MACHINE LEARNING FOR SENTIMENT ANALYSIS OF CHILDREN'S DIARIES

MARJOLEIN DE VRIES (4302486)

Master Computing Science
Algorithmic Data Analysis Group
Department of Information and Computing Sciences
Faculty of Science
Utrecht University

May 19, 2017 – Final Version

SUPERVISORS:
dr. Maya Sappelli (TNO)
dr. Ad Feelders (UU)
prof. dr. Arno Siebes (UU, second reader)

ABSTRACT

In this thesis, the automatic detection of sentiments expressed by children using machine learning is investigated. The automatic sentiment detection is used within a robotic companion that helps children with their daily struggles of living with Type 1 diabetes. The problem statement that guides this research is: *To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of automatic text analysis?* Results show that machine learning models yield a significantly higher performance than a symbolic sentiment scoring algorithm. Machine learning models have shown to be better at capturing context and at capturing complex negations. The usage of an ordinal or time-correlated adaptation of a machine learning model is evaluated as well, but results show that such a model does not have an advantage over a regular machine learning model. Additionally, a new algorithm for semantic normalization on top of standard morphological normalization is introduced in this thesis. Results show that using this newly created step consistently improves the performance in the current study. The new algorithm for semantic normalization is especially useful in this thesis, because the dataset is sparse and contains highly infrequent words.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Type 1 diabetes mellitus (T1DM) is one of the most common diseases among children and youngsters in the United States and Europe. In the United States, approximately 215,000 children and youngsters under the age of 20 have to endure this type of diabetes, and every year another 13,000 young people are diagnosed with T1DM. In Europe, the statistics show an average increase among all children and youngsters of 3.9% per year, and the increase is largest among children under four years old. [22]

Having T1DM requires a vast number of lifestyle changes, such as blood glucose monitoring, insulin administration, and diet modification. Children often struggle to manage T1DM and all its consequent lifestyle changes by themselves. Self-managing blood glucose checking and insulin administrating can be difficult, especially at a young age [4]. Children indicate for example the importance of having someone nearby whom can help when blood glucose is low [22].

Moreover, having T1DM also comes with emotional challenges [4]. The most important concern for children with T1DM is to still have a 'normal life' which does not differ from their peers [42]. Children with T1DM can feel left out as they have to spend time for example during lunch on testing glucose and administering insulin, which reduces the amount of time to play with their friends. Because of the additional activities which have to be incorporated into their lives, children with T1DM can sometimes feel alone. [22]

A robotic companion can be a solution to some of these challenges. The Personal Assistant for a healthy Lifestyle (PAL [1]) project develops such a social robot (using the Nao [2]) and an accompanying mobile application for children with T1DM. One possible use of the robot indicated by both children and their parents is to be a sensitive listener. A robot can for example function as a friend when children feel emotional and can comfort children when they are sad or anxious. In this way, a robot can help improve self-esteem and make the child feel less different from their peers. Another important use of the robot is to help with the self-management of all the daily tasks, and to increase children's knowledge on what to do in certain situations. [4]

Studies evaluating the perception of this robot show positive results [33] [34] [39]. Children with T1DM demonstrate bonding with the robot [39]. Moreover, children with T1DM indicate that they perceive

1 http://www.pal4u.eu

2 Aldebaran Robotics (http://www.aldebaran.com)

the robot as a peer [33] and as a friend [34]. Children feel at ease when interacting with the robot, and appreciate the shown interest by the robot. Children indicate an interest in further interactions with the robot [34]. Therefore, the robot can be a solution to some of the challenges faced by children with T1DM.

One way of helping children with their emotional challenges is to ask the child to keep a diary at the end of the day [49]. The robot can use such a diary entry to detect feelings of sadness, fear, anger, and happiness in the child, and act accordingly. The robot could for example comfort the child when it is feeling sad, or engage in the enjoyment of the child when it is feeling happy. Since a robot it is not straightforward for a robot to understand such sentiments from text, an algorithm is needed which analyzes the sentiment of the diary entries. Currently, pilots for evaluation of this robot are running in both the Netherlands and Italy, where some textual data from children is collected.

The current algorithm [49] for this task in the PAL project is based on sentiment scoring. The algorithm parses the text entry, translates the entry into English, and assigns a sentiment value obtained from the *SentiWordNet* dictionary [3][19] to the entry. The final sentiment is then obtained by evaluating whether the sentiment value is below or above certain manually set classification thresholds in order to label the entry with either "positive", "neutral", or "negative".

A data-driven algorithm, which uses machine learning, might find different patterns in the data entries than a sentiment scoring approach. It is important to consider multiple text representations and machine learning models, in order to research which representation together with which model and which parameters works best. This study will first be performed on the Dutch data entries, but the approach can be duplicated for the Italian data entries.

Algorithms for sentiment analysis have been widely studied [37]. In our study, we do not focus on texts from adults but rather on data from children in the age of 8 to 14 [49]. Most algorithms however are based on reviews, discussions, blogs or other social network data which are almost always written by adults. Hence, almost all algorithms and available datasets which can be used for training sentiment classification algorithms are focused on data from adults. Since this research focuses on data from children, this research will broaden the current knowledge on sentiment analysis for children's text entries.

Within the PAL project, only a small dataset has been collected so far. Because the dataset is relatively small, it will probably contain words that only occur once in the dataset. Training on these words is not useful, as weights learned for these words cannot be applied to other words. Therefore, a new algorithm for semantic normaliza-

tion is designed and evaluated, which is able to map these highly infrequent words to more frequent words.

The data also has some interesting properties. In general, machine learning models assume that the data entries are independent of each other. In this case, however, different diary entries of the same child possibly relate to each other. In other words, the sentiment of a specific child today might be dependent on his sentiment yesterday. Therefore, using a model which takes into account time-correlation of entries might be interesting to study. Moreover, the target variable (sentiment) can be interpreted to be at an ordinal level rather than a nominal level, as sentiment is defined as either positive, neutral, or negative. Therefore, using an ordinal model which takes this ordering into account is studied as well.

Altogether, the problem statement that guides the research of this thesis is the following:

To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of automatic text analysis?

To provide a specific answer, the problem statement breaks down into the following research questions:

1. To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of a sentiment scoring algorithm?
2. To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of machine learning models?
3. Does using the new algorithm for semantic normalization improve the performance?
4. Does using an ordinal model or a time-correlated model improve the performance?

The outline of the thesis is as follows. Chapter 2 provides a literature review describing studies that are relevant for this research. Chapter 3 describes the method by outlining different text representations and machine learning models. Chapter 4 specifies the data collection, description, and pre-processing. Chapter 5 studies the performance of the sentiment scoring model, in order to answer the first research question. Chapter 6 evaluates various text representation methods in the field of natural language processing and sentiment analysis. Chapter 7 aims at learning different machine learning models from these text representations, in order to answer the second research question. Chapter 8 evaluates the usage of a new algorithm for semantic normalization, in order to answer the third research question. In Chapter 9, the effect of using an ordinal adaptation of Logistic Regression and a time-correlated approach are evaluated, in order

to answer the fourth research question. Chapter 10 studies the performance across different datasets and evaluates the errors made by the machine learning models models. Chapter 11 comprises the discussion and conclusion, including an answer to the problem statement and recommendations for future research.

2 LITERATURE REVIEW

Algorithms for sentiment analysis have been widely studied. In this literature overview, a selection of research on sentiment analysis is discussed, starting with sentiment analysis in general. Moreover, classifying emotion, which is close to sentiment, from text is discussed. Finally, work done on automatic text analysis especially for Dutch texts is evaluated.

2.1 SENTIMENT ANALYSIS

Most of the sentiment analysis research is focused on data retrieved from the World Wide Web, and this data is often obtained from social media websites. Sentiment analysis is mostly directed at reviews, blogs, forum texts, and tweets [8][37][54]. In this context, sentiment analysis is mostly used for companies to get a grip on how customers are talking about their company on the internet [54]. Sentiment analysis in this case is focused on how people express their opinion regarding a certain consumption product [8] or regarding the customer service [37].

Sentiment analysis can be approached from both a *symbolic* and a *machine learning* setting. In a *symbolic* setting, sentiment analysis algorithms consist of rules which are manually created [8]. The current algorithm for classifying the children's diaries [49] is an example of such a *symbolic* setting, as it uses *SentiWordNet* [3][19] sentiment values together with manually set thresholds. Another *symbolic* method is to use semantic similarity, which can be calculated by using *WordNet* [20], a lexical database where words and relations among words are indicated by using a network structure. The path from a word in the document to semantic words such as 'good' and 'bad' in *WordNet* can be used as an indicator of sentiment [9]. Most symbolic approaches used are pre-trained with one dataset, and can afterwards be applied to other datasets. The critical point of *symbolic* approaches such as sentiment scoring and using a *WordNet* is that the dataset on which the scoring or network algorithm is trained should be comparable enough to the dataset the algorithm is used on.

Machine learning approaches are generally better than *symbolic* approaches at capturing context and measuring the (un)certainty of a sentiment classification [8]. The most common approach to sentiment classification with machine learning is to use lowercase features and split documents and sentences into n-grams, i.e. subsets of size n (either n words or n characters). n-Grams are shown to be effective for

word sense disambiguation [56] and capturing negation [53], both important when performing sentiment analysis. These *n*-grams are then used as features in machine learning models, such as Logistic Regression or Naive Bayes.

2.2 EMOTION ANALYSIS

Multiple researchers have focused on automatically classifying a person's emotion from his writings, and have worked with machine learning techniques for this purpose. Chaffar and Inkpen [10] have researched the usage of supervised learning to recognize the six basic emotions [18] in text. Their dataset consisted of a mixture of different text domains, including sentences from blog posts, sentences from fairytales, and headlines from newspapers and the Google News search engine. Stopwords were removed from the data and all data was stemmed, i.e. a word such as 'fishing' was replaced by its stem 'fish'. The data was transformed into three types of features: Bag-Of-Words (BOW) features, n-grams, and lexical emotion features. A Bag-Of-Words representation consists of a feature vector with Boolean or frequency attributes for each word that occurs in the sentence. An n-gram representation consists of one feature for every subset of n words of the sentence instead of one feature per word. A Bag-Of-Words representation is thus the same as an unigram (1-gram) representation. Lexical emotion features were extracted from a lexical resource called WordNetAffect [66], which contains sets of emotional words based on *WordNet* [20]. A Decision Tree model, a Naive Bayes model, and a Support Vector Machine (SVM) model were evaluated. Results show that the Support Vector Machine performs best on all types of datasets. Moreover, results show that an n-gram representation (with $n \geq 2$) performs only better than a Bag-Of-Words representation on some datasets. Finally, the usage of lexical emotion features did not improve the accuracy rates. Hence, a simple uni- or n-gram machine learning approach performed better than using complex lexical features.

Leshed and Kaya [35] also studied machine learning approaches for classifying emotions in blog posts. The emotion expressed in a blog post was annotated by the bloggers themselves. The authors however indicate that this is a limitation, as the labels are not objectively assigned. Stopwords were removed from the blog entries. Blog entries were transformed into a Bag-Of-Words representation with a standard tf-idf (term frequency - inverse document frequency) scheme instead of a Boolean approach used by Chaffar and Inkpen [10]. Only the 5,000 most frequent words were used. A Support Vector Machine (SVM) model was again used to learn the patterns. The authors note that using a Bag-of-Words (or a uni-gram) representation which does not take negations such as 'not sad' into account was a limitation of

their research. Hence, studying whether to use a Bag-Of-Words or an n-gram representation is important in sentiment or emotion analysis research.

Alm, Roth, and Sproat [2] researched machine learning techniques for emotion analysis of sentences from children stories. Seven emotions were grouped into two categories: positive valence $PE = \{$happy, positively surprised$\}$ and negative valence $NE = \{$anger, disgusted, fearful, sad, negatively surprised $\}$. Their model aimed at classifying a sentence as either positive, negative, or neutral. Emotions expressed in a sentence were labelled by annotators who worked in pairs. Inter-annotator agreement however was reached in approximately 50% of the cases, due to the subjective nature of the annotation task and the large number of different classes (seven classes). Hence, labelling text entries is not a straightforward task. Features used included the positive and negative word counts, calculated by using a dictionary of positive and negative words. Additionally, the usage of *sequencing*, i.e. taking into account the emotion of adjacent sentences, was studied as well. Results show that *sequencing* was beneficial in some cases, showing that taking into account emotions from related sentences can possibly improve classification results.

Finally, a totally different approach is presented by Danisman and Alpkocak [14], who used a Vector Space Model to classify emotions in text. The aim of their research was to classify sentences from the International Survey on Emotion Antecedents and Reactiosn (ISEAR) dataset as either anger, disgust, fear, sad, or joy. In their vector space model, each sentence is represented as a vector similar to the Bag-of-Words approach with a tf-idf scheme. Each emotion class is represented by a set of sentences which are labelled with that emotion. A model vector for an emotion is calculated by taking the mean of all the vectors in the corresponding emotion class. A new sentence which needs to be classified is then transformed into a tf-idf vector as well, and the cosine similarity between this new vector and all emotion model vectors is calculated. The emotion corresponding to the most similar emotion model vector is then selected as the predicted emotion class. Results show that this Vector Space Model approach has in some cases similar performance as machine learning models such as Naive Bayes and SVM. Moreover, the effect of stemming and stopword removal is studied, and results show that sometimes these techniques decrease the emotional meaning of a sentence. Hence, using a Vector Space Model with similarity measures might provide interesting results. Moreover, the usage of stemming and stopword removal should be studied, because research results do not agree on the performance of these techniques.

2.3 DUTCH TEXT ANALYSIS

Some research has specifically focused on automatic text analysis of Dutch texts, which is useful to discuss as Dutch texts are used in this study as well. Hollink, Kamps, Monz, and de Rijke [27] studied the performance of different pre-processing techniques used for document retrieval on different European languages, including Dutch. Techniques were evaluated on datasets provided by the Cross-Language Evaluation Forum (CLEF), and were compared by looking at the mean average precision scores of all documents of a certain language. The baseline consisted of a simple Bag-Of-Words representation of the texts in lower-case without any pre-processing. Replacing diacritical characters, i.e. characters with an accent placed on, with their corresponding unmarked character improved the precision compared to the baseline with 9.6% for Dutch texts. Stemming of words with the Snowball stemmer [63] improved upon the baseline precision only slightly with 1.2%. In the work of Kraaij and Pohlmann [32] on Dutch texts, stemming does improve recall but at the cost of a decrease in precision. Using a lemmatizer does not yield a significant result over using a stemmer for Dutch texts [27].

Additionally, the impact of splitting a compounded word, i.e. 'iceberg' into 'ice' and 'berg', is evaluated by Hollink, Kamps, Monz, and de Rijke [27]. Compound splitting on Dutch texts improved the stemmed representation with 3.6% and improves the baseline with 4.0%. Other researchers [8][11][32][46] also report that improved performance can be reached by using compound splitting on Dutch texts.

3 METHOD

In this chapter, the methods and approaches used in this thesis are described. This study is divided into four parts. First, the sentiment scoring model is explained (Section 3.1). Moreover, various text representation methods are evaluated (Section 3.2). These text representations are tested together with different machine learning models (Section 3.3). All these representations and models are evaluated according to several measures (Section 3.4).

3.1 SENTIMENT SCORING

Sentiment scoring can be seen as a *symbolic* approach, rather than a machine learning approach, and is currently used for the children's diary classification task. The algorithm that is currently being used in the PAL project [49] parses the text entry, translates each word into English, and assigns to each translated word a sentiment value from the *SentiWordNet* dictionary [3][19]. *SentiWordNet*, a lexical resource for sentiment mining, assigns to words a positivity, negativity, and objectivity sentiment score. The final classification into either positive, neutral, or negative is assigned to an entry when its sentiment value is below or above a threshold. The performance is however not reported in the paper. Internal resources indicate that the algorithm, tested on 42 diary entries, has an accuracy of 71.43%. Currently, more training examples are available, as will be described in Chapter 4. In this thesis, the sentiment scoring approach will also be studied, as a baseline or reference for the machine learning models.

Since *SentiWordNet* [3][19] is based on English data, and this thesis uses Dutch data, another resource is needed. The CLiPS (Computational Linguistics and Psycholinguistics) research center at the University of Antwerp has developed a linguistic tool called *Pattern* [62], which also has a Dutch module [55]. The Dutch module contains textual functions such as a Part-Of-Speech (POS) tagger and a noun singularizer and pluralizer. Moreover, it contains the sentiment function `sentiment` which returns a valence (positive/negative) and a subjectivity value, based on a lexicon of adjectives. The valence value is between -1 (negative) and 1 (positive), and the subjectivity value is between 0 (objective) and 1 (subjective). The Dutch algorithm is trained on book reviews, but it is not documented where these reviews come from. In Chapter 5, the usage of this sentiment scoring algorithm will be studied. Both the usage of the valence value and the

subjectivity value will be studied. Moreover, an analysis of the errors made by sentiment scoring will be included.

3.2 TEXT REPRESENTATION

An alternative to a sentiment scoring approach is a machine learning approach. In order to train machine learning models, textual data needs to be transformed into numerical data by using words as features and counting their occurrences in a sentence. Optionally, normalization of the text entries is performed before transforming text entries into numerical data, in order to reduce the number of unique words or to overcome the sparseness of the data. A pipeline for preprocessing textual data into numerical data can contain the following steps in relative order: morphological normalization (Section 3.2.1), semantic normalization (Section 3.2.2), n-grams (Section 3.2.3), and term frequencies (Section 3.2.4). An exploration of the impact of all methods except semantic normalization will be performed in Chapter 6, and Chapter 7 will study their performance in machine learning models. Chapter 8 discusses the performance of using a new algorithm for semantic normalization.

3.2.1 *Morphological normalization*

Morphological normalization, i.e. normalizing words in the input text to a standard form, is widely used in the field of automatic text analysis [27]. Morphological normalization was introduced for two main reasons: an increase in performance as related words are mapped to the same standard form [28], and a large reduction in storage size because the number of distinct words reduces [6].

One type of morphological normalization is *stemming*, which automatically removes suffixes from words. The most well-known algorithm for stemming of English words is constructed by Porter [57]. The Porter stemmer maps different words to their stem, i.e. 'connected', 'connection', 'connecting', and 'connections' are all mapped to the stem 'connect'. Using a stemmer can possibly improve the performance. A stemmer can however also be very disruptive and may produce non-words or may map non-related words to the same stem, for example both 'universe' and 'university' to the stem 'univers' [27]. A stemmer does not take different meanings of a word into account and does not look at the context of a word [41].

Therefore, instead of using a traditional stemmer, morphological normalization will be performed by using functions from the Dutch module of *Pattern* [55]. The function `singularize`, `lemma`, and `predicative` will be used. The function `singularize` returns the singular form of plural nouns. The function `lemma` returns the base form of a verb, for example the base form of 'ben' is 'zijn'. Here we can see that using a

lemma function instead of a stemmer function produces more qualitative results, as a stemmer is not able to map 'ben' to 'zijn' as it does not take the meaning of a word into account. The `predicative` function returns the base of an adjective with an $-e$ suffix, for example the base of 'lieve' is 'lief'. We will refer to this type of morphological normalization with the term *stemming*.

Generally, stopwords are removed before applying morphological normalization to the text [27]. A list with stopwords for Dutch [17] is provided in the *Natural Language Toolkit (NLTK)* documentation [48] for Python. The stopwords are derived from a large sample of Dutch texts, and are ranked according to the number of occurrences found in the sample. Stopwords include for example: 'de', 'en', 'van', 'ik', 'te', 'dat', and 'die'. A total of 101 stopwords are present in the Dutch stopword list. Some stopwords, however, are likely to be important for capturing negation and intensity. Therefore, the words 'niet', 'geen', 'veel', 'zo', and 'niets' will not be seen as stopwords and will not be removed, resulting in a list of 96 stopwords.

Different variations of morphological normalization will be evaluated in this thesis. The following variations will be tested: 1) no normalization, 2) stopword removal, and 3) stopword removal and stemming.

3.2.2 *Semantic Normalization*

Because we are dealing with a relatively small dataset, it is very likely that words occur in the test data that are not in the training data. In such a case, no weight is learned for such a word, and the machine learning model will have more difficulty predicting the target value as it cannot use these unseen words. Leaving these 'unknown' words untreated thus might effect the classification performance. For this, a new algorithm will be designed that maps words that are not present in the training set to similar words that do occur in the training set. This will be done with the usage of synonyms, which can be extracted from synonym websites. The precise workings of the new algorithm will be discussed in Chapter 8.

One method of selecting the most appropriate synonym is to use Word2Vec [70]. Word2Vec is a method of transforming words from text entries into numerical vectors, or *neural word embeddings*. Word2Vec does this by using a two-layer neural network with a full corpus as input data. Word2Vec can be compared to an *autoencoder*, but does not train input words through reconstruction but rather against other words in the neighborhood in the corpus. This is done by predicting a target word based on its context (continuous Bag of Words, CBOW), and conversely by predicting a target context based on the word (skip-gram), see Figure 1. Similar words will be placed close

together in vector space [44][45]. By training a Word2Vec model on a dataset, words similar to a specific word can be found.

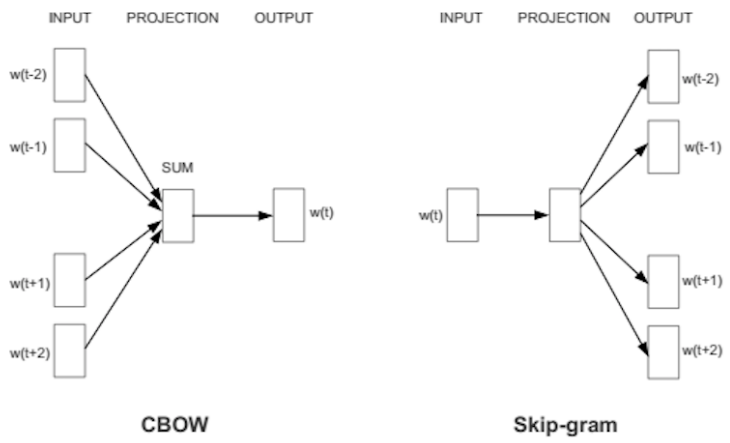

Figure 1: CBOW architecture and Skip-gram. Reprinted from *Efficient Estimation of Word Representations in Vector Space*, by Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean (2013), arXiv preprint arXiv:1301.3781.

