Lichtenberg (TSABL) program (Section 5.6) has parts that are based on some of Twitty's modules.

Twitty is programmed entirely in the Python programming language, and is split up in multiple modules by function. Each module will push a set of data further along the pipeline, which comprises the stages depicted in Figure 5.2. Each module is designed as a Command Line Interface (CLI). The user can choose which stage to start at by using the appropriate module.

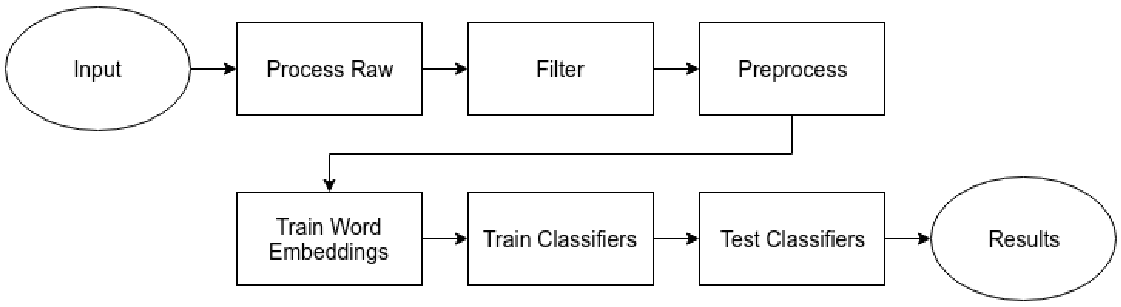


Figure 5.2.: Overview of the Twitty Pipeline.

5.4.1. Processing Raw Files

Raw files are files that contain a tweet ID and optionally a sentiment annotation per line. To fetch a tweet for each ID, the `process_raw` module connects to the Twitter REST API. New files will be saved where the IDs have been replaced with JSON objects representing the respective tweet. Note that the IDs might refer to tweets that have been deleted. In that case, no tweet will be returned, and the resulting file will contain fewer tweets than the original ID file.

5.4.2. Filtering

Not all tweets are of further interest. All retweets and all non-English tweets are removed using the `filter_tweets` module. Retweets are copies of other tweets, and these are removed to avoid having duplicates in the corpora. JSON objects for tweets contain metadata such as values for language and retweet status, which are used to filter unwanted tweets. If a tweet JSON object does not have a value for language, the module tries to classify the language in the tweet using the *langid*[14] (Lui and Baldwin, 2012) module.

[14] *langid* is a standalone language identification tool (`https://github.com/saffsd/langid.py`).

5.4.3. Preprocessing

Tweets are preprocessed to make them better fit for computational processing. The `tweet_preprocessor` module employs Twokenize (see Section 2.5.4, p. 13) to split tweets into their constituent parts, and the Preprocessor (see Section 2.5.5) to do tokenization and cleaning. The result is a corpus with a preprocessed tweet per line. The module can also reduce elongated words (see Section 2.1.5, p. 6) and lowercase the tweets.

5.4.4. Train Word Embeddings

The `train_word_embeddings` module trains word embeddings on a given corpus. It supports word2vec and GloVe models. The word2vec models are trained using MultiVec (see Section 2.5.2, p. 13), and both the Skip-gram model and the Continuous Bag-of-Words (CBOW) model can be trained with varying dimensions. The GloVe model is trained using *glove-python* (see Section 2.5.3, p. 13), and can be trained with various dimensions, learning rates and epochs. The trained model and resulting vectors will be saved in separate files. The vector files are on the same format, where each line starts with a token followed by its vector values.

5.4.5. Train Classifier

The `train` module for training sentiment classifiers supports a Support Vector Machine (SVM) and a Maximum Entropy classifier. The classifiers are implemented in the *scikit-learn* (Section 2.5.1, p. 13) library. After training, the models are saved. The machine learning classifiers utilized require input vectors of fixed length. This means that the combined embeddings of tweets need to be of the same size regardless of the length of each tweet. In this system the input to the classifiers, the combined tweet embedding, is created by averaging the word embeddings for each word in a tweet.

5.4.6. Test Classifier

The module `test` is used to test the performance of the trained classifier models. The module takes a word embedding file, a training and test file, and the trained models, and calculates F_1-scores for both classifiers.

Since each module will save its results, there is no need to redo all steps if one only wants to do one of the last. This makes the program flexible and easily configurable.

5.5. Distant Supervision Program

The Distant Supervision Program was created for implementing, tweaking, and comparing methods for distant supervision of tweets contained in a MongoDB database. The experiments of Chapter 6 are performed using this program. It is written in Python and is available on GitHub[15]. An object-oriented architecture is used, where each distant supervision method inherits from an abstract base class. This makes developing new methods fast and easy. The methods support both evaluation through testing toward a test set, and processing through unlabeled data to save them as distant-supervised datasets.

5.6. TSABL

To train sentiment embeddings and test them with various classifiers as part of a complete Twitter Sentiment Analysis (TSA) system, the *Twitter Sentiment Analysis Byrkjeland–de Lichtenberg* (TSABL) program was created. The program consists of independent modules for fetching Twitter data, preprocessing data, training word embeddings, and training and testing classifiers. Most of the program is written using the Python programming language, while one implementation of the word embedding trainer is written using Java.

5.6.1. Fetching Twitter Data

The modules `process_raw` and `filter_tweets` from the Twitty program described in Section 5.4 have been reused as parts of the TSABL program. The functionality of the modules remains as described in Sections 5.4.1 and 5.4.2, but the CLI modules have been changed to Python modules to better fit the bigger system.

5.6.2. Preprocessing

Preprocessing is done similarly as for the `tweet_preprocessor` module of the Twitty program described in Section 5.4.3. The *Twokenize* library described in Section 2.5.4 (p. 13) is used to split tweets to separate punctuation from words as well as split contractions. The split tweets are subsequently passed to the *Preprocessor* library described in Section 2.5.5 (p. 14) to remove Twitter-specific elements. In the TSABL program, the Preprocessor library removes URLs, user mentions, reserved words and numbers. Explanations of these terms are provided in Section 2.5.5. Hashtags, emojis and smileys are preserved as they can convey sentiment information of a tweet and are used by some of the distant supervision methods described in Chapter 6. All tweets are lower-cased and elongated words are reduced (see Section 2.1.5, p. 6), before the preprocessed tweets are saved to a text file.

5.6.3. Training Word Embeddings

The TSABL program contains two separate implementations of both the Hybrid Ranking Model and the Ternary Sentiment Embedding Model presented in Chapter 4 written in the programming languages Python and Java. Both implementations are structured

[15]Distant Supervision Program Code – `https://github.com/draperunner/distant-supervised-tweets`

in the same manner. Initially, the program reads Twitter data from files separated by sentiment. For binary training, the program reads two text files with a tweet on each line, one file with positive tweets and one with negative tweets. In the ternary case, the program also reads a file containing neutral tweets. All the tweets are assigned a label according to the data file the tweet came from before the tweets were shuffled. The program subsequently creates a vocabulary over all the words in the data files, and each unique word is given a unique index. During this procedure, words that occur less than five times are not given an index and are consequently not used for training.

Each tweet is then split up into context windows of specified size. The program creates context windows consisting of word indices for all the words in a tweet that are surrounded by enough context words. Tweets that are shorter than the context window are ignored completely. For each context window, a corrupted context window is also created where the focus word is swapped for another randomly drawn word from the vocabulary.

The program then creates the neural network model according to hyperparameters the user can specify. The final model is created by having two identical models, one for the correct and one for the corrupted context window. One of the two identical models is initialized with random values according to the initialization described in Section 4.4 (p. 29) before the second is created by copying the first model and its initial parameters. The two models are needed in order to compare the scores given by passing correct and corrupt context windows to the model.

When training the model, each pair of context windows is passed as input. The model parameters are updated in a feed-forward pass before the loss is calculated for each top layer. The loss is combined as explained in Section 4.3 (p. 28). The program then calculates the gradient for each parameter, before updating the model parameters. This procedure is repeated for all context window pairs, before starting the next epoch. After the model is trained for the specified number of epochs, the weights of the lookup layer, the trained word embeddings, are stored along with the word they represent in a text file.

Python implementation

A module for training word embeddings models was created using the Python programming language. The module is implemented using the Keras library (Section 2.5.14, p. 16) and can utilize Theano (Section 2.5.12) or TensorFlow (Section 2.5.13) as backend. The user can specify the model type and hyperparameters, meaning that the module can be used to build and train the Collobert and Weston model (Section 3.2, p. 18), the Hybrid Ranking Model (Section 3.3.4, p. 22), and the proposed Ternary Sentiment Embedding Model (Chapter 4) with various hyperparameter configurations. The module can run on CPU or GPU, and a script is provided for running the module on Nvidia GPUs using the parallel computing platform CUDA.

Java implementation

During testing of the Python module for training word embeddings it was discovered that the program would require approximately eleven days running on an NVIDIA GeForce GTX Titan X GPU to train the model on three million tweets using Stochastic Gradient Descent (SGD) for 20 epochs. As it was desired to train and test the model using different configurations, it was decided that another, faster implementation was needed. As a result, another module for training the word embedding models was implemented using Java.

The module extends the source code used by Tang et al. (2016) for training the Hybrid Ranking Model by adding the new model architecture and loss function. Duyu Tang's original source code is available on his website[16]. Similarly to the Python implementation, this module allows the user to specify the hyperparameters, but the different model architectures are split into separate files. The module is written using only the standard Java libraries. The runtime performance of training the model was significantly improved compared to the Python implementation, using approximately 24 hours to train the model using SGD for 20 epochs on a single AMD Opteron 6128 CPU core. This made it possible to train multiple model configurations in parallel, reducing the time needed for training and testing significantly. The Java implementation writes the word embeddings produced after each epoch to file.

5.6.4. Training and Testing Classifiers

The TSABL program contains a module for training and testing a Support Vector Machine (SVM) classifier using different word embeddings as input features. The SVM classifier is implemented in Python using LinearSVC from the scikit-learn library (see Section 2.5.1, p. 13).

During training, the SVM classifier takes as input an array of features and an array of labels, and creates a decision function that can be used to classify new, previously unseen features to one of the relevant labels. For the task of classifying tweets, each feature represents a tweet while each corresponding label is the tweet's sentiment. The feature vector of a tweet is created by averaging the word embeddings for all the words in the tweet. Any word that does not have a word embedding created for it will not contribute to the tweet's feature vector. Before being averaged, each word embedding is scaled to have zero mean and unit variance.

After the classifier has been trained on the training dataset, it can be used to predict a class for new samples. Feature vectors for test tweets are created by scaling and averaging the word embeddings of the tweet's constituent word in the same manner as when training the classifier. The classifier will produce a predicted label for each tweet, and the results are compared to the actual labels and scored using the different evaluation metrics described in Section 2.3.2 (p. 7).

[16]`http://ir.hit.edu.cn/~dytang/`

6. Distant Supervision of Tweets

The idea of *distant supervision* is to automatically label data in order to be able to leverage large amounts of it. These data are called *distant supervised* or *weakly annotated* data, as the quality is not great, but the quantity is. To train sentiment embeddings, large amounts of weakly annotated tweets are needed. In this section the approach of extracting weak labels from the corpus of collected tweets is described. As of May 2, 2017, this was about 547 million.

In order to evaluate the different methods, all the manually annotated International Workshop on Semantic Evaluation (SemEval) datasets from 2013 to 2016[1] were downloaded. In this chapter the dataset names are abbreviated by removing "20" and "-A" (for instance, "2013-dev-A" becomes "13-dev"). The datasets are organized for the SemEval 2017 Task 4, and made available on the task's data "Data and Tools web" page[2]. The datasets contain IDs for 50,333 tweets, but 10,251 of those tweets have been deleted and cannot be downloaded. Thus, 40,082 tweets remain. 532 of these are duplicates, and are removed, giving 39,550 tweets. After the filtering steps explained in Section 6.1 there are 28,120 tweets left[3]. Each method is run on these tweets and evaluated using various quality evaluation metrics. Additionally, the processing time and the fraction of tweets included are noted, as these are important properties of distant supervised sentiment analysis methods.

The filtering steps are described in Section 6.1 before the implementation of the distant supervision methods are described in Section 6.2. In Section 6.3 the grid parameter searches done are explained, for the methods where this was relevant. Lastly, in Section 6.4, the methods are compared using the SemEval sets.

[1]2013-dev-A, 2013-test-A, 2013-train-A, 2014-test-A, 2015-test-A, 2014-sarcasm-A, 2015-train-A, 2016-dev-A, 2016-devtest-A, 2016-test-A, 2016-train-A

[2]`http://alt.qcri.org/semeval2017/task4/index.php?id=data-and-tools`

[3]One tweet was removed due to an error in the process of inserting tweets into the database.

Dataset	Original	Downloaded	Duplicates	After filtering
13-dev	1654	1231	2	961
13-test	3547	2708	1	1839
13-train	9684	7323	26	5425
14-sarcasm	85	60	0	52
14-test	1853	1461	0	997
15-test	2390	1881	18	1376
15-train	489	364	0	281
16-dev	1999	1662	2	1662
16-devtest	2000	1652	2	1173
16-test	20632	16815	4	12075
16-train	6000	4925	22	3273
Total	50333	40082	532	29114

Table 6.1.: Statistics for SemEval training and test sets. Fetched March 21, 2017. Filtering is described in Section 6.1. The total of duplicates is not the sum of above rows, but the number of duplicates over all the sets.

Dataset	Tweet count	Positive		Negative		Neutral	
13-dev	961	353	36.73%	198	20.6%	408	42.46%
13-test	1839	827	44.97%	318	17.29%	694	37.74%
13-train	5425	2171	40.02%	878	16.18%	2362	43.54%
14-sarcasm	52	20	38.46%	26	50.0%	6	11.54%
14-test	997	556	55.77%	134	13.44%	307	30.79%
15-test	1376	610	44.33%	249	18.1%	504	36.63%
15-train	281	103	36.65%	39	13.88%	139	49.47%
16-dev	1662	453	27.26%	207	12.45%	391	23.53%
16-devtest	1173	574	48.93%	193	16.45%	404	34.44%
16-test	12075	4328	35.84%	1899	15.73%	5845	48.41%
16-train	3273	1714	52.37%	515	15.73%	1027	31.38%
Total	29114	11709	40.22%	4656	15.99%	12087	41.52%

Table 6.2.: Distribution of sentiment for the SemEval datasets.

6.1. Filtering

Similarly to Fredriksen and Jahren (2016), tweets with certain properties are ignored:

Retweets Retweets are copies of original tweets. Including retweets might therefore lead to over-representation of certain phrases.

Contains URL Usually, to analyze tweets with URLs, it is also necessary to analyze the contents of the link destination. Often the tweets only contain the title of a news article, or another reference to the link destination's content. These tweets are ignored as they often on their own do not provide enough information needed to predict sentiment.

Contained ° symbol Most tweets containing the degree symbol are weather reports.

Ends with number Spammers might append a number to a tweet, to avoid Twitter's spam detection.

6.2. Methods

The following sections explain each of the sentiment analysis systems that are compared for use as distant supervision. The methods use sentiment lexica, and most are easily available as Python libraries.

6.2.1. Emoticons

Go et al. (2009) introduced a novel approach to automatically classifying tweet sentiment using distant supervised tweets. Go et al. use the Twitter Search API's *attitude* operator to fetch tweets with positive and negative attitude by querying for ":)" and ":(", respectively. According to Go et al., this will return tweets with certain positive and negative emoticons, given by Table 6.3[4].

Emoticons mapped to :)	Emoticons mapped to :(
:)	:(
:-)	:-(
:)	: (
:D	
=)	

Table 6.3.: Emoticons mapped to positive ":)" and negative ":(" attitude by the Twitter API.

An implementation of the emoticon method of Go et al. (2009) is created using the Python programming language. Tweets are not fetched using the Twitter Search API as described above, but instead the pre-collected corpus of tweets is used. Tweets that contains any of the positive emoticons in Table 6.3 are assigned a positive sentiment, and equivalently for negative tweets. Tweets that contain both a positive and negative emoticon are removed, as is also done by Go et al. To adapt the method for the ternary classification task, tweets containing none of the emoticons are classified as neutral. The Emoticons method is abbreviated to "Em." in some of the following tables and figures.

[4]The Twitter API documentation does not state how its attitude operator works. This might have changed since Go et al. (2009).

6.2.2. Emoticons Extended

The sets of positive and negative emoticons used by Go et al. (2009) are arguably quite sparse compared to the vast amount of emojis and emoticons used today. Thus, the sets are extended with more emoticons and emojis. The western emoticons that are found to have clear sentiment from the "List of emoticons"[5] Wikipedia page are manually annotated.

For emojis, the emoji-emotion valence lexicon[6] was used, assigning emojis with negative score as negative and emojis with positive score as positive. Emojis with score 0 are not included.

The resulting sets of emoticons are shown in Table 6.4. The Emoticons Extended method will be abbreviated to "Em. Ext." in some of the following tables and figures.

6.2.3. AFINN

This method uses the *AFINN* library (see Section 2.5.8, p. 15). As AFINN was found to give a large number of tweets a score of 0, only these are classified as neutral. Tweets with a score greater than 0 are classified as positive, and tweets with scores lower than 0 as negative. The sentiment of a tweet is given by:

$$sentiment(tweet) = \begin{cases} \text{negative}: & sentimentScore(tweet) < 0 \\ \text{neutral}: & sentimentScore(tweet) = 0 \\ \text{positive}: & sentimentScore(tweet) > 0 \end{cases} \tag{6.1}$$

where

$$sentimentScore(tweet) = \sum_{n=1}^{N} sentimentScore(word_n) \tag{6.2}$$

where N is the number of words in the tweet.

