UNIVERSITY OF TWENTE.

Improving text representations for NLP from bags of words to strings of words

Luc Schoot Uiterkamp

Master thesis, august 2019

Supervisor:
Prof. Dr. Frank van der Velde

Second reader:
Dr. Martin Schmettow

Cognitive Psychology and Ergonomics
Faculty of behavioural and management sciences, University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

Natural language as we use it daily is often ambiguous, as words and sentences can have different meanings depending on the context in which they are used. Humans derive meaning from words using contextual information and word relations, which is something we do automatically. Most computer algorithms rely on rules and an absence of ambiguity to process information. Since the contextual cues we use to communicate meaning cannot be readily captured in rules, such algorithms cannot reliably interpret natural language. Machine learning algorithms enable computers to model in some way what human-made texts mean. In this thesis, a machine learning algorithm is used to estimate whether reviews of movies and tv shows taken from IMDB give a positive or negative appraisal of their respective movie or show. The reviews are represented as vectors of numbers to enable the machine learning algorithm to process the text. The method of representing these reviews that is currently standard, the 'bag of words' representation, represents the reviews in terms of how frequently each word in a predefined word list occurs in each review. The original arrangement of the words in the review is lost, as the representation is ordered in terms of the predefined word list. This limits the use of contextual information and word relations, which forms a barrier to interpreting what was meant. An alternative to the bag of words text representation is presented, which enables the original arrangement of the words in a text to be used. The alternative 'string of words' representation represents texts in terms of the original words of a text, in the original order. This differs from the bag of words representation, which represents the text in the order of the words that are known to the model. To find out if a machine learning classifier can be improved by the string of words representation, it is tested against the bag of words representation in a neural network that classifies movie and tv show reviews into two categories: positive and negative reviews. For both conditions in the comparison between word representations, the same neural network layers were used as a basis for the machine learning model.

To compare the two representations, performance and time measures were taken. The performance was measured as the MCC value, a combined confusion matrix measure, of the classifier model that was yielded using that representation. The impact of representation length of the string of words representation and the lengths of the classified reviews were assessed in an exploratory analysis.

The string of words representation outperforms the bag of words in time measures as well as performance measures but does come with its own limitations. The string of words representation performs best with texts that deviate little in length from the training texts and offers an advantage over the bag of words representation only if the text length is shorter than the number of words known to a machine learning model.

Keywords: Natural language processing, feature extraction, machine learning, deep learning.

Contents

Abstract 2

1 Introduction 7

2 Sentiment analysis 8

3 Machine learning 10

3.1 Vocabulary 11

3.2 Neural networks 13

3.3 Feature extraction 14

3.4 Bag of Words 14

3.4.1 Loss of word arrangement 15

3.4.2 Sparsity of data 16

3.5 String of words 18

3.5.1 Advantages of SOW 20

4 Neural network 22

4.1 Overview 22

4.2 Importing and normalization 24

4.3 Processing 27

4.4 Neural network layers 29

4.4.1 Embedding 30

4.4.2 LSTM 31

4.4.3 Fully connected 34

4.4.4 Regression 35

4.4.5 Updating weights 35

5 Research 35

5.1 Hypotheses 35

5.2 Dataset 36

5.3 Model implementation 37

5.3.1 Training variables 37

5.3.2 Program versions 37

5.3.3 Design 39
5.4 Data visualization 39
5.5 Model comparison methods 40
5.6 Comparison of measures 41
5.7 Exploratory analysis: Truncation length and text length 42

6 Results **43**
6.1 Data properties 43
6.1.1 Review lengths 43
6.1.2 Part of speech tags 43
6.2 Analysis results 44
6.2.1 Vocabulary type 44
6.2.2 MCC values 45
6.2.3 Time measures 45
6.3 Truncation length and text length 47

7 Discussion **51**
7.1 Vocabulary types 51
7.2 BOW vs SOW 52
7.3 Durations 52
7.4 Model performance 53
7.5 Truncation length 55
7.6 Assumptions 55
7.7 Limitations 56
7.7.1 Limitations of sentiment analysis in general 56
7.7.2 Limitations to this study 56
7.7.3 Limitations of the string of words model 57
7.8 Future research 57

8 Summary **59**
8.1 Conclusions 59

9 References **61**

Appendix A Background reading on probabilistic classifiers **64**

A.1 Generative classifiers 64

A.1.1 Bayesian classifiers 64

A.1.2 Training Bayesian classifiers 65

A.2 Discriminative classifiers 67

A.2.1 Logistic regression classifier 67

A.2.2 Training logistic regression classifiers 68

Appendix B Other word representation structures **70**

B.2 Tree-like structures 70

B.3 N-grams and n-grams as bag of words 71

B.4 N-grams as feature vector 72

Appendix C Worked out example of the model **74**

Appendix D R markdown **75**

Appendix E List of unused reviews **80**

Appendix F POS tags **81**

1 Introduction

Natural language is language as we use it in our day-to-day lives. It is often ambiguous; words and sentences can have different meanings, dependent on the context in which they are used (Jurafsky & Martin, 2017). Most humans have very little trouble with disambiguating natural language and interpreting it. We achieve this by using contextual cues and previously learned information.