The Python package *gensim* [24] provides a function for building a Word2Vec model. In this study, the Word2Vec model will be trained on the BasiLex-corpus [5]. The BasiLex-corpus is an annotated collection of texts written for children of primary school age. It consists of 11.5 million words. Approximately 40 percent of the corpus consists of texts for educational purposes, another approximately 40 percent of the corpus consists of child literature, and the other 20 per cent consists of media texts aimed at children. Because of the large size of the corpus, it is suitable to be used for building a Word2Vec model. Moreover, the BasiLex-corpus consists of texts written for young children. The diaries in this research are, however, written by children. Although a difference exists in texts written for children or by children, the words in the corpus are somewhat comparable to words used in children's diaries and the corpus is the most suitable one available for this research.

The *gensim* package [24] and its model `Word2Vec` transform words into word vectors based on their context. Moreover, they have a similarity function which can calculate the cosine similarity between two word vectors which indicates the similarity between the two corresponding words in the following way:

Definition 1. *The cosine similarity between two word vectors x and y is calculated by:*

$$similarity(x, y) = cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|}$$

The similarity ranges between -1 and 1, where -1 means exactly the opposite and 1 means exactly the same. This can be used to find

similar words, which are the words with the highest cosine similarity. The performance of using Word2Vec for mapping unknown words to words which are present in the training data in the new algorithm for semantic normalization will be studied in Chapter 8.

3.2.3 *n-Grams*

After performing morphological normalization, the resulting text entries can be further pre-processed into n-grams: text input of size n (either n words or n characters). n-Grams are widely used in automatic text algorithms [8]. n-Grams are constructed by sliding an n-long window along the text entry while moving one word or character at a time. At every position, the sequence of words or characters in the window is stored [27]. In this study, we will work with subsets of n words. The most appropriate value for n depends on the dataset. When using bigrams (2-grams) or trigrams (3-grams) instead of unigrams (1-grams, Bag-of-Words), negations such as 'niet leuk' or 'niet zo leuk' can be captured. However, the larger the value for n, the sparser the resulting feature set will be. Therefore, unigrams, bigrams, and trigrams are all incorporated in this study.

The Scikit-Learn package [60], a machine learning package for Python, has the function `CountVectorizer` [13] for transforming text entries into a matrix of token (n-grams) counts. The function has amongst others the parameter *ngram_range* which can be used to indicate which sizes of n-grams will be produced. Other parameters are the maximum number of features (*max_features*), the minimum document frequency (*min_df*), and the maximum document frequency (*max_df*). When a maximum number of features is given, a subset of tokens (n-grams) is selected according to their term frequency across the corpus. Minimum document frequency can be used to remove highly infrequent words. Maximum document frequency can be used to remove corpus-specific stop words [21].

In this study, different values for these parameters will be evaluated. Appropriate values to consider depend on the dataset. Therefore, the specific values will be determined in Chapter 6.

3.2.4 *Term Frequency (- Inverse Document Frequency)*

After applying the `CountVectorizer` [13] function, the resulting matrix consists of occurrence counts of the selected n-grams for every text entry. Hence, the feature value is the number of times an n-gram occurs in the text entry. With this method, however, longer text entries have higher average count values [58]. Another representation to use is *term frequency*, which normalizes the occurrence counts by dividing them by the total number of words in the text entry, also

called document. Note that *term* is in this case used to refer to *n-gram*. Hence, *term frequency* is defined as follows:

Definition 2. *Term frequency tf_{ij} of a term t_i in a text entry d_j is calculated by [69]:*

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}$$

where n_{ij} is the occurrence count of term t_i in text entry d_j, which is divided by the total number of words in the text entry.

Moreover, words that occur in many documents can be downscaled in order to make them less dominant [58]. This can be useful as most frequent words are most of the times non-informative words or stopwords. Frequent words are, however, not by definition non-informative, and an improved performance of tf-idf over term frequencies or occurrence counts depends on the data. A widely used representation for doing this is using *term frequency - inverse document frequency (tf-idf)*. The term frequency, as defined in Definition 2, is multiplied by the *inverse document frequency*, which is defined as follows:

Definition 3. *Inverse document frequency idf_i of a term i is calculated by [69]:*

$$idf_i = log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

where $|D|$ is the total number of documents (or text entries) in the corpus, and $|\{j : t_i \in d_j\}|$ is the number of documents (text entries) the term t_i occurs once or more in.

Definition 4. *Term frequency - inverse document frequency for a term t_i in a document d_j is calculated by:*

$$tf_{ij} \cdot idf_i$$

The Scikit-Learn package for Python [60] has a function `TfidfTransformer` [71] for transforming the occurrence counts into either term frequencies or tf-idf. In this study, the usage of occurrence counts, term frequency and tf-idf will be evaluated.

3.3 MACHINE LEARNING MODELS

With these different text representations, different machine learning models are trained. First, Logistic Regression is explained (Section 3.3.1). Secondly, the usage of the lasso penalty within Logistic Regression as a feature selection step is discussed (Section 3.3.2). Moreover, the two other often used machine learning models are reviewed: Naive Bayes (Section 3.3.3) and Support Vector Machine (Section 3.3.4).

3.3.1 *Logistic Regression*

The Logistic Regression model is related to the Linear Regression model, but uses an inverse logistic (logit) function to transform the output value into a value between 0 and 1. The output variable can now be interpreted as a probability. The Logistic Regression model function uses weights β_j for all features X_j, similar to a Linear Regression model. The output value when working with a binary target variable Y is then calculated by: [29]

Definition 5. *The logistic function in a binary case with q features, $X \equiv (X_1, \ldots, X_q)$, is calculated by [29]:*

$$p(X) \equiv Pr(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_q X_q}}{1 + e^{\beta_0+\beta_1 X_1+\ldots+\beta_q X_q}}$$

$p(X)$ is equal to $Pr(Y = 1|X)$ in a binary case. Because of the logit function, the output value is between 0 and 1. In a binary case, if the value calculated by the logistic function is larger or equal to some hand-set threshold, then the predicted value is 1, and otherwise 0. The threshold can be set manually, but one often used threshold is 0.5. After manipulating the function in Definition 5, we get what is also called the *log-odds*:

Definition 6. *The log-odds in a binary case are calculated by [29]:*

$$log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_q X_q$$

The *log-odds* do not indicate a straight-line relationship between the logistic function $p(X)$ and the inputs X_j, as is in linear regression. The relationship is rather complicated, and the rate of change in $p(X)$ per one-unit increase in an X_j depends on the current value of X_j and the values of the other variables. The only conclusion that can be drawn is that if β_j is positive, then an increasing X_j is associated with an increasing $Pr(Y = 1)$ if all other variables remain the same. [29]

When working with a multinomial target variable, the formula in Definition 5 is slightly adapted:

Definition 7. *The logistic function in a multinomial case with q features and K classes is calculated by [7]:*

$$p_k(X) \equiv Pr(Y = k|X) = \frac{e^{B_k X}}{\sum_{l=1}^{K} e^{B_l X}}$$

where $B_k X$ equals $\beta_{k,0} + \beta_{k,1} X_1 + \ldots + \beta_{k,q} X_q$ with weights specific for every class k, and where one class is set as reference with $B_1 = 0$.

In this thesis, however, a one-versus-rest scheme is used, which trains a binary classifier for all classes after which the predictions for all binary classifiers are combined into a single prediction. From now on, the binomial case is discussed again. Learning the weights (or coefficient estimates) β_j is usually done by maximum likelihood. Maximum likelihood aims at optimizing the *likelihood function*, which is calculated as follows in the binary case when using only a β_0 and a β_1 [29]:

Definition 8. *The likelihood function in a binary case when using only β_0 and β_1 is calculated as follows [29]:*

$$\mathcal{L}(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

Various numerical optimization algorithms are developed for finding the estimators of β_0 and β_1 by minimizing the negative log-likelihood. The negative log-likelihood is a transformation of the likelihood function by taking the logarithm of the likelihood function and multiplying it by -1.

3.3.1.1 *Regularization*

In order to prevent the model from overfitting, regularization can be used. This will be explained in the binary case, but the difference with a multinomial case is only the number of coefficients for each variable. One type of regularization is shrinkage of the weights (coefficient estimates) β_j to a lower value or even to zero. The two best-known techniques for doing so are the usage of a *ridge* or a *lasso* penalty. The *ridge* penalty is also called a *l2* penalty, and the *lasso* penalty is also called a *l1* penalty. When using a *ridge* penalty, the goal is not to minimize the negative log-likelihood ($-log\ \mathcal{L}$), but rather to minimize the following: [29]

Definition 9. *When adding the ridge penalty, the aim is to minimize [29]:*

$$-log\ \mathcal{L} + \lambda \sum_{j=1}^{q} \beta_j^2$$

where $\lambda \geq 0$ is a tuning parameter which needs to be specified separately.

The *lasso* penalty is defined as follows: [29]

Definition 10. *When adding the lasso penalty, the aim is to minimize [29]:*

$$-log\ \mathcal{L} + \lambda \sum_{j=1}^{q} |\beta_j|$$

where $\lambda \geq 0$ is a tuning parameter which needs to be specified separately.

Figure 2 shows the constraint regions of the lasso and ridge penalty and the contours of the error function. The solution without regularization is marked as $\hat{\beta}$. The blue solid areas are the constraint regions, with $|\beta_1| + |\beta_2| \leq s$ for lasso (left) and $\beta_1^2 + \beta_2^2 \leq s$ for ridge (right). The red ellipses are the contours of the error function. The $\hat{\beta}$ solution is the same as the regularization solution if s is sufficiently large, where a large value for s corresponds to $\lambda = 0$ in Definitions 9 and 10. However, if s decreases, coefficient estimates have to shrink in order to fall within the constraint region.

The different effect of using lasso or ridge penalty can also be seen from Figure 2. When the ellipse intersects with the constraint region at an axis, one of the coefficients will be equal to zero. Because the constraint area of the ridge penalty has no sharp points, an intersection between the constraint region and the ellipse will generally not occur on an axis. With the lasso penalty, the constraint area has corners on each of the axes. Therefore, the constraint region and the ellipse will often intersect at an axis, and feature elimination will thus occur more often. [29].

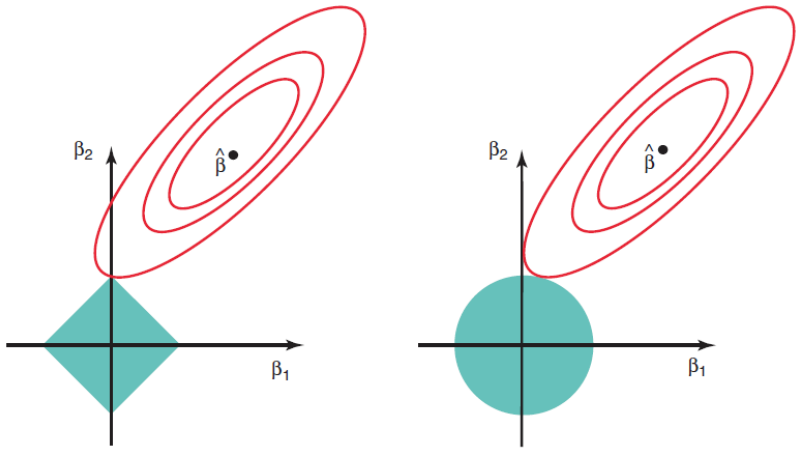

Figure 2: Contours of the error and constraint functions for the lasso (left) and ridge regression (right). Reprinted from *An Introduction to Statistical Learning*, by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013), Springer Texts in Statistics.

In this thesis, the model `LogisticRegression` [38] from the Scikit-Learn package for Python will be used for modelling Logistic Regression. The *liblinear* solver will be used, which uses one-versus-rest schemes for multinomial target variables. The usage of both a *l1* (lasso) and a *l2* (ridge) penalty will be evaluated. Moreover, different values for λ will be evaluated. The `LogisticRegression` model uses a parameter C instead of a λ value. The parameter C is an inverse of regularization strength. Hence, smaller values indicate a stronger regularization [38].

3.3.1.2 *Proportional Odds*

Because the classes in this research can be interpreted to be at an ordinal level (negative-neutral-positive), instead of at a nominal level, an ordinal Logistic Regression model will be evaluated as well. The most common used ordinal Logistic Regression model is the proportional odds model, also called the cumulative logit model. The proportional odds model is formulated differently than with the regular Logistic Regression model, namely as $Pr(Y \leq k|X)$ instead of $Pr(Y = k|X)$ [1]. Moreover, in the case of K different target values, only one weight β_j for each of the features X_j is present, instead of $K-1$ different weights $\beta_{k,j}$ for each feature X_j with traditional Logistic Regression [31]. Hence, the form is as follows:

Definition 11. *The proportional odds model has the following form [1]:*

$$Pr(Y \leq k|X) = \frac{exp(\alpha_k + \sum_{j=1}^{q} \beta_j X_j)}{1 + exp(\alpha_k + \sum_{j=1}^{q} \beta_j X_j)}$$

where α_k is the intercept for a specific class k.

The odds are subsequently calculated differently than in Definition 6, namely as:

Definition 12. *The proportional odds are calculated by [1]:*

$$\frac{Pr(Y \leq k|X)}{1 - Pr(Y \leq k|X)} = \frac{Pr(Y \leq k|X)}{Pr(Y > k|X)}$$

The proportional odds model is implemented in R with the `ordinal.gmifs` function from the *ordinalgmifs* package [52] for R. The function `ordinal.gmifs` performs lasso penalization by using the generalized monotone incremental forward stagewise method. Incremental forward stagewise regression solves the lasso regression problem when enforcing monotonicity. The only difference is that forward stage-wise is optimal per unit L_1 arc-length traveled along the coefficient path, where lasso reduces the residual sum-of-squares per unit increase in L_1-norm of the coefficient β [26]. The assumption of an ordinal model is that variables are monotonically related to the order. Before training the ordinal model, the function `nominal_test` from the *ordinal* package [51] can be used to test whether variables are non-monotonically related to the order, which is a violation of the assumption of the ordinal model. The usage of this model will be evaluated in Chapter 9.

3.3.2 *Dimension Reduction*

As explained in the previous subsection, the usage of a lasso (or *l1*) penalty in Logistic Regression leads to a reduction of dimensions be-

cause coefficient estimates can be shrunk down to zero. The outcome of a lasso penalty can also be used as a feature selection step for the two upcoming models, Naive Bayes and Support Vector Machine. The function `SelectFromModel` [61] in Scikit-Learn can be used for this purpose. The function `SelectFromModel` asks for a model, such as a Logistic Regression with lasso penalty, and returns the features which have non-zero coefficients. The usage of a feature selection step for both Naive Bayes and Support Vector Machine will be studied.

3.3.3 *Naive Bayes*

Another machine learning model is Naive Bayes, which is a classifier based on the Bayes' rule, i.e. [65]:

$$Pr(\text{hypothesis}|\text{data}) = \frac{Pr(\text{data}|\text{hypothesis}) \times Pr(\text{hypothesis})}{Pr(\text{data})}$$

The structure of the Naive Bayes model is shown in Figure 3. The Naive Bayes classifier learns the conditional probability of each feature X_i given the class label Y, hence $Pr(X_i|Y)$. With the Bayes' rule, the probability of Y given a particular valuation of features $X_1, \ldots, X_q$ (in the case of q features) can be calculated. The predicted value is then the class with the highest posterior probability. The assumption that Naive Bayes makes, namely that all features are conditionally independent given the class label Y, is however rather strong. In a text dataset, it cannot be assumed that words are chosen independent of each other in a sentence. Nevertheless, the performance of Naive Bayes is competitive with state-of-the-art classifiers. [23]

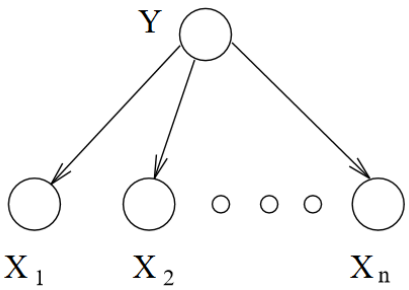

Figure 3: The structure of the naive Bayes network. Reprinted from *Bayesian Network Classifiers*, by Nir Friedman, Dan Geiger, and Moises Goldszmidt (1997), in *Machine Learning*, 29, p. 131-163.

A Multinomial Naive Bayes model is implemented in Scikit-Learn with the function `MultinomialNB` [47]. A Multinomial model is chosen rather than a Bernoulli model because the Multinomial model can work with word counts and fractional counts (such as tf-idf) which will be used in this thesis as well. McCallum and Nigam [43] also show an improved performance of using a Multinomial Naive Bayes

over a Bernouilli Naive Bayes for text classification. The probability distribution is represented by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yq})$ for a training set with y class labels and q features. θ_{yi} is the probability $Pr(x_i|y)$ of feature x_i given the class y [47].

Additionally, a smoothing parameter can be used to prevent from having zero probabilities for future computations. The smoothing parameter transforms the distribution θ_{yi} into $\hat{\theta}_{yi}$ in the following way [47]:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha q}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the occurrence count of word i in the vocabulary of class y, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all words for class y. Smoothing is called Laplace smoothing when $\alpha = 1$ and Lidstone smoothing when $\alpha < 1$ (but always $\alpha \geq 0$) [47]. In this thesis, different values for the smoothing parameter α will be evaluated.

3.3.4 *Support Vector Machine*

A Support Vector Machine (SVM) is a specific instance of a *support vector classifier*. A *support vector classifier* aims at finding a hyperplane (or decision boundary) where the distance from the hyperplane to the closest point of each class is maximal. The margin is then the distance between the hyperplane and the closest data points of each class. Because a support vector classifiers aims at finding the maximal margin, low confidence classifications are avoided. A cost parameter C is used as a penalty for variables which lie within the margin or on the wrong side of the hyperplane [7]. Slack variables $\epsilon_1, \ldots, \epsilon_n$ allow observations to be on the wrong side of the hyperplane or margin. If ϵ_i (of the ith observation) is zero, the observation is on the correct side of the margin. If $\epsilon_i > 0$, the observation is within the margin but on the correct side of the hyperplane, and when $\epsilon_i > 1$, the observation is on the wrong side of the hyperplane [29]. A high value for C results in a smaller margin with less misclassifications, whereas a low value for C will result in a larger margin with more misclassifications [7]. In the book of James, Witten, Hastie, and Tibshirani [29], however, C is defined as a budget rather than a cost. Hence, the effect of the parameter C as a budget is opposite to the effect of C as a cost. Formally speaking, the goal of a support vector classifier according to James, Witten, Hastie, and Tibshirani [29] is defined as follows:

Definition 13. *The goal of the support vector classifier is to find the hyperplane that optimizes [29]:*

$$\underset{\beta_0, \beta_1, \ldots, \beta_q, \epsilon_1, \ldots, \epsilon_n}{\textit{maximize}} \quad M$$

$$\textit{subject to} \sum_{j=1}^{q} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_q x_{iq}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$$

with weights β_j, q features, n training examples, and where M is the width of the margin, C is the budget parameter, and $\epsilon_1, \ldots, \epsilon_n$ are slack variables.

A support vector classifier aims at finding a hyperplane that linearly separates the two classes from each other. In some cases, however, a well performing linear boundary cannot be found. Figure 4 provides an example of a dataset which is difficult to divide linearly. In such a situation, a kernel function that maps the input features to a higher dimensional space in which the data is linearly separable can be a solution. A *support vector machine* is a *support vector classifier* with a non-linear kernel function. [29]

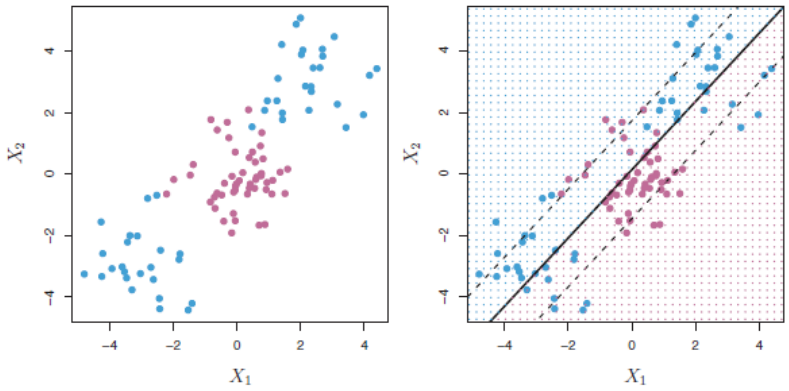

Figure 4: Left: The observations fall into two classes, with a non-linear boundary between them. Right: The support vector classifier seeks a linear boundary, and consequently performs very poorly. Reprinted from *An Introduction to Statistical Learning*, by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013), Springer Texts in Statistics.

A kernel function quantifies the similarity of two observations. The kernel used in the *support vector classifier* is the *linear kernel*, which uses the inner product of two observations x_i and $x_{i'}$. The linear kernel is comparable to using a Pearson (standard) correlation. The kernel function K of the linear kernel is as follows [29]:

Definition 14. *The linear kernel function is defined as [29]:*

$$K(x_i, x_{i'}) = \sum_{j=1}^{q} x_{ij} x_{i'j}$$

Two other type of kernels are the *polynomial* kernel and the *radial* kernel. These kernels are non-linear. When a non-linear kernel is used, the resulting classifier is known as a *support vector machine* instead of a *support vector classifier*. The polynomial and radial kernel are calculated as follows [29]:

Definition 15. *The polynomial kernel function of degree d is defined as [29]:*

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{q} x_{ij} x_{i'j})^d$$

Definition 16. *The radial kernel function is defined as [29]:*

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{q} (x_{ij} - x_{i'j})^2)$$

The effect of using a polynomial or radial kernel function is displayed in Figure 5. The figure shows that the data that was not linearly separable before, can now be separated by using a non-linear kernel function.