6.2.4. TextBlob

The *TextBlob* library (see Section 2.5.10, p. 15) is used to compute polarity and subjectivity scores. Tweets with subjectivity score less than a threshold st are defined as neutral. The polarity threshold pt is set so that tweets with polarity score less than $-pt$ are classified as negative and tweets with score greater than pt as positive.

$$sentiment(tweet) = \begin{cases} \text{neutral}: & subj(tweet) \leq st \\ \text{positive}: & subj(tweet) > st \wedge pol(tweet) \geq pt \\ \text{negative}: & subj(tweet) > st \wedge pol(tweet) \leq -pt \end{cases} \tag{6.3}$$

where $subj(tweet)$ is the subjectivity score of a tweet, and $pol(tweet)$ is the polarity score of a tweet.

[5] `https://en.wikipedia.org/wiki/List_of_emoticons`
[6] `https://github.com/wooorm/emoji-emotion`

Emoticons :)		Emojis :)		Emoticons :(		Emojis :(	
:)	:-)	😁	😼	:(	DX	😠	🤢
:)	:D	😀	😛	:-(	:-/	😦	🤓
=D	:-]	😍	😎	: (	:/	😲	😮
:]	:-3	😻	😅	:'(	:-.	😊	😔
:3	:->	🤗	😉	:-(	>:\	🤡	😣
:>	8-)	😇	😋	:(	>:/	😥	😾
8)	:-}	😂		:-c	:\	😖	😡
:}	:o)	😹		:c	=/	😕	🙄
:c)	:^)	😗		:-<	=\	😢	😱
=]	=)	😽		:<	:L	😿	☹️
:-D	8-D	😚		:-[	=L	😞	🙀
8D	x-D	😘		:[	:S	😓	😈
xD	X-D	😙		:-||	</3	😵	🤧
XD	=D	😆		>:[	<\3	🤕	
=3	B-^D	☺️		:{	>.<	🤒	
:-))	:'-)	😌		:@	v.v	😨	
:')	:-*	🤣		>:(		😳	
:*	:×	🙂		D-':		😧	
;-)	;)	😄		D:<		😬	
*-)	*)	😸		D:		😯	
;-]	;]	😃		D8		👿	
;^)	:-,	😺		D;		🤥	
;D	<3	😏		D=		😷	

Table 6.4.: Emoticons and emojis mapped to positive ":)" and negative ":(" attitude by the Emoticons Extended method.

6.2.5. VADER Sentiment Analysis

Using the sentiment scores computed by the *VADER* library (see Section 2.5.9, p. 15), the confidence threshold t is set so that:

$$sentiment(tweet) = \begin{cases} \text{positive}: & positiveScore(tweet) \geq t \\ \text{negative}: & negativeScore(tweet) \geq t \\ \text{neutral}: & neutralScore(tweet) \geq t \end{cases} \qquad (6.4)$$

The vector is normalized, which means that the sum of *positiveScore*, *negativeScore* and *neutralScore* equals 1. By setting the threshold $t > 0.5$ it is certain that the other sentiment scores must be below 0.5. The higher the threshold is set, the more confident the classification is. However, with a high threshold, more tweets are ignored. If none of the scores are above the threshold for a tweet, the tweet is skipped and the resulting set of annotated tweets is smaller than if a lower threshold is set.

6.2.6. Combo Average

A combination of the classifiers AFINN, TextBlob, and VADER is created. Initially, the scores for the three classifiers are normalized to be in the same range, a single polarity score between -1 (very negative) and 1 (very positive). For AFINN, the score is divided by five times the number of words in the tweet. Five, because that is the absolute value of the highest score a single word can get. For VADER, the *compound* score (see Section 2.5.9, p. 15) is used. For TextBlob, the score is already normalized as desired.

The scores are combined using a weighted average, given by the equation:

$$sentimentScore = \frac{a \cdot \mathit{afinnScore} + b \cdot \mathit{vaderScore} + c \cdot \mathit{textblobScore}}{a + b + c} \tag{6.5}$$

where the parameter weights a, b and c are tuned using a grid parameter search, which is explained in Section 6.3.

To classify the tweets, a threshold t is chosen so that every tweet with $sentimentScore > t$ is classified as positive, every tweet with $sentimentScore < -t$ is classified as negative, and all other tweets are classified as neutral.

$$\mathit{sentiment}(\mathit{tweet}) = \begin{cases} \text{negative}: & \mathit{sentimentScore}(\mathit{tweet}) < -t \\ \text{positive}: & \mathit{sentimentScore}(\mathit{tweet}) > t \\ \text{neutral}: & \text{otherwise} \end{cases} \tag{6.6}$$

6.2.7. Fredriksen–Jahren Lexicon Classifier (FJLC)

The port of the Lexicon Classifier of Fredriksen and Jahren (2016) is used to classify tweets, using their best performing lexicon and parameters. The classifier and the port are described in Section 5.3 (p. 32).

6.3. Grid Parameter Searches

Some of the methods explained above take hyperparameters. These methods are tuned by performing grid searches. The results for each method are described in the following subsections. The abbreviation “Incl.” is used for inclusion rate, which is the rate of tweets that are classified by the method in question.

6.3.1. VADER

t	Macro F_1	F_1^{PN}	F_1^{POS}	F_1^{NEG}	F_1^{NEU}	Incl.
0.1	**0.5306**	**0.5323**	**0.6312**	**0.4334**	0.5272	1
0.2	0.5086	0.4486	0.5379	0.3593	0.6286	1
0.3	0.3659	0.2381	0.3161	0.1601	0.6216	1
0.4	0.2585	0.0848	0.1282	0.0414	0.6060	0.9996
0.5	0.2130	0.0197	0.0346	0.0048	0.5995	0.9933
0.6	0.2073	0.0040	0.0080	0	0.6139	0.9364
0.7	0.2159	0.0007	0.0014	0	0.6462	0.8072
0.8	0.2325	0	0	0	**0.6974**	0.5838

Table 6.5.: Results for the VADER method using different confidence thresholds t.

The table shows that VADER struggles to classify positive and negative tweets as the threshold increases. It is only the neutral tweets that VADER is strongly confident about. $t = 0.1$ is the best performing threshold.

6.3.2. TextBlob

st	pt	Macro F_1	F_1^{PN}	F_1^{POS}	F_1^{NEG}	F_1^{NEU}	Incl.
0.1	0.1	0.5001	0.5094	0.6314	0.3874	0.4816	0.8626
0.1	0.2	0.5292	**0.5221**	0.6474	**0.3967**	0.5436	0.7224
0.1	0.3	**0.5478**	0.5205	**0.6545**	0.3866	0.6022	0.5904
0.2	0.1	0.5015	0.5051	0.6274	0.3828	0.4944	0.8837
0.2	0.2	0.5273	0.5116	0.6382	0.3850	0.5587	0.7585
0.2	0.3	0.5383	0.5013	0.6367	0.3660	0.6122	0.6379
0.3	0.1	0.5030	0.5008	0.6227	0.3789	0.5074	0.9064
0.3	0.2	0.5238	0.4988	0.6244	0.3732	0.5739	0.8061
0.3	0.3	0.5259	0.4778	0.6108	0.3449	**0.6221**	0.7039

Table 6.6.: Results for the TextBlob method using different subjectivity thresholds st and polarity thresholds pt.

The table shows that TextBlob performs best with a low subjectivity threshold. $st = 0.1$ and $pt = 0.3$ are chosen for the final TextBlob classifier, as these parameters achieve the best Macro F_1 score, although the inclusion rate is rather low.

6.3.3. Combo Average Method

a	b	c	t	Macro F_1	F_1^{PN}	F_1^{POS}	F_1^{NEG}	F_1^{NEU}	Incl.
0.0	0.4	0.4	0.2	0.5573	**0.5463**	**0.638**	**0.4545**	0.5794	1
0.2	0.3	0.1	0.2	0.5574	0.5411	0.6315	0.4506	0.5901	1
0.0	0.3	0.1	0.3	0.5575	0.5402	0.6309	0.4494	0.5923	1
0.0	0.4	0.1	0.3	0.5575	0.5427	0.6322	0.4533	0.5871	1
0.3	0.4	0.1	0.2	0.5577	0.5408	0.6312	0.4504	0.5915	1
0.1	0.4	0.3	0.2	0.5579	0.5445	0.6365	0.4525	0.5848	1
0.1	0.2	0.1	0.2	0.5581	0.5432	0.6346	0.4518	0.5880	1
0.2	0.4	0.2	0.2	0.5581	0.5432	0.6346	0.4518	0.5880	1
0.1	0.3	0.2	0.2	0.5583	0.5446	0.6357	0.4535	0.5857	1
0.3	0.1	0.1	0.1	**0.5587**	0.5387	0.6345	0.4429	**0.5986**	1

Table 6.7.: The 10 best results for the Combo Average method using different weights a, b, and c and threshold t.

A grid search is conducted for the four parameters a, b, c, t. The top ten performing combinations of parameters are shown in Table 6.7. The table shows that the combination that achieves the top Macro F_1 score is not the same as the one achieving the top F_1^{PN} score. Both scores are included in the further comparisons in the next section. The F_1^{PN} winner is called *Combo A* ($a = 0.0$, $b = 0.4$, $c = 0.4$, $t = 0.2$) and the Macro F_1 winner *Combo B* ($a = 0.3$, $b = 0.1$, $c = 0.1$, $t = 0.1$). These methods are shortened to "Cb. A" and "Cb. B" in some of the tables to come.

6.4. Comparisons

This section comprises comparisons of the methods with tuned parameters against all the SemEval datasets as one (Table 6.8), then against each of the SemEval sets (Table 6.10).

Method	Macro F_1	F_1^{PN}	F_1^{POS}	F_1^{NEG}	F_1^{NEU}	Incl.	Runtime
FJLC	**0.5703**	0.5323	0.5931	**0.4715**	**0.6463**	1	0.93
Combo B	0.5610	0.5317	0.6261	0.4374	0.6195	1	2.27
Combo A	0.5574	0.5371	0.6283	0.4458	0.5981	1	2.26
TextBlob	0.5409	0.5018	**0.6428**	0.3608	0.6190	0.5991	0.48
AFINN	0.5368	**0.5425**	0.6201	0.4649	0.5255	1	1.21
VADER	0.5316	0.5245	0.6209	0.4281	0.5458	1	0.63
Em. Ext.	0.2592	0.1300	0.1011	0.1590	0.5174	0.9911	0.11
Em.	0.2506	0.0611	0.0860	0.0363	0.6295	0.9999	**0.09**

Table 6.8.: Comparison of F_1 scores for the different distant supervision methods, sorted by descending Macro F_1 score. Evaluated against all SemEval sets from 2013–2016. Runtime is given in milliseconds per tweet (including skipped tweets).

Method	P^{POS}	P^{NEG}	P^{NEU}	R^{POS}	R^{NEG}	R^{NEU}
FJLC	0.6951	0.4549	0.5928	0.5172	0.4894	0.7104
Combo A	0.6199	0.4739	0.5943	0.6369	0.4208	0.6020
Combo B	0.6442	0.4930	0.5876	0.6089	0.3930	0.6550
TextBlob	0.6636	0.4204	0.5815	0.6232	0.3160	0.6617
AFINN	0.5799	0.3984	0.6190	0.6664	**0.5581**	0.4565
VADER	0.5535	0.4234	**0.6365**	**0.7071**	0.4330	0.4777
Em.	**0.8136**	**0.6215**	0.4616	0.0454	0.0187	**0.9896**
Em. Ext.	0.7359	0.1195	0.4336	0.0543	0.2375	0.6412

Table 6.9.: Comparison of precision and recall for the different distant supervision methods. Evaluated against all SemEval sets from 2013–2016.

Dataset	FJLC	Cb. A	Cb. B	TextBlob	AFINN	VADER	Em.	Em. Ext.
13-dev	**0.6000**	0.5642	0.5674	0.5502	0.5832	0.5496	0.2973	0.3156
13-test	**0.6342**	0.6095	0.6084	0.5956	0.6075	0.6035	0.2661	0.2525
13-train	**0.6275**	0.5980	0.6071	0.5970	0.5912	0.5792	0.3024	0.2863
14-sarcasm	0.3365	0.3290	0.3438	0.3953	0.3919	**0.4380**	0.0765	0.0917
14-test	**0.6203**	0.5941	0.6003	0.5866	0.5713	0.5730	0.2624	0.2125
15-test	**0.5847**	0.5769	0.5811	0.5629	0.5490	0.5703	0.2394	0.2246
15-train	0.5765	0.5648	0.5754	**0.5948**	0.5648	0.5436	0.3116	0.2962
16-dev	0.4725	**0.4944**	0.4891	0.4451	0.4716	0.4708	0.2026	0.2640
16-devtest	0.4687	**0.4869**	0.4856	0.4654	0.4799	0.4564	0.2022	0.2378
16-test	**0.5775**	0.5596	0.5641	0.5341	0.5216	0.5183	0.2484	0.2633
16-train	0.4649	**0.4797**	0.4789	0.4748	0.4760	0.4735	0.1887	0.2064

Table 6.10.: Comparison of Macro F_1 scores for the different distant supervision methods. Evaluated against the different SemEval sets from 2013–2016.

6.4.1. Runtime

The Emoticons methods are extremely fast (0.09 and 0.11 ms/tweet), but their scores are substantially worse than the others. The Combo Average Methods are slow (2.26 and 2.27 ms/tweet), because they have to calculate the scores for each component method. Note that the runtimes mentioned in Table 6.8 are not including saving to file. The comparisons were run on a computer with four AMD Opteron 6128 CPUs and 125 GB of RAM running Ubuntu 16.04.

The distant supervision experiments of this thesis have been carried out on a database of already acquired tweets. The tweets for this database have been collected using the Tweet Collector Program (see Section 5.1, p. 31), which simply stores tweets from the Twitter Streaming API. The program is able to collect about 54 tweets a second, which is 18.5 ms per tweet.

6.4.2. Prediction Quality

We see from Table 6.8 that the top performing Macro F_1 score is 0.5703, a score that does not seem very impressive. Fredriksen and Jahren (2016) found that the score of their classifiers dropped significantly from tests on the 2013 dataset to the 2016 dataset. Therefore they investigated the quality of the SemEval datasets, checking the number of duplicate tweets and whether or not the duplicate tweets had different annotations. They found that the 2015 and 2016 datasets had significantly more duplicates and more inter-annotator disagreement than those of 2013 and 2014, leading to lower scores.

To our knowledge, no previous sentiment analysis research has been evaluated against the complete set of SemEval datasets, like we have done. This makes the results hard to compare to other work. Combined with the skepticism of the SemEval sets of Fredriksen and Jahren, we wanted to compare the methods against each of the datasets, to see whether some methods perform much better or worse on certain datasets. The results are shown in Table 6.10. From these results we can see that there is a trend that the scores decrease for the later datasets.

The 2014-sarcasm-A dataset is the worst performing dataset by a good margin. All tweets in this dataset contain the hashtag #sarcasm. The results for this dataset might be an indication of the methods' inability to detect sarcasm. For instance, the tweet "On the bright side we have school today... Tomorrow and the day after ! #killmenow #sarcasm" is annotated as negative. The hashtags indicate sarcasm, and the otherwise positive-seeming tweet is negated. None of the methods classify this particular tweet as negative. The entire dataset comprises only 60 tweets, so the scores might not represent the general performance on sarcastic tweets. Also, note from Table 6.2 (p. 40) that the sarcasm set is the only one to have a majority of negative tweets (50 percent), while the average ratio of negative tweets for the SemEval datasets is 15.99 percent. The distribution of sentiment, the sarcastic content of the tweets, and its small size make this dataset different from the rest, and low scores are to be anticipated.

7. Optimizing System

The goal of the Ternary Sentiment Embedding Model is to create sentiment embeddings that can be used for the ternary Twitter sentiment classification task, i.e. classifying tweets as *positive*, *negative* or *neutral*. This chapter presents tests to find the best performing hyperparameters of the model for this task. The sentiment embeddings are tested by using a Support Vector Machine (SVM) classifier specified in Section 7.2.4. In addition, the different methods for distant supervision presented in Chapter 6 are compared, before the SVM classifier is optimized.

7.1. Optimization Plan

Initially, a search of the hyperparameters of the Ternary Sentiment Embedding Model is performed to find the model that performs best on the ternary classification task. The hyperparameters are tested by performing a search of manually selected values. For each hyperparameter tested, all others are kept fixed to test how varying each hyperparameter affects the model performance. However, the overall goal of the hyperparameter search is not to provide a study of how the model is affected by its hyperparameter, but rather to find optimal values to be used in further testing. The hyperparameter search is explained in Section 7.3.

The model is retrained using the optimal[1] hyperparameters found using datasets produced by different distant supervision methods. This is done to find the distant supervision method that is most effective in combination with the new sentiment embedding model, and that will be used for further testing.

Finally, a search is performed on the C parameter of the Support Vector Machine (SVM) classifier. When testing the classifier parameters, the sentiment embeddings produced when training the Ternary Sentiment Embedding Model with the optimal hyperparameters and best performing distant supervision method are used.