This can be very simple: the sentence *"I saw her duck"* can be interpreted in two ways. Firstly, someone might have seen a duck belonging to a girl or woman or, secondly, someone might have seen a girl or woman bend or crouch down. If a person would be asked what this sentence meant, they would use contextual cues to figure out the meaning. If the girl or woman in question was about to be hit by, say, a paper airplane, this would indicate that she was crouching down. Contrarily, if the girl or woman is known to own a duck, the likelihood of the first meaning of the sentence would increase.

The use of contextual cues and previous knowledge to disambiguate sentences such as "I saw her duck" is something people do automatically. We learn language by doing, and get better at it by coming into contact with language around us. Even though we do learn some explicit grammar rules, intuition plays a big part in our language processing. Our procedural knowledge of language is very high, it is not difficult to recognize the two meanings of "I saw her duck" but it is more difficult to explain exactly why there are two meanings, and why one would be more likely than another. It requires more than just the grammatical rules of our language to do this.

Unlike humans, computers rely on rules and unambiguous information to process an input. Although our language is structured, in a grammatical sense, what is meant by a sentence may not be obvious from the grammatical rules alone, as could be seen in the duck example. For this reason, computer programs that are solely based on rules have a limited ability of determining the correct interpretation of a certain piece of natural language. Hence, to enable computer programs to extract meaning from natural language, algorithms that are not solely rule based are required.

In this thesis, the automated interpretation of one aspect of the meaning of a piece of natural language called 'sentiment analysis' is investigated. Reviews from movies and tv shows taken from IMDB[1] are classified as either positive or negative reviews, based on the

[1] https://www.imdb.com/

content of the review. A neural network based machine learning algorithm will be used to perform the sentiment analysis in this study. To enable the reviews to be processed by the machine learning model, they are represented as a vector. This is often done using the so called 'bag of words' method, with which it is not possible evaluate words in their original arrangement. This hinders the use of context to determine the meaning of a piece of text which is problematic because, as we have seen in the duck-example, context is very important for a correct interpretation.

An alternative to the bag of words representation is presented and tested. This 'string of words' representation is developed to operate more similarly to how humans process language and to improve the ability of machine learning models to take context into account.

The goal of the thesis is to find out which of the two word representations performs better in a number of conditions. Performance is measured as the ratio of true versus false predictions made by a machine learning model. The machine learning model will be tested with both the bag of words and string of words representations, and performance measures will be compared using a Bayesian linear model. Time measures are also taken into consideration and divided into preparation time, training time and total time.

In section 2 some background is given on the main topic of this thesis, which is sentiment analysis. Machine learning, described in section 3, will be used for this. The Bag of Words (BOW) word representation is explained in section 3.4 and some problems are identified. Previous solutions to these problems are identified in section B and the String of Words (SOW) solution is presented in section 3.5. Some examples of traditional classification algorithms are given in section A to give a background on machine learning classifiers, which relate to the machine learning model that stands at the core of this thesis, which is described in sections 4.1 and 4.4. The methods for evaluating the word representations are presented in section 5, the results in section 6.

2 Sentiment analysis

The "I saw her duck" example is a fairly simple and straightforward piece of natural language, where the correct interpretation out of two options needed to be determined. This is an example of language disambiguation: there are multiple interpretations for a certain text and it is not immediately obvious which one is correct.

One step further in extracting meaning from a text is 'sentiment analysis', the extraction of the author's feeling towards what is described in the text. Sentiment analysis tries to capture what an author is trying to get across using the natural language as a medium. This analysis can be done on multiple levels of abstraction, from identifying the emotional connotation of a sentence - for example answering the question 'does the author feel happy or sad regarding the subject of the text?' - to identifying whether the stance of an author regarding the subject of their text is positive or negative.

Sentiment analysis can be seen as an estimation task, in which sentiment can be estimated on a continuous scale ranging from for example positive to negative, or as a classification task, in which a text is classified as either 'positive' or 'negative'. In this thesis, a sentiment analysis task is performed but is limited to classifying stance toward the subject of a text into one of two classes: positive or negative. This analysis is carried out on reviews of movies and tv shows, generated by IMDB users. The dataset used was compiled by Maas et al. (2011) and made available for research in machine learning. The reviews have been selected by Maas et al. (2011) to represent extremes of the scale, only distinctly positive or distinctly negative reviews are used. Reviews were considered positive if they had a rating of seven out of ten or higher and negative if they had a rating of four out of ten or lower. Reviews with scores between four and seven were not included in the dataset. Because of this dichotomy in the dataset, a binary classification task was performed instead of a continuous scaling. Figure 1 shows an example of a positive review (1a) and a negative review (1b) as they are used in this thesis.