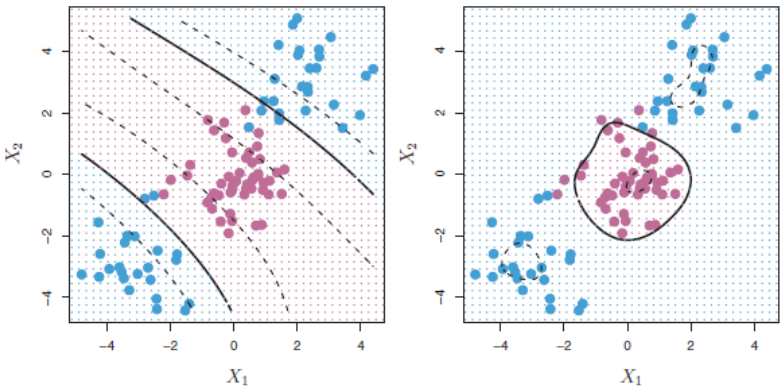

Figure 5: Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 4, resulting in a far more appropriate decision rule. Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary. Reprinted from *An Introduction to Statistical Learning*, by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013), Springer Texts in Statistics.

Support Vector Machines are modeled with the function `SVC` [68] in the Scikit-Learn package for Python. The `SVC` function uses the C parameter as a cost parameter rather than a budget parameter. Hence, a low value for C means a smooth decision surface and a high value for C means more focus on correctly classifying the training examples. Different values for the cost parameter C, different kernels (linear, polynomial, rbf), different values for γ for the polynomial and radial kernel, and different degrees for the polynomial kernel will be evaluated in this study.

3.4 EVALUATION METHODS

Because of the small size of the dataset, multiple training and test splits are used to evaluate the machine learning models. The approach of nested cross-validation is used. The data is split into ten folds with the `StratifiedKFold` [67] function of Scikit-Learn, where the folds are made by preserving the percentage of samples for each class. For $j = 1..10$, cross-validation is performed on all folds except j to determine the optimal parameters. A model is then trained on all folds except j with the found parameters, after which the prediction performance is estimated on j. The final performance of a model is then the average and standard deviation of each of the 10 folds. The same division of the training data among all outer and inner folds will be used for all models.

3.4.1 *Accuracy*

Machine learning models will be mainly evaluated on their accuracy. The accuracy of a model is defined as follows:

Definition 17. *The accuracy of a model is computed by:*

$$\frac{\textit{number of correctly classified training examples}}{\textit{total number of training examples}}$$

Moreover, the confusion matrices of each model will be evaluated. The confusion matrix can be build with the Scikit-Learn function `confusion_matrix`. In the case of working with the three classes {negative, neutral, positive}, the confusion matrix looks as follows (Table 1):

		predicted		
		negative	neutral	positive
actual	negative	a	b	c
	neutral	d	e	f
	positive	g	h	i

Table 1: Example confusion matrix

From such a confusion matrix, the accuracy can also be calculated by dividing the sum of the diagonal by the sum of the whole matrix, indicated by the green cell colors:

$$\frac{a+e+i}{a+b+c+d+e+f+g+h+i}$$

When it is the case that the number of diary entries per class differs substantially, it might be more relevant to use a more balanced approach. The balanced accuracy can then be used, which is the sum of recall for each class, averaged over the number of classes [64].

3.4.2 *Mean Absolute Error*

A measure related to accuracy is the mean absolute error (MAE). MAE is defined as follows, when the classes $\{negative, neutral, positive\}$ are coded respectively as $\{1, 2, 3\}$:

Definition 18. *The mean absolute error (MAE) is calculated by:*

$$\frac{1}{n}\sum_{i=1}^{n}|f_i - y_i|$$

where n is the size of the test set, f_i is the predicted class and y_i is the true class of a test example i.

Note that the goal is to minimize the MAE, and to maximize accuracy and balanced accuracy.

3.4.3 *Precision and Recall*

Next to solely evaluating models on their accuracy, it can also be important to look more specific to the classification errors made by the algorithm. Another type of measure is to use *precision* and *recall*. Precision is also called the positive predictive value. Recall is the same as the True Positive Rate, and is also known as sensitivity [15]. Precision and recall are calculated as follows:

Definition 19. *Precision is computed by [15]:*

$$\frac{true\ positives}{true\ positives + false\ positives}$$

Definition 20. *Recall is computed by [15]:*

$$\frac{true\ positives}{true\ positives + false\ negatives}$$

Precision and recall are often combined into one measure, called the *F-score*. The F-score has a parameter β which can be used to indicate a preference for recall or precision. The F-score is calculated as follows:

Definition 21. *The F_β-score is computed by [25]:*

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

The F-score with $\beta = 1$ is often used, and is also called the F_1-score. The F_1-score has a maximum of 1 and a minimum of 0, where 1 means perfect performance and 0 means the worst performance. The F_1-score will be used in this study next to classification accuracy. The F_1-score is then calculated as follows:

Definition 22. *The F_1-score is computed by [25]:*

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

4

DATA

In this chapter, the data used in this study is discussed. In Section 4.1, the data collection and labelling of the data is described. Section 4.2 explains the pre-processing of the dataset.

4.1 COLLECTION AND LABELLING

Datasets are obtained from internal resources at TNO. The data was collected by TNO from children in the age of 8 to 14 at diabetes related environments (hospitals and diabetes camps). An overview of the datasets used can be seen in Table 2. All text entries in the datasets were written by children. From the *vriendenboekje* and *year1_results* datasets, only text entries which resemble diary entries were selected.

dataset	type of data	nr of text entries
sentiment_test	diary entries	44
a_note	activities performed today	49
e_note	current emotion	61
mike	activities performed today	154
vriendenboekje	friendly message from a child to the robot	13
year1_results	questionnaire regarding the usability of the robot	30
total		351

Table 2: Overview datasets

Text entries were split into multiple entries when they indicated sentiments or activities at different points in time (i.e. this morning, tomorrow, next week) or when they were very long. The number of resulting text entries is shown in Table 3.

Out of the 395 text entries, 282 text entries were labelled by the child itself. The labels, however, did in many cases not relate to the content of the text entry. Moreover, the sentiment analysis algorithm should not make any assumptions when analyzing the text entries. Text entries concerning birthdays or parties should thus be seen as neutral, except in the case a sentiment word such as 'leuk' (fun) was used in the text entry as well. Thus, even though a child might find a specific activity positive or negative, if the child does not express his sentiment clearly in the text, the text entry should be classified as

dataset	nr of original entries	nr of split entries
sentiment_test	44	70
a_note	49	52
e_note	61	65
mike	154	165
vriendenboekje	13	13
year1_results	30	30
total	351	395

Table 3: Overview datasets after splitting text entries

neutral. Therefore, all text entries were labelled by annotators and the labels of the children were not used.

Three annotators all labelled all text entries as either negative, neutral, or positive. The first annotator was the writer of this thesis (woman, aged 23). The second annotator was a woman aged 53, and the third annotator was a man aged 56. Annotators were told to objectively label the text entries, thus not making assumptions on the sentiment of specific activities. The inter-annotator agreement is measured by Cohen's Kappa and is displayed in Table 4. A Cohen's Kappa of above 0.8 indicates 'very good agreement'. Hence, the agreement between every pair of annotators was very good.

	1	2	3
1		0.86	0.85
2	0.86		0.82
3	0.85	0.82	

Table 4: Inter-annotator agreement measured by Cohen's Kappa

Total agreement was not reached in 56 cases. In those cases, the majority rule was used to select the most appropriate label. Exceptions to the majority rule were text entries indicating illness. Some text entries indicating illness were labelled as negative and others as neutral by the majority rule. Therefore, in order to remain consistent, all text entries indicating illness were labelled as negative. The division of the resulting labels across the different datasets is shown in Table 5.

4.2 PRE-PROCESSING

Because the text entries were written by children, some text entries contained spelling errors. Spelling errors were corrected manually, because a well-performing spelling correction algorithm for Dutch was not found and the size of the dataset was relatively small. In

dataset	negative	neutral	positive
sentiment_test	8	31	31
a_note	1	30	21
e_note	12	34	19
mike	26	66	73
vriendenboekje	0	0	13
year1_results	13	6	11
total	60	167	168

Table 5: Division of labels across the datasets

practice, however, automatic spelling correction should be employed in order to guard the performance, as only weights are learned for correct spellings of words.

Previous research [27] has indicated the improvement of performance when using compound splitting, i.e. splitting 'iceberg' into 'ice' and 'berg'. In this thesis, some words are split as well, thereby making it easier for the algorithm to recognize similar feelings such as 'buikpijn' and 'keelpijn'. An overview of the compound splitting performed is shown in Table 6. An '*' indicates any word.

compounded word	first word	second word
*pijn	*	pijn
*feest	*	feest
school*	school	*
kerst*	kerst	*
super*	super	*
feestweek	feest	week
gymles	gym	les

Table 6: Words split into compounds

Furthermore, the following punctuation marks were removed: commas (,), full stops (.), colons (:), apostrophes ('), inverted commas ("), hyphens (-), and parenthesis (()). Exclamation marks (!) were not removed, but are treated as a word. Multiple consecutive exclamation marks were replaced by one single exclamation mark, in order to prevent from extreme outliers, because the usage of exclamation marks varied widely among diaries. Finally, diacritical characters, i.e. characters with an accent, were replaced with their corresponding unmarked character. All characters were transformed into lowercase characters.

5 SENTIMENT SCORING

In this chapter, the usage of sentiment scoring is explored. Sentiment scoring can be seen as a *symbolic* approach, rather than a machine learning approach, and is currently used for the children's diary classification task. The performance of a sentiment scoring algorithm can function as a baseline or reference for the machine learning models. The sentiment scoring function `sentiment` from the Dutch *Pattern* module of CLiPS [55] is used. This function returns a valence value between -1 (negative) and 1 (positive), and a subjectivity value between 0 (objective) and 1 (subjective). In this chapter, both the usage of the valence value (Section 5.1) and both the valence and subjectivity value (Section 5.2) is studied.

5.1 VALENCE VALUE

After feeding a sentence to the `sentiment` function, a valence value between -1 and 1 is returned. The predicted label is then assigned with the usage of thresholds: if the value is larger than zero, the classification is 'positive'. Similarly, if the value is smaller than zero, the classification is 'negative'. If the valence value is exactly zero, then the classification is 'neutral'. The algorithm is thus as follows:

Input : valence value v_i of text entry t_i
Output : label l_i
if $v_i > 0$ **then**
  $l_i =$ positive
else if $v_i < 0$ **then**
  $l_i =$ negative
else
  $l_i =$ neutral

Algorithmus 1 : Labelling algorithm based on the valence value

Performance is measured on the whole dataset by means of the accuracy, balanced accuracy, F1-score and mean absolute error (MAE) scores. The performance is shown in Table 7. Note that the goal is to minimize MAE, whereas the goal is to maximize the other three measures.

The corresponding confusion matrix is shown in Table 8. Colors indicate the percentage of examples for each class which are classified correctly.

measure	performance
accuracy	0.7570
balanced accuracy	0.7410
F1-score	0.7184
MAE	0.2937

Table 7: Performance of sentiment scoring with valence value

		predicted		
		negative	neutral	positive
actual	negative	41	8	11
	neutral	35	108	24
	positive	9	9	150

Table 8: Confusion matrix sentiment scoring with valence value

An analysis of negative text entries with the highest valence values and positive text entries with the lowest valence values is displayed in Table 9.

text entry	valence
negative text entries	
daarna gingen we een modeshow lopen dat was dan weer niet zo leuk	0.60
veel buik pijn en niet zo lekker	0.60
school school niet zo leuk	0.60
niet helemaal goed het lukte een beetje	0.55
ik snapte het niet helemaal goed niet hoe het werkte	0.55
positive text entries	
we gaan nog naar burger zoo daar heb ik zin in	-0.65
met de vogels ontbijten de vogels kwamen naar mij en naar mijn zusje en mijn moeder en het was grappig en leuk	-0.23
naar huis ik en een vriendin en een vriend gingen	-0.23
naar huis het was wel donker maar ik had een leuke schoen ervoor	-0.10
ik ging naar charlie in het ziekenhuis en dat was leuk	-0.03

Table 9: Analysis of negative text entries with the highest valence values and positive text entries with the lowest valence values

The trend in the negative text entries in Table 9 appears to be the usage of 'niet zo' or 'niet helemaal', which is not detected as a negation of positiveness by sentiment scoring. The valence of 'niet lekker' and 'niet leuk' is -0.3, whereas the valence of 'niet zo lekker' and 'niet zo leuk' is 0.6. Apparently, sentiment scoring does not detect phrases such as 'niet zo' or 'niet helemaal' as a negation of the positive sentiment, because 'niet' is not directly in front of the positive sentiment.

The trend in the positive text entries is the usage of the word 'naar'. In these text entries, the word 'naar' is a preposition and is used as an indication of a movement towards a place, such as the zoo. However, the word 'naar' as an adjective is a negative word. Seemingly, sentiment scoring does not take the usage of 'naar' as a preposition into account. The difference between the two usages of the word 'naar' can be detected by means of a Part-Of-Speech tagger, in order to better classify the text entries.

5.2 VALENCE AND SUBJECTIVITY VALUE

The usage of the subjectivity value as a method for detecting neutral text entries is explored in this section. The subjectivity value is between 0 (objective) and 1 (subjective). Therefore, a threshold of 0.5 is chosen to detect neutral (objective) text entries, which is exactly between 0 and 1.

Text entries are now classified as 'positive' if their corresponding valence value is larger than zero and if their subjectivity value is larger than 0.5. Text entries are classified as 'negative' if their corresponding valence value is smaller than zero and if their subjectivity value is larger than 0.5. In all other cases, the classification is 'neutral'. Hence, the algorithm is as follows:

Input : valence value v_i and subjectivity value s_i of text entry t_i
Output : label l_i
if $v_i > 0$ *and* $s_i > 0.5$ **then**
  $l_i =$ positive
else if $v_i < 0$ *and* $s_i > 0.5$ **then**
  $l_i =$ negative
else
  $l_i =$ neutral

Algorithmus 2 : Labelling algorithm based on the valence and subjectivity value

Table 10 shows that the usage of the subjectivity value next to the valence value increases performance according to all measures. The accuracy is increased from 0.7570 to 0.7844, the balanced accuracy from 0.7410 to 0.7629, the F1-score from 0.7184 to 0.7447, and the MAE from 0.2937 to 0.2656. The division of classes according to this model is displayed in Figure 6. Figure 6 shows that a low score for

subjectivity can be an indication for a neutral sentiment. The confusion matrix is displayed in Table 11. The confusion matrix shows that more neutral text entries are labelled correctly compared to the algorithm which only uses the valence value. Colors indicate the percentage of examples for each class which are classified correctly.

measure	performance
accuracy	0.7848
balanced accuracy	0.7629
F1-score	0.7447
MAE	0.2656

Table 10: Performance of sentiment scoring with valence and subjectivity value

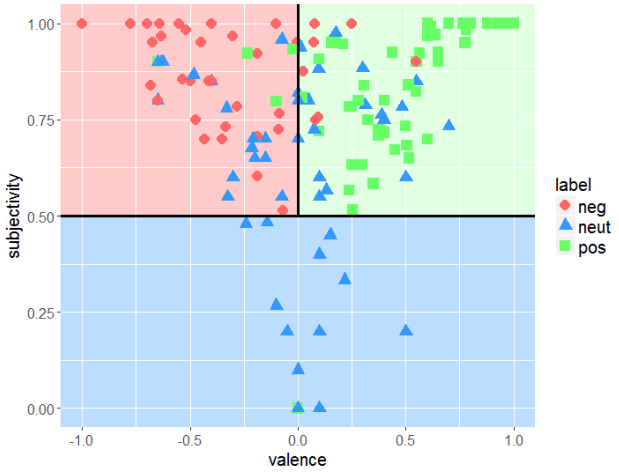

Figure 6: Scatter plot of data points based on their valence (x-axis) and subjectivity (y-axis) values, labelled by the usage of valence and subjectivity value

		predicted		
		negative	neutral	positive
actual	negative	41	8	11
	neutral	31	119	17
	positive	9	9	150

Table 11: Confusion matrix sentiment scoring with valence and subjectivity value

6 TEXT REPRESENTATION

In this chapter, an exploration of the different text representations is reported. This chapter is intended solely as an exploration of the workings of the transformations, not as a base to make decision upon. All transformations in this chapter are performed on the whole dataset. When training machine learning models, however, the transformations are first fit and applied on the training data, after which they are applied on the test data. The chapter describes the three steps performed before learning models, namely morphological normalization (Section 6.1), n-grams (Section 6.2), and term frequency weighting (Section 6.3).

6.1 MORPHOLOGICAL NORMALIZATION

The first step is morphological normalization. In this thesis, three different variations are evaluated: 1) no normalization, 2) stopword removal, and 3) stopword removal and stemming. In this section, all variations are discussed.

For stopword removal, a list with stopwords for Dutch [17] is provided in the *Natural Language Toolkit (NLTK)* documentation [48] for Python. The stopwords are derived from a large sample of Dutch texts. Stopwords include for example: 'de', 'en', and 'ik'. A total of 101 stopwords are present in the Dutch stopword list. Some stopwords, however, are likely to be important for capturing negation and intensity. Therefore, the words 'niet', 'geen', 'veel', 'zo', and 'niets' are not seen as stopwords and are not removed, resulting in a list of 96 stopwords. When performing stopword removal, the stopwords in the stopword list are removed from the sentences. The effect of stopword removal on the average sentence length is shown in Table 12.

	original	stopword removal
all	9.93 (6.77)	5.63 (3.49)
negative	9.48 (4.84)	5.90 (2.96)
neutral	9.55 (6.87)	5.38 (3.45)
positive	10.46 (7.24)	5.77 (3.71)

Table 12: Average sentence length (and standard deviation) after stopword removal

Table 12 shows that both the average sentence length and the standard deviation decrease after performing stopword removal. The ta-

ble also shows that the smallest reduction in average sentence length takes place in the negative text entries. This indicates that negative text entries contain less stopwords.

For stemming, an algorithm is built manually with the usage of functions from the Dutch module of *Pattern* [55]. The functions `singularize`, `lemma`, and `predicative` are used. The function `singularize` returns the singular form of plural nouns. The function `lemma` returns the base form of a verb: for example the base form of 'ben' is 'zijn'. The `predicative` function returns the base of an adjective with an $-e$ suffix: for example the base of 'lieve' is 'lief'. The Part-of-Speech (POS) tag of a word (e.g. 'plural noun', 'singular noun', 'adjective', etc.) in a sentence can be found with the function `parse`, which is also from the Dutch module of *Pattern* [55]. The stemming algorithm is then as follows:

Input : sentence s
Output : sentence s' with stemmed words
$s' = ''$
forall *words* w *occurring in sentence* s **do**
  if w *is a plural noun* **then**
    w = `singularize`(w)
  else if w *is a verb not in base form* **then**
    w = `lemma`(w)
  else if w *is an adjective* **then**
    w = `predicative`(w)
  $s' = s' + w$
end

Algorithmus 3 : Stemming algorithm

The effect of both 1) stopword removal and 2) stopword removal and stemming on the number of distinct words in the dataset is shown in Table 13. Stemming is always performed after stopword removal, hence the term 'stemming' in the table refers to both stopword removal and stemming.

	original	stopword removal	stemming
all	821	759	692
negative	219	171	163
neutral	526	467	429
positive	417	357	326

Table 13: Number of distinct words after stopword removal and stemming

Table 13 shows that the number of distinct words in the dataset decreases as stopword removal and stemming are performed. The number of distinct words in negative text entries is lower, because less negative text entries occur in the dataset. The number of positive

and neutral text entries in the dataset differs with only 1 text entry, but the number of distinct words differs quite substantially: the set of neutral text entries contains more distinct words than the set of positive text entries.

6.2 N-GRAMS

After morphological normalization, text entries are transformed into n-Grams. The Scikit-Learn package [60], a machine learning package for Python, has the function `CountVectorizer` [13] for transforming text entries into a matrix of n-grams counts. The function has amongst others the parameter *ngram_range* which can be used to indicate which sizes of n-grams are produced. Other parameters are the maximum number of features (*max_features*), the minimum document frequency (*min_df*), and the maximum document frequency (*max_df*). In this research, the following parameter settings are used:

- ngram_range = (1,3), hence all uni-, bi-, and tri-grams
- max_features = None
- min_df = 2, in order to eliminate non-words or highly infrequent words
- max_df = None

In this section, the usage of different values for n is explored. For the machine learning models, however, all uni-, bi-, and trigrams (with minimum document frequency of 2) are used. The exploration in this section is done on the dataset after stopword removal and stemming. In this section, transformations are performed on the whole dataset. When training machine learning models, however, the transformations are first fit and applied on the training data, after which they are applied on the test data. Table 14 shows the number of distinct n-grams for different values of n.

	unigrams	bigrams	trigrams
all	243	156	47
negative	97	40	8
neutral	195	63	8
positive	177	108	36

Table 14: Number of distinct n-grams for $n = \{1, 2, 3\}$ with minimum document frequency = 2

Table 14 shows that the number of distinct n-grams decreases as n increases, which is expected. Moreover, it shows that the set of positive text entries contains more distinct trigrams than the set of

neutral text entries, even though they contain approximately the same number of text entries. This can be explained by the fact that the set of neutral text entries contains more distinct words (unigrams), thereby decreasing the possibility of higher-order n-grams occurring twice or more.