7.2. Experimental Setup

This section provides an overview of the data used in the experiments following.

7.2.1. Datasets for Training Word Embeddings

Using the Tweet Collector program (Section 5.1, p. 31), more than 500 million tweets had been collected at the time of the start of the experiments. Using the methods described

[1]Some revisions were made in order to save computation time.

in Section 6.2 (p. 41), the collection is iterated through, and datasets are saved for each distant supervision method. The filtering of Section 6.1 (p. 40) is applied. The datasets were not created at the same time, and tweets were collected simultaneously as datasets were created. Three datasets of one million tweets from each sentiment class is extracted from the total datasets for each of the distant supervision methods. This is done to create datasets with an even number of tweets from each method and sentiment label. The exception is for the Emoticon method, which only has 151,538 tweets annotated as negative. Therefore, the Emoticon datasets are limited to 150,000 tweets for each label. Table 7.1 shows the counts of tweets for each dataset before extracting the one million of each.

Dataset	Count	Positive		Negative		Neutral	
AFINN	43,014,617	15,751,203	36.62%	9,917,128	23.06%	17,346,286	40.33%
Combo A	47,154,022	15,397,901	32.65%	7,188,117	15.24%	24,568,004	52.1%
Combo B	47,154,022	14,769,815	31.32%	7,098,267	15.05%	25,285,940	53.62%
Emoticon	42,826,459	363,696	0.85%	151,538	0.35%	42,311,225	98.8%
Em. Ext.	42,236,058	2,843,462	6.73%	5,686,644	13.46%	33,705,952	79.8%
FJLC	44,715,931	10,415,622	23.29%	8,619,184	19.28%	25,681,125	57.43%
TextBlob	34,533,751	10,144,522	29.38%	3,792,504	10.98%	20,596,725	59.64%
VADER	43,130,805	19,160,339	44.42%	7,606,142	17.64%	16,364,324	37.94%

Table 7.1.: Distribution of sentiment for distant supervised datasets before limiting to 1 million of each sentiment (150,000 for Emoticon).

7.2.2. Datasets for Testing and Training Classifiers

In order to compare results to those of Tang et al. (2016), the datasets of SemEval 2013 are chosen for the optimization experiments. Training is done on the 2013-train-A set. Testing is done on the 2013-dev-A set as this was the validation set of the 2013 workshop. The 2013-test-A dataset is used for evaluation of the optimized model when comparing to the SemEval 2013 results in Section 8.5.2 (p. 69).

Dataset	Tweet count	Positive		Negative		Neutral	
2013-train-A	7129	2656	37.26%	1019	14.29%	3454	48.45%
2013-dev-A	1231	428	34.77%	250	20.31%	553	44.92%
Total	8360	3084	36.89%	1269	15.18%	4007	47.93%

Table 7.2.: Datasets for training and testing classifiers for optimizing the Ternary Sentiment Embedding Model.

7.2.3. Preprocessing

The collected tweets are preprocessed before they are used for training word embeddings. URLs, mentions, reserved words and numbers are removed. For an explanation of these terms see Section 2.5.5 (p. 14). The tweets are lower-cased and elongated words are reduced to contain a maximum of three repeating letters (see Section 2.1.5, p. 6).

7.2.4. Classifier

The classifier used in the experiments is the SVM classifier with a linear kernel as described in Section 5.6.4 (p. 38). For the experiments in Section 7.3 and 7.4 the classifier's C parameter is set to 1. This value is the recommended starting point for working with the SVM classifier according to the scikit-learn documentation[2]. In Section 7.5 a search on the SVM C hyperparameter is performed to further optimize the classifier.

7.2.5. Hyperparameters

A manual search is performed for finding the near-optimal hyperparameters to use when training the Ternary Sentiment Embedding Model. The hyperparameters and the values to test are explained in this section. Initially, the same hyperparameters are used as the ones used by Tang et al. (2016) for training Sentiment-Specific Word Embeddings (SSWE). These values are shown in bold in Table 7.3. When performing the search for one of the hyperparameters, the rest are kept at their initial values.

Alpha The α value decides the balance between context and sentiment influence on the word embeddings. Tang et al. (2014) used $\alpha = 0.5$, an even combination. An α of 0 would be a solely context-aware embedding, while an α of 1.0 would be a solely sentiment-aware embedding. See Section 4.3 (p. 28) for more about the α hyperparameter.

Context Window Size The size of the context windows. That is, the number of token embedding vectors to concatenate and feed to the linear layer at once, as described in Section 4.1 (p. 27).

Embedding Length The embedding length, or *dimension*, is the length of the word embedding vectors.

Hidden Layer Size The number of nodes in the hidden linear layer and the hardTanh layer.

Learning Rate The learning rate of the neural network. This is a rate that decides the impact of new observations on the network's weights.

Margin The objective of the Ternary Sentiment Embedding Model is to separate the context score of a correct context window and a corrupted context window as well as the sentiment scores of the correct context window by a given *margin*. See Section 4.1 (p. 27) for more details about the margin hyperparameter.

[2]`http://scikit-learn.org/stable/modules/svm.html`

Hyperparameter	Hyperparameter search values									
Alpha	0.0	0.1	0.2	0.4	**0.5**	0.6	0.8	1.0		
Context Window Size	1	2	3	5	**7**	9				
Embedding Length	**50**	75	100	125	150					
Hidden Layer Size	10	**20**	30	50	100					
Learning Rate	0.001	0.01	0.03	0.05	0.07	0.09	**0.1**	0.2	0.3	0.5
	0.7	0.9	1.1							
Margin	0.5	0.7	0.9	**1.0**	1.1	1.3	1.5	1.7	1.9	2.0
	3.0	4.0	5.0	10.0						

Table 7.3.: Values for the hyperparameter search. Initial values are in bold type.

7.3. Hyperparameter Search Results

The hyperparameter searches as explained in Section 7.2.5 are performed using the dataset created by the Fredriksen–Jahren Lexicon Classifier (FJLC) distant supervision method. This supervision method is chosen as it is the top Macro F_1 performer of the experiments in Section 6.4 (p. 46).

The following subsections show the results and graphs for each of the hyperparameters. As can be observed in the detailed results in Appendix A, the scores vary for each epoch. In order to get a more robust comparison for each hyperparameter, the Macro F_1 scores are averaged over epochs 10 to 20.

7.3.1. Alpha

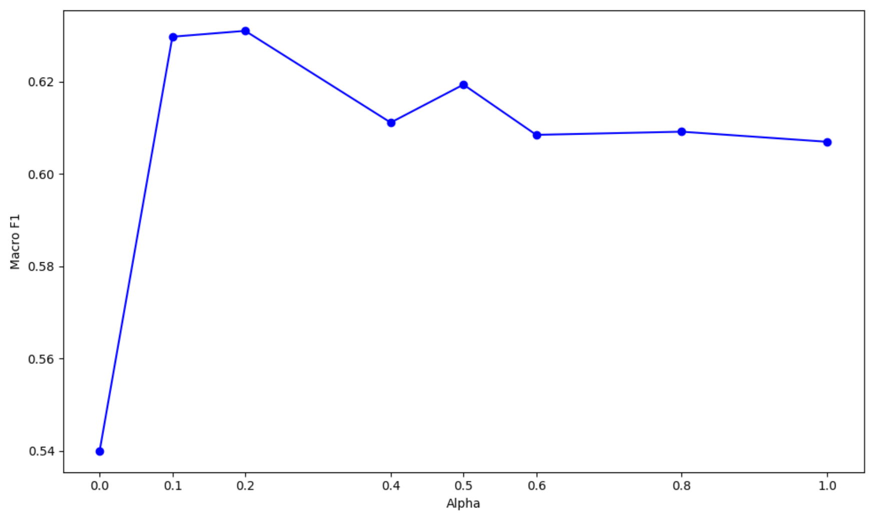


Figure 7.1.: α comparison.

Alpha	0.0	0.1	0.2	0.4	0.5	0.6	0.8	1.0
Macro F_1	0.5400	0.6297	**0.6310**	0.6111	0.6193	0.6084	0.6091	0.6069

Table 7.4.: α comparison values.

The hyperparameter α is the weighting between the sentiment loss and the context loss in the combined loss function (Equation 4.3, p. 29) used when training the Ternary Sentiment Embedding Model. Figure 7.1 and Table 7.4 show the results when training the Ternary Sentiment Embedding Model using different values for the α hyperparameter. For $\alpha = 0.0$ the model obtains its worst score, while the best score is achieved for $\alpha = 0.2$. This indicates that the contexts of the words are more important than the sentiment of the tweets. However, leaving out sentiment information altogether ($\alpha = 0$) is far worse than as for any other value for α, as can be seen in Table 7.4 and Table A.1. Interestingly, leaving out context information ($\alpha = 1$) does not perform as badly.

7.3.2. Context Window Size

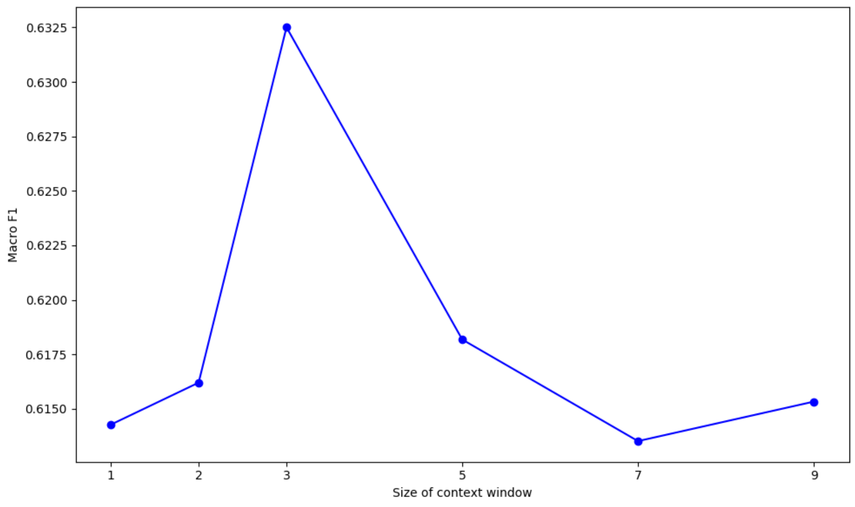


Figure 7.2.: Context window size comparison.

Size of context window	1	2	3	5	7	9
Macro F_1	0.6143	0.6162	**0.6325**	0.6182	0.6135	0.6153

Table 7.5.: Context window size comparison values.

The context window size specifies the number of words around a focus word to consider as context when learning word embeddings. Figure 7.2 and Table 7.5 show that the best performing window size is 3. This means that the model only considers the two words surrounding a focus word when learning word embeddings. Tweets are typically short texts with informal language due to the limit of their size. It is possible that having larger context windows will lead to the model considering excerpts that are too long for the language of the tweets, and that would fit better for more formal texts. However, the differences in the results are too small to draw such conclusions.

By having a smaller context window, the model is able to consider shorter tweets that would otherwise be ignored. This leads to the model using a smaller context window size being trained on more tweets that can provide additional context information when training. The effect of having more data to train on is also likely to increase the quality of the produced word embeddings, but a thorough comparison of dataset size is not provided in this thesis.

However, using a context window size of 1 or 2 (i.e. unigrams or bigrams) does not seem to improve the performance of the model. This indicates that some context information is indeed useful for training the model.

7.3.3. Embedding Length

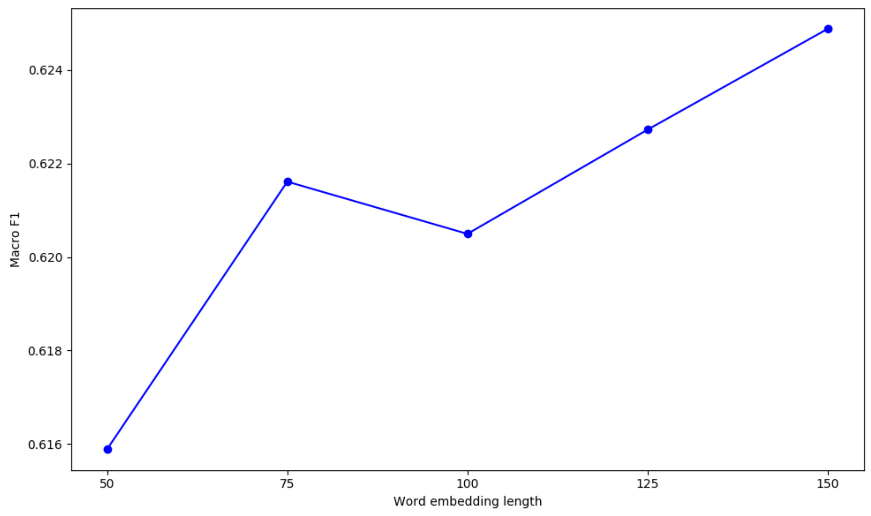


Figure 7.3.: Embedding length comparison.

Word embedding length	50	75	100	125	150
Macro F_1	0.6159	0.6216	0.6205	0.6227	**0.6249**

Table 7.6.: Embedding length comparison values.

The embedding length is the length, or *dimension*, of each word embedding vector. The larger the dimension, the more fine-grained information the vectors can hold. The best performing embedding length is 150. Though, if one inspects Table A.3, one can see that the top values are achieved for embedding length 125 for epochs 7 and 9.

The results indicate that larger embedding lengths result in better scores for the model. This is not so surprising, as other word embeddings like GloVe or word2vec are commonly trained with a dimension of 200 or 300. Training with such large embedding lengths would severely impact the processing time for the worse. Better results are to be expected with larger embedding lengths, but only slightly. Looking at Table 7.6, we see that the increase in Macro F_1 from embedding length 100 to 150 is $0.6249 - 0.6205 \approx 0.0044$, a minor improvement.

7.3.4. Hidden Layer Size

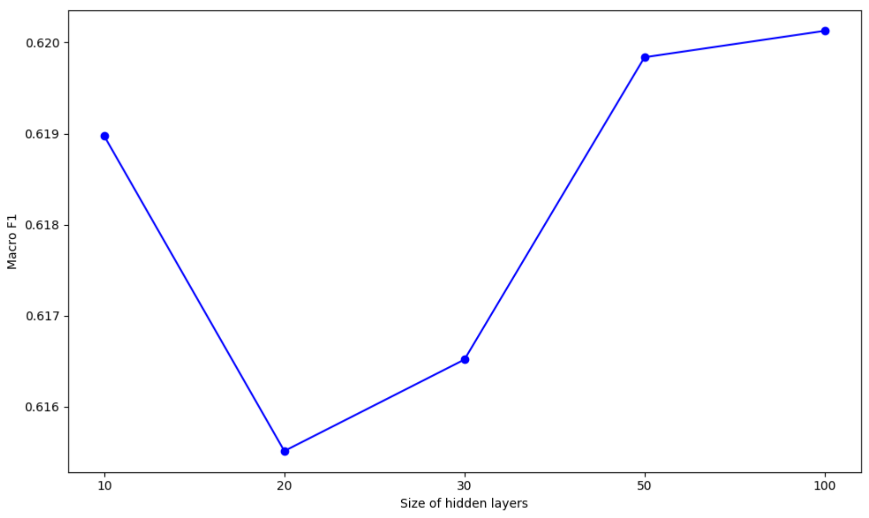


Figure 7.4.: Hidden length comparison.

Size of hidden layers	10	20	30	50	100
Macro F_1	0.6190	0.6155	0.6165	0.6198	**0.6201**

Table 7.7.: Hidden length comparison values.

The hidden layer size is the number of nodes, also called neurons, in the hidden layers of a neural network model. For the Ternary Sentiment Embedding Model, the hidden layers are the Linear Layer and the HardTanh Layer. The results in Figure 7.4 and Table 7.7 show that there is minimal impact on the score of the total system when varying the hidden layer size, having a range on the score values of only 0.0046. The best performance is achieved with the size 100. These results correspond well to the work of Collobert et al. (2011) who report that the size of the hidden layer, given it is of sufficient size, has limited impact on the generalization performance.

The size of the hidden layers has a significant impact on the runtime of training word embeddings using the model. Since the difference in score values were small compared to other hyperparameters, and because of the notes made by Collobert et al., it was decided to use a hidden layer size of 50 in the final model.

7.3.5. Learning Rate

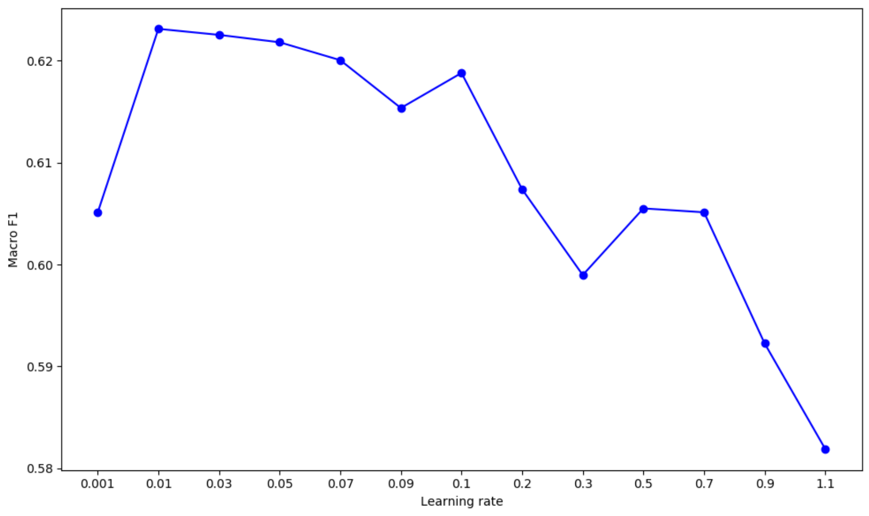


Figure 7.5.: Learning rate comparison.