It is often difficult for us to make explicit why we interpret language in a certain way, even for ourselves. Humans process language in a procedural manner, not in an explicitly descriptive manner. This means that we have the ability to correctly apply language rules and interpret meaning in language correctly, but that it is difficult to specify exactly how we come to the correct conclusions. In the examples of movie reviews in figure 1 it is quite easy to determine in which class - positive or negative - both reviews belong. When trying to point out why, the positive and negative words in both reviews might be mentioned. However, looking at what words are used in both reviews might be misleading. The positive review features words like 'coaxed', 'reluctant' and 'wrong', seemingly negative words. The negative review on the other hand features words like 'legendary', 'lovely' and 'engages', words that seem positive on first hand. A rule based algorithm could be put

(a) Positive review example

I went and saw this movie last night after being coaxed to by a few friends of mine. I'll admit that I was reluctant to see it because from what I knew of Ashton Kutcher he was only able to do comedy. I was wrong. Kutcher played the character of Jake Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The sign of a good movie is that it can toy with our emotions. This one did exactly that. The entire theater (which was sold out) was overcome by laughter during the first half of the movie, and were moved to tears during the second half. While exiting the theater I not only saw many women in tears, but many full grown men as well, trying desperately not to let anyone see them crying. This movie was great, and I suggest that you go see it before you judge.

(b) Negative review example

Blake Edwards' legendary fiasco, begins to seem pointless after just 10 minutes. A combination of The Eagle Has Landed, Star!, Oh! What a Lovely War!, and Edwards' Pink Panther films, Darling Lili never engages the viewer; the aerial sequences, the musical numbers, the romance, the comedy, and the espionage are all ho hum. At what point is the viewer supposed to give a damn? This disaster wavers in tone, never decides what it wants to be, and apparently thinks it's a spoof, but it's pathetically and grindingly square. Old fashioned in the worst sense, audiences understandably stayed away in droves. It's awful. James Garner would have been a vast improvement over Hudson who is just cardboard, and he doesn't connect with Andrews and vice versa. And both Andrews and Hudson don't seem to have been let in on the joke and perform with a miscalculated earnestness. Blake Edwards' SOB isn't much more than OK, but it's the only good that ever came out of Darling Lili. The expensive and professional look of much of Darling Lili, only make what it's all lavished on even more difficult to bear. To quote Paramount chief Robert Evans, "24 million dollars worth of film and no picture".

Figure 1. Examples of a positive and negative reviews from the dataset from IMDB as collected by Maas et al. (2011)

on the wrong track by words like these, as a purely rule based algorithm relates positive words to positive reviews because positive words *are* often related to positive reviews. However, in some contexts they might indicate a negative tone, like the legendary fiasco in the negative review example. More than a rule based approach is required to take this context into account, and machine learning is used to do this.

3 Machine learning

As rule-based algorithms for interpreting language are not desirable, and since humans learn to interpret language in a procedural manner, it makes sense to employ a similar procedural strategy for letting computers understand language. This means letting a program learn by example instead of solely by rules, just like we do. These data-driven

algorithms are known as machine learning models and are mostly based on probability theory or neural networks.

Machine learning algorithms are programs that can derive parameters, called weights, from a dataset. This process is called training. Each entry in the IMDB dataset used in this thesis is labeled with the correct category. Training using such a labeled dataset is referred to as supervised learning. Training on unlabeled datasets, called unsupervised learning, is also possible but yields classes that are unlabeled as well, it resembles a principle component analysis in this way[2]. In this thesis, training refers to supervised training on a fully labeled dataset. There are multiple methods for the determination of weights, usually based on gradient descent or maximum likelihood estimation. In short, these two methods come down to iteratively adjusting weights until a minimum in error is reached or determining probability from frequency data. These methods are elaborated upon further in the additional background reading on probabilistic classifiers in appendix A. Before these weights can be learned however, a set of words needs to be learned, in order for the model to interpret them.

3.1 Vocabulary

The construction of a set of 'known' words is necessary for machine learning models to process natural language such as the IMDB reviews, because much like we as humans, the models need to learn words before it can interpret sentences. In the context of machine learning models 'known words' are words that were encountered during the training of the model and which have a known relation to the categories that the model learns about. These categories are positive and negative reviews in the case of the model used in this thesis. This set of known words is called the vocabulary of the model and is usually between a few hundred and a few thousand words. As these words occur in the training data, the model learns what relation each word has to the two categories. These are the words that will be used when reviews that were not included in the training are to be evaluated when the machine learning model is used in practice. The words that are known are used to determine the meaning of the new data.