6.2.1 *Unigrams*

In this section, the usage of unigrams on the dataset after stopword removal and stemming is analyzed. Table 15, Table 16, and Table 17 show the top five most occurring words in respectively negative, neutral, and positive text entries. Colors indicate the percentages displayed in the tables.

word	negative	neutral	positive
niet	43.33	11.38	3.57
heel	20.00	1.20	17.86
leuk	18.33	3.59	70.83
vinden	16.67	2.40	18.45
moeilijk	16.67	0.06	0.00

Table 15: Top five most occurring words in negative text entries, scores indicate the percentage of text entries per class that contain the word

word	negative	neutral	positive
school	13.33	26.95	25.60
gaan	13.33	21.56	15.48
we	6.67	19.76	14.29
spelen	1.67	14.37	7.74
niet	43.33	11.38	3.57

Table 16: Top five most occurring words in neutral text entries, scores indicate the percentage of text entries per class that contain the word

Table 15, Table 16, and Table 17 show that some words overlap between the tables, such as 'leuk', 'heel', 'school', 'gaan', and 'niet'. The high occurrence of 'leuk' in negative text entries can be explained by the usage of phrases such as 'niet leuk'. Children also use 'school' approximately double as much in positive text entries than in negative text entries, thereby indicating that going to school is not always unpleasant.

word	negative	neutral	positive
leuk	18.33	3.59	70.83
school	13.33	26.95	25.60
vinden	16.67	2.40	18.45
heel	20.00	1.20	17.86
gaan	13.33	21.56	15.48

Table 17: Top five most occurring words in positive text entries, scores indicate the percentage of text entries per class that contain the word

6.2.2 *Bigrams*

In this section, the usage of bigrams on the dataset after stopword removal and stemming is analyzed. Table 18, Table 19, and Table 20 show the top five most occurring bigrams in respectively negative, neutral, and positive text entries. Colors indicate the percentages displayed in the tables.

bigram	negative	neutral	positive
niet leuk	11.67	0.60	0.00
niet zo	11.67	0.00	0.00
vinden niet	8.33	1.20	0.00
heel moeilijk	5.00	0.00	0.00
buik pijn	5.00	0.00	0.00

Table 18: Top five most occurring bigrams in negative text entries, scores indicate the percentage of text entries per class that contain the word

bigram	negative	neutral	positive
we gaan	0.00	4.79	4.17
school we	0.00	4.19	1.10
we hebben	1.67	2.40	1.79
film kijken	0.00	2.40	0.60
pannenkoek eten	0.00	1.80	0.00

Table 19: Top five most occurring bigrams in neutral text entries, scores indicate the percentage of text entries per class that contain the word

Table 18, Table 19, and Table 20 show that bigrams do not overlap much between the tables. All bigrams occurring in the negative table are not present in the positive text entries, and vice versa. Bigrams occurring in the negative and positive tables indicate a strong sense of sentiment, especially compared to the unigrams occurring in the

bigram	negative	neutral	positive
heel leuk	0.00	0.00	8.93
ik vinden	0.00	0.00	8.33
leuk dag	0.00	0.00	7.14
super leuk	0.00	0.00	6.55
vinden leuk	0.00	0.00	5.95

Table 20: Top five most occurring bigrams in positive text entries, scores indicate the percentage of text entries per class that contain the word

previous subsection which are more general. Four out of the five bigrams in the positive table use the word 'leuk', which is apparently a strong indicator of positive sentiment as it is also present in 70.83% of the positive text entries.

6.2.3 *Trigrams*

In this section, the usage of trigrams on the dataset after stopword removal and stemming is analyzed. Table 21, Table 22, and Table 23 show the top five most occurring trigrams in respectively negative, neutral, and positive text entries. Colors indicate the percentages displayed in the tables.

trigram	negative	neutral	positive
vinden niet leuk	8.33	0.60	0.00
heel moeilijk vragen	3.33	0.00	0.00
niet helemaal goed	3.33	0.00	0.00
niet zo lekker	3.33	0.00	0.00
niet zo leuk	3.33	0.00	0.00

Table 21: Top five most occurring trigrams in negative text entries, scores indicate the percentage of text entries per class that contain the word

trigram	negative	neutral	positive
school we gaan	0.00	1.20	0.60
spelen vogels gaan	0.00	1.20	0.00
vogel spelen vogels	0.00	1.20	0.00
vogels gaan vliegen	0.00	1.20	0.00
niks minder gespoot	1.67	0.60	0.00

Table 22: Top five most occurring trigrams in neutral text entries, scores indicate the percentage of text entries per class that contain the word

trigram	negative	neutral	positive
vinden super leuk	0.00	0.00	4.17
ik vinden leuk	0.00	0.00	2.38
ik vinden super	0.00	0.00	2.38
best wel leuk	0.00	0.00	1.79
leuk dag hebben	0.00	0.00	1.79

Table 23: Top five most occurring trigrams in positive text entries, scores indicate the percentage of text entries per class that contain the word

Table 21, Table 22, and Table 23 show that trigrams do not occur often in the text entries. Table 21 on the negative text entries displays the trigrams the Sentiment Scorer from the previous chapter had trouble with, namely 'niet zo leuk', 'niet zo lekker', and 'niet helemaal goed'. Table 22 shows a lot of trigrams referencing birds ('vogel'). Because neutral text entries contain a large number of distinct words (Table 13), few trigrams occur in more than one entry. A few (1.20%) neutral entries refer to birds, and those topics are thereby the most occurring trigrams, as not many other trigrams exist that occur in more than one text entry.

6.3 TERM FREQUENCY (- INVERSE DOCUMENT FREQUENCY)

After applying the `CountVectorizer` [13] function, the resulting matrix consists of occurrence counts of the selected n-grams for every text entry. Another representation to use is *term frequency*, which normalizes the occurrence counts by dividing them by the total number of words in the text entry. Moreover, words that occur in many documents can be downscaled in order to make them less dominant [58]. An often used representation for doing this is using *term frequency - inverse document frequency (tf-idf)*.

The Scikit-Learn package for Python [60] has a function `TfidfTransformer` [71] for transforming the occurrence counts into either term frequencies or tf-idf. In this study, the usage of occurrence counts, term frequency and tf-idf is evaluated.

In this section, transformations are performed on the whole dataset. When training machine learning models, however, the transformations are first fit and applied on the training data, after which they are applied on the test data. Figure 7 shows the distribution of the often occurring variables 'leuk' and 'niet' for the three different representations: occurrence count (top), term frequency (middle), and tf-idf (bottom).

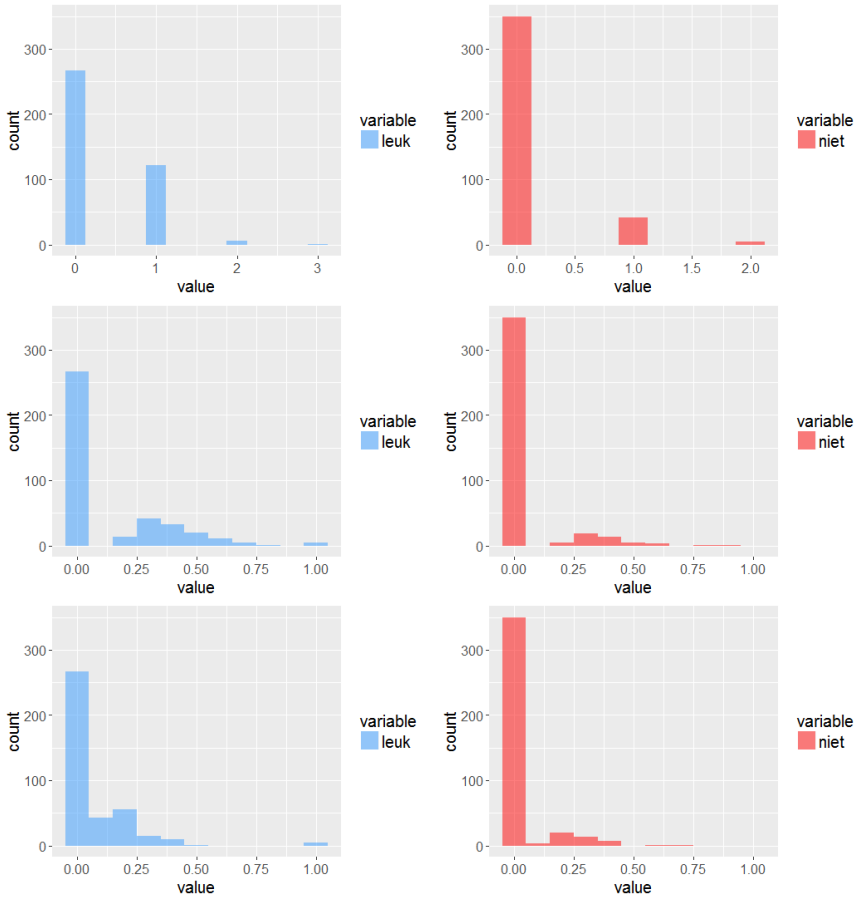

Figure 7: Histogram of the variables 'leuk' and 'niet' with an occurrence count (top), term frequency (middle), and tf-idf (bottom) representation

Figure 7 shows that the distribution becomes more spread after applying term frequencies. Moreover, the distributions shifts a bit to the left after applying tf-idf. This makes sense, as tf-idf downscales variables that occur in many text entries and both 'leuk' and 'niet' occur often in the dataset. A difference can be seen between 'leuk' and 'niet' though': 'niet' occurs less often than 'leuk' and is subsequently less downscaled.

The effect of using all these text representations is evaluated in the next chapter on machine learning models.

7 MACHINE LEARNING MODELS

In this chapter, the usage of different machine learning models is studied. In the first three sections, Logistic Regression (Section 7.1), Naive Bayes (Section 7.2) and Support Vector Machine (Section 7.3) are discussed. In Section 7.4, the three models are compared.

Because of the small size of the dataset, multiple training and test splits are used to evaluate the machine learning models. The approach of nested cross-validation is used. The data is split into ten folds with the `StratifiedKFold` [67] function of Scikit-Learn, where the folds are made by preserving the percentage of samples for each class. For $j = 1..10$, cross-validation is performed on all folds except j to determine the optimal parameters. A model is then trained on all folds except j with the found parameters, after which the prediction performance is estimated on j. The final estimated performance of a model is then the average and standard deviation of each of the 10 folds.

All models use the same division of data among the folds in both the outer and inner loop. All transformations on the data (e.g. term frequency) are fit and applied on the training data first, after which they are applied on the test data.

7.1 LOGISTIC REGRESSION

The model `LogisticRegression` [38] from the Scikit-Learn package for Python is used for modelling Logistic Regression. The *liblinear* solver is used, which uses one-versus-rest schemes for multinomial target variables. The model specific parameter which is optimized is the C parameter, which is the inverse of regularization strength λ. Hence, smaller values indicate stronger regularization. The values for C considered at every cross-validation step are: $\{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\} \times 10^{\{3,2,1,0,-1\}}$. It is checked that optimal C-values are not among the most extreme C-values considered. Hence, the search range for C-values is appropriate. Different types of text representations and the usage of a lasso (l1) or ridge (l2) penalty are studied as well. Resulting models are evaluated according to four measures: accuracy, mean absolute error (MAE), balanced accuracy, and F1-score.

Table 24 shows the accuracy scores and Table 25 shows the mean absolute errors for the corresponding models. The term 'occ' in the tables refers to occurrence counts and 'stopwords' refers to stopword removal. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 25 are differ-

ent, because the objective is to minimize MAE and to maximize the other three measures.

		occ	tf	tf-idf
lasso (l1)	text	85.28 (4.65)	85.51 (6.04)	86.29 (3.77)
	stopwords	86.06 (3.89)	86.59 (4.16)	86.80 (4.14)
	stemming	88.84 (3.88)	88.85 (4.31)	89.85 (2.65)
ridge (l2)	text	82.70 (6.23)	81.20 (6.94)	79.47 (4.57)
	stopwords	84.28 (5.30)	84.30 (4.21)	83.02 (5.79)
	stemming	84.54 (4.49)	85.78 (3.60)	84.29 (4.20)

Table 24: Results of Logistic Regression (mean and standard deviation), measured with accuracy $\times 100$

		occ	tf	tf-idf
lasso (l1)	text	16.49 (4.81)	16.01 (6.19)	15.48 (5.09)
	stopwords	16.45 (4.53)	15.67 (5.72)	14.97 (5.05)
	stemming	12.68 (4.95)	13.18 (5.67)	12.16 (3.37)
ridge (l2)	text	19.83 (6.41)	21.83 (6.86)	23.33 (4.94)
	stopwords	18.25 (6.28)	18.24 (5.17)	19.51 (6.78)
	stemming	17.99 (5.93)	16.75 (4.69)	17.99 (5.34)

Table 25: Results of Logistic Regression (mean and standard deviation), measured with mean absolute error $\times 100$

Tables 24 and 25 show that a lasso (l1) penalty performs better than a ridge (l2) penalty. The lasso (l1) penalty performs dimension reduction by setting some of the coefficients to zero. In this case, the dataset contains a lot of features and the dataset is sparse, and dimension reduction thus provides an improvement on the performance. The tables moreover show the following ordering: *stemming* > *stopword removal* > *text* representation. Hence, more intense morphological normalization leads to higher scores. By using morphological normalization, words with the same base are mapped to the same stem. This results in fewer distinct n-grams, which make it easier for the model to learn the pattern in the training set as less weights need to be learned. Tables 24 and 25 do not show a clear preference for occurrence counts, term frequencies or tf-idf.

Since the dataset is unbalanced, balanced accuracy and F1-score are also evaluated. Table 26 shows the balanced accuracy scores and Table 27 shows the F1-scores for the corresponding models. Colors indicate the scores.

Tables 26 and 27 show approximately the same performance pattern as in Tables 24 and 25. The values in Tables 26 and 27 have slightly higher standard deviations than in the other two tables. Higher standard deviations are probably caused by the increased influence

		occ	tf	tf-idf
lasso (l1)	text	78.44 (5.65)	78.62 (8.43)	79.96 (7.43)
	stopwords	79.40 (5.77)	81.23 (5.55)	82.11 (5.05)
	stemming	83.74 (6.86)	83.38 (7.49)	84.16 (5.74)
ridge (l2)	text	76.41 (7.01)	74.51 (7.50)	73.86 (5.13)
	stopwords	79.78 (6.63)	79.07 (6.78)	79.85 (6.55)
	stemming	79.28 (6.78)	80.62 (5.58)	80.15 (5.57)

Table 26: Results of Logistic Regression (mean and standard deviation), measured with balanced accuracy $\times 100$

		occ	tf	tf-idf
lasso (l1)	text	79.56 (5.27)	79.58 (7.91)	80.78 (6.30)
	stopwords	80.65 (6.21)	82.36 (5.91)	82.99 (5.02)
	stemming	84.68 (6.71)	84.29 (7.48)	85.38 (5.53)
ridge (l2)	text	77.30 (6.98)	75.60 (7.75)	75.02 (5.10)
	stopwords	80.15 (6.43)	79.67 (6.26)	80.17 (6.80)
	stemming	79.48 (6.09)	81.08 (5.01)	80.82 (5.45)

Table 27: Results of Logistic Regression (mean and standard deviation), measured with F1-score $\times 100$

of negative entries on the score. Balanced accuracy and F1-score are calculated by taking the mean of the accuracies for each class. Far less negative text entries are present in the dataset than positive and neutral entries. As a result, a wrongly classified negative text entry has approximately three times as much influence on the score as positive and neutral text entries. Consequently, the scores fluctuate more than when using normal accuracy or MAE.

In general, Logistic Regression shows the following patterns: *lasso* > *ridge*, and *stemming* > *stopword removal* > *text* representation. When using the lasso penalty and the stemming representation, a tf-idf representation is a best match. This combination scores highest on all measures. Hence, for comparing Logistic Regression to Naive Bayes and Support Vector Machine in Section 7.4, the combination lasso-stemming-tf-idf is used for Logistic Regression.

7.1.1 *Coefficient Evaluation*

Because Logistic Regression learns weights for predicting classes, an exploratory analysis of the coefficient values might give insights into the workings of the model. In this subsection, the coefficients of the lasso-stemming-tf-idf combination are evaluated. Because a tf-idf representation is used, all variables are scaled into the same range. Table 28 shows the variables and their coefficients with the highest sum

of absolute coefficient values across all one-versus-rest models, when training on the whole dataset with a lasso-stemming-tfidf combination.

variable	negative	neutral	positive
leuk	-4.80	-35.71	42.40
lekker		-20.90	21.26
goed		-19.17	19.07
niet zo	15.67	-3.85	-5.50
jammer	14.76	-10.03	
lastig	1.87	9.22	-13.49
gezellig	-0.02	-8.98	15.43
saai	14.15	-9.38	
hard		10.76	-12.53
niet leuk	13.76		-9.52

Table 28: Top ten variables and their coefficients with highest sum of absolute coefficient values for Logistic Regression

Table 28 shows that some variables indicate clear positive sentiments, such as 'leuk', 'lekker', 'goed', and 'gezellig'. Some variables indicate clear negative sentiments, such as 'jammer', 'lastig', 'saai', and 'niet leuk'. The words 'niet zo' probably indicate a negation of a positive sentiment. The feature set also contains the trigrams the Sentiment Scorer had trouble with, namely 'niet zo leuk', niet echt leuk', and 'niet helemaal goed'. This indicates that these trigrams are strong indicators of sentiment.

The usage of the lasso penalty also performs as a feature selection step. In this case, out of the 446 features, 315 have a coefficient of zero for all classes. Tables 24, 25, 26, and 27 have shown that using a lasso (l1) penalty leads to a better performance. The outcome of a lasso penalty can also be used as a feature selection step for the two upcoming models, Naive Bayes and Support Vector Machine. The function `SelectFromModel` [61] in Scikit-Learn can be used for this purpose. The function `SelectFromModel` asks for a model, such as a Logistic Regression with lasso penalty and default C parameter, and returns the features which have non-zero coefficients. The usage of this feature selection step for both Naive Bayes and Support Vector Machine is studied. This feature selection step will be called 'lasso' in the upcoming sections, where 'no lasso' indicates no feature selection step.

7.2 NAIVE BAYES

A Multinomial Naive Bayes model is implemented in Scikit-Learn with the function `MultinomialNB` [47]. A Multinomial model is chosen rather than a Bernoulli model because the Multinomial model can work with word counts and fractional counts (such as tf-idf) which are used in this thesis as well. McCallum and Nigam [43] also show an improved performance of using a Multinomial Naive Bayes over a Bernouilli Naive Bayes for text classification. The model specific parameter which is optimized is the smoothing parameter, which can be used to prevent from having zero probabilities for future computations. The values for smoothing considered at every cross-validation step are: $\{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\} \times 10^{\{0,-1,-2,-3\}}$.

Table 29 shows the accuracy scores and Table 30 shows the mean absolute errors for the corresponding models. The term 'occ' in the tables refers to occurrence counts. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 30 are different, because the objective is to minimize MAE and to maximize the other three measures.

		occ	tf	tf-idf
lasso	text	78.21 (7.99)	75.40 (7.20)	71.36 (7.02)
	stopwords	75.91 (6.59)	70.63 (6.95)	69.87 (7.06)
	stemming	74.40 (6.66)	72.38 (8.87)	71.12 (8.79)
no lasso	text	75.95 (3.57)	75.42 (3.75)	75.20 (4.79)
	stopwords	76.68 (5.45)	78.69 (6.29)	77.94 (5.95)
	stemming	79.23 (3.36)	80.22 (5.12)	76.68 (4.76)

Table 29: Results of Naive Bayes (mean and standard deviation), measured with accuracy $\times 100$

		occ	tf	tf-idf
lasso	text	25.62 (10.17)	29.66 (6.09)	35.98 (7.76)
	stopwords	27.90 (8.09)	35.20 (8.08)	36.72 (9.57)
	stemming	29.68 (8.29)	32.97 (11.30)	33.44 (9.58)
no lasso	text	29.64 (4.82)	29.92 (5.16)	29.89 (4.76)
	stopwords	28.62 (8.33)	25.35 (8.56)	26.08 (8.00)
	stemming	26.06 (6.08)	23.56 (6.86)	27.37 (5.67)

Table 30: Results of Naive Bayes (mean and standard deviation), measured with mean absolute error $\times 100$

Tables 29 and 30 show a different pattern than with Logistic Regression. Not a clear improvement of performance can be seen when using a feature selection step based on the lasso Logistic Regression model instead of no feature selection step. This can be explained be-

cause Logistic Regression and Naive Bayes have very different workings: Logistic Regression works with weights and Naive Bayes works with a probability distribution. Hence, features that are important for Logistic Regression are not necessarily the most useful for Naive Bayes.

The difference in performance of using feature selection with occurrence counts or tf/tf-idf can be explained by the number of features selected: an exploratory study indicates that lasso Logistic Regression sets less coefficients to zero for occurrence counts than for term frequencies or tf-idf. Hence, more features are selected with occurrence counts. This might be because the distribution of variables with occurrence counts differs from the distribution with tf or tf-idf, as has been shown in Section 6.3. A more detailed study on this aspect could be performed, but results for occurrence counts indicate that feature selection will probably not improve the performance vastly. Moreover, results in general indicate that it is not very likely that Naive Bayes will perform better than Logistic Regression. Hence, this aspect is not further evaluated.

Tables 31 shows the balanced accuracy scores and 32 shows the F1-scores for the corresponding models. Colors indicate the scores.

		occ	tf	tf-idf
lasso	text	74.33 (7.78)	66.38 (6.69)	63.21 (8.59)
	stopwords	74.27 (7.45)	65.87 (8.30)	65.61 (8.40)
	stemming	73.80 (6.75)	69.40 (10.52)	68.02 (8.67)
no lasso	text	67.89 (5.26)	68.17 (3.74)	67.30 (7.01)
	stopwords	70.60 (5.11)	74.32 (7.19)	72.31 (7.12)
	stemming	72.62 (5.47)	74.83 (6.79)	70.60 (7.52)

Table 31: Results of Naive Bayes (mean and standard deviation), measured with balanced accuracy $\times 100$

		occ	tf	tf-idf
lasso	text	74.74 (8.81)	67.76 (8.08)	64.25 (10.24)
	stopwords	74.11 (7.42)	66.99 (9.08)	66.59 (8.89)
	stemming	73.56 (6.78)	70.36 (10.89)	69.20 (8.68)
no lasso	text	68.58 (5.55)	69.54 (4.49)	68.15 (8.34)
	stopwords	71.71 (5.99)	75.33 (7.53)	73.37 (7.28)
	stemming	73.20 (5.42)	75.75 (6.78)	70.69 (8.55)

Table 32: Results of Naive Bayes (mean and standard deviation), measured with F1-score $\times 100$

Both Table 31 and 32 show the same pattern, which quite resembles the patterns seen in Table 29 and Table 30. The main difference is, however, that feature selection with occurrence counts is now clearly

among the best combinations. Rennie, Shih, Teevan, and Karger [59] indicate that Naive Bayes select poor weights for the decision boundary when the dataset is unbalanced. An under-studied bias effect of Naive Bayes is that it shrinks weights for classes with few training examples. In our case, negative text entries make up for approximately 15% of the dataset, whereas neutral and positive entries each make up for 42.5% of the dataset. Feature selection performed by lasso Logistic Regression thus may have the effect that it prevents from overfitting on neutral and positive text entries. Balanced accuracy and F1-score use a macro average over all the classes, thereby giving more weight to the negative text entries than normal accuracy. Hence, this may be an explanation that feature selection with occurrence count is among the best combinations for balanced accuracy and F1-score. The bias effect of Naive Bayes might also be an explanation of the decreased performance when using Naive Bayes over Logistic Regression.