Learning rate	0.001	0.01	0.03	0.05	0.07	0.09	0.1
Macro F_1	0.6051	**0.6231**	0.6225	0.6218	0.6200	0.6153	0.6188
Learning rate	0.2	0.3	0.5	0.7	0.9	1.1	
Macro F_1	0.6074	0.5990	0.6055	0.6051	0.5922	0.5819	

Table 7.8.: Learning rate comparison values.

The learning rate is the rate at which the parameters of the neural network are updated during backpropagation. When training a neural network, a small learning rate means that the network slowly converges towards a possible optimal score, while a large learning rate can make the network overshoot the optimum.

Figure 7.5 and Table 7.8 show that the best performing learning rate is 0.01. The total range of the scores is 0.0412. Note that the learning rates tested are not as evenly spread as Figure 7.5 seems to indicate.

7.3.6. Margin

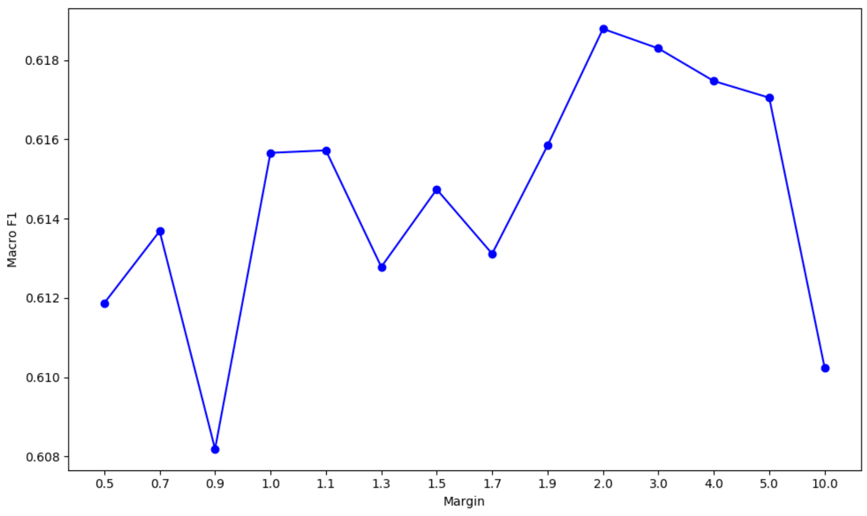


Figure 7.6.: Margin comparison.

Margin	0.5	0.7	0.9	1.0	1.1	1.3	1.5
Macro F_1	0.6119	0.6137	0.6082	0.6157	0.6157	0.6128	0.6147
Margin	1.7	1.9	2.0	3.0	4.0	5.0	10.0
Macro F_1	0.6131	0.6158	**0.6188**	0.6183	0.6175	0.6171	0.6102

Table 7.9.: Margin comparison values.

The margin hyperparameter is the value at which the scores produced by the Ternary Sentiment Embedding Model should be separated in the loss functions specified in Equation 4.1 (p. 27) and Equation 4.2 (p. 28). Having a larger margin leads to similar scores for each sentiment class giving a larger total loss, which leads to the model parameters being updated by a larger value during backpropagation. It is hard to predict the impact of having a higher margin, but since the loss is greater when sentiment scores are close, this leads us to believe that it gives a better separation of words from tweets belonging to each of the sentiment classes.

7.3.7. Summary of Best Parameters

The best parameter values from the hyperparameter search of this section are summarized in the middle column of Table 7.10. For the further experiments, the embedding length was reduced from 150 to 100 and hidden layer size reduced from 100 to 50. This was done to save computation time, compromising a minor loss in performance. The chosen hyperparameters are shown in the rightmost column of Table 7.10.

Hyperparameter	Best	Selected
Alpha	0.2	0.2
Embedding Length	150	100
Hidden Layer Size	100	50
Learning Rate	0.01	0.01
Margin	2	2
Window Size	3	3

Table 7.10.: Best and selected hyperparameter values found.

7.4. Comparing Distant Supervision Methods

The Ternary Sentiment Embedding Model with the hyperparameter values shown in the rightmost column of Table 7.10 is trained on datasets created by using each of the distant supervision methods from Chapter 6. The datasets comprise one million tweets from each sentiment class, with the exception of the dataset created by the Emoticon method that comprises 150,000 tweets for each class.

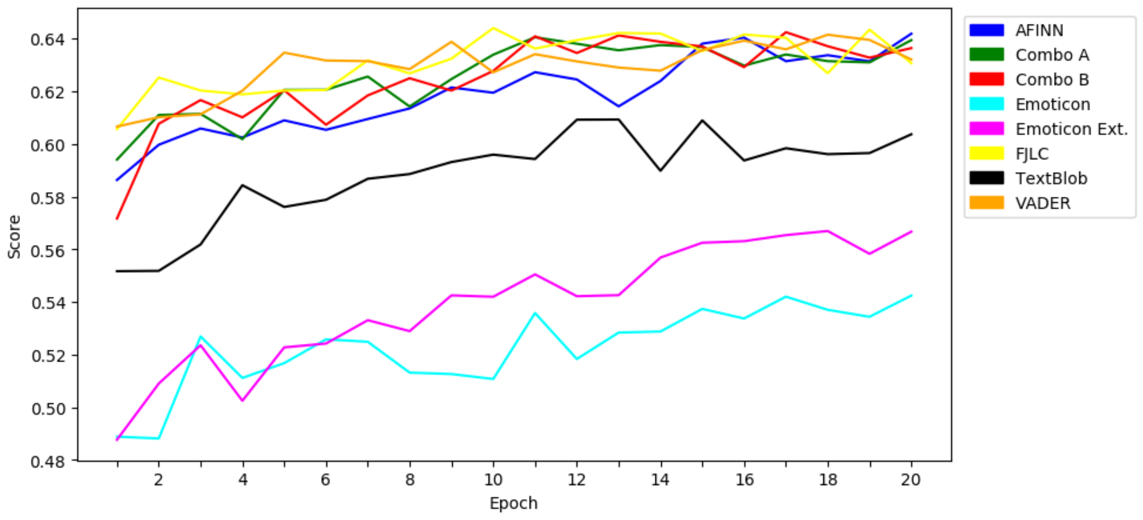


Figure 7.7.: Macro F_1 scores for the Ternary Sentiment Embedding Model on different datasets.

	Distant supervision method							
Epoch	AFINN	Cb. A	Cb. B	Em.	Em. Ext.	FJLC	TextBlob	VADER
1	0.5863	0.5940	0.5717	0.4889	0.4877	0.6056	0.5517	0.6066
2	0.5997	0.6110	0.6076	0.4882	0.5090	0.6252	0.5518	0.6101
3	0.6059	0.6114	0.6166	0.5269	0.5236	0.6203	0.5618	0.6112
4	0.6024	0.6018	0.6101	0.5112	0.5025	0.6188	0.5843	0.6202
5	0.6089	0.6205	0.6204	0.5168	0.5228	0.6202	0.5761	0.6346
6	0.6053	0.6205	0.6073	0.5258	0.5242	0.6205	0.5788	0.6317
7	0.6095	0.6256	0.6184	0.5249	0.5331	0.6317	0.5868	0.6314
8	0.6135	0.6142	0.6249	0.5132	0.5289	0.6268	0.5886	0.6284
9	0.6214	0.6246	0.6202	0.5126	0.5425	0.6324	0.5931	0.6388
10	0.6194	0.6339	0.6277	0.5108	0.5420	**0.6440**	0.5959	0.6272
11	0.6272	0.6405	0.6408	0.5358	0.5505	0.6362	0.5943	0.6341
12	0.6245	0.6380	0.6345	0.5184	0.5422	0.6394	0.6092	0.6313
13	0.6142	0.6355	0.6412	0.5284	0.5426	0.6421	0.6093	0.6290
14	0.6239	0.6375	0.6387	0.5288	0.5569	0.6419	0.5898	0.6278
15	0.6381	0.6368	0.6370	0.5374	0.5625	0.6352	0.6089	0.6356
16	0.6404	0.6297	0.6291	0.5337	0.5631	0.6415	0.5937	0.6392
17	0.6314	0.6339	0.6424	0.5420	0.5654	0.6403	0.5984	0.6359
18	0.6337	0.6314	0.6371	0.5370	0.5669	0.6269	0.5961	0.6414
19	0.6313	0.6309	0.6327	0.5344	0.5583	0.6434	0.5965	0.6395
20	0.6418	0.6394	0.6364	0.5424	0.5667	0.6307	0.6037	0.6320
$Avg_{10,20}$	0.6296	0.6352	0.6361	0.5317	0.5561	**0.6383**	0.5996	0.6339

Table 7.11.: Macro F_1 scores for word embeddings trained with the Ternary Sentiment Embedding Model using different distant supervision methods. $Avg_{10,20}$ depicts the average score over epochs 10 to 20, inclusive.

7.5. SVM Hyperparameter Search Results

A coarse parameter search on the C parameter of the SVM classifier is performed with values ranging from 0.001 to 1000. The word embeddings used are the ones produced by the Ternary Sentiment Embedding Model using the dataset created with the FJLC distant supervision method. The embeddings are trained for 20 epochs. The results are shown in Table 7.12.

C	Macro F_1
0.001	0.6355
0.01	**0.6404**
0.1	0.6326
1	0.6319
10	0.6300
100	0.6319
1000	0.6321

Table 7.12.: Coarse parameter search for C values between 0.001 and 1000.

From the coarse parameter search the observed best performing C value is 0.01. A finer search is conducted for C values between 0.001 and 0.009, and between 0.01 and 0.09. The results of these searches are found in Table 7.13 and Table 7.14.

C	Macro F_1
0.001	0.6355
0.002	0.6397
0.003	0.6404
0.004	0.6423
0.005	0.6428
0.006	**0.6429**
0.007	0.6416
0.008	0.6400
0.009	0.6414

Table 7.13.: Parameter search for C values between 0.001 and 0.009.

C	Macro F_1
0.01	0.6404
0.02	**0.6408**
0.03	0.6376
0.04	0.6356
0.05	0.6333
0.06	0.6328
0.07	0.6325
0.08	0.6314
0.09	0.6312

Table 7.14.: Parameter search for C values between 0.01 and 0.09.

The parameter search shows that the C-value of 0.006 yields the best performing classifier for the model on the 2013-dev-A set, which achieved a Macro F_1 score 0.6429. The most commonly used C value for SVM is 1, which achieved a score of 0.6319. This leads to an increase of $0.6429 - 0.6319 = 0.011$.

The C parameter specifies the balance between separating samples by a large margin and the amount of misclassified samples. A small C value means the classifier favors more misclassified samples over a large margin.

The results show that the small C value of 0.006 performs best for our system, indicating that it is hard to avoid misclassifying some samples. However, the differences in scores are very low even for large variations of the parameter, meaning the samples to classify are not easily linearly separable either way.

8. Evaluating the Final System

This chapter presents tests performed to evaluate the performance of the Ternary Sentiment Embedding Model for creating sentiment embeddings for the ternary classification task. A test is performed to examine how the different classifiers of Chapter 6 compare when used as distant supervision for the proposed model. The model is compared to the Hybrid Ranking Model by Tang et al. (2016) using different distant supervision methods. The model is compared to a range of baseline systems, among them other popular word embeddings models. Finally, the performance of the total system is evaluated against published results of state-of-the-art systems.

8.1. Experimental Setup

This section explains the setup of the final Twitter Sentiment Analysis (TSA) system and the datasets used during testing.

8.1.1. Final System

The TSA system comprises the Ternary Sentiment Embedding Model and the Support Vector Machine (SVM) classifier with the hyperparameters found in Chapter 7. The final system has the following properties:

Hyperparameter	Value
Alpha	0.2
Context Window Size	3
Embedding Length	100
Hidden Layer Size	50
Learning Rate	0.01
Margin	2.0
Number of Epochs	20

Table 8.1.: Hyperparameter values of final model.

For all tests with word embeddings, the same SVM classifier is used. The classifier is found by the hyperparameter search in Section 7.5. The SVM classifier has a linear kernel and parameter $C = 0.006$.

When not explicitly stated otherwise, the Ternary Sentiment Embedding Model is trained on a dataset created by using the Fredriksen–Jahren Lexicon Classifier (FJLC) as distant supervision method. The dataset is created by collecting one million tweets for each sentiment class.

8.1.2. Datasets for Testing and Training Classifiers

The International Workshop on Semantic Evaluation (SemEval) provides labeled datasets to be used when training, validating and testing TSA systems.

In order to compare the proposed TSA system against results from the SemEval workshop, two additional datasets shown in Table 8.3 are created by combining datasets from Table 8.2. The 2013-2016-train-dev-A dataset is created by combining the 2013-dev-A, 2013-train-A, 2014-sarcasm-A, 2015-train-A and 2016-train-A datasets. The 2013-2016-all-A is created by combining all the datasets from 2013 to 2016 in Table 8.2. The reasoning for these combinations are further explained in Section 8.5.2 (p. 69).

The tweets were downloaded on March 21, 2017. The tweet counts shown in Table 8.2 and Table 8.3 are after downloading[1], removing duplicates[2], and removing retweets and non-English tweets as described in Section 5.4.2 (p. 34).

Dataset	Tweet count	Positive		Negative		Neutral	
2013-dev-A	1228	430	35.02%	245	19.95%	553	45.03%
2013-test-A	2695	1090	40.45%	400	14.84%	1205	44.71%
2013-train-A	7109	2660	37.42%	1010	14.21%	3439	48.38%
2014-sarcasm-A	56	22	39.29%	27	48.21%	7	12.5%
2014-test-A	1460	762	52.19%	149	10.21%	549	37.6%
2015-test-A	1865	793	42.52%	285	15.28%	787	42.2%
2015-train-A	352	118	33.52%	43	12.22%	191	54.26%
2016-dev-A	1657	702	42.37%	324	19.55%	631	38.08%
2016-devtest-A	1645	811	49.3%	254	15.44%	580	35.26%
2016-test-A	16771	5790	34.52%	2512	14.98%	8469	50.5%
2016-train-A	4893	2535	51.81%	696	14.22%	1662	33.97%
2017-test-A	12284	2375	19.33%	3972	32.33%	5937	48.33%

Table 8.2.: Datasets for training and testing classifier.

Dataset	Tweet count	Positive		Negative		Neutral	
2013-2016-train-dev-A	13638	5765	42.27%	2021	14.82%	5852	42.91%
2013-2016-all-A	39731	15713	39.55%	5945	14.96%	18073	45.49%

Table 8.3.: Combined datasets for training and testing classifier.

[1]The number of downloaded tweets is likely lower than the number of tweet IDs to fetch, because tweets that have been deleted will not be downloaded.

[2]If duplicate tweets with the same sentiment label are found, only one is kept. If duplicate tweets are found with different labels, both are deleted.

8.2. Comparison of Distant Supervision Methods

The Ternary Sentiment Embedding Model is trained for 20 epochs using the different distant supervision methods of Chapter 6. The produced sentiment embeddings are tested using the SVM classifier specified in Section 8.1.1 using 10-fold cross validation on the 2013-2016-all-A dataset from Table 8.3. Table 8.4 shows different metrics for the tests. The metrics used are described in Section 2.3.2 (p. 7).

Method	Macro F_1	F_1^{PN}	F_1^{POS}	F_1^{NEG}	F_1^{NEU}
Combo B	0.6091	0.5952	0.6681	0.5222	0.6369
Combo A	0.6082	0.5958	0.6673	0.5244	0.6330
FJLC	0.6036	0.5871	0.6651	0.5091	0.6366
AFINN	0.6022	0.5892	0.6602	0.5181	0.6283
VADER	0.5965	0.5831	0.6555	0.5108	0.6231
TextBlob	0.5837	0.5714	0.6566	0.4862	0.6083
Em. Ext.	0.5479	0.5247	0.6303	0.4192	0.5943
Em.	0.5043	0.4814	0.5953	0.3676	0.5501

Table 8.4.: Comparison of distant supervision methods, sorted by descending Macro F_1 score. The tests are performed using 10-fold cross validation on the 2013-2016-all-A dataset.

8.3. Comparison with Hybrid Ranking Model

To get a robust comparison of the performance of the sentiment embeddings produced by the Ternary Sentiment Embedding Model and the Hybrid Ranking Model, a 10-fold cross-validation is performed. This cross-validation allows for using all manually annotated SemEval datasets, while producing robust scores with low bias. Both architectures are trained on three million tweets weakly annotated using the different classifiers presented in Chapter 6 as distant supervision methods. The Ternary Sentiment Embedding Model is trained on tweets labeled as "positive", "negative" or "neutral", with one million of each. The Hybrid Ranking Model only utilizes tweets labeled as "positive" or "neutral", and is as a result trained on 1.5 million tweets of each sentiment class. The models were trained for 20 epochs.