The combination stemming-tf without feature selection can be considered the best combination for Naive Bayes: it scores highest on all four measures. Hence, for comparing Naive Bayes with the other models in Section 7.4, this combination is used.

7.3 SUPPORT VECTOR MACHINE

Support Vector Machines (SVM) are modeled with the function `SVC` [68] in the Scikit-Learn package for Python. Two kernels are evaluated: Linear kernel and RBF kernel. The model specific parameters which are optimized during cross-validation are the cost parameter C (both kernels) and the gamma value γ (RBF).

7.3.1 *Linear Kernel*

For the Linear kernel, the cost parameter C needs to be optimized. The values for C considered at every cross-validation step are the same as in Logistic Regression: $\{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\} \times 10^{\{3,2,1,0,-1\}}$. Tables 33, 34, 35, and 36 show the accuracy scores, mean absolute errors, balanced accuracy scores, and F1-scores for the corresponding models. The term 'occ' in the tables refers to occurrence counts. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 34 are different, because the objective is to minimize MAE and to maximize the other three measures.

Tables 33, 34, 35, and 36 all show the same pattern. More intense morphological normalization improves the performance. Moreover, the tables show that feature selection with lasso Logistic Regression improves upon the best performance of the SVM with Linear kernel. A clear preference for either occurrence counts, term frequencies of tf-idf is not clear.

		occ	tf	tf-idf
lasso	text	86.06 (7.42)	83.22 (7.57)	82.96 (7.40)
	stopwords	86.03 (5.56)	85.02 (5.62)	86.78 (5.24)
	stemming	90.11 (7.19)	86.53 (5.85)	87.80 (6.03)
no lasso	text	83.52 (4.11)	81.21 (6.29)	80.72 (5.89)
	stopwords	84.48 (7.12)	84.07 (5.59)	82.53 (6.54)
	stemming	85.99 (7.58)	87.57 (6.55)	85.03 (3.78)

Table 33: Results of SVM with Linear kernel (mean and standard deviation), measured with accuracy $\times 100$

		occ	tf	tf-idf
lasso	text	16.22 (5.51)	20.85 (8.30)	20.85 (6.87)
	stopwords	17.54 (6.67)	18.27 (5.07)	16.76 (5.73)
	stemming	12.44 (7.32)	17.54 (7.12)	15.74 (6.65)
no lasso	text	19.26 (4.01)	21.84 (6.91)	22.07 (6.44)
	stopwords	18.29 (6.73)	18.47 (4.77)	20.49 (6.14)
	stemming	17.06 (7.27)	15.24 (6.10)	18.02 (4.22)

Table 34: Results of SVM with Linear kernel (mean and standard deviation), measured with mean absolute error $\times 100$

		occ	tf	tf-idf
lasso	text	80.84 (8.15)	77.19 (7.24)	76.97 (7.06)
	stopwords	80.80 (5.84)	78.96 (5.72)	81.75 (4.50)
	stemming	86.14 (6.96)	81.55 (5.16)	83.62 (5.72)
no lasso	text	78.46 (4.08)	75.94 (7.03)	76.98 (6.63)
	stopwords	78.53 (7.05)	78.91 (6.09)	78.77 (6.70)
	stemming	79.72 (8.13)	83.79 (6.61)	81.44 (4.30)

Table 35: Results of SVM with Linear kernel (mean and standard deviation), measured with balanced accuracy $\times 100$

		occ	tf	tf-idf
lasso	text	80.84 (7.42)	77.77 (7.57)	78.05 (7.40)
	stopwords	81.14 (5.56)	79.92 (5.62)	82.95 (5.24)
	stemming	86.65 (7.19)	82.25 (5.85)	84.01 (6.03)
no lasso	text	78.70 (4.11)	76.32 (6.29)	77.15 (5.89)
	stopwords	79.00 (7.12)	79.05 (5.59)	78.99 (6.54)
	stemming	79.70 (7.58)	83.61 (6.55)	81.23 (3.78)

Table 36: Results of SVM with Linear kernel (mean and standard deviation), measured with F1-score $\times 100$

The combination stemming-occurrence counts with feature selection can be considered the best combination for Linear SVM: it scores

best on all four measures. Therefore, for comparing Linear SVM to the other models in Section 7.4, this combination is used.

7.3.2 *RBF Kernel*

For the RBF kernel, both the cost parameter C and the gamma value γ are optimized. The values for C are the same as previously used. The values for γ considered at every cross-validation step are: $2 \times 10^{\{-15,-13,-11,-9,-7,-5,-3,-1,1,3\}}$. Tables 37, 38, 39, and 40 show the accuracy scores, mean absolute errors, balanced accuracy scores and F1-scores for the corresponding models. The term 'occ' in the tables refers to occurrence counts. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 38 are different, because the objective is to minimize MAE and to maximize the other three measures.

		occ	tf	tf-idf
lasso	text	85.26 (4.75)	82.47 (5.55)	83.47 (6.47)
	stopwords	86.56 (2.92)	85.01 (3.14)	86.78 (4.56)
	stemming	90.09 (3.97)	85.50 (5.15)	88.05 (4.47)
no lasso	text	83.28 (2.85)	81.74 (4.41)	79.96 (5.09)
	stopwords	84.00 (3.70)	84.07 (3.51)	82.52 (5.48)
	stemming	86.26 (5.34)	87.32 (3.64)	84.78 (4.00)

Table 37: Results of SVM with RBF kernel (mean and standard deviation), measured with accuracy $\times 100$

		occ	tf	tf-idf
lasso	text	16.76 (5.08)	21.35 (7.46)	19.56 (5.54)
	stopwords	17.50 (4.61)	18.29 (5.31)	16.51 (5.91)
	stemming	12.46 (6.52)	17.82 (7.93)	14.73 (5.59)
no lasso	text	19.50 (3.23)	20.80 (5.38)	22.83 (5.65)
	stopwords	18.54 (4.28)	18.46 (4.88)	20.24 (6.10)
	stemming	16.28 (6.61)	14.97 (5.48)	18.26 (4.90)

Table 38: Results of RBF kernel (mean and standard deviation), measured with mean absolute error $\times 100$

Tables 37, 38, 39, and 40 all show the same pattern, which is similar to the SVM with a Linear kernel. More intense morphological normalization improves the performance. Moreover, feature selection with lasso Logistic Regression improves upon the best performance of the SVM with RBF kernel. A preference for or pattern in the performance on either occurrence counts, term frequencies of tf-idf is not clear.

		occ	tf	tf-idf
lasso	text	79.88 (6.51)	76.61 (7.01)	77.39 (6.48)
	stopwords	81.92 (4.72)	80.02 (4.07)	81.40 (6.90)
	stemming	85.77 (5.71)	80.05 (6.31)	84.18 (5.24)
no lasso	text	77.91 (4.57)	76.36 (6.74)	74.96 (6.18)
	stopwords	77.79 (5.49)	79.26 (6.55)	79.11 (6.95)
	stemming	79.93 (7.90)	83.24 (6.41)	81.26 (5.58)

Table 39: Results of SVM with RBF kernel (mean and standard deviation), measured with balanced accuracy $\times 100$

		occ	tf	tf-idf
lasso	text	80.28 (5.91)	77.30 (7.21)	78.66 (6.75)
	stopwords	81.94 (4.25)	80.56 (3.98)	82.32 (6.78)
	stemming	86.39 (5.79)	81.09 (6.56)	84.78 (5.35)
no lasso	text	78.18 (4.11)	76.71 (6.22)	75.69 (5.90)
	stopwords	78.08 (4.93)	79.18 (5.51)	79.06 (6.40)
	stemming	80.15 (7.42)	83.08 (5.96)	80.99 (5.44)

Table 40: Results of SVM with RBF kernel (mean and standard deviation), measured with F1-score $\times 100$

The combination stemming-occurrence counts with feature selection can be considered the best combination for RBF SVM: it scores best on all four measures. Therefore, for comparing RBF SVM to the other models in Section 7.4, this combination is used.

The RBF kernel does not show any improvement on performance when compared to the Linear kernel, even though the model is more complex than the Linear kernel. Therefore, a decision is made to not evaluate the Polynomial kernel, as it is even more complex than the other two kernels. A more complex model does not only take up more computing time, but is also more prone to overfitting, especially with a small dataset as is currently the case.

7.4 COMPARING MODELS

In this section, two studies are performed on the statistical significance of observed differences between the accuracy of predictions made by cross-validation in the outer loop by the models: first on differences in morphological normalization and secondly on differences in machine learning models. The statistical significance of observed differences between the accuracy of predictions made by all models is computed by comparing the number of correct predictions made by one model and incorrect by another, and vice versa, i.e. counting the number of wins and losses of each model against the other. These

are compared with a two-tailed binomial test (with the R function `pbinom`) against the null hypothesis that both classifiers have the same accuracy. If the algorithms would perform similarly, as assumed under the null hypothesis, each model should win approximately N/2 times where N is the number of cases on which the two models do not agree. This statistical test is also known as the sign test [16].

7.4.1 *Morphological Normalization*

In the previous sections, the pattern *stemming* > *stopword removal* > *text* can be seen from the results. In this section, this pattern is statistically evaluated. The best combinations for all four machine learning models include stemming. In order to statistically test the effect of morphological normalization, the predictions made by each machine learning models with their best combination (which always includes stemming) are compared to the same combination but then with stopword removal or no morphological normalization at all. Table 41 shows the results, where the resulting p-values which are significant at $\alpha = 0.05$ are indicated with an asterisk. The abbreviation 'stop' refers to stopword removal and 'stem' refers to stemming.

	text	stop	stem
text		.868	.016*
stop			.029*
stem			

Logistic Regression

	text	stop	stem
text		.104	.027*
stop			.362
stem			

Naive Bayes

	text	stop	stem
text		.999	.005**
stop			.004**
stem			

Linear SVM

	text	stop	stem
text		.486	< .001***
stop			< .001***
stem			

RBF SVM

Table 41: Statistical differences (reported with an asterisk if significant at $\alpha = 0.05$) in accuracy between different types of morphological normalization

Table 41 shows that using the stemming algorithm performs significantly different in accuracy from no morphological normalization for all four models. Table 41 also shows that using the stemming algorithm performs significantly different in accuracy from using only stopword removal for all models except Naive Bayes. Hence, it can be concluded that using stemming significantly improves the accuracy over using the other two types of morphological normalization. By using stemming, words with the same base are mapped to the same stem, such as 'ben' to 'zijn'. This results in fewer distinct n-grams,

which make it easier for the model to learn the pattern in the training set as less weights need to be learned. The difference in accuracy between stopword removal and no morphological normalization is not significant.

7.4.2 *Machine Learning Models*

All four machine learning models are compared to each other on the basis of their performance with their best combination. Table 42 shows an overview of the performance of all models according to all measures. For completeness, the performance of the Sentiment Scorer on each of the ten folds in the outer loop is calculated as well. The values in Table 42 are visualized in a histogram in Figure 8. Note that the goal is to minimize mean absolute error (MAE), whereas the goal is to maximize the other three measures.

model	acc.	bal. acc.	F1	MAE
Sent. Scorer	78.48 (7.16)	76.29 (7.56)	74.47 (7.56)	26.56 (9.09)
Log. Reg.	89.85 (2.65)	84.16 (5.74)	85.38 (5.53)	12.16 (3.37)
Naive Bayes	80.22 (5.12)	74.83 (6.79)	75.75 (6.78)	23.56 (6.86)
Linear SVM	90.11 (7.19)	86.14 (6.96)	86.65 (7.19)	12.44 (7.32)
RBF SVM	90.09 (3.97)	85.77 (5.71)	86.39 (5.79)	12.46 (6.52)

Table 42: Performance (×100) of the best combinations for each model

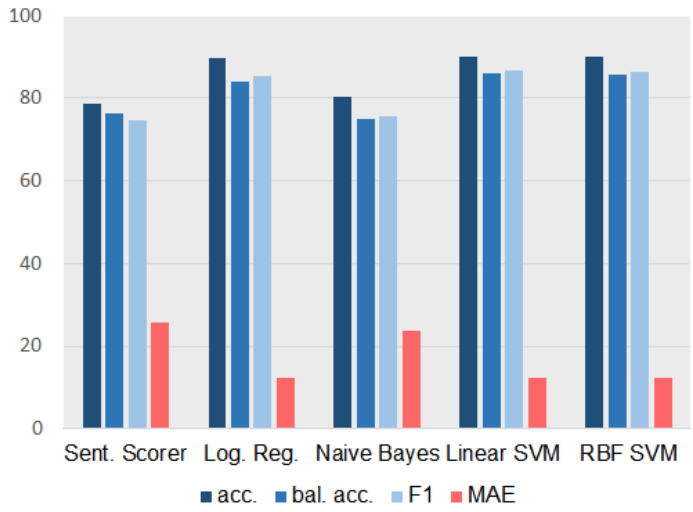

Figure 8: Visualized performance (×100) of the best combinations for each model

Table 42 and Figure 8 show a clear difference in performance between Sentiment Scoring and Naive Bayes on one hand, and Logistic Regression and the SVM's on the other hand. A study of the signifi-

cance of observed differences between the accuracy of predictions of all models is performed, as explained at the beginning of this section. Table 43 shows the results, where the resulting p-values are reported when the difference is significant at $\alpha = 0.05$. 'LSVM' refers to SVM with Linear kernel and 'RBFSVM' refers to SVM with RBF kernel.

	SS	LR	NB	LSVM	RBFSVM
SS		$< .001^{***}$		$< .001^{***}$	$< .001^{***}$
LR			$< .001^{***}$		
NB				$< .001^{***}$	$< .001^{***}$
LSVM					
RBFSVM					

Table 43: Statistical differences (reported with a p-value if significant at $\alpha = 0.05$) in accuracy between machine learning models

Table 43 shows that Sentiment Scoring and Naive Bayes perform significantly different in accuracy from Logistic Regression and the SVM's. Sentiment Scoring and Naive Bayes do not differ significantly, and neither do Logistic Regression and the SVM's. A closer look at the confusion matrices might still show the differences between them.

Tables 44 and 45 show the confusion matrix with predictions for all data entries made by cross-validation in the outer loop, for respectively Sentiment Scoring and Naive Bayes. Colors indicate the percentage of examples for each class which are classified correctly.

		predicted		
		negative	neutral	positive
actual	negative	41	8	11
	neutral	31	119	17
	positive	9	9	150

Table 44: Confusion matrix of Sentiment Scoring

		predicted		
		negative	neutral	positive
actual	negative	33	15	12
	neutral	8	132	27
	positive	3	13	152

Table 45: Confusion matrix of Naive Bayes

Tables 44 and 45 show that even though Sentiment Scoring and Naive Bayes are not statistically different, their predictions differ from each other. The bias effect [59] of Naive Bayes towards classes with

more text entries can be seen from its confusion matrix: the number of correctly classified negative text entries is relatively lower than for neutral and positive text entries. Almost half of the negative text entries are classified incorrect. This is also the main difference between Naive Bayes and Sentiment Scoring, as the latter classifies more negative text entries correctly (at the cost of neutral entries).

Tables 46, 47, and 48 show the confusion matrix with predictions for all data entries made by cross-validation in the outer loop, for respectively Logistic Regression, Linear SVM, and RBF SVM. Colors indicate the percentage of examples for each class which are classified correctly.

		predicted		
		negative	neutral	positive
actual	negative	38	15	7
	neutral	3	161	3
	positive	1	11	156

Table 46: Confusion matrix of Logistic Regression

		predicted		
		negative	neutral	positive
actual	negative	43	13	4
	neutral	6	158	3
	positive	6	7	155

Table 47: Confusion matrix of Linear SVM

		predicted		
		negative	neutral	positive
actual	negative	42	14	4
	neutral	5	159	2
	positive	6	7	155

Table 48: Confusion matrix of RBF SVM

Tables 46, 47, and 48 show that Logistic Regression and the SVM's perform quite similar. The main difference between Logistic Regression and Linear SVM is a slightly more accurate prediction for either positive and neutral text entries (Logistic Regression) or negative text entries (Linear SVM). RBF SVM is a sort of trade-off between the two.

Figure 9 shows the learning curves for Logistic Regression, Naive Bayes, Linear SVM and RBF SVM. The curves show the average accuracy on the test sets (y-axis) for an increasing random proportion of the dataset size (x-axis) on which nested cross-validation is performed.

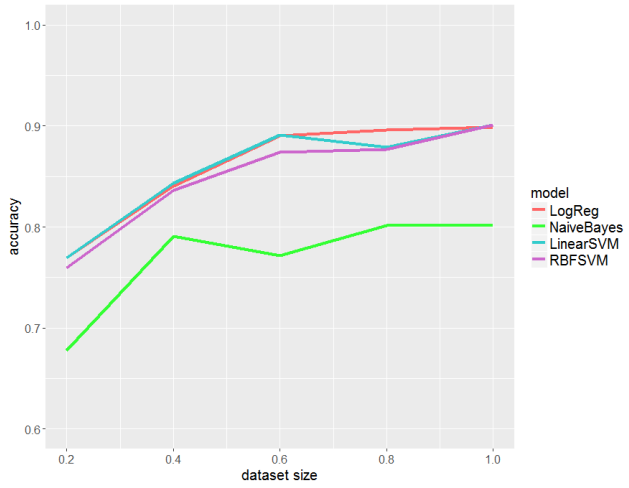

Figure 9: Learning curves for Logistic Regression, Naive Bayes, Linear SVM and RBF SVM

The learning curves show that it might be beneficial for Logistic Regression, Linear SVM and RBF SVM to obtain more data in order to improve performance, as the lines are still slightly increasing. The learning curve of Naive Bayes shows that obtaining more data will not help very much, as the line is converging.

All in all, the results from this chapter indicate that using Logistic Regression or Support Vector Machine machine learning models make significantly better predictions than a symbolic approach. Machine learning models can take complex negations such as 'niet zo leuk' into account, as can be seen in the feature sets of these models. Logistic Regression and Support Vector Machine are thereby advisable to use for the sentiment classification task of the robot.

8

INTRODUCING SEMANTIC NORMALIZATION

In the previous chapter, the results show that more intense morphological normalization yields a better performance. In addition to morphological normalization, a new algorithm for semantic normalization is designed. An effect of using this newly designed semantic normalization algorithm is expected, particularly because the dataset is sparse and contains many infrequent words. Because the dataset is relatively small, it is very likely that words occur in the test data that are not in the training data. In such a case, no weight is learned for such a word, and the machine learning model will have more difficulty predicting the target value as it cannot use these unseen words. Moreover, only words in the training data that occur twice or more often in the training data are taken into account during learning. The training algorithm thus does not take into account words that occur only once, thereby possibly missing informative training patterns. Leaving these 'unknown' words in the training and test data untreated thus might affect the classification performance.

In this chapter, these 'unknown' words (i.e. words that do not occur or occur only once in the training set) are mapped to 'known' words (i.e. words that occur twice or more often in the training data) by means of a newly designed algorithm. We call this semantic normalization, as it operates on the meaning of the words (semantics) rather than syntax. In this chapter, semantic normalization is first explained (Section 8.1), after which the usage of the new algorithm is evaluated (Section 8.2).

8.1 WORKINGS

For the new algorithm for semantic normalization, a synonym dictionary is first constructed. Synonyms are extracted from the Dutch website *http://synoniemen.net*, with the usage of Python libraries *urllib2* and *Beautiful Soup 4*, which are able to open (X)HTML websites and parse them. The website *synoniemen.net* only contains synonyms for verbs if they are in base form, for nouns if they are singular, and for adjectives if they are in base form. Therefore, the synonym replacement step is performed after stemming. For every of the 692 words in the dataset after stemming, synonyms were extracted from the *synoniemen.net* website. Synonyms are only considered for the dictionary if they occur in the vocabulary of our dataset. Then, 'unknown' words are mapped to 'known' words in two ways: 1) by using Part-Of-Speech-tags (POS-tags) and 2) by using Word2Vec.

8.1.1 *POS Normalization*

Because the synonyms are not ordered, a selection procedure needs to be designed in order to select an appropriate synonym for a word. One straightforward approach is to iterate over the synonyms for an 'unknown' word and select the first one with the same POS-tag, as some words have multiple interpretations and functions (such as the word 'naar'). The POS-tag can be found with the function `parse` from the Dutch module of *Pattern* [55]. Hence, an 'unknown' word is replaced with a 'known' word if the POS-tag of the synonym in the sentence is the same as the POS-tag of the original word in the sentence. Therefore, the following algorithm is constructed, where the blue part indicates the part which differs from the Word2Vec approach:

Input : training set with sentences t_i, other set with sentences o_j, synonym dictionary d
Output : other set with sentences o_j with words possibly replaced with their synonym
construct feature set f = all words in training set sentences t_i with minimum document frequency of 2
forall *sentences in the other set* o_j **do**
  forall *words* w_k *occurring in sentence* o_j **do**
    if w_k *is not in the feature set* f **then**
      forall *synonyms* m_l *for word* w_k *from the synonym dictionary* d*, which are in the feature set* f **do**
        o'_j = sentence with synonym m_l instead of w_k
        if *POS-tag*(m_l *in* o'_j) == *POS-tag*(w_k *in* o_j) **then**
          $o_j = o'_j$
          break (move to the next word in the sentence)
      end
  end
end

Algorithmus 4 : Synonym replacement with POS algorithm

The 'other set' in the algorithm can be both the training set and the test set. In the first case, the algorithm replaces words that occur only once in the training set with words that occur twice or more often. In the latter case, the algorithm replaces words in the test set that do not occur in the training set with words that occur twice or more often in the training set.

8.1.2 *Word2Vec Normalization*

Another method could be to select the synonym according to its Word2Vec similarity score with the original word. As explained in Chapter 3, Word2Vec transforms words from text entries into numerical vectors by using a two-layer neural network. These word vectors

can be used to find the most similar words by calculating cosine similarity between two word vectors, which indicates to what extent the two words occur in the same context. In this thesis, a Word2Vec model is trained on the BasiLex-corpus [5], consisting of 11.5 million words in texts written for young children. Hence, the text in the corpus is somewhat comparable to the words in the diary entries.