The Ternary Sentiment Embedding Model is trained with the hyperparameters stated in Section 8.1.1, while the Hybrid Ranking Model is trained using the hyperparameters used by Tang et al. (2016), the bold values found in Table 7.3 (p. 52). The produced sentiment embeddings are fed to the SVM classifier stated in Section 8.1.1. The dataset used in the 10-fold cross-validation is the 2013-2016-all-A dataset. The Macro F_1 metric is used, and the results are shown in Table 8.5.

Dataset	Ternary Sentiment Embedding Model	Hybrid Ranking Model
AFINN	**0.6022**	0.5775
Combo A	**0.6082**	0.5873
Combo B	**0.6091**	0.5920
Em.	0.5043	**0.5284**
Em. Ext.	**0.5479**	0.5357
FJLC	**0.6036**	0.5919
TextBlob	**0.5837**	0.5746
VADER	0.5964	**0.5965**

Table 8.5.: Comparison between the Ternary Sentiment Embedding Model and the Hybrid Ranking Model of Tang et al. (2016) using different distant supervision methods.

8.4. Comparison with Baselines

In order to see how well the final TSA system performs, a comparison between existing sentiment analysis systems is done. The systems are tested using 10-fold cross-validation on the 2013-2016-all-A dataset from Table 8.3. Most of these baseline methods were implemented as part of the experiments of Chapter 6. As explained in Section 7.2.1 (p. 49), the Ternary Sentiment Embedding Model is trained on three million tweets, with one million from each of the sentiment classes. The tweets are classified using the FJLC distant supervision method. The same dataset of tweets is used to train GloVe and word2vec word embeddings. The Hybrid Ranking Model, which only utilizes positive and negative tweets, is trained using 1.5 million positive and 1.5 million negative tweets in order to keep the size of the dataset the same as for the other word embedding models.

8.4.1. Description of Baselines

AFINN The AFINN library as explained in Section 2.5.8 (p. 15).

Combo A The Combo Average model of Section 6.2.6 (p. 44) with parameters $a = 0.0$, $b = 0.4$, $c = 0.4$, $t = 0.2$, as found from the grid search of Section 6.3.3 (p. 46).

Combo B The Combo Average model of Section 6.2.6 with parameters $a = 0.3$, $b = 0.1$, $c = 0.1$, $t = 0.1$, as found from the grid search of Section 6.3.3.

Emoticons The Emoticons method of Go et al. (2009), as explained in Section 6.2.1 (p. 41), with the variation that tweets containing both negative and positive emoticons are regarded as neutral.

Emoticons Extended The extended Emoticons method as explained in Section 6.2.2 (p. 42), with the variation that tweets containing both negative and positive emoticons are regarded as neutral.

GloVe The GloVe model from Section 2.4.2 (p. 12) is used to train word embeddings of dimension 100 on a set of three million tweets, the same tweets as the proposed system. The model is trained using the Twitty program described in Section 5.4 (p. 34). The word embeddings are fed to the SVM classifier specified in Section 8.1.1.

Hybrid Ranking Model (w/FJLC) Word embeddings are produced using the Hybrid Ranking Model of Tang et al. (2016) trained on a set of three million tweets created by using the FJLC distant supervision method. The Hybrid Ranking Model only uses tweets labeled as "positive" or "negative" when training word embeddings, and 1.5 million tweets of each class are used for training. The word embeddings are fed to the SVM classifier specified in Section 8.1.1.

FJLC Using the Python port of the Lexicon Classifier of Fredriksen and Jahren (2016) explained in Section 6.2.7 (p. 44).

Random Uniform Picks a random label from a uniform probability distribution, that is, each label has equal chance of getting picked.

Random Weighted Picks a random label from the same distribution as in the training set. If 50 percent of the training data are neutral tweets, "neutral" has a probability of 50 percent of getting picked.

TextBlob The TextBlob method as described in Section 6.2.4 (p. 42) with sentiment threshold 0.1 and polarity threshold 0.3. Tweets achieving scores that do not satisfy the thresholds are assigned the label "neutral".

VADER The method described in Section 6.2.5 (p. 43) with threshold $t = 0.1$. Tweets achieving scores not satisfying the threshold for any of the sentiments were labeled as neutral.

word2vec (CBOW) Word embeddings are trained using the Continuous Bag-of-Words (CBOW) word2vec model described in Section 2.4.1 (p. 11). The model is trained using the Twitty program described in Section 5.4 (p. 34). The embeddings are trained on the dataset created by using the FJLC distant supervision method, and the dimension of the embeddings is 100. The word embeddings are fed to the SVM classifier specified in Section 8.1.1.

word2vec (Skip-gram) Word embeddings are trained using the Continous Skip-gram word2vec model described in Section 2.4.1. The model is trained using the Twitty program described in Section 5.4. The embeddings are trained on the dataset created by using the FJLC distant supervision method, and the dimension of the embeddings is 100. The word embeddings are fed to the SVM classifier specified in Section 8.1.1.

8.4.2. Results

Table 8.6 shows the results for each TSA system using the Macro F_1 metric.

Model	Macro F_1
Final Ternary Sentiment Embedding Model	0.6036
word2vec (CBOW)	0.6015
Hybrid Ranking Model (w/FJLC)	0.5919
word2vec (Skip-gram)	0.5886
FJLC	0.5706
GloVe	0.5662
Combo B	0.5621
Combo A	0.5579
AFINN	0.5381
VADER	0.5286
TextBlob	0.3826
Random Weighted	0.3315
Random Uniform	0.3174
Emoticon Extended	0.2542
Emoticon	0.2462

Table 8.6.: Comparison between the Ternary Sentiment Embedding Model and baselines using 10-fold cross validation on the 2013-2016-all-A dataset, ordered by decreasing Macro F_1 score.

8.5. Comparison with Published Results

The final TSA system is trained and tested using similar training and testing sets as other published TSA systems to provide a fair comparison with published results.

8.5.1. Comparison with Tang et al. (2016)

Tang et al. (2016) present the Hybrid Ranking Model by which the proposed Ternary Sentiment Embedding Model is heavily influenced. Tang et al. train sentiment embeddings on five million positive and five million negative distant-supervised tweets. The sentiment embeddings produced by their model are tested with a SVM classifier on the SemEval 2013 test dataset. The score from Tang et al. (2016) is presented in Table 8.7[3]. Tang et al. achieved improved results using additional lexical features. These are not provided, as we only compare to the most similar system to ours.

[3]Three decimals are used as in Tang et al. (2016).

Model	Macro F_1
Final Ternary Sentiment Embedding Model	0.655
Hybrid Ranking Model	0.634

Table 8.7.: Comparison between the scores of the final proposed TSA system and published results from Tang et al. (2016).

8.5.2. Comparison with SemEval

To see how the final TSA system compares to state-of-the-art systems from SemEval, tests are done against the appropriate test sets and compared with the published results. The workshops and tasks that are compared with are SemEval 2013 Task 2B (Nakov et al., 2013), SemEval 2016 Task 4A (Nakov et al., 2016), and SemEval 2017 Task 4A (Rosenthal et al., 2017).

The proposed TSA system is trained using the training sets provided by the respective conference. For 2013, the model is trained on 2013-train-A and tested on 2013-test-A. For 2016, the model is trained on 2013-2016-train-dev-A given in Table 8.3 (p. 64), and tested on 2016-test-A. The SemEval 2016 workshop allowed training on the training and development datasets of previous years, as well as the training dataset from 2016. For 2017, the model is trained on 2013-2016-all-A from Table 8.3 as the SemEval 2017 workshop allowed systems to be trained on all datasets from previous years. The system is tested on the 2017-test-A dataset.

Note that the number of tweets that could be downloaded is smaller than at the time of the respective workshops, as tweets have been deleted in the meantime. The metric used is F_1^{PN}-score, which is the official SemEval metric.[4] The results are presented in Table 8.8 with the same number of decimals as in the published papers.

Year	Top SemEval Result	Ternary Sentiment Embedding Model
2013	0.6902	0.6178_{9}
2016	0.633	0.5805_{12}
2017	0.685	0.6291_{9}

Table 8.8.: Comparison of the Ternary Sentiment Embedding Model with top results from different SemEval years. The scores are F_1^{PN} scores. The subscripts denote the ranking the systems would have achieved for each year.

[4]The official metric of SemEval 2017 is *AvgRec*, but F_1^{PN} scores are also included.

9. Discussion

In this chapter, the results from Chapters 6, 7, and 8 are evaluated and discussed.

9.1. Distant Supervision

From the comparison results in Table 6.8 (p. 47), we see that the Lexicon Classifier (FJLC) of Fredriksen and Jahren (2016) achieved the top Macro F_1 score. AFINN achieved the top F_1^{PN} score, which is the official International Workshop on Semantic Evaluation (SemEval) metric for both the binary and ternary tasks.

The SemEval tweets are pretty clean and similar, with very little spam. The collected tweets on the other hand, are noisy and probably contain a lot more spam. Consequently, we expect the inclusion rates to be lower when the distant supervision methods are run on the collected data compared to when run on SemEval data.

The slowest of the distant supervision methods (Combo B) uses 2.27 ms to classify a tweet, which is minuscule compared to the 18.5 ms it takes to download a tweet. Therefore, all distant supervision methods discussed here could be used for real-time annotation of tweets from the Twitter Streaming API. When processing a collection of saved tweets, the runtime has a larger impact: processing 500 million tweets would take about 13 days with the Combo B method (2.27 ms/tweet), but only about twelve hours with the Emoticon method (0.09 ms/tweet).

It is interesting that it is not consistent which method performs best for all datasets. While FJLC comes out on top for most of the sets, also the Combo A, TextBlob and VADER methods have at least one top performing result each.

9.1.1. Combination Methods

Both Combo methods perform better than all of their component methods. This indicates that the averaging scheme is able to overcome weaknesses of the component methods. Other component method combinations could be investigated.

The number of component methods could also be experimented with. Bonab and Can (2016) found that theoretically, having the same number of classifiers in an ensemble as the number of classes, achieves the best scores. Therefore, combining three methods is not an unreasonable choice. One should also keep in mind that more components lead to higher processing times, which is not favorable for distant supervision.

Other approaches to combining the method scores could prove to be better than averaging. The *most confident* approach would let the component method that is most confident in its classification determine the label for the tweet. The *majority* voting scheme would select the label that the majority of component methods labeled the tweet as. An advantage of the averaging scheme over the most confident and majority approaches is that the average works on the continuous sentiment values that each component method generates, generating a new score which is then mapped to a label. In other words, the averaging approach works on a more accurate level than the other approaches where each component method maps its value to a class before combining them.

9.1.2. Speed versus Quality

Although the combination methods perform well in terms of prediction quality, they are by far the slowest methods, as they have to compute scores for each of their component methods. As mentioned in the introduction of Chapter 6 (p. 39), speed might be a more important trait than quality for distant supervision methods. Therefore, the VADER method, for instance, might be preferred over any of the Combo methods, even if it gets a 6.37 percent worse Macro F_1 score, since it is nearly 3.5 times faster. A clearer choice perhaps is the FJLC, which is the top Macro F_1 performer and runs approximately 2.3 times faster than the Combo methods.

While the Emoticon method is doing poorly in terms of prediction, it classifies positive and negative tweets with high precision (Table 6.9, p. 47). Its weakness is that it predicts all tweets with none of its emoticons as neutral, which leads to a very low precision on neutral. This shows that the emoticons of Table 6.3 (p. 41) represent their sentiment classes well, and the proper way to extend this method for the ternary task would be to find a set of symbols that represent the neutral class equally well. Such a method would combine both high speed and high precision. When building a distant-supervised dataset from a large corpus of tweets, recall is not as important, as tweets can be ignored, at the cost of a smaller resulting set.

9.2. Optimizing System

In this section, the procedure of finding the optimal system performed in Chapter 7 is discussed.

9.2.1. Limitations of the Hyperparameter Search

Chapter 7 presents a manual search of the hyperparameters of the Ternary Sentiment Embedding Model. The model is trained with various, manually selected, hyperparameter values. During the search, only one parameter was changed at a time, while the other values were set to the initial values chosen. An exhaustive grid search, where all combinations of parameter values are tested against each other, is needed for finding the optimal hyperparameters. This, however, would have been too time consuming, as training the Ternary Sentiment Embedding Model once takes between 24 and 36 hours depending on the hyperparameters and the size of the dataset.

One classifier, the FJLC, was chosen as distant supervision method for the search. It is not clear that the best hyperparameters found using this method will generalize to the system when trained on data created when using another distant supervision method. Optimally, a new hyperparameter search should have been performed using each of the distant supervision methods.

9.2.2. Data for the Hyperparameter Search Classifier

The word embeddings produced by using the different hyperparameters were used to train and classify tweets using a Support Vector Machine (SVM) classifier. As specified in Section 7.2.2 (p. 50), the classifier was trained on the 2013-train-A dataset from the SemEval workshop. The 2013-dev-A dataset was used as validation set. These datasets were chosen to get training and validation as close as possible to the procedure in Tang et al. (2016) in order to get a better comparison of the two systems. These datasets are fairly small in size. The model is also randomly initialized as reported in Section 4.4 (p. 29), which could lead to varying results, especially when the differences between the scores are as low as they are. The purpose of the hyperparameter search, however, is not to establish the effect of varying each hyperparameter, but rather to create a model that could be reasonably compared to previous models, and be functionally used to create word embeddings for ternary sentiment classification.

9.2.3. Evaluation of Hyperparameter Search Results

Because of the implementation of the Ternary Sentiment Embedding Model, which outputs embeddings for each epoch trained, it was easy to include embeddings at all epochs up to a given limit. For the hyperparameter search, the model was trained for a total of 20 epochs. During the search, we observed that results varied from epoch to epoch and we chose to average the scores over a certain interval of epochs. This way, the scores were more robust for picking optimal hyperparameters. The tendency observed was that after 10 epochs, the results seemed to converge. Therefore, scores were averaged over epochs 10 to 20, inclusive, and the number of epochs is not considered an experimental hyperparameter like the rest.

The scoring metric for the hyperparameter search was the Macro F_1-score. This metric was chosen as we found it to better represent all three classes in the ternary classification task compared to the F_1^{PN} score, the previous official metric of the SemEval workshop, which only looks at F-scores for the positive and negative class. The Macro F_1 score is also the one used in Tang et al. (2016), and to get a better comparison to their model we chose the same score as metric.

9.2.4. Distant Supervision Method

After selecting the hyperparameters that performed best in each hyperparameter search, the Ternary Sentiment Embedding Model was trained on datasets created by using each of the distant supervision methods from Chapter 6. The tests are done in order to find a distant supervision method that should be used for the final experiments. By using the 2013-dev-A dataset as validation set, the selection of distant supervision method is not dependent of results from the test set, which is in accordance to the procedure used by the systems in the SemEval workshop.

The results in Section 7.4 (p. 59) show that the performance is very similar for five of the distant supervision methods, with the FJLC achieving the best results. As discussed in Section 9.1, the FJLC method is the best performing classifier from Chapter 6 using the Macro F_1 score as metric. It is also the distant supervision method used in the hyperparameter search. This is a likely explanation for why this distant supervision method is the best performing in this comparison as well.

9.3. Comparison of Distant Supervision Methods

As seen in Table 8.4 (p. 65), Combo B is the best performing distant supervision method. Interestingly, this is not the same as the top performer of Chapter 6 nor Section 7.4 (p. 59), which was the FJLC.

The results show the performance is similar for all the distant supervision methods, with the clear exceptions of the Emoticons methods and arguably the TextBlob method. The weakness of the Emoticon methods is clear — the low number of emoticons in tweets lead to the Emoticon methods classifying the vast majority of tweets as neutral, as can be seen in Table 7.1 (p. 50). The extended method performs slightly better, because of its larger sets of emoticons and inclusion of emojis. Also, the Emoticon Extended dataset with its three million tweets is much larger than the Emoticons set, which comprises 450,000 tweets.

The TextBlob does not perform very well in this comparison. When looking at the results from comparing the distant supervision methods in Table 6.8 (p. 47), the TextBlob method does not perform notably worse than the other methods when looking at the Macro F_1 score. The method achieves high F_1 scores for the positive and neutral class, but the F_1 score for the negative class is significantly lower compared to the other methods (excluding the Emoticon methods). The scores in Table 6.9 (p. 47) show that the recall for the negative class is low, meaning that the classifier is not good at classifying true negative tweets as negative. This weakness of the TextBlob classifier seems to persist when used as distant supervision method as the F_1 score for the negative class in Table 8.4 (p. 65) is the lowest, when excluding the Emoticon methods.

The Combo methods perform best in this comparison. Since they average over three methods, as mentioned in Section 9.1.1, they can overcome weaknesses of their component methods. While the combo methods were not the top performers in the comparison of the distant supervision methods in Section 6.4 (p. 46), the ability to balance other weak classifiers seems to be important when used as distant supervision methods for the proposed model.

The results show that the Ternary Sentiment Embedding Model performs best when trained on data from a distant supervision method that is good at classifying all tweets into all three sentiment classes. Methods such as the Emoticon classifiers and the

TextBlob classifier have weaknesses when classifying tweets into one or more of the classes, and as a results these distant supervision methods yield the worst results for the total system. The Combo methods seem to overcome the weaknesses of individual methods, resulting in the best performance of the model on the ternary sentiment classification task.

9.4. Comparison with Hybrid Ranking Model

The proposed Ternary Sentiment Embedding Model is closely related to the Hybrid Ranking Model by Tang et al. (2016). The main goal of the new model is to improve the performance when used as part of a ternary classification system. Since both models are trained using distant supervised datasets, it is interesting to test how such distant supervision methods affect both models, and test if the new model achieves improved results on the task.

Table 8.5 (p. 66) shows the results when testing word embeddings produced by both models, using the different distant supervision methods from Chapter 6. The results show that the Ternary Sentiment Embedding Model outperforms the Hybrid Ranking Model using all but two distant supervision methods.

As explained in Section 7.2.1 (p. 49), the dataset created using the Emoticon distant supervision method only consists of 150,000 tweets for each sentiment class. The word embeddings for the other distant supervision methods are trained using three million tweets, one million per class. As discussed in Section 9.3, the smaller size of the Emoticon dataset likely explains the poor performance when using this distant supervision method. The method is the worst performing for both models.

The Emoticon method classifies a very large percentage of tweets as neutral (98.8 percent of our manually collected tweets seen in Table 7.1, p. 50). Only tweets that contain one or more emoticons from a small set of emoticons are classified as positive or negative, while all others are seen as neutral. This conservative classification likely means that the tweets that are actually classified as positive or negative have a high likelihood of being classified correctly. This assumption is reflected by the results found in Table 6.9 (p. 47). The precision for the negative and positive class for the Emoticon method is the highest of all the methods.

The Emoticon dataset is the only dataset where the Hybrid Ranking Model performs significantly better than the proposed model. The Hybrid Ranking Model is trained using only tweets labeled as positive or negative, while the Ternary Sentiment Embedding Model also utilizes neutral tweets in its training. As seen above, the Emoticon distant supervision method performs well for classifying tweets as positive or negative, but not for neutral, meaning the quality of the positive and negative tweets are likely higher than for neutral tweets. This possibly explains why the Hybrid Ranking Model performs better when using this distant supervision method.

Tang et al. (2016) use a distant supervision method similar to the Emoticon method, looking only at emoticons for creating positive and negative tweets to use during training of the Hybrid Ranking Model. This seems like a good choice for their model considering the high precision for such datasets. For our model, however, it is not sufficient to only find some tweets that are likely positive or negative, and a more sophisticated distant supervision method seems essential. This also means that when training the proposed model, we are able to utilize a much larger corpus of distant supervised tweets since no tweets are discarded. This means that our model is able to train on much more varied tweets, learning sentiment information from tweets that are not limited by their use of emoticons.

When using the more sophisticated distant supervision methods, the Ternary Sentiment Embedding Model outperforms the Hybrid Ranking Model, with the exception of VADER where the scores are almost identical. This indicates that the proposed model is able to better take advantage of sentiment information from a larger set of tweets, increasing performance when used for the ternary sentiment classification task.

9.5. Comparison with Baselines

In Section 8.4 (p. 66) the final TSA system is compared to a range of baseline systems to assert its performance on the ternary classification task. We create baselines using the popular word2vec and GloVe word embedding models to test how the sentiment embeddings produced by the Ternary Sentiment Embedding Model compares to the widely used models. The word embeddings produced are used as input features for the SVM classifier described in Section 8.1.1 (p. 63). The classifiers described in Chapter 6 are also tested on the same data, as well as two classifiers that randomly classify the data.

From the results in Table 8.6 (p. 68), we observe that the best scores are achieved by the word embedding systems, with our system outperforming the baselines. The word embeddings produced by the Ternary Sentiment Embedding Model achieve slightly better results than the word embeddings produced by the Continuous Bag-of-Words (CBOW) word2vec model, however, the difference is small. One of the strengths of the word2vec models is that they require much less time for training compared to the larger neural network models like the Collobert and Weston model and the proposed Ternary Sentiment Embedding Model. The word2vec models use approximately three minutes, while the Ternary Sentiment Embedding Model uses 24 hours to train on three million tweets for the tests. This advantage of the word2vec models means that they could be trained using a much larger dataset, which would likely yield an even better performance for these word embedding models. The word2vec models do not utilize sentiment information of the tweets, which is necessary to create sentiment embeddings with the Ternary Sentiment Embedding Model. This is another advantage of the word2vec models, as they do not have the need for a separate distant supervision method. The word2vec models are, however, slightly outperformed by the Ternary Sentiment Embedding Model in terms of the final score, and with further optimization the difference could increase.

The Hybrid Ranking Model is created for training using binary distant-supervised data, and performs best when used on the binary classification task, i.e. classifying tweets as positive or negative. By incorporating the neutral class, the Ternary Sentiment

Embedding Model is able to improve the performance on the ternary classification task.

The final Ternary Sentiment Embedding Model is trained on tweets that are labeled using the FJLC. From the results we observe that by training word embeddings and using these as features for an SVM classifier, the performance is improved. This indicates that the proposed system is able to learn additional information, such as context information, from the tweets in addition to the sentiment information provided by the FJLC classifier.

9.6. Comparison with Published Results

The focus of the work in this thesis has been to create a new model for training sentiment embeddings that can be used for the ternary Twitter sentiment classification task. However, to get a sense of the performance of the proposed system, it is compared to other TSA systems.

9.6.1. Comparison with Tang et al. (2016)

The results shown in Table 8.7 (p. 69) indicate that the Ternary Sentiment Embedding Model performs better on the ternary classification task than the Hybrid Ranking Model of Tang et al. (2016), even though their embeddings were trained on a much larger set of data than the embeddings used in the present work.

9.6.2. Comparison with SemEval

The results from Table 8.8 (p. 69) show that the Ternary Sentiment Embedding Model does not match the state-of-the-art systems of the different conferences. There are multiple possible reasons to this. Our model is optimized for achieving the best Macro F_1 score possible in Chapter 7. The SemEval uses F_1^{PN} as official metric. Had our system been optimized for F_1^{PN}, better scores for this metric would probably have been achieved. Also, the SemEval contenders might have trained their systems on other or more data than we have. As tweets have been deleted over time, we could not download as many tweets as were available at the time the respective conferences were held.

10. Conclusion and Future Work

This chapter comprises a conclusion of the thesis and description of future work to further improve the proposed model or the overall Twitter Sentiment Analysis system.

There are three main contributions of this thesis. A comparison of various distant supervision methods was performed. The Ternary Sentiment Embedding Model was developed to improve sentiment embeddings for the ternary classification task. To perform the above, several tools and programs were developed, many of these open-source.

10.1. Distant Supervision

We performed a comparison of the speed and prediction quality of various simple and sophisticated lexicon-based classifiers. The method that achieved the best Macro F_1 score was the Fredriksen–Jahren Lexicon Classifier (FJLC) by Fredriksen and Jahren (2016). The classifier obtained the best F_1 scores for both the negative and neutral classes. The Emoticon classifier was the fastest of the classifiers. It performed well for creating small datasets of positive and negative tweets with high precision. However, the classifier came short when used to classify tweets for all three classes. An attempt at improving the Emoticon classifiers was made by creating the Emoticon Extended classifier. The classifier achieved slightly better Macro F_1 score, but was significantly worse than the Emoticon classifier for retrieving negative tweets, meaning it is less suitable for creating binary distant-supervised datasets. TextBlob obtained the best F_1 score for the positive class, but performed substantially worse for negative tweets compared to the other classifiers. The Combo methods performed better than each of the constituent classifiers. Since the method calculates scores for three methods, it is the slowest of all the tested classifiers.

10.2. Ternary Sentiment Embedding Model

In this thesis we have proposed the Ternary Sentiment Embedding Model, a model for training sentiment embeddings, specialized for ternary sentiment classification. The model is based on the Hybrid Ranking Model of Tang et al. (2016), but considers the three classes *positive*, *negative* and *neutral* instead of just *positive* and *negative*. The experiments of Chapter 8 show that the Ternary Sentiment Embedding Model generally performs better than the Hybrid Ranking Model, except when the embeddings have been trained using the Emoticon or the VADER distant supervision methods. Our results show that the quality of the distant-supervised dataset has a great impact on the quality of the produced sentiment embeddings, and transitively the entire Twitter Sentiment Analysis system.

10.3. Future Work

Various work has been done to extend or improve the sentiment embeddings of Tang et al. (2014) in recent years. This work show promising areas to investigate for further improving upon the Ternary Sentiment Embedding Model, but were out of scope for this thesis. In this section this work is mentioned, as well as areas of improvement that were discovered during our experiments.

10.3.1. Distant Supervision

While more sophisticated classifiers provide better prediction quality than the simple Emoticon classifier, the Emoticon approach is unquestionably the fastest. The precision of the Emoticon method is very good for positive and negative, but fails for neutral. A thorough analysis of neutral tweets should be done in order to find features that are close to unique for them, which could act as the neutral counterpart to the emoticons. If such high-precision features are found, one could create a distant supervision model that could create high-quality data from a set of pre-collected tweets at a very high speed. The method would neglect tweets with none of the features, which creates a need for a large set of tweets in order to exploit the speed of the method to the fullest.

Various ensembles of classifiers could be investigated for distant supervision. The Combination methods (Combo A and Combo B) average the scores of their component methods (AFINN, VADER and TextBlob). Other methods could be used, as well as combination schemes, as discussed in Section 9.1.1 (p. 71).

10.3.2. Exhaustive Optimization

In the hyperparameter search of Section 7.3 (p. 52), the dataset annotated by the FJLC method was used. Preferably, an individual hyperparameter search should be done with each of the distant-supervised datasets, as it is not clear that the best values found with the FJLC set will generalize to the other sets.

The manual hyperparameter search done might have found local maxima. A more thorough grid search should be done in order to find the globally best hyperparameter values. The coarse search does not consider how hyperparameters affect each other. For instance, the best value for one hyperparameter might not be the best performing when changing to another hyperparameter. This would not be a problem for a full grid search.

Hyperparameters such as the embedding length and hidden layer size seem to improve the scores as they get larger. A more thorough search should investigate even larger values of these to find if there is any increase in performance to gain from it.

10.3.3. Investigate Impact of Using More Data

Due to strict filtering (Section 6.1, p. 40), a low percentage of the collected tweets were included in the distant-supervised datasets. The datasets used in this thesis comprised three million tweets, one million of each sentiment. Due to time constraints we did not get the chance to train with larger datasets and compare how the data size impacts the quality of the embeddings. We expect larger datasets to further improve the quality of the embeddings.

10.3.4. Word-Sense Disambiguation

Many words have different meanings based on the context they are in. The different meanings might have different sentiments associated with them. An example from Ren et al. (2016) illustrates the issue:

- Monday before I leave Singapore, I am going to post something that might be **offensive. (NEGATIVE)**
- #Patriots Tom Brady wins AFC **offensive** player of the week for 22nd time.... http://t.co/WIFHyQ0I - #NFL (**POSITIVE**)

In the above example, the word "offensive" has two different meanings, and different sentiments. Ren et al. proposed a model for training topic-enriched multi-prototype word embeddings (TMWE) that addresses the issue of polysemy. The new model of Ren et al. significantly improved upon the results of SemEval 2013 on the binary classification task. Extending the Ternary Sentiment Embedding Model with the ability to discrimate sentiment of polysemous words in three classes, would be expected to further improve its performance.

10.3.5. Word-Specific Sentiment

Both the Hybrid Ranking Model and the Ternary Sentiment Embedding Model base the training on the assumption that all words in a tweet have the same sentiment. This ignores the prior sentiment polarity of the word on its own. Xiong (2016) addressed this problem by exploiting both lexicon resource and distant supervised information in his proposed multi-level sentiment-enriched word embedding learning method, which outperformed state-of-the-art methods.

Xiong used the sentiment lexicon of Hu and Liu (2004), which is based on customer reviews. Further work could look at using state-of-the-art sentiment lexica for Twitter Sentiment Analysis. Word-sense aware lexica could be used in order to combine the works of Ren et al. (2016) and Xiong (2016).

Bibliography

Abadi, Martín; Agarwal, Ashish; Barham, Paul; Brevdo, Eugene; Chen, Zhifeng; Citro, Craig; Corrado, Greg S.; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Goodfellow, Ian; Harp, Andrew; Irving, Geoffrey; Isard, Michael; Jia, Yangqing; Jozefowicz, Rafal; Kaiser, Lukasz; Kudlur, Manjunath; Levenberg, Josh; Mané, Dan; Monga, Rajat; Moore, Sherry; Murray, Derek; Olah, Chris; Schuster, Mike; Shlens, Jonathon; Steiner, Benoit; Sutskever, Ilya; Talwar, Kunal; Tucker, Paul; Vanhoucke, Vincent; Vasudevan, Vijay; Viégas, Fernanda; Vinyals, Oriol; Warden, Pete; Wattenberg, Martin; Wicke, Martin; Yu, Yuan, and Zheng, Xiaoqiang (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org.

Agarwal, Apoorv; Xie, Boyi; Vovsha, Ilia; Rambow, Owen, and Passonneau, Rebecca (2011). "Sentiment Analysis of Twitter Data." In: *Proceedings of the Workshop on Language in Social Media (LSM 2011).* Portland, OR, USA: Association for Computational Linguistics, pp. 30–38.

Alain, Guillaume; Almahairi, Amjad; Angermueller, Christof; Bahdanau, Dzmitry; Ballas, Nicolas; Bastien, Frédéric; Bayer, Justin; Belikov, Anatoly; Belopolsky, Alexander; Bengio, Yoshua; Bergeron, Arnaud; Bergstra, James; Bisson, Valentin; Bleecher Snyder, Josh; Bouchard, Nicolas; Boulanger-Lewandowski, Nicolas; Bouthillier, Xavier; Brébisson, Alexandre de; Breuleux, Olivier; Carrier, Pierre-Luc; Cho, Kyunghyun; Chorowski, Jan; Christiano, Paul; Cooijmans, Tim; Côté, Marc-Alexandre; Côté, Myriam; Courville, Aaron; Dauphin, Yann N.; Delalleau, Olivier; Demouth, Julien; Desjardins, Guillaume; Dieleman, Sander; Dinh, Laurent; Ducoffe, Mélanie; Dumoulin, Vincent; Ebrahimi Kahou, Samira; Erhan, Dumitru; Fan, Ziye; Firat, Orhan; Germain, Mathieu; Glorot, Xavier; Goodfellow, Ian; Graham, Matt; Gulcehre, Caglar; Hamel, Philippe; Harlouchet, Iban; Heng, Jean-Philippe; Hidasi, Balázs; Honari, Sina; Jain, Arjun; Jean, Sébastien; Jia, Kai; Korobov, Mikhail; Kulkarni, Vivek; Lamb, Alex; Lamblin, Pascal; Larsen, Eric; Laurent, César; Lee, Sean; Lefrancois, Simon; Lemieux, Simon; Léonard, Nicholas; Lin, Zhouhan; Livezey, Jesse A.; Lorenz, Cory; Lowin, Jeremiah; Ma, Qianli; Manzagol, Pierre-Antoine; Mastropietro, Olivier; McGibbon, Robert T.; Memisevic, Roland; Merriënboer, Bart van; Michalski, Vincent; Mirza, Mehdi; Orlandi, Alberto; Pal, Christopher; Pascanu, Razvan; Pezeshki, Mohammad; Raffel, Colin; Renshaw, Daniel; Al-Rfou, Rami; Rocklin, Matthew; Romero, Adriana; Roth, Markus; Sadowski, Peter; Salvatier, John; Savard, François; Schlüter, Jan; Schulman, John; Schwartz, Gabriel; Serban, Iulian Vlad; Serdyuk, Dmitriy; Shabanian, Samira; Simon, Étienne; Spieckermann, Sigurd; Subramanyam, S. Ramana; Sygnowski, Jakub; Tanguay, Jérémie; Tulder, Gijs van; Turian, Joseph; Urban, Sebastian; Vincent, Pascal; Visin, Francesco; Vries, Harm de; Warde-Farley, David; Webb, Dustin J.; Willson, Matthew; Xu, Kelvin; Xue, Lijun; Yao, Li; Zhang, Saizheng, and Zhang, Ying (2016). "Theano: A Python framework for fast computation of mathematical expressions." In: *arXiv e-prints* abs/1605.02688.

Balikas, Georgios and Amini, Massih-Reza (2016). "TwiSE at SemEval-2016 Task 4: Twitter Sentiment Classification." In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016).* San Diego, CA, USA: Association for Computational Linguistics, pp. 85–91.

Bengio, Yoshua; Ducharme, Réjean; Vincent, Pascal, and Jauvin, Christian (2003). "A Neural Probabilistic Language Model." In: *Journal of Machine Learning Research* 3.Feb, pp. 1137–1155.

Bérard, Alexandre; Servan, Christophe; Pietquin, Olivier, and Besacier, Laurent (2016). "MultiVec: a Multilingual and Multilevel Representation Learning Toolkit for NLP." In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016).* Portorož, Slovenia: European Language Resources Association, pp. 4188–4192.

Boag, William; Potash, Peter, and Rumshisky, Anna (2015). "TwitterHawk: A Feature Bucket Based Approach to Sentiment Analysis." In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval-2015).* Denver, CO, USA: Association for Computational Linguistics, pp. 640–646.

Bonab, Hamed R. and Can, Fazli (2016). "A Theoretical Framework on the Ideal Number of Classifiers for Online Ensembles in Data Streams." In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management.* Indianapolis, IN, USA: ACM, pp. 2053–2056.

Byrkjeland, Mats and de Lichtenberg, Frederik Gørvell (2016). "Exploring Word Embeddings in Twitter Sentiment Analysis." Unpublished Specialization Project, Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU).