A Word2Vec model is trained with a window size of 5 and 200 dimensions. The window size is chosen because Levy and Goldberg [36] have shown that a larger window (size=5) tend to capture more topical and domain similarities, whereas a smaller window (size=2) tend to capture more functionally similar words. The dimension is chosen as an exploratory study has shown that no substantial differences occur in similarities in this research between similarity scores with 20 or 200 dimensions. Hence, no more dimensions are evaluated. Moreover, only words which occur 10 times or more in the BasiLex corpus are transformed into word vectors.

Figure 10 shows a t-Distributed Stochastic Neighbor Embedding (t-SNE) of the word vectors of frequently occurring words in our dataset and words which have a high sum of absolute coefficient values for Logistic Regression (Table 28). T-SNE [40] is a technique which can display high-dimensional datasets in two-dimensional space. It uses random walks on neighborhood graphs to extract the implicit structure of the dataset, which can then be used to find suitable visualizations of the dataset. The colors are done manually, in order to visualize the different observed groups.

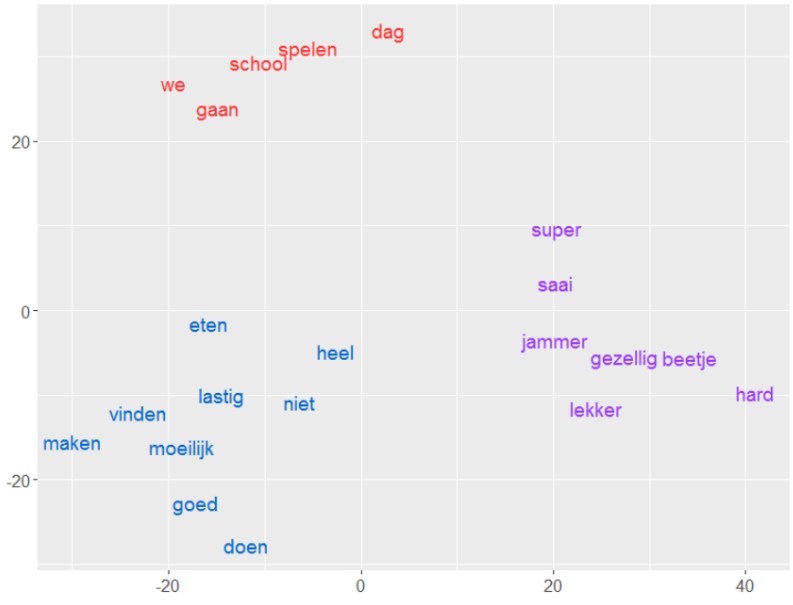

Figure 10: t-SNE of Word2Vec word vectors

Figure 10 shows some patterns. Most of the words in the purple group represent subjective opinions. The purple group contains both

positive ('lekker', 'gezellig') and negative ('jammer', 'saai') words, and moderator words ('beetje', 'super'). Most of the words in the red group represent daily activities ('dag'), indicated by phrases such as going to school ('gaan', 'school'). Most of the words in the blue group represent slightly more objective opinions, indicated by words such as 'goed', 'moeilijk', 'vinden' and 'lastig'.

Figure 10 also shows that antonyms can have high similarity scores. The similarity score for example between the word vectors for 'leuk' and 'stom' is 0.91 and the score for 'lekker' and 'vies' is 0.90, where the maximum score is 1.0. Hence, Word2Vec cannot be used as a standalone method for synonym replacement. Therefore, Word2Vec is used in collaboration with the synonym dictionary described in the previous section. Instead of iterating over all the synonyms and selecting the first one with the same POS-tag, however, the synonym is chosen with the highest Word2Vec similarity score. In this way, words which occur in the same context are mapped to each other, without the possibility of selecting antonyms. Hence, the strength of synonyms and of Word2Vec is combined. Words are, however, only replaced by one of their synonyms if their similarity score is 0.75 or higher. Therefore, the following algorithm is designed, where the blue part indicates the part which differs from the POS-tag approach:

Input : training set with sentences t_i, other set with sentences o_j, synonym dictionary d, Word2Vec model $w2v$
Output : other set with sentences o_j with words possibly replaced with their synonym
construct feature set f = all words in training set sentences t_i with minimum document frequency of 2
forall *sentences in the other set* o_j **do**
  forall *words* w_k *occurring in sentence* o_j **do**
    if w_k *is not in the feature set* f **then**
      $best\_score = 0$
      $best\_word = ''$
      forall *synonyms* m_l *for word* w_k *from the synonym dictionary* d, *which are in the feature set* f **do**
        sim_{lk} = similarity(w_k, m_l, $w2v$)
        if $sim_{kl} > best\_score$ **then**
          $best\_score = sim_{lk}$
          $best\_word = m_l$
      end
      if $best\_score \geq 0.75$ **then**
        o'_j = sentence with synonym *best_word* instead of w_k
        $o_j = o'_j$
  end
end

Algorithmus 5 : Synonym replacement with Word2Vec algorithm

The workings of this newly created algorithm are shown in Figure 11 and Figure 12, which show all synonyms for the words 'boos' and 'toernooi'. Colors indicate the ordering according to our Word2Vec model, where a darker color means higher similarity, and the selected synonym with the highest similarity score is underlined.

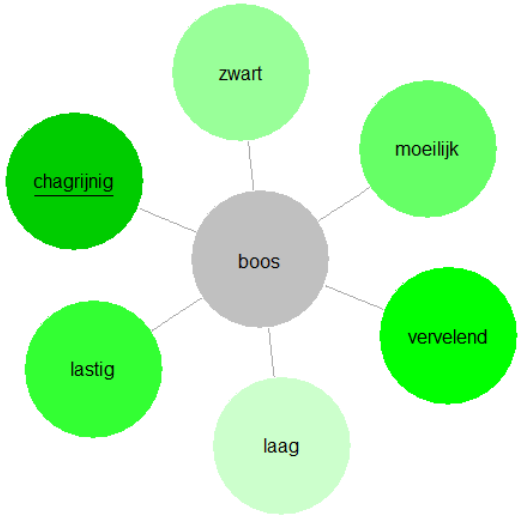

Figure 11: Synonyms of the word 'boos', colors indicate the ordering according to Word2Vec

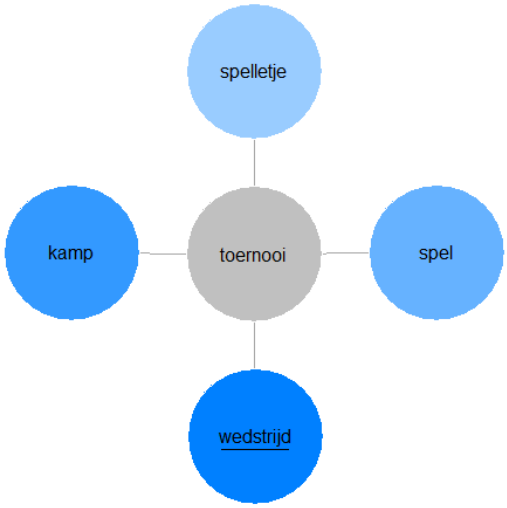

Figure 12: Synonyms of the word 'toernooi', colors indicate the ordering according to Word2Vec

Figure 11 and Figure 12 show that Word2Vec performs quite well on selecting the best synonym from a list of synonyms. Whether Word2Vec selection outperforms POS-tags or vice versa is studied in the next section.

8.2 RESULTS

In this section, the usage of the new algorithm for semantic normalization is evaluated. The approach is the same as in the previous chapter: all models use the same division of data among the folds in both the outer and inner loop of the nested cross-validation. Table 49 shows the average number of entries with a replaced word and the average number of words replaced in the folds in the outer loop, in both the train and the test set. The training sets consist approximately of 355.5 entries and the test sets of 39.5 entries.

	train		test	
	entries	words	entries	words
POS	80.50 (1.72)	107.30 (2.95)	12.10 (3.28)	17.20 (4.18)
W2V	61.70 (3.47)	78.40 (4.60)	9.10 (3.98)	11.80 (5.75)

Table 49: Average number (and standard deviation) of entries and words replaced by semantic normalization

Table 49 shows that the Word2Vec (W2V) approach is more selective, as it replaces less entries and less words with a synonym. The Word2Vec approach has the criterion that the similarity score should be at least 0.75 (out of 1.00). The criterion of the POS approach is that the word should have the same POS-tag. Hence, the similarity score criterion is more selective than the POS-tag criterion.

In this chapter, only Logistic Regression and SVM with Linear kernel are evaluated because Naive Bayes has shown to have significant lower performance than the other models and SVM with RBF kernel performs similar to SVM with Linear kernel but has a slightly lower performance. Four different text variations are evaluated: 1) POS normalization on both train and test set (tt-POS), 2) POS normalization on the test set only (test-POS), 3) Word2Vec normalization on both train and test set (tt-W2V), and 4) W2V normalization on test set only (test-W2V).

Section 8.2.1 describes the results with Logistic Regression, Section 8.2.2 evaluates the results with Linear SVM, and Section 8.2.3 provides a comparison to the results in the previous chapter.

8.2.1 *Logistic Regression*

The same implementation of the Logistic Regression model and the same parameters as used in the previous chapter are used in this chapter as well.

Tables 50, 51, 52, and 53 show the accuracy scores, mean absolute errors, balanced accuracy scores, and F1-scores for the corresponding models. The term 'occ' in the tables refers to occurrence counts. The

scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 51 are different, because the objective is to minimize MAE and to maximize the other three measures.

		occ	tf	tf-idf
lasso (l1)	tt-POS	89.12 (5.04)	90.17 (4.51)	89.65 (5.29)
	test-POS	89.63 (5.40)	89.65 (5.14)	89.89 (3.84)
	tt-W2V	89.88 (4.67)	89.11 (5.34)	90.65 (4.18)
	test-W2V	90.37 (4.89)	90.88 (4.25)	91.37 (3.49)
ridge (l2)	tt-POS	84.83 (4.61)	84.81 (3.72)	84.07 (5.09)
	test-POS	85.59 (4.72)	86.82 (3.36)	85.07 (5.38)
	tt-W2V	86.55 (4.01)	87.35 (4.38)	86.61 (4.30)
	test-W2V	86.32 (5.11)	87.07 (3.53)	85.31 (4.32)

Table 50: Results of Logistic Regression (mean and standard deviation) with semantic normalization, measured with accuracy ×100

		occ	tf	tf-idf
lasso (l1)	tt-POS	12.65 (5.00)	11.59 (5.34)	12.36 (6.03)
	test-POS	11.87 (6.49)	12.11 (6.05)	12.37 (4.33)
	tt-W2V	11.38 (4.93)	12.39 (5.10)	11.10 (4.72)
	test-W2V	11.14 (5.39)	10.88 (4.61)	10.91 (3.63)
ridge (l2)	tt-POS	17.19 (4.25)	17.47 (4.23)	18.20 (5.61)
	test-POS	16.69 (4.70)	15.96 (3.76)	17.45 (6.89)
	tt-W2V	15.49 (4.73)	14.93 (5.23)	15.16 (4.84)
	test-W2V	15.97 (5.60)	15.48 (3.95)	17.23 (5.27)

Table 51: Results of Logistic Regression (mean and standard deviation) with semantic normalization, measured with mean absolute error ×100

		occ	tf	tf-idf
lasso (l1)	tt-POS	85.02 (6.57)	85.85 (6.01)	85.80 (6.90)
	test-POS	85.08 (8.49)	85.08 (8.03)	84.92 (6.65)
	tt-W2V	85.98 (8.76)	85.74 (8.33)	86.94 (8.66)
	test-W2V	86.02 (8.54)	86.41 (7.93)	86.79 (6.98)
ridge (l2)	tt-POS	79.86 (6.28)	80.56 (6.37)	79.99 (7.37)
	test-POS	80.82 (6.23)	81.44 (5.89)	81.13 (6.75)
	tt-W2V	83.01 (7.27)	83.64 (8.54)	83.77 (7.93)
	test-W2V	82.46 (7.15)	82.70 (6.68)	82.38 (7.51)

Table 52: Results of Logistic Regression (mean and standard deviation) with semantic normalization, measured with balanced accuracy ×100

		occ	tf	tf-idf
lasso (l1)	tt-POS	86.09 (6.32)	87.32 (5.70)	87.10 (6.22)
	test-POS	85.75 (8.38)	85.82 (8.27)	85.80 (6.63)
	tt-W2V	86.44 (8.63)	86.16 (7.93)	87.25 (8.46)
	test-W2V	86.51 (8.00)	87.05 (7.70)	87.43 (6.70)
ridge (l2)	tt-POS	80.60 (5.78)	80.60 (5.85)	80.53 (7.03)
	test-POS	81.13 (5.84)	81.75 (5.34)	81.35 (6.54)
	tt-W2V	83.01 (6.65)	83.50 (7.65)	83.52 (7.41)
	test-W2V	82.44 (7.05)	82.57 (5.77)	81.84 (6.61)

Table 53: Results of Logistic Regression (mean and standard deviation) with semantic normalization, measured with F1-score $\times 100$

Tables 50, 51, 52, and 53 show that Word2Vec performs better than POS-tags in most cases. Word2Vec is a more advanced method of synonym selection than just selecting the first synonym found that has the same POS-tag, and it is shown that this more advanced method is paying off.

The results also show that when using Word2Vec, semantic normalization on the test set only (test-W2V) performs better in some cases than normalization on both the training and the test set (tt-W2V). In such a case, words that occur only once in the training set are possibly not useful to train on. Moreover, the improved performance of using a lasso penalty is shown in this section as well. A clear preference for either occurrence counts, term frequencies of tf-idf is not clear.

The combination test-W2V with tf-idf and lasso penalty can be considered the best combination for Logistic Regression: it scores best on accuracy and F1-score, and is runner-up on MAE and balanced accuracy. Therefore, for comparing Logistic Regression with semantic normalization to the other models in Section 8.2.3, the combination test-W2V with tf-idf and lasso penalty is used.

8.2.2 *Support Vector Machine*

The same implementation of the SVM model with Linear kernel and the same parameters as used in the previous chapter are used in this chapter as well.

Tables 54, 55, 56, and 57 show the accuracy scores, mean absolute errors, balanced accuracy scores, and F1-scores for the corresponding models. The term 'occ' in the tables refers to occurrence counts. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 55 are different, because the objective is to minimize MAE and to maximize the other three measures.

		occ	tf	tf-idf
lasso (l1)	tt-POS	88.85 (3.08)	87.33 (4.13)	86.83 (2.95)
	test-POS	88.87 (6.44)	86.82 (4.06)	88.60 (3.80)
	tt-W2V	89.88 (2.94)	88.27 (6.52)	90.33 (4.96)
	test-W2V	91.14 (5.97)	87.54 (5.33)	89.07 (5.50)
ridge (l2)	tt-POS	84.53 (2.85)	85.28 (4.35)	82.52 (3.13)
	test-POS	86.01 (5.52)	87.34 (4.76)	85.30 (3.62)
	tt-W2V	87.05 (5.80)	88.09 (4.63)	87.09 (5.16)
	test-W2V	86.74 (7.06)	88.08 (4.97)	85.78 (4.52)

Table 54: Results of SVM with Linear kernel (mean and standard deviation) with semantic normalization, measured with accuracy $\times 100$

		occ	tf	tf-idf
lasso (l1)	tt-POS	13.43 (2.35)	15.72 (4.43)	16.46 (2.57)
	test-POS	13.68 (7.98)	16.48 (5.03)	14.44 (4.83)
	tt-W2V	11.37 (3.87)	15.05 (9.13)	12.22 (6.74)
	test-W2V	10.89 (7.31)	16.03 (7.46)	13.97 (6.53)
ridge (l2)	tt-POS	17.77 (3.26)	17.76 (4.66)	20.27 (3.44)
	test-POS	16.53 (6.11)	15.72 (6.04)	17.75 (4.80)
	tt-W2V	14.74 (5.97)	14.44 (5.15)	15.97 (6.54)
	test-W2V	15.31 (8.53)	13.73 (6.22)	16.52 (5.69)

Table 55: Results of SVM with Linear kernel (mean and standard deviation) with semantic normalization, measured with mean absolute error $\times 100$

		occ	tf	tf-idf
lasso (l1)	tt-POS	84.82 (5.52)	82.18 (5.68)	83.21 (4.71)
	test-POS	85.54 (8.77)	82.50 (5.48)	85.33 (5.76)
	tt-W2V	85.96 (6.08)	85.81 (9.99)	88.50 (8.34)
	test-W2V	88.39 (9.08)	84.13 (6.93)	86.40 (7.05)
ridge (l2)	tt-POS	78.57 (5.01)	81.30 (6.90)	78.43 (6.13)
	test-POS	80.09 (8.50)	83.62 (8.46)	81.32 (6.19)
	tt-W2V	83.39 (8.47)	84.58 (8.02)	84.86 (7.98)
	test-W2V	81.38 (8.88)	84.91 (8.36)	83.11 (6.80)

Table 56: Results of SVM with Linear kernel (mean and standard deviation) with semantic normalization, measured with balanced accuracy $\times 100$

Tables 54, 55, 56, and 57 show a resemblance with the results of Logistic Regression with semantic normalization. Word2Vec performs better than POS-tags in most cases, and using semantic normalization on test set only performs better than both train and test set in some

		occ	tf	tf-idf
lasso (l1)	tt-POS	85.58 (4.73)	82.82 (5.50)	83.26 (3.84)
	test-POS	85.56 (8.57)	82.88 (5.67)	85.52 (5.39)
	tt-W2V	86.70 (5.35)	85.42 (10.03)	88.23 (7.93)
	test-W2V	88.30 (8.78)	84.19 (7.03)	86.18 (6.92)
ridge (l2)	tt-POS	78.98 (3.79)	81.13 (5.87)	77.93 (5.13)
	test-POS	79.98 (7.72)	83.16 (7.75)	81.10 (5.34)
	tt-W2V	83.37 (8.30)	84.25 (7.22)	84.27 (7.48)
	test-W2V	81.50 (9.02)	84.15 (7.71)	82.26 (6.11)

Table 57: Results of SVM with Linear kernel (mean and standard deviation) with semantic normalization, measured with F1-score ×100

cases. An improved performance of using a lasso penalty is clear as well. Moreover, a clear preference for either occurrence counts, term frequencies of tf-idf is not clear.

The combination test-W2V with occurrence counts and lasso penalty can be considered the best combination for Logistic Regression: it scores best on accuracy, MAE, and F1-score, and is runner-up on and balanced accuracy. Therefore, for comparing Linear SVM with semantic normalization to the other models in Section 8.2.3, the combination test-W2V with occurrence counts and lasso penalty is used.

8.2.3 *Comparing models*

The Logistic Regression (LR) and Linear SVM (LSVM) model without and with the new algorithm for semantic normalization (*sem*) are compared to each other on the basis of their performance with their best combination. Table 58 shows an overview of the performance of all models according to all measures. The values in Table 58 are visualized in a histogram in Figures 13 and 14.

model	acc.	bal. acc.	F1	MAE
LR	89.85 (2.65)	84.16 (5.74)	85.38 (5.53)	12.16 (3.37)
LR^{sem}	91.37 (3.49)	86.79 (6.98)	87.43 (6.70)	10.91 (3.63)
LSVM	90.11 (7.19)	86.14 (6.96)	86.65 (7.19)	12.44 (7.32)
$LSVM^{sem}$	91.14 (5.97)	88.39 (9.08)	88.30 (8.78)	10.89 (7.31)

Table 58: Performance (×100) of the best combinations for each model with and without semantic normalization

Table 58 and Figures 13 and 14 show a clear improvement in performance when using the new algorithm for semantic normalization. A study on the statistical significance of observed differences in the accuracy of predictions made by the models with and without semantic

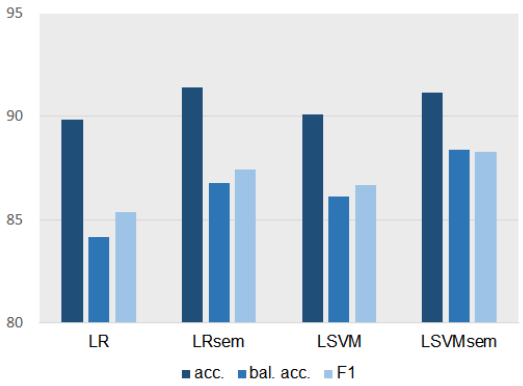

Figure 13: Visualized performance (×100) of the best combinations for each model with and without semantic normalization

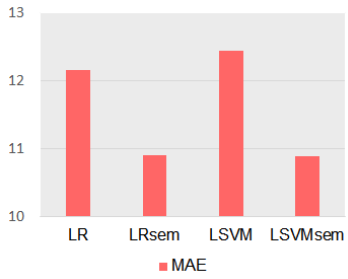

Figure 14: Visualized mean absolute error (×100) of the best combinations for each model with and without semantic normalization

normalization is performed, with the same two-tailed binomial test as explained in Section 7.4. The p-value for respectively Logistic Regression and Linear SVM is $p = .146$ and $p = .424$. Hence, the difference in the accuracy of predictions is not significant at $\alpha = 0.05$.

A second statistical test performed is the paired Wilcoxon Signed-Rank Test, which is a non-parametric test that ranks the performance of two classifiers and compares the ranks rather than the actual performance [16]. The test is performed in *R* with the function `wilcox.test`. The performance of using semantic normalization (with test-W2V) against only using stemming is compared on six scores: the average accuracy for occurrence counts, term frequencies, and tf-idf for both lasso and ridge penalty.

The p-value on the paired Wilcoxon Signed-Rank Test for respectively Logistic Regression and Linear SVM is $p = .031$ and $p = .036$. Hence, the difference in ranks is significant at $\alpha = 0.05$. Thus, it can be concluded that using semantic normalization performs consistently better across different combinations.

Tables 59 and 60 show the confusion matrix with predictions for all data entries made by cross-validation in the outer loop, for respectively Logistic Regression and Linear SVM, both with semantic

normalization. Colors indicate the percentage of examples for each class which are classified correctly.

		predicted		
		negative	neutral	positive
actual	negative	42	11	7
	neutral	4	159	4
	positive	2	6	160

Table 59: Confusion matrix of Logistic Regression with semantic normalization

		predicted		
		negative	neutral	positive
actual	negative	47	9	4
	neutral	9	153	5
	positive	4	4	160

Table 60: Confusion matrix of Linear SVM with semantic normalization

Tables 59 and 60 show that the main difference between the Logistic Regression model and the Linear SVM model are the predictions for negative and neutral text entries. Logistic Regression classifies more neutral entries correctly, while Linear SVM classifies more negative entries correctly. The same difference can also be seen in the confusion matrices of the models without semantic normalization in Chapter 7 (Tables 46 and 47). With the application in mind, one could argue that the predictions of Linear SVM are more useful for the robot. The task of the robot is to detect sentiments expressed by children, and a higher classification rate for negative entries contributes more to this goal than a higher classification rate for neutral entries.