Collobert, Ronan and Weston, Jason (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning." In: *Proceedings of the 25th International Conference on Machine Learning.* Helsinki, Finland: ACM, pp. 160–167.

Collobert, Ronan; Weston, Jason; Bottou, Léon; Karlen, Michael; Kavukcuoglu, Koray, and Kuksa, Pavel (2011). "Natural Language Processing (Almost) from Scratch." In: *Journal of Machine Learning Research* 12.Aug, pp. 2493–2537.

Cortes, Corinna and Vapnik, Vladimir (1995). “Support-Vector Networks.” In: *Machine Learning* 20.3, pp. 273–297.

Deerwester, Scott; Dumais, Susan T.; Furnas, George W.; Landauer, and Harshman, Richard (1990). “Indexing by Latent Semantic Analysis.” In: *Journal of the American Society for Information Science* 41.6, pp. 391–407.

Deriu, Jan; Gonzenbach, Maurice; Uzdilli, Fatih; Lucchi, Aurelien; De Luca, Valeria, and Jaggi, Martin (2016). “SwissCheese at SemEval-2016 Task 4: Sentiment Classification Using an Ensemble of Convolutional Neural Networks with Distant Supervision.” In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016).* San Diego, CA, USA: Association for Computational Linguistics, pp. 1124–1128.

Fielding, Roy Thomas (2000). “Architectural Styles and the Design of Network-based Software Architectures.” PhD thesis. University of California, Irvine.

Firth, J. R. (1957). “A Synopsis of Linguistic Theory 1930-55.” In: *Studies in Linguistic Analysis* Special Volume of the Philological Society.

Fredriksen, Valerij and Jahren, Brage Ekroll (2016). “Twitter Sentiment Analysis: Exploring Automatic Creation of Sentiment Lexica.” Master’s thesis. Norwegian University of Science and Technology (NTNU).

Giorgis, Stavros; Rousas, Apostolos; Pavlopoulos, John; Malakasiotis, Prodromos, and Androutsopoulos, Ion (2016). “aueb.twitter.sentiment at SemEval-2016 Task 4: A Weighted Ensemble of SVMs for Twitter Sentiment Analysis.” In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016).* San Diego, CA, USA: Association for Computational Linguistics, pp. 96–99.

Go, Alec; Bhayani, Richa, and Huang, Lei (2009). “Twitter Sentiment Classification using Distant Supervision.” In: *CS224N Project Report, Stanford*, pp. 1–12.

Gomez-Adorno, Helena and Sidorov, Grigori (2016). “CICBUAPnlp at SemEval-2016 Task 4-A: Discovering Twitter Polarity using Enhanced Embeddings.” In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016).* San Diego, CA, USA: Association for Computational Linguistics, pp. 145–148.

Hinton, Geoffrey E. (1986). “Learning Distributed Representations of Concepts.” In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* Vol. 1. Amherst, MA, USA: Lawrence Erlbaum Associates, pp. 1–12.

Hu, Minqing and Liu, Bing (2004). “Mining and Summarizing Customer Reviews.” In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Seattle, WA, USA: ACM, pp. 168–177.

Hutto, C.J. and Gilbert, Eric (2014). “VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text.” In: *Proceedings of the Eighth International Conference on Weblogs and Social Media (ICWSM-14)*. Ann Arbor, MI, USA: The AAAI Press, pp. 216–225.

Jiang, Long; Yu, Mo; Zhou, Ming; Liu, Xiaohua, and Zhao, Tiejun (2011). “Target-dependent Twitter Sentiment Classification.” In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*. Portland, OR, USA: Association for Computational Linguistics, pp. 151–160.

Levy, Omer; Goldberg, Yoav, and Dagan, Ido (2015). “Improving Distributional Similarity with Lessons Learned from Word Embeddings.” In: *Transactions of the Association for Computational Linguistics* 3, pp. 211–225.

Lui, Marco and Baldwin, Timothy (2012). “langid.py: An Off-the-shelf Language Identification Tool.” In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*. Jeju, Republic of Korea: Association for Computational Linguistics, pp. 25–30.

Mikolov, Tomas; Chen, Kai; Corrado, Greg, and Dean, Jeffrey (2013a). “Efficient Estimation of Word Representations in Vector Space.” In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S., and Dean, Jeff (2013b). “Distributed Representations of Words and Phrases and their Compositionality.” In: *Advances in Neural Information Processing Systems*. Vol. 26. Lake Tahoe, NV, USA, pp. 3111–3119.

Mikolov, Tomas; Yih, Wen-tau, and Zweig, Geoffrey (2013c). “Linguistic Regularities in Continuous Space Word Representations.” In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Vol. 13. Atlanta, GA, USA, pp. 746–751.

Nakov, Preslav; Ritter, Alan; Rosenthal, Sara; Sebastiani, Fabrizio, and Stoyanov, Veselin (2016). “SemEval-2016 Task 4: Sentiment Analysis in Twitter.” In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, CA, USA: Association for Computational Linguistics, pp. 1–18.

Nakov, Preslav; Rosenthal, Sara; Kozareva, Zornitsa; Stoyanov, Veselin; Ritter, Alan, and Wilson, Theresa (2013). “SemEval-2013 Task 2: Sentiment Analysis in Twitter.” In: *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013)*. Atlanta, GA, USA: Association for Computational Linguistics, pp. 312–320.

Nielsen, Finn Årup (2011). “A New ANEW: Evaluation of a Word List for Sentiment Analysis in Microblogs.” In: *Proceedings of the ESWC2011 Workshop on ’Making Sense of Microposts’: Big things come in small packages*. Vol. 718. Heraklion, Crete, pp. 93–98.

Nigam, Kamal; Lafferty, John, and McCallum, Andrew (1999). "Using Maximum Entropy for Text Classification." In: *The International Joint Conference on Artificial Intelligence-99 Workshop on Machine Learning for Information Filtering.* Vol. 1. Stockholm, Sweden, pp. 61–67.

Owoputi, Olutobi; O'Connor, Brendan; Dyer, Chris; Gimpel, Kevin; Schneider, Nathan, and Smith, Noah A. (2013). "Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters." In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Vol. 13. Atlanta, GA, USA, pp. 138–147.

Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand; Grisel, Olivier; Blondel, Mathieu; Prettenhofer, Peter; Weiss, Ron; Dubourg, Vincent, et al. (2011). "Scikit-learn: Machine learning in Python." In: *Journal of Machine Learning Research* 12.Oct, pp. 2825–2830.

Pennington, Jeffrey; Socher, Richard, and Manning, Christopher D. (2014). "GloVe: Global Vectors for Word Representation." In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543.

Ren, Yafeng; Wang, Ruimin, and Ji, Donghong (2016). "A Topic-enhanced Word Embedding for Twitter Sentiment Classification." In: *Information Sciences* 369, pp. 188–198.

Rosenthal, Sara; Farra, Noura, and Nakov, Preslav (2017). "SemEval-2017 Task 4: Sentiment Analysis in Twitter." In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017).* Vancouver, Canada: Association for Computational Linguistics, pp. 493–509.

Rouvier, Mickael and Favre, Benoit (2016). "SENSEI-LIF at SemEval-2016 Task 4: Polarity embedding fusion for robust sentiment analysis." In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016).* San Diego, CA, USA: Association for Computational Linguistics, pp. 202–208.

Ræder, Johan Georg Cyrus Mazaher (2016). "Automatic Sarcasm Detection in Twitter Messages." Master's thesis. Norwegian University of Science and Technology (NTNU).

Steinskog, Asbjørn Ottesen and Therkelsen, Jonas Foyn (2016). "Characterizing Twitter Data using Sentiment Analysis and Topic Modeling." Master's thesis. Norwegian University of Science and Technology (NTNU).

Tang, Duyu; Wei, Furu; Qin, Bing; Yang, Nan; Liu, Ting, and Zhou, Ming (2016). "Sentiment Embeddings with Applications to Sentiment Analysis." In: *IEEE Transactions on Knowledge and Data Engineering* 28.2, pp. 496–509.

Tang, Duyu; Wei, Furu; Yang, Nan; Zhou, Ming; Liu, Ting, and Qin, Bing (2014). "Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, MD, USA: Association for Computational Linguistics, pp. 1555–1565.

Xiong, Shufeng (2016). "Improving Twitter Sentiment Classification via Multi-Level Sentiment-Enriched Word Embeddings." In: *CoRR* abs/1611.00126.

Zhang, Edmond and Mayo, Michael (2010). "Improving Bag-of-Words Model with Spatial Information." In: *Proceedings of the 25th Conference of Image and Vision Computing*. Queenstown, New Zealand: ACM, pp. 1–8.

Zhou, Yunxiao; Zhang, Zhihua, and Lan, Man (2016). "ECNU at SemEval-2016 Task 4: An Empirical Investigation of Traditional NLP Features and Word Embedding Features for Sentence-level and Topic-level Sentiment Analysis in Twitter." In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, CA, USA: Association for Computational Linguistics, pp. 256–261.

A. Hyperparameter Search

This appendix contains detailed results from the hyperparameter search presented in Section 7.3 (p. 52).

The following plots and tables show test results from all the 20 epochs that the system was trained for. The figures show that the scores vary over epochs. The overall best results are marked in bold type.

A.1. Alpha

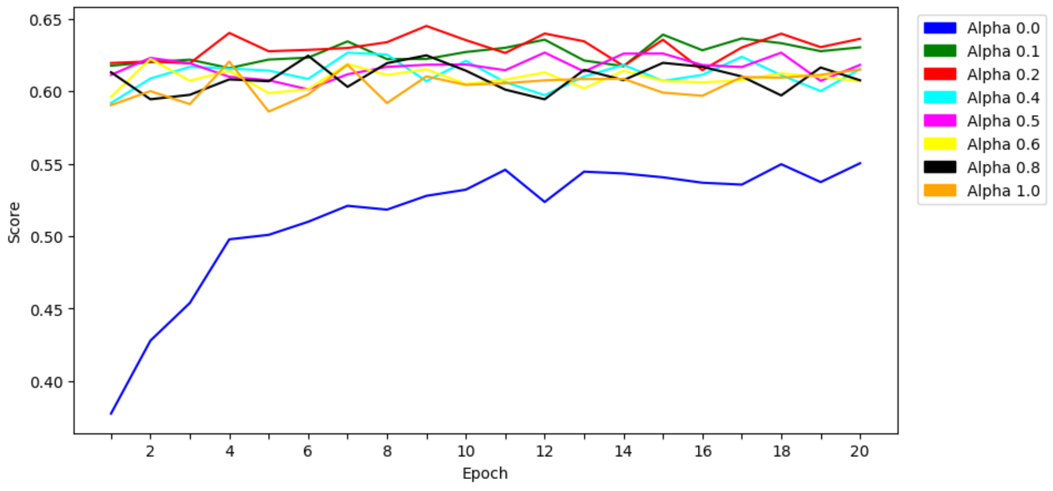


Figure A.1.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different values for α.

Epoch	α 0.0	0.1	0.2	0.4	0.5	0.6	0.8	1.0
1	0.3775	0.6177	0.6195	0.5919	0.6113	0.5961	0.6132	0.5904
2	0.4278	0.6201	0.6206	0.6088	0.6231	0.6223	0.5945	0.6001
3	0.4538	0.6219	0.6193	0.6169	0.6193	0.6073	0.5976	0.5911
4	0.4978	0.616	0.6403	0.6157	0.6101	0.6141	0.6082	0.6205
5	0.5009	0.6219	0.6277	0.6143	0.6076	0.5988	0.6070	0.5860
6	0.5099	0.6232	0.6286	0.6085	0.6014	0.6014	0.6246	0.5978
7	0.5209	0.6345	0.6299	0.6267	0.6117	0.6189	0.6031	0.6184
8	0.5183	0.6222	0.6339	0.6254	0.6169	0.6113	0.6194	0.5919
9	0.5278	0.6223	**0.6451**	0.6070	0.6183	0.6152	0.6248	0.6103
10	0.5321	0.6271	0.6353	0.621	0.6185	0.6050	0.6143	0.6044
11	0.5458	0.6300	0.6265	0.6063	0.6146	0.608	0.6012	0.6056
12	0.5236	0.6356	0.6399	0.5973	0.6267	0.6131	0.5945	0.6076
13	0.5445	0.6213	0.6345	0.6102	0.6137	0.602	0.6148	0.6085
14	0.5433	0.6173	0.6175	0.6183	0.6260	0.6142	0.6077	0.6086
15	0.5406	0.6391	0.6356	0.6072	0.6261	0.6072	0.6196	0.5991
16	0.5369	0.6283	0.6146	0.6113	0.6181	0.6062	0.6170	0.5969
17	0.5356	0.6365	0.6303	0.6239	0.6168	0.6078	0.6102	0.6099
18	0.5497	0.6332	0.6398	0.6111	0.6267	0.6124	0.5972	0.6094
19	0.5373	0.6277	0.6305	0.6002	0.6072	0.6100	0.6164	0.6114
20	0.5503	0.6304	0.6363	0.6153	0.6182	0.6070	0.6076	0.6150

Table A.1.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different values for α.

A.2. Context Window Size

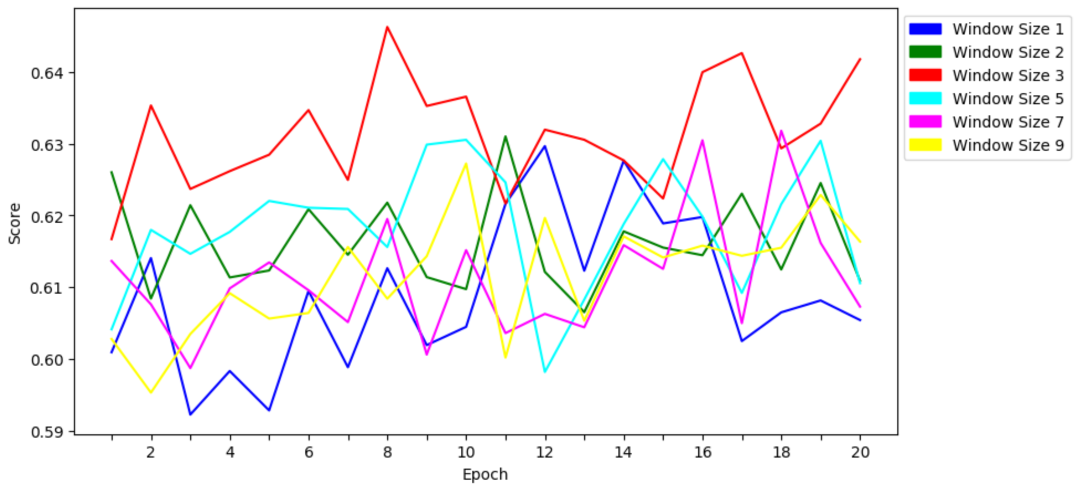


Figure A.2.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different context window sizes.

Epoch	1	2	3	5	7	9
1	0.6009	0.6260	0.6167	0.6041	0.6137	0.6028
2	0.6141	0.6084	0.6354	0.6180	0.6076	0.5953
3	0.5922	0.6214	0.6237	0.6147	0.5987	0.6035
4	0.5983	0.6114	0.6262	0.6177	0.6098	0.6092
5	0.5928	0.6123	0.6285	0.622	0.6135	0.6056
6	0.6095	0.6209	0.6347	0.6211	0.6096	0.6064
7	0.5988	0.6145	0.6250	0.6209	0.6051	0.6156
8	0.6126	0.6218	**0.6463**	0.6156	0.6195	0.6084
9	0.6019	0.6114	0.6353	0.6299	0.6006	0.6143
10	0.6045	0.6097	0.6366	0.6306	0.6152	0.6273
11	0.6216	0.6310	0.6217	0.6246	0.6036	0.6002
12	0.6297	0.6121	0.6320	0.5982	0.6063	0.6197
13	0.6123	0.6065	0.6306	0.6083	0.6044	0.6053
14	0.6277	0.6178	0.6277	0.6188	0.6159	0.6171
15	0.6189	0.6155	0.6224	0.6279	0.6126	0.6141
16	0.6198	0.6145	0.6400	0.6198	0.6305	0.6158
17	0.6025	0.6231	0.6427	0.6092	0.6050	0.6144
18	0.6065	0.6125	0.6294	0.6216	0.6318	0.6155
19	0.6082	0.6245	0.6328	0.6304	0.6162	0.6229
20	0.6054	0.6109	0.6418	0.6106	0.6073	0.6164

Context Window Size

Table A.2.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different context window sizes.

A.3. Embedding Length

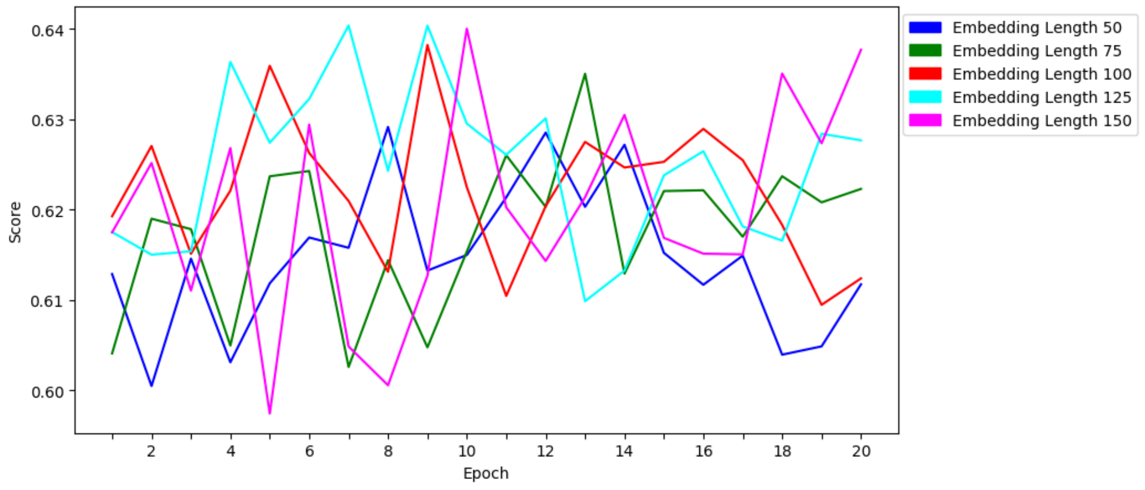


Figure A.3.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different embedding lengths.