Tables 59 and 60 also show an interesting aspect of using the new algorithm for semantic normalization as compared to the confusion matrices of the corresponding models without semantic normalization (Tables 46 and 47). When using the new semantic normalization algorithm, the accuracy for neutral entries slightly decreases, but the accuracy for negative and positive entries increases. Thus, with the application in mind, using the new semantic normalization algorithm has a real advantage.

All in all, the results for the new algorithm for semantic normalization on top of standard morphological normalization show that using this newly created step consistently improves the performance in the current study. The new semantic normalization algorithm is especially useful in this thesis, because the dataset is sparse and contains highly infrequent words.

9 MACHINE LEARNING ADAPTATIONS

In this chapter, two adaptations of machine learning models are evaluated. Section 8.1 discusses ordinal Logistic Regression, and in specific the proportional odds model. Section 8.2 studies the usage of using time-correlations within machine learning models.

9.1 ORDINAL LOGISTIC REGRESSION

The target variable can be interpreted to be at an ordinal level rather than a nominal level, as we can order the sentiment as negative-neutral-positive according to an increasing valence level. Hence, the classes $\{negative, neutral, positive\}$ can be ordered respectively as $\{1, 2, 3\}$. For this approach, ordinal Logistic Regression is studied as compared to regular Logistic Regression. The most commonly used ordinal Logistic Regression model is the proportional odds model, also called the cumulative logit mode. In the case of K different target values, only one weight β_j for each of the features X_j is present, instead of $K-1$ different weights $\beta_{k,j}$ for each feature X_j with traditional Logistic Regression [31].

The assumption of the ordinal Logistic Regression is that variables are monotonically related to the order negative-neutral-positive. Before training an ordinal model, it is tested whether this assumption applies to the variables in our dataset with the function `nominal_test` from the *ordinal* package [51] for R. The test performs an analysis of variance (anova) on the difference between an ordinal model that uses one weight β_j for each of the variables, and an ordinal model with one variable that uses weights that vary with the class (and all else equal): in our case weights for 'negative | neutral' and 'neutral | positive' [12]. The difference in log-likelihood between the two models is compared with a chi-square distribution, and a significant difference indicates that a variable is not monotonically related to the order. This test is applied to all variables, and results indicate that no variable is statistically non-monotonically related to the order with $\alpha = 0.05$. The variable with the lowest p-value is 'school' with $p = .319$. Hence, the assumption of ordinal Logistic Regression is met for all variables.

The usage of the proportional odds model is evaluated by comparing its performance against the results of regular Logistic Regression from the previous chapters. Moreover, the usage of semantic normalization for ordinal Logistic Regression is studied in this section as well. Consequently, four different models are compared to each other in this section: with or without semantic normalization, and regular

or ordinal Logistic Regression. The best representations for regular Logistic Regression from the previous chapters are used for ordinal Logistic Regression as well.

Both models had a higher performance with a lasso penalty than with a ridge penalty. Therefore, the `ordinal.gmifs` function from the *ordinalgmifs* package [52] is chosen instead of other ordinal functions such as the `polr` function from *MASS* package [50], as `ordinal.gmifs` performs penalization by using the generalized monotone incremental forward stagewise method, which behaves like a monotone version of the lasso [26]. The model is trained and tested with the same folds in the outer loop as all models in the previous chapters, and does not have an inner loop.

Tables 61, 62, 63, and 64 show the accuracy scores, mean absolute errors, balanced accuracy scores, and F1-scores for the corresponding models. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. The colors in Table 62 are different, because the objective is to minimize MAE and to maximize the other three measures.

	LR	LR^{sem}
regular	89.85 (2.65)	91.37 (3.49)
ordinal	85.78 (4.39)	87.28 (5.73)

Table 61: Performance of ordinal Logistic Regression (mean and standard deviation), measured with accuracy $\times 100$

	LR	LR^{sem}
regular	12.16 (3.37)	10.91 (3.63)
ordinal	14.98 (4.40)	13.47 (5.58)

Table 62: Performance of ordinal Logistic Regression (mean and standard deviation), measured with mean absolute error $\times 100$

	LR	LR^{sem}
regular	84.16 (5.74)	86.79 (6.98)
ordinal	78.47 (5.67)	82.15 (7.30)

Table 63: Performance of ordinal Logistic Regression (mean and standard deviation), measured with balanced accuracy $\times 100$

Tables 61, 62, 63, and 64 show that ordinal Logistic Regression models perform worse than regular Logistic Regression. An explanation for the lower performance may be that the proportional odds model is too restricted, as it only uses one weight for each of the features, whereas regular Logistic Regression has a one-vs-rest scheme and thus three weights for each feature. It might still be the case that a less restricted ordinal model could perform better.

	LR	LR^{sem}
regular	85.38 (5.53)	87.43 (6.70)
ordinal	80.14 (5.33)	83.23 (6.94)

Table 64: Performance of ordinal Logistic Regression (mean and standard deviation), measured with F1-score $\times 100$

A study on the statistical significance of observed differences in the accuracy of predictions made made by the models is performed by means of a two-tailed binominal test on the wins and losses of one model against another, as described in Section 7.4. The p-value for the test of the LR-regular model against the LR-ordinal model is $p = .002$, and for the LR^{sem}-regular model against the LR^{sem}-ordinal model is $p = .002$. Hence, the ordinal model performs significantly different in accuracy from the regular model at $\alpha = 0.05$. Thus, it can be concluded that the proportional odds model makes significantly worse predictions than a regular Logistic Regression model.

Moreover, the tables show that the performance increases when using the new algorithm for semantic normalization for ordinal Logistic Regression. Therefore, the usage of ordinal Logistic Regression with semantic normalization is evaluated from this point on.

Table 65 shows the coefficients for ordinal Logistic Regression for the variables with the highest sum of absolute coefficient values for regular Logistic Regression (from Table 28). For the proportional odds model, it holds that a positive slope indicates a tendency for a decreasing response level as the variable decreases. Hence, a positive coefficient indicates a negative monotone relationship.

variable	negative	neutral	positive	ordinal
leuk	-4.80	-35.71	42.40	-4.63
lekker		-20.90	21.26	-1.31
goed		-19.17	19.07	-1.39
niet zo	15.67	-3.85	-5.50	0.95
jammer	14.76	-10.03		0.40
lastig	1.87	9.22	-13.49	0.55
gezellig	-0.02	-8.98	15.43	-0.75
saai	14.15	-9.38		0.78
hard		10.76	-12.53	0.18
niet leuk	13.76		-9.52	1.59

Table 65: Top ten variables and their coefficients with highest sum of absolute coefficient values for regular Logistic Regression, with corresponding coefficients for ordinal Logistic Regression

The coefficients in Table 65 show that the ordinal coefficients are quite intuitive: all positive variables ('leuk', 'lekker', 'goed', 'gezellig') have a negative coefficient, and all negative variables ('niet zo', 'jammer', 'lastig', 'saai', 'niet leuk') have a positive coefficient.

Table 66 shows the ten variables with highest absolute coefficient for ordinal Logistic Regression.

variable	coefficient
leuk	-4.63
niet leuk	1.59
goed	-1.39
lekker	-1.31
lol	-0.96
niet zo	0.95
saai	0.78
gezellig	-0.75
ziek	0.68
helemaal goed	0.64

Table 66: Top ten variables with highest absolute coefficient value for ordinal Logistic Regression

Table 66 shows an overlap of seven variables with Table 65, indicating that similar variables are important for both regular Logistic Regression and ordinal Logistic Regression. The coefficients are again intuitive: positive variables have negative coefficients and vice versa. The bigram 'helemaal goed' also indicates a negative sentiment, as it is a subpart of the phrase 'niet helemaal goed' and the phrase 'helemaal goed' does not occur in positive text entries.

Table 67 shows the confusion matrix with predictions for all data entries made by cross-validation in the outer loop, for ordinal Logistic Regression with semantic normalization. Colors indicate the percentage of examples for each class which are classified correctly.

		predicted		
		negative	neutral	positive
actual	negative	38	19	3
	neutral	8	151	8
	positive	0	12	156

Table 67: Confusion matrix of ordinal Logistic Regression with semantic normalization

The confusion matrix shows that, when compared to regular Logistic Regression (Table 59), negative and positive text entries are more often classified as neutral than as respectively positive and negative. Hence, predictions made by ordinal Logistic Regression are more centered around the middle class.

9.2 TIME-CORRELATED MODELS

Another approach is to assume that the sentiment of a child is correlated in time, i.e. that a child's current sentiment has a relation with it's sentiment in the past couple of days. For this approach, only text entries for which it is known which child wrote the diary are taken into account. This is the data from the *a_note*, *e_note*, and *mike* datasets. If a diary was split into multiple diaries, only the first part of the split is taken into account in order to avoid interdependency between diaries. Because the model uses less training examples ($n = 264$) than in the previous experiments ($n = 395$), the results of a time-correlated approach cannot be compared to the results in the previous chapter.

All text entries of a specific child ordered according to the time the entries were expressed are called the history of a child. For each class $c \in \{negative, neutral, positive\}$, two dummy variables are constructed: whether the label of the text entry one step back in history (denoted as t_1) of the child is c , and whether the label of the text entry two steps back in history (denoted as t_2) of the child is c. Hence, six dummy variables are created in total.

The performance when using these dummy variables is evaluated in this section for both Logistic Regression (LR) and Linear SVM (LSVM) without or with semantic normalization (*sem*). For regular Logistic Regression and Linear SVM, the best performing combinations from Chapter 7 are used. For these models with semantic normalization, the best performing combinations from Chapter 8 are used. All these combinations include Lasso penalty selection. Nested cross-validation is used as described in the previous chapters, although now on a smaller dataset.

Tables 68, 69, 70, and 71 show the accuracy scores, mean absolute errors, balanced accuracy scores, and F1-scores for the corresponding models. The scores are multiplied with 100 for better readability of the table.

	LR	LR^{sem}	LSVM	LSVM^{sem}
regular	84.82 (5.50)	86.00 (6.45)	85.48 (5.70)	86.75 (6.78)
regular+t_1	85.13 (6.36)	87.09 (6.41)	85.48 (5.70)	87.13 (6.51)
regular+t_1+t_2	84.39 (6.07)	86.70 (6.43)	84.75 (6.90)	86.40 (7.36)

Table 68: Performance of time-correlated models (mean and standard deviation), measured with accuracy $\times 100$

	LR	LR^{sem}	LSVM	LSVM^{sem}
regular	17.91 (7.81)	17.44 (8.16)	16.85 (7.06)	16.30 (8.52)
regular+t_1	16.81 (7.93)	15.57 (7.91)	16.85 (7.06)	15.53 (7.77)
regular+t_1+t_2	17.94 (8.44)	15.95 (8.56)	17.56 (8.17)	16.24 (8.91)

Table 69: Performance of time-correlated models (mean and standard deviation), measured with mean absolute error $\times 100$

	LR	LR^{sem}	LSVM	LSVM^{sem}
regular	74.16 (9.65)	79.99 (9.03)	73.77 (11.18)	79.64 (9.62)
regular+t_1	72.98 (10.95)	79.95 (9.85)	73.77 (11.18)	79.95 (9.59)
regular+t_1+t_2	72.42 (10.86)	79.67 (9.80)	73.44 (12.41)	79.62 (10.07)

Table 70: Performance of time-correlated models (mean and standard deviation), measured with balanced accuracy $\times 100$

	LR	LR^{sem}	LSVM	LSVM^{sem}
regular	75.07 (11.11)	81.36 (8.42)	75.02 (13.08)	81.55 (9.19)
regular+t_1	73.98 (13.33)	82.00 (9.35)	74.87 (13.27)	82.15 (8.85)
regular+t_1+t_2	73.20 (13.18)	82.01 (9.41)	74.36 (13.95)	80.94 (9.91)

Table 71: Performance of time-correlated models (mean and standard deviation), measured with F1-score $\times 100$

Tables 68, 69, 70, and 71 show that the performance increases when using semantic normalization on this smaller dataset.

The results show a small preference for the regular+t_1 model, although the preference is not clear-cut. The regular+t_1 model scores best on accuracy and mean absolute error, but the difference with the other models is small. The balanced accuracy scores and F1-scores do not clearly show this preference, but also do not show a preference for any other model. All scores have a relatively high standard deviation, which indicates that the performance is unstable across different folds. An increased performance when taking time-correlation into account could thus merely be a coincidence, rather than that an actual time-correlation exists in the data.

Another explanation is that the machine learning models also take into account the words in the sentence and not only the time features t_1 and t_2. Hence, a time relation might still be present, but does not add any valuable information given the words occurring in the sentence. Or in other words, if a diary contains n-grams clearly indicating one sentiment, this is more informative than time features.

The accuracy when training a Logistic Regression model (with lasso penalty) on only the t_1 and t_2 dummy variables is 0.5038, compared to a majority baseline of 0.4228. This result might indicate that some time correlation is present in the dataset. This model, however, does not predict any entry as negative, hence the recall of negative senti-

ment is zero. Moreover, the Cohen's Kappa between the current label and the label one step back in time is only 0.15, and the Cohen's Kappa between the current label and the label two steps back in time is only 0.05. Thus, the presence of an actual time correlation rather than a lucky coincidence is still questionable.

Table 72 shows the coefficient values for the regular+t_1 Logistic Regression model.

variable	negative	neutral	positive
t_1-negative	0.64	-0.46	
t_1-neutral	-0.19	0.57	
t_1-positive	0.15	-0.20	0.90

Table 72: Coefficient values of time variables for regular+t_1 Logistic Regression model

Table 72 shows that most coefficients are quite intuitive: the coefficient for the class itself (i.e. t_1-negative for the negative class) always has a positive sign, i.e. it is most likely that a negative entry will occur after another negative entry. The table also shows that the lasso penalty has set the t_1-negative and t_1-neutral variable to zero for the positive class. The coefficients are, however, not very influential as they are relatively small. The maximum absolute coefficient value is for example 33.85 (for the word 'leuk' for the positive class).

Tables 68, 69, 70, and 71 have shown that overall the LSVM^{sem} regular+t_1 model performs best. Therefore, the confusion matrix of this model is compared to the confusion matrix of the LSVM^{sem} regular model. Tables 73 and 74 show these confusion matrices with predictions for all data entries made by cross-validation in the outer loop, for LSVM^{sem} respectively without and with time-correlation (t_1). Colors indicate the percentage of examples for each class which are classified correctly.

		predicted		
		negative	neutral	positive
actual	negative	19	8	6
	neutral	2	115	6
	positive	2	11	95

Table 73: Confusion matrix of LSVM^{sem} with the child-only dataset

Tables 73 and 74 show a very small difference in predictions, one neutral and one positive training example are both predicted respectively as negative (regular) or positive (t_1). Because of this small difference in confusion matrices and the small difference in performance as previously reported, one can conclude that using a time-correlation

		predicted		
		negative	neutral	positive
actual	negative	19	8	6
	neutral	1	115	7
	positive	1	11	96

Table 74: Confusion matrix of time-correlated (t_1) LSVM^{sem} with the child-only dataset

model does not have a machine learning model that does not take time-correlation into account.

10 FINAL STUDIES

In this chapter, two final remarks are made on the machine learning models studied in this thesis. In Section 10.1, the robustness of the machine learning algorithms across the different datasets is evaluated. In Section 10.2, the errors made by the best machine learning model are studied.

10.1 ROBUSTNESS ACROSS DATASETS

In this section, the robustness of the machine learning algorithms across the different datasets is studied. Therefore, instead of nested cross-validation, a leave-one-dataset-out study is performed. Hence, machine learning models are trained on five out of the six datasets, and are tested on the sixth dataset. The outer loop of nested cross-validation is thus replaced by a leave-one-dataset-out scheme, but the workings of the inner loop are similar to the previous chapters. Both Logistic Regression and Linear SVM with and without semantic normalization are studied. For regular Logistic Regression and Linear SVM, the best performing combinations from Chapter 7 are used. For the models with semantic normalization, the best performing combinations from Chapter 8 are used.

Tables 75, 76, 77, and 78 show the accuracy scores, mean absolute errors, balanced accuracy scores, and F1-scores for the corresponding models. The scores are multiplied with 100 for better readability of the table. Colors indicate the scores. For balanced accuracy and F1-score, no results are reported for the *a_note* and *vriendenboekje* datasets as these datasets are highly unbalanced. *A_note* contains only one negative text entry, and *vriendenboekje* does not contain any neutral or negative text entries at all (Table 5). Therefore, the balanced accuracy and F1-score for these two datasets present a distorted view, as balanced accuracy and F1-score are computed at a macro level.

Tables 75, 76, 77, and 78 show some patterns. For almost all datasets, using semantic normalization either improves or preserves the scores for performance.

Moreover, for some datasets it is more difficult to get good predictions than for others. This is most probably due to the unbalanced division of labels across the datasets (see also Table 5). Due to a lack of negative text entries to train on, predictions for negative text entries are more often incorrect. This can be seen with *year1_results*, which contains relatively more negative text entries than the other datasets, and scores worse than all other datasets on accuracy and MAE.

test set	LR	LR^{sem}	LSVM	LSVM^{sem}
sentiment_test	87.14	95.71	85.71	88.57
a_note	82.69	84.62	80.77	80.77
e_note	78.46	80.00	80.00	80.00
mike	81.82	84.24	84.85	86.67
vriendenboekje	84.62	84.62	92.31	92.31
year1_results	70.00	73.33	76.67	80.00

Table 75: Robustness across different datasets, measured with accuracy $\times 100$

test set	LR	LR^{sem}	LSVM	LSVM^{sem}
sentiment_test	17.14	7.14	20.00	17.14
a_note	17.31	15.38	19.23	19.23
e_note	24.62	20.00	21.54	20.00
mike	23.03	20.61	16.36	14.55
vriendenboekje	15.38	15.38	7.69	7.69
year1_results	43.33	36.67	26.67	23.33

Table 76: Robustness across different datasets, measured with mean absolute error $\times 100$

test set	LR	LR^{sem}	LSVM	LSVM^{sem}
sentiment_test	81.05	90.59	76.88	79.03
a_note				
e_note	69.82	71.58	72.60	71.58
mike	73.53	77.79	76.74	80.54
vriendenboekje				
year1_results	73.93	76.50	82.05	84.62

Table 77: Robustness across different datasets, measured with balanced accuracy $\times 100$

test set	LR	LR^{sem}	LSVM	LSVM^{sem}
sentiment_test	81.90	91.75	77.46	79.03
a_note				
e_note	72.23	74.66	75.40	74.66
mike	74.42	78.67	78.53	82.63
vriendenboekje				
year1_results	67.90	71.48	75.16	78.75

Table 78: Robustness across different datasets, measured with F1-score $\times 100$

Another trend in the results is that Logistic Regression performs clearly better than Linear SVM for the *sentiment_test* dataset, but Linear SVM performs clearly better than Logistic Regression for the

vriendenboekje and *year1_results* datasets. The *sentiment_test* dataset has relatively more neutral text entries than the *vriendenboekje* and *year1_results* datasets, and the *vriendenboekje* and *year1_results* datasets have relatively more negative and positive text entries. The confusion matrices in Chapters 7 and 8 have shown that Logistic Regression is more accurate in predicting neutral text entries than Linear SVM, and that Linear SVM is more accurate in predicting negative text entries. Since *year1_results* and *vriendenboekje* have relatively more negative text entries and *sentiment_test* has relatively more neutral text entries, this result is not surprising.

The highest performance is seen when *sentiment_test* is used as test set. *Sentiment_test* is the dataset which contains actual diary entries, whereas the other five datasets are diary-like text entries. The text entries from the *sentiment_test* thus contain the most informative words from which to extract sentiment. An advice for the developers of the robot is thus to collect more data like the entries in *sentiment_test*, as these are most informative for the robot.

10.2 ANALYSIS OF ERRORS

In Chapter 8, it is argued that the Linear SVM model with the new algorithm for semantic normalization (LSVM^{sem}) produces the most useful predictions for the robot. The accuracy of this model is 0.9114 (Table 58), and the confusion matrix of this model is shown in Table 60. The LSVM^{sem} model makes 35 errors in 395 predictions in the outer loop of nested cross-validation.

Table 81 in Appendix A.1 shows all the 35 errors the LSVM^{sem} model has made, and the actual and predicted labels. Out of the 35 incorrect labels predicted by the model, seven entries have a predicted label which was given to the text entry by one of the annotators. Thus, no full consensus was reached on these seven text entries during the annotation phase. These entries are colored in green in the appendix. These seven diaries are also represented here in Table 79, where 'act' indicates the actual label and 'pred' indicates the predicted label. The diaries are given an index number for reference.

Most text entries in Table 79 contain some contradictions, indicated by the word 'maar' (entries 1, 3, 5, 7). Giving a label to text entries with such a construction is a subjective task and depends on the interpretation of the annotator. Most of these diaries have been given a neutral label, but contain both a positive and negative sentiment. Moreover, diary entry 2 indicates illness, which was labelled as negative but can also be seen as a neutral sentiment. Some of the constructed labels in this study, and labels for sentiment data in general, are thus quite subjective and a ground truth is very difficult to establish.

nr	diary	act	pred
1	bso het was leuk maar het regent te hard	neut	pos
2	ik moest spugen	neg	neut
3	bso het was niet echt leuk maar ik ging toch iets doen	neg	pos
4	ik zit op 229 pfffffffff net gefietst en had niks minder gespoten en ook geen dextro jammer	neg	neut
5	boek lezen was leuk maar ook niet echt leuk ik had geen echte leuke boek	neut	pos
6	ziek nog blijven het gaat beter dan gisteren en mijn been doet iets minder pijn dan gisteren	neut	neg
7	boodschappen doen het was leuk maar een bloem vinden vond ik lastig	neut	pos

Table 79: Text entries wrongly classified by the LSVM^{sem} model, but with a label which was given to the text entry by one of the annotators

Seven errors out of the remaining 28 errors made by the algorithm are 'extreme', i.e. a positive text entry is predicted as negative or vice versa. These errors are colored in red in the appendix and displayed here in Table 80. The diaries are given an index number for reference.

nr	diary	act	pred
8	ik voel mij blij want ik hoef niet naar school	pos	neg
9	hij was hoog dus mijn humeur was niet goed	neg	pos
10	omdat ik mij duizelig voel	neg	pos
11	bij nizo is sinterklaas! het was fantastisch en er waren veel pepernoten maar door heel veel snoepen kom als je niet op let kom je hoog te zitten! dus toen zat ik op 229 en dat is natuurlijk heel hoog!	pos	neg
12	toen ik weer naar huis ging ben ik uitgegleden en dat was het minder leuke!	neg	pos
13	het was gezellig thuis dus zo is het	pos	neg
14	ik voel mij top	pos	neg

Table 80: Text entries which are extremely wrongly classified by the LSVM^{sem} model

The coefficients used for predicting the labels of these seven entries are studied, by looking at the coefficients of the fold each of the predictions belong to. In the folds entries 8 and 14 belong to, the word 'voelen' is associated with a negative sentiment, thereby classifying these entries as negative. In entry 10, this is probably vice versa, with

'voelen' associated with a positive sentiment. The wrong classification of diary 9 is probably due to the occurrence of 'goed', which indicates a positive feeling, whereas the bigram 'niet goed' is not present in the feature set. This might also be the case for entry 12, which contains the word 'leuke', whereas 'minder leuke' is not in the feature set. Entry 11 contains a lot of exaggerating terms, such as 'heel' and 'echt' (replacement for 'natuurlijk' by Word2Vec), which are associated with negative sentiments. The same holds for entry 13, which contains the word 'zo', which is also more associated with negative sentiments than positive.

The other 21 errors are a confusion between neutral and positive, or neutral and negative. The seven 'extreme' errors are most disturbing. For the other 21 errors it might be argued that humans sometimes also make small mistakes in conversations with others, for example due to noise.

To conclude, the accuracy of the algorithm is 91.14%. The classifications will most likely be accepted by the end-user in even more cases, considering the disagreement between the annotators and the target application where neutral-positive and neutral-negative confusion might not be a strong problem.

11 DISCUSSION AND CONCLUSION

11.1 DISCUSSION

Previous research has indicated that machine learning approaches are generally better than *symbolic* approaches at word sense disambiguation [56], capturing negation [53], and capturing context [8]. The results from this research are in line with previous research. A significantly higher accuracy is reached when using machine learning over sentiment scoring. Moreover, the sentiment scoring algorithm has shown to have difficulty with some negations and word sense disambiguation of some words. The machine learning models in this research use an n-gram representation with uni-, bi-, and trigrams, which enable the models to take complex negations such as 'niet zo leuk' into account, as also proposed by Leshed an Kaya [35].

Furthermore, earlier research done on Dutch text analysis [27][32] did not show a clear improvement when using stemming. Additionally, an unstable effect of using stemming was also found in research on English texts [14]. Most researchers [27] use the Snowball stemmer [63], for which the algorithm is incorporated in the *Natural Language Toolkit (NLTK)* documentation [48]. This type of stemming does not take different meanings of a word into account and does not look at the context of a word [41]. In our research, however, the positive effect of using stemming has been proven throughout. Results have shown that stemming significantly improves the accuracy over using stopword removal or no morphological normalization. In contrast to earlier research which uses the Snowball stemmer, a stemmer was built manually for this thesis with the functions `singularize`, `lemma`, and `predicative` from *Pattern* [62]. These functions do take meaning and context into account, and can map for example the verb 'ben' to the base form 'zijn', something the Snowball stemmer cannot do. One can conclude that using a stemmer which does take meaning into account rather than the Snowball stemmer may be more valuable.

In addition to previous research, this study also designed and evaluated a new algorithm for semantic normalization by using synonyms and Word2Vec to map highly infrequent words to similar more frequent words. The results in this thesis have shown that using semantic normalization performs consistently better than using only stemming. Future research could investigate the workings and the effect of this new semantic normalization step on other datasets, and could think of different methods for mapping infrequent words to more frequent words besides using synonyms and Word2Vec.

Naturally, it would be beneficial to collect more negative text entries, as the current dataset is highly unbalanced. Researchers [30] have, however, shown that people suppress negative emotions more often than positive emotions in public settings. In other words, people are not keen to express negative emotions to others. Hence, obtaining a more balanced dataset with relatively more negative text entries is a difficult task.

Earlier research has also indicated the difficulty of assigning sentiments to text entries. Alm, Roth, and Sproat [2] have researched machine learning techniques for emotion analysis of sentences from children stories. They have also indicated that agreement between annotators is sometimes difficult to achieve. In this thesis, it is argued that labelling the sentiment expressed in a text entry is not straightforward, due to the subjective nature of the annotation task. Future research could focus on how to deal with the subjectivity of this type of text entry.

Regarding the usage of the LSVM^{sem} algorithm in practice in a robot-child setting, results have shown that the performance is quite satisfying. The algorithm makes an extremely incorrect prediction in 1.17% of the cases. For all other cases it can be argued that it makes a prediction which could be made by a human as well, and in 91.14% of the cases the prediction is a perfect fit. Naturally, improving the accuracy by acquiring more data could improve the performance. But since the number of extremely incorrect predictions is quite low, the current algorithm can already be implemented in the robot.

11.2 RESEARCH QUESTIONS

The aim of this research is to classify the sentiment of a Dutch child's diary entry as correctly as possible by means of automatic text analysis, which has been researched addressing four main research questions:

1. To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of a sentiment scoring algorithm?

2. To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of machine learning models?

3. Does using the new algorithm for semantic normalization improve the performance?

4. Does using an ordinal model or a time-correlated model improve the performance?

The aim of the first research question is to study the workings of Sentiment Scorer from the Dutch *Pattern* module of CLiPS [55], which is a *symbolic* approach. The results show that incorporating not only the valence value but also the subjectivity value results in a reasonable performance. When analyzing the errors made by the Sentiment Scorer, it is revealed that the Sentiment Scorer has difficulty with some negations of positive sentiments ('niet zo leuk') and different functions and usages of the same word ('naar').

The aim of the second research question is to evaluate whether using a machine learning approach has an advantage over using the Sentiment Scorer. The results show that a Logistic Regression or a SVM model yield a significantly higher accuracy than the Sentiment Scorer. A machine learning approach is better at capturing context [8], and is better at capturing the negations which the Sentiment Scorer has difficulty with. Moreover, machine learning approaches are better at adapting to domain-specific word usages, as symbolic approaches are pre-trained on datasets which are often not fully similar to the datasets they are applied on. Additionally, results have shown that stemming significantly improves the accuracy over using stopword removal or no morphological normalization.

The aim of the third research question is to design and explore the possible advantage of using the new algorithm for semantic normalization as a next level of morphological normalization. The results show that using this new semantic normalization algorithm on top of stemming results in models with consistently better performance. Moreover, using the new semantic normalization algorithm leads to a higher accuracy for negative and positive text entries, although at the cost of the accuracy for neutral text entries. This is an advantage for the application whose goal it is to detect sentiments, in order to react to children interacting with the robot.

The aim of the fourth research question is to examine whether using adaptations of the regular machine learning models does improve the performance. The proportional odds model appears to be an unfortunate adaptation as it makes significantly worse predictions than a regular model. The proportional odds model is probably too restricted. Another less restricted ordinal model, however, might still perform better. The results of a time-correlated model also show no increased performance, possibly due to a lack of time correlation in the data or possibly because time features do not have additional information over the content of the text entries in this study.

11.3 CONCLUSION

The problem statement that guided the research of this thesis is: *To what extent is it possible to correctly classify the sentiment of a Dutch child's diary entry by means of automatic text analysis?* The results show that

this is best possible with a Support Vector Machine with a Linear kernel and Word2Vec semantic normalization on the test set, which then has an accuracy of 0.9114, a balanced accuracy of 0.8839, a F1-score of 0.8830, and a mean absolute error of 0.1089. The results for using an ordinal or time-correlated adaptation of a machine learning model show that such a model does not have an advantage over a regular machine learning model.

This thesis has also introduced a new algorithm for semantic normalization. The results of this thesis demonstrate a clear increased performance when using this new algorithm. Using semantic normalization performs consistently better than using only stemming in this thesis. Semantic normalization can be a viable addition to an automatic text analysis pipeline, especially for small and sparse datasets such as the one used here.

a

APPENDIX

A.1 ERRORS MADE BY LSVM^{sem}

Table 81 shows the errors made by the LSVM^{sem} model. The table shows the actual label ('act') and the label predicted by the LSVM^{sem} model ('pred'). The diary entries in green have a predicted label which was also given to the text entry by one of the annotators, and can thus be considered a 'human' mistake. The other diary entries have a predicted label which was not annotated by one of the annotators, of which the diary entries in red are extremely confused, i.e. a negative text entry is predicted as positive or a positive text entry is predicted as negative.

DIARY	ACT	PRED
ik voel mij blij want ik hoef niet naar school	pos	neg
bso het was leuk maar het regent te hard	neut	pos
vogel op halen het was heel lang en we hadden verkeerde huisnummer gebeld	neut	pos
mie mie gegeten	neut	neg
tijdens het afzwemmen ik zat rond de 30 hoog maar dat voel ik niet ik was wel wat stuiterig	neut	neg
ik moest spugen	neg	neut
maar het nadeel van deze dag was dat we verloren hadden	neg	neut
hij was hoog dus mijn humeur was niet goed	neg	pos
in het weekend ga ik zwemmen met mijn neefje!	neut	neg
ik ga zo naar de bruiloft 42 jaar getrouwd van mijn opa en oma maar eerst even douchen	neut	neg
ik verveel me	neg	neut
heb daar mijn school foto met mijn broertje gezien mijn broertje vind het niet leuk dat iemand zijn foto ziet dus heb ik er mijn eigen foto bij gedaan	neut	neg
ben chagrijnig ben moe	neg	neut

bso het was niet echt leuk maar ik ging toch iets doen	neg	pos
heel veel achter elkaar quiz gespeeld	neut	neg
ik ben naar school geweest maar vond het saai	neg	neut
mijn vriend heeft een ongelukje gehad dat was wel erg jammer	neg	neut
ik zit op 229 pfffffffff net gefietst en had niks minder gespoten en ook geen dextro jammer	neg	neut
ik was het super blije poppetje maar was vergeten in te vullen	pos	neut
school er was een feest van iemand die had zijn doel stil zijn behaald want hij praat overal door heen door de les door het voor lezen	neut	pos
boek lezen was leuk maar ook niet echt leuk ik had geen echte leuke boek	neut	pos
een beetje hoog en net nog een keer geprikt toen zat ik op 84	neut	neg
omdat ik mij duizelig voel	neg	pos
bij nizo is sinterklaas! het was fantastisch en er waren veel pepernoten maar door heel veel snoepen kom als je niet op let kom je hoog te zitten! dus toen zat ik op 229 en dat is natuurlijk heel hoog!	pos	neg
feestje van tom en chris nou er was veel te eten dus ik zat best wel hoog	neut	neg
ziek nog blijven het gaat beter dan gisteren en mijn been doet iets minder pijn dan gisteren	neut	neg
toen ik weer naar huis ging ben ik uitgegleden en dat was het minder leuke!	neg	pos
het was gezellig thuis dus zo is het	pos	neg
ik voel mij top	pos	neg
heb een hekel aan vandaag en mezelf krijg allemaal de schijt!	neg	neut
biologie we hoefden niet in ons werkboek	pos	neut

te schrijven yes		
thuis zijn ik was een beetje boos op mijn diabetes!	neg	neut
cool	pos	neut
taal het was interessant	pos	neut
boodschappen doen het was leuk maar een bloem vinden vond ik lastig	neut	pos

Table 81: Errors made by the LSVM^{sem} model

BIBLIOGRAPHY

[1] Alan Agresti. *Analysis of ordinal categorical data*. Vol. 656. John Wiley & Sons, 2010.

[2] Cecilia Ovesdotter Alm, Dan Roth, and Richard Sproat. "Emotions from text: machine learning for text-based emotion prediction." In: *Proceedings of the conference on human language technology and empirical methods in natural language processing*. Association for Computational Linguistics. 2005, pp. 579–586.

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining." In: *LREC*. Vol. 10. 2010, pp. 2200–2204.

[4] Ilaria Baroni, Marco Nalin, Paul Baxter, Clara Pozzi, Elettra Oleari, Alberto Sanna, and Tony Belpaeme. "What a robotic companion could do for a diabetic child." In: *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*. IEEE. 2014, pp. 936–941.

[5] *BasiLex-corpus*. URL: http://tst-centrale.org/nl/tst-materialen/corpora/basilex-corpus-detail (visited on 12/06/2016).

[6] Colin Bell and Kevin P Jones. "Towards everyday language information retrieval systems via minicomputers." In: *Journal of the American Society for information Science* 30.6 (1979), pp. 334–339.

[7] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer Science and Business Media, 2006.

[8] Erik Boiy and Marie-Francine Moens. "A machine learning approach to sentiment analysis in multilingual Web texts." In: *Information retrieval* 12.5 (2009), pp. 526–558.

[9] Alexander Budanitsky and Graeme Hirst. "Evaluating wordnet-based measures of lexical semantic relatedness." In: *Computational Linguistics* 32.1 (2006), pp. 13–47.

[10] Soumaya Chaffar and Diana Inkpen. "Using a heterogeneous dataset for emotion analysis in text." In: *Canadian Conference on Artificial Intelligence*. Springer. 2011, pp. 62–67.

[11] Aitao Chen. "Cross-language retrieval experiments at CLEF 2002." In: *Workshop of the Cross-Language Evaluation Forum for European Languages*. Springer. 2002, pp. 28–48.

[12] Rune Haubo B Christensen. *A Tutorial on fitting Cumulative Link Models with the ordinal Package*. 2015. URL: https://cran.r-project.org/web/packages/ordinal/vignettes/clm_tutorial.pdf (visited on 05/19/2017).

[13] *CountVectorizer*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (visited on 12/05/2016).

[14] Taner Danisman and Adil Alpkocak. "Feeler: Emotion classification of text using vector space model." In: *AISB 2008 Convention Communication, Interaction and Social Intelligence*. Vol. 2. 2008, pp. 53–59.

[15] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 233–240.

[16] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets." In: *Journal of Machine learning research* 7.Jan (2006), pp. 1–30.

[17] *Dutch stop word list*. URL: http://snowballstem.org/algorithms/dutch/stop.txt (visited on 12/05/2016).

[18] Paul Ekman. "Facial expression and emotion." In: *American psychologist* 48.4 (1993), p. 384.

[19] Andrea Esuli and Fabrizio Sebastiani. "Sentiwordnet: A publicly available lexical resource for opinion mining." In: *Proceedings of LREC*. Vol. 6. Citeseer. 2006, pp. 417–422.

[20] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

[21] Alice Frain and Sander Wubben. "SatiricLR: a Language Resource of Satirical News Articles." In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. 2016.

[22] Donna Freeborn, Tina Dyches, Susanne O Roper, and Barbara Mandleco. "Identifying challenges of living with type 1 diabetes: child and youth perspectives." In: *Journal of clinical nursing* 22.13-14 (2013), pp. 1890–1898.

[23] Nir Friedman, Dan Geiger, and Moises Goldszmidt. "Bayesian network classifiers." In: *Machine learning* 29.2-3 (1997), pp. 131–163.

[24] *Gensim 0.13.3*. URL: https://pypi.python.org/pypi/gensim (visited on 12/06/2016).

[25] Cyril Goutte and Eric Gaussier. "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation." In: *European Conference on Information Retrieval*. Springer. 2005, pp. 345–359.

[26] Trevor Hastie, Jonathan Taylor, Robert Tibshirani, Guenther Walther, et al. "Forward stagewise regression and the monotone lasso." In: *Electronic Journal of Statistics* 1 (2007), pp. 1–29.

[27] Vera Hollink, Jaap Kamps, Christof Monz, and Maarten De Rijke. "Monolingual document retrieval for European languages." In: *Information retrieval* 7.1-2 (2004), pp. 33–52.

[28] David A Hull et al. "Stemming algorithms: A case study for detailed evaluation." In: *JASIS* 47.1 (1996), pp. 70–84.

[29] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer Texts in Statistics, 2013.

[30] Alexander H Jordan, Benoît Monin, Carol S Dweck, Benjamin J Lovett, Oliver P John, and James J Gross. "Misery has more company than people think: Underestimating the prevalence of others' negative emotions." In: *Personality and Social Psychology Bulletin* 37.1 (2011), pp. 120–135.

[31] David G Kleinbaum and Mitchel Klein. "Ordinal logistic regression." In: *Logistic Regression*. Springer, 2010, pp. 463–488.

[32] Wessel Kraaij and Renée Pohlmann. "Viewing stemming as recall enhancement." In: *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1996, pp. 40–48.

[33] Ivana Kruijff-Korbayová, Elettra Oleari, et al. "Let's Be Friends: Perception of a Social Robotic Companion for children with T1DM." In: *Conference Proceedings New Friends 2015*.

[34] Ivana Kruijff-Korbayová, Elettra Oleari, Anahita Bagherzadhalimi, Francesca Sacchitelli, Bernd Kiefer, Stefania Racioppa, Clara Pozzi, and Alberto Sanna. "Young Users' Perception of a Social Robot Displaying Familiarity and Eliciting Disclosure." In: *International Conference on Social Robotics*. Springer. 2015, pp. 380–389.

[35] Gilly Leshed and Joseph'Jofish' Kaye. "Understanding how bloggers feel: recognizing affect in blog posts." In: *CHI'06 extended abstracts on Human factors in computing systems*. ACM. 2006, pp. 1019–1024.

[36] Omer Levy and Yoav Goldberg. "Dependency-Based Word Embeddings." In: *ACL (2)*. Citeseer. 2014, pp. 302–308.

[37] Bing Liu. *Sentiment analysis and opinion mining*. Morgan & Claypool Publishers, 2012.

[38] *Logistic Regression*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 12/12/2016).

[39] Rosemarijn Looije, Mark A Neerincx, and Johanna Peters. "How do diabetic children react on a social robot during multiple sessions in a hospital?" In: *Conference Proceedings New Friends 2015*.

[40] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[41] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[42] Marie Marshall, Bernie Carter, Karen Rose, and Ailsa Brotherton. "Living with type 1 diabetes: perceptions of children and their parents." In: *Journal of clinical nursing* 18.12 (2009), pp. 1703–1710.

[43] Andrew McCallum, Kamal Nigam, et al. "A comparison of event models for naive bayes text classification." In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. Citeseer. 1998, pp. 41–48.

[44] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[45] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013).

[46] Christof Monz and Maarten De Rijke. "Shallow morphological analysis in monolingual information retrieval for Dutch, German, and Italian." In: *Workshop of the Cross-Language Evaluation Forum for European Languages*. Springer. 2001, pp. 262–277.

[47] *Multinomial Naive Bayes*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (visited on 12/12/2016).

[48] *Natural Language Toolkit stemming package*. URL: http://www.nltk.org/api/nltk.stem.html (visited on 12/05/2016).

[49] Anouk Neerincx, Francesca Sacchitelli, et al. "Child's Culture-related Experiences with a Social Robot at Diabetes Camps." In: *The Eleventh ACM/IEEE International Conference on Human Robot Interation*. IEEE Press. 2016, pp. 485–486.

[50] *Package 'MASS'*. URL: https://cran.r-project.org/web/packages/MASS/MASS.pdf.

[51] *Package 'ordinal'*. URL: https://cran.r-project.org/web/packages/ordinal/ordinal.pdf (visited on 05/09/2017).

[52] *Package 'ordinalgmifs'*. URL: https://cran.r-project.org/web/packages/ordinalgmifs/ordinalgmifs.pdf (visited on 02/27/2017).

[53] Bo Pang and Lillian Lee. "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts." In: *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2004, p. 271.

[54] Bo Pang and Lillian Lee. "Opinion mining and sentiment analysis." In: *Foundations and trends in information retrieval* 2.1-2 (2008), pp. 1–135.

[55] *Pattern.nl*. URL: http://www.clips.ua.ac.be/pages/pattern-nl (visited on 12/07/2016).

[56] Ted Pedersen. "A decision tree of bigrams is an accurate predictor of word sense." In: *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*. Association for Computational Linguistics. 2001, pp. 1–8.

[57] Martin F Porter. "An algorithm for suffix stripping." In: *Program* 14.3 (1980), pp. 130–137.

[58] Rajendra Prasath, Philip O'Reilly, and T. Kathirvalavakumar. *Mining Intelligence and Knowledge Exploration*. Springer, 2014.

[59] Jason D Rennie, Lawrence Shih, Jaime Teevan, David R Karger, et al. "Tackling the poor assumptions of naive bayes text classifiers." In: *ICML*. Vol. 3. Washington DC. 2003, pp. 616–623.

[60] *Scikit-Learn*. URL: http://scikit-learn.org (visited on 12/05/2016).

[61] *SelectFromModel*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html (visited on 12/20/2016).

[62] Tom De Smedt and Walter Daelemans. "Pattern for python." In: *Journal of Machine Learning Research* 13.Jun (2012), pp. 2063–2067.

[63] *Snowball Stemmers*. URL: http://snowballstem.org/ (visited on 11/30/2016).

[64] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation." In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2006, pp. 1015–1021.

[65] James Stone. *Bayes' Rule: A Tutorial Introduction to Bayesian Analysis*. Sebtel Press, 2013.

[66] Carlo Strapparava, Alessandro Valitutti, and Oliviero Stock. "The affective weight of lexicon." In: *Proceedings of the fifth international conference on language resources and evaluation*. 2006, pp. 423–426.

[67] *StratifiedKFold*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html (visited on 02/14/2017).

[68] *Support Vector Classifier*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (visited on 12/12/2016).

[69] *Term Frequency and Inverted Document Frequency*. URL: http://disi.unitn.it/\~bernardi/Courses/DL/Slides_11_12/measures.pdf (visited on 12/06/2016).

[70] *Word2Vec*. URL: https://deeplearning4j.org/word2vec (visited on 12/06/2016).

[71] *Working With Text Data*. URL: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html (visited on 12/06/2016).