Epoch	50	75	100	125	150
1	0.6129	0.6041	0.6193	0.6175	0.6175
2	0.6005	0.6190	0.6270	0.6150	0.6252
3	0.6146	0.6178	0.6151	0.6154	0.6111
4	0.6031	0.6050	0.6221	0.6363	0.6268
5	0.6119	0.6237	0.6359	0.6274	0.5975
6	0.6169	0.6243	0.6263	0.6322	0.6294
7	0.6158	0.6026	0.6210	**0.6404**	0.6049
8	0.6292	0.6144	0.6131	0.6243	0.6006
9	0.6133	0.6048	0.6382	**0.6404**	0.6127
10	0.6150	0.6154	0.6225	0.6295	0.6400
11	0.6214	0.6260	0.6105	0.6261	0.6203
12	0.6285	0.6203	0.6204	0.6301	0.6143
13	0.6203	0.6350	0.6275	0.6099	0.6214
14	0.6272	0.6129	0.6247	0.6133	0.6305
15	0.6152	0.6221	0.6253	0.6238	0.6169
16	0.6117	0.6222	0.6289	0.6265	0.6151
17	0.6149	0.6170	0.6255	0.6181	0.6151
18	0.6040	0.6237	0.6183	0.6166	0.6351
19	0.6049	0.6208	0.6095	0.6284	0.6273
20	0.6117	0.6223	0.6124	0.6277	0.6377

Table A.3.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different embedding lengths.

A.4. Hidden Layer Size

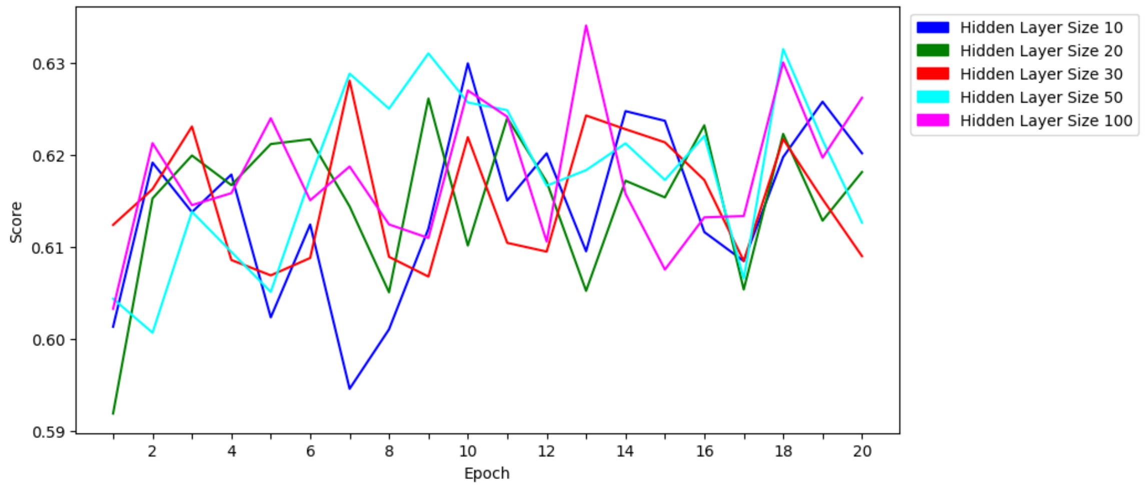


Figure A.4.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different hidden layer sizes.

	Hidden Layer Size				
Epoch	10	20	30	50	100
1	0.6013	0.5919	0.6124	0.6043	0.6033
2	0.6191	0.6153	0.6163	0.6007	0.6213
3	0.6138	0.6199	0.6231	0.6138	0.6145
4	0.6178	0.6167	0.6086	0.6094	0.6158
5	0.6023	0.6212	0.6069	0.6051	0.6240
6	0.6124	0.6217	0.6088	0.6175	0.6150
7	0.5946	0.6144	0.6280	0.6288	0.6187
8	0.6010	0.6050	0.6089	0.6250	0.6124
9	0.6120	0.6261	0.6068	0.6310	0.6110
10	0.6299	0.6101	0.6219	0.6257	0.6270
11	0.6150	0.6240	0.6104	0.6248	0.6241
12	0.6201	0.6171	0.6095	0.6166	0.6106
13	0.6095	0.6052	0.6243	0.6183	**0.6340**
14	0.6247	0.6172	0.6228	0.6212	0.6158
15	0.6237	0.6154	0.6213	0.6173	0.6075
16	0.6116	0.6232	0.6172	0.6220	0.6132
17	0.6084	0.6053	0.6084	0.6065	0.6133
18	0.6197	0.6223	0.6217	0.6315	0.6300
19	0.6258	0.6128	0.6152	0.6216	0.6197
20	0.6202	0.6181	0.6090	0.6126	0.6262

Table A.4.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different hidden layer sizes.

A.5. Learning Rate

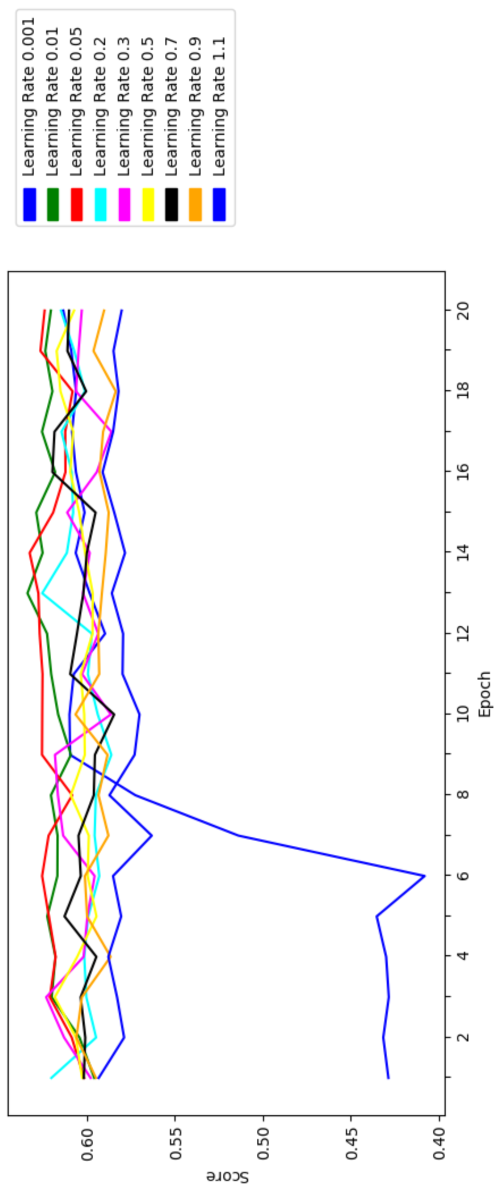


Figure A.5.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different learning rates.

	Learning Rate								
Epoch	0.001	0.01	0.05	0.2	0.3	0.5	0.7	0.9	1.1
1	0.4285	0.5958	0.6016	0.6202	0.5979	0.6026	0.6017	0.5948	0.5933
2	0.4314	0.6042	0.6086	0.5948	0.6129	0.6063	0.6009	0.6060	0.5788
3	0.4283	0.6200	0.6211	0.6003	0.6232	0.6182	0.6035	0.6028	0.5828
4	0.4298	0.6179	0.6178	0.6015	0.6019	0.6055	0.5946	0.5865	0.5878
5	0.4352	0.6225	0.6216	0.5993	0.5997	0.5945	0.6127	0.5999	0.5805
6	0.4079	0.6168	0.6255	0.5929	0.5956	0.5995	0.6035	0.6011	0.5851
7	0.5138	0.6167	0.6218	0.5956	0.6135	0.5989	0.6048	0.5877	0.5631
8	0.5724	0.6205	0.6083	0.5949	0.6165	0.6091	0.5961	0.5934	0.5872
9	0.6095	0.6091	0.6255	0.5861	0.6181	0.6012	0.5954	0.5882	0.5729
10	0.6099	0.6163	0.6254	0.5938	0.5864	0.6015	0.5845	0.6065	0.5701
11	0.6074	0.6203	0.6252	0.5994	0.6027	0.6035	0.6095	0.5930	0.5797
12	0.5896	0.6226	0.6268	0.5971	0.5935	0.5968	0.6057	0.5935	0.5794
13	0.5984	**0.6338**	0.6276	0.6254	0.6022	0.5967	0.6020	0.5915	0.5858
14	0.6064	0.6252	0.6325	0.6114	0.5984	0.6010	0.6000	0.5893	0.5783
15	0.6013	0.6288	0.6192	0.6078	0.6112	0.6044	0.5950	0.5876	0.5844
16	0.6062	0.6180	0.6122	0.6086	0.5944	0.6094	0.6196	0.5928	0.5910
17	0.6085	0.6255	0.6123	0.6145	0.5861	0.6078	0.6184	0.5908	0.5851
18	0.6062	0.6196	0.6081	0.6011	0.6062	0.6150	0.6004	0.5836	0.5820
19	0.6088	0.6236	0.6264	0.6072	0.6047	0.6172	0.6111	0.5962	0.5848
20	0.6134	0.6205	0.6240	0.6147	0.6029	0.6073	0.6102	0.5902	0.5802

Table A.5.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different learning rates.

A.6. Margin

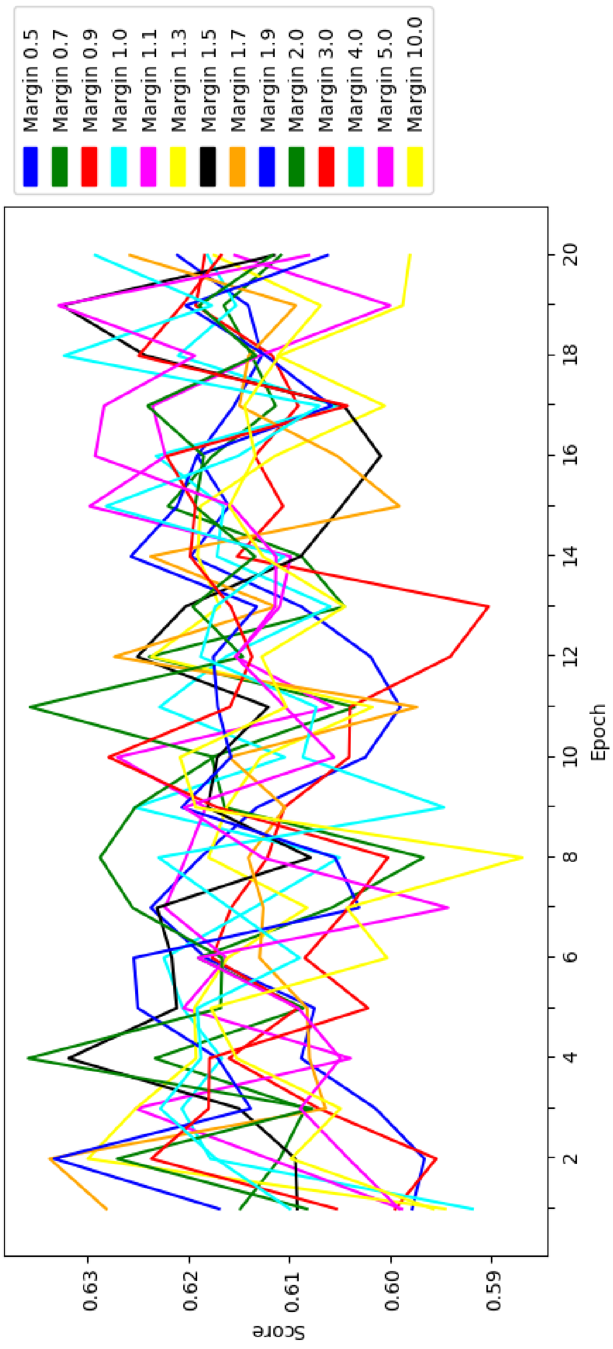


Figure A.6.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different margins.

	Margin													
Epoch	0.5	0.7	0.9	1.0	1.1	1.3	1.5	1.7	1.9	2.0	3.0	4.0	5.0	10.0
1	0.5978	0.6149	0.5995	0.5919	0.5989	0.5946	0.6092	0.6281	0.6170	0.6083	0.6053	0.6100	0.5991	0.5957
2	0.5966	0.6110	0.5954	0.6178	0.6120	0.6300	0.6094	0.6338	0.6333	0.6270	0.6237	0.6173	0.6041	0.6098
3	0.6015	0.6084	0.6071	0.6207	0.6252	0.6252	0.6150	0.6064	0.6138	0.6077	0.6180	0.6228	0.6089	0.6049
4	0.6088	0.6233	0.6160	0.6167	0.6039	0.6192	0.6319	0.6080	0.6171	**0.6359**	0.6179	0.6188	0.6049	0.6152
5	0.6075	0.6086	0.6092	0.6206	0.6205	0.6194	0.6212	0.6082	0.6250	0.6168	0.6022	0.6191	0.6091	0.6178
6	0.6183	0.6190	0.6176	0.6224	0.6162	0.6161	0.6216	0.6129	0.6254	0.6167	0.6085	0.6090	0.6190	0.6003
7	0.6237	0.6058	0.6158	0.6136	0.6223	0.6082	0.6231	0.6126	0.6031	0.6255	0.6041	0.6159	0.5943	0.6043
8	0.6188	0.5967	0.6122	0.6050	0.6202	0.6180	0.6079	0.6141	0.6055	0.6288	0.6002	0.623	0.6127	0.5869
9	0.6132	0.6164	0.6105	0.6253	0.6183	0.6162	0.6181	0.6104	0.6207	0.6253	0.6172	0.5947	0.6204	0.6193
10	0.6025	0.6177	0.6041	0.6105	0.6270	0.6128	0.6171	0.6160	0.6158	0.6175	0.6279	0.6087	0.6056	0.6208
11	0.5990	0.6035	0.6040	0.6229	0.6057	0.6018	0.6121	0.5973	0.6171	0.6357	0.6159	0.6073	0.6104	0.6103
12	0.6019	0.6239	0.5941	0.6163	0.6154	0.6237	0.6251	0.6273	0.6175	0.6146	0.6137	0.6188	0.6153	0.6128
13	0.6088	0.6045	0.5903	0.6059	0.6109	0.6169	0.6203	0.6114	0.6132	0.6196	0.6158	0.6173	0.6114	0.6045
14	0.6198	0.6088	0.6152	0.6172	0.6099	0.6191	0.6089	0.6238	0.6257	0.6134	0.6198	0.6105	0.6113	0.6126
15	0.616	0.6220	0.6106	0.6165	0.6298	0.6188	0.6048	0.5991	0.6211	0.6194	0.6193	0.6282	0.6156	0.6160
16	0.6191	0.6176	0.6134	0.6232	0.6223	0.6114	0.6010	0.6052	0.6192	0.6185	0.6222	0.6151	0.6292	0.6134
17	0.6155	0.6114	0.6091	0.6053	0.6236	0.6006	0.6046	0.6149	0.6058	0.6241	0.6043	0.6070	0.6284	0.6144
18	0.6127	0.6137	0.6117	0.6210	0.6130	0.6112	0.6242	0.6140	0.6123	0.6133	0.6249	0.6323	0.6193	0.6110
19	0.6141	0.6165	0.6193	0.6153	0.6000	0.6069	0.6325	0.6094	0.6203	0.6190	0.6206	0.6177	0.6329	0.5988
20	0.6211	0.6108	0.6184	0.6181	0.6154	0.6175	0.6115	0.6258	0.6062	0.6117	0.6168	0.6292	0.6081	0.5980

Table A.6.: Macro F_1 scores for the Ternary Sentiment Embedding Model with different margins.

B. SemEval 2017 Results

Rank	System	AvgRec	F_1^{PN}	Acc
1	DataStories	0.681_1	0.677_2	0.651_5
	BB twtr	0.681_1	0.685_1	0.658_3
3	LIA	0.676_3	0.674_3	0.661_2
4	Senti17	0.674_4	0.665_4	0.652_4
5	NNEMBs	0.669_5	0.658_5	0.664_1
6	Tweester	0.659_6	0.648_6	0.648_6
7	INGEOTEC	0.649_7	0.645_7	0.633_{11}
8	SiTAKA	0.645_8	0.628_9	0.643_9
9	TSA-INF	0.643_9	0.620_{11}	0.616_{17}
10	UCSC-NLP	0.642_{10}	0.624_{10}	0.565_{30}

Table B.1.: The *AvgRec*, F_1^{PN} and *Accuracy* scores for the top ten systems of SemEval 2017 Task 4A