The words in the vocabulary can be defined as either the most prevalent words, which

[2] See for example Englebienne (2016) or Jurafsky and Martin (2017) for more information on unsupervised training.

would be the most occurring words in the positive and negative reviews combined, or as a set of words that have the highest ability to distinguish between positive and negative reviews (Jurafsky & Martin, 2017).

Frequency. If the former definition is used, the most prevalent words are taken from all categories in the dataset. This means that the most used words from both the positive and negative film reviews form the vocabulary. This approach to building a vocabulary is called the 'frequency' approach and will be referred to as such from now on. The frequency approach forms a vocabulary which is ordered according to the frequency with which each word in the training corpus occurs. The length of the vocabulary is defined beforehand based on how long and complex the texts that are to be processed are, what size the training data set is, and what a balance between computational efficiency and classifier precision is desired.

Information gain. If the latter definition is used, the vocabulary is build up based on how well the words distinguish between positive and negative reviews. A formula such as the chi-square formula is used to define the distinguishing ability of different words. This vocabulary contains words that are indicative of one or the other category and is sorted by how indicative they are. Words that occur often in the positive reviews and little in the negative reviews are indicative of the positive reviews and will get a high chi-square score. Similarly, words that occur often in negative reviews and little in positive reviews will also get a high chi-square score. These words have a high differentiating ability for the used classes, which in turn should result in a more reliable classification. The vocabulary is composed of the list of words with the highest chi-square scores and is ordered by the chi square scores. Alternatively, the vocabulary may be pre-made with words that are known to be relevant or at least prevalent in the context that the model is applied to. Maas et al. (2011), who published the IMDB dataset, did not publish such a vocabulary.

Each type of vocabulary has its own advantages. The most frequently used words are likely to also occur in future data, even if that future data differs from the training data. The frequency vocabulary may be useful if the future data is expected to differ in word use from the training set. The chi-square vocabulary will contain words that have a higher differentiating capability and will likely form better predictors of classes (Englebienne, 2016). Because the way the vocabulary is assembled will likely influence the bag of words

and string of words representations differently, both the frequency and information gain approach to vocabulary building are tested in this thesis.

The machine learning model used in this thesis is a neural network, but before considering how the text representations are used in neural networks, an overview of more traditional[3] machine learning algorithms is given as background for the deep neural network type machine learning algorithm elaborated upon later.

3.2 Neural networks

Neural network classifiers are usually considered discriminative classifiers, like the logistic regression classifier mentioned above. While they do differ a lot in both complexity and performance, the similarities between logistic regression and neural networks can be found at the root of all neural networks: the perceptron. The first neural algorithms consisted of a single formula for combining a number of inputs with a number of weights, much like the logistic regression classifier. The weights are updated during training by adding the difference between the correct class and estimated class times a learning rate and the input, which resembles the logistic regression classifier but with a simpler loss function.

$$\theta_{t+1} = \theta_t + \eta(y - \hat{y})x \tag{1}$$

Perceptrons were inspired by how biological neurons function, giving certain outputs based on an array of inputs. Modern neural networks still adhere to this principle but use multiple layers that can perform different transformations on the input. Layers can be combined in different ways to make a neural network behave in a certain way. Weights are still updated during training using gradient descent, but because of the multitude of layers, much more weights need to be changed, making training a neural network much more difficult than training a simple perceptron. As with the logistic regression classifier, a loss function is calculated using the estimated and correct classes, which is used to assess the performance of the model. The weights in the model are updated iteratively through gradient descent until a minimum in loss is reached or a certain number of training steps has been reached.

The models that are elaborated upon in this section are only a selection of natural lan-

[3] Traditional in the sense that they are not multi-layered deep learning models.

guage processing algorithms that exist, and of the selected algorithms, there is much more to say than can be said in this thesis. For more estimation algorithms, more classification algorithms and a more mathematical background on the classification algorithms that are listed here, refer to Bishop (2011) and Jurafsky and Martin (2017).

Neural networks are considered to be more flexible than the traditional machine learning models and perform better in natural language classification tasks. This greater flexibility is due to the relatively many degrees of freedom in the functions they approximate compared to Bayesian and regression models. They are considered the state of the art in machine learning and for this reason a neural network will be used to compare the string of words representation with the bag of words one in this thesis.

3.3 Feature extraction

Before neural networks can be trained, they need something to train on. As can be seen in the examples in figure 1, it is not trivial to extract elements out of a text to be used to determine whether a review is positive or negative. These elements, called 'features', are dependent on the context in which they occur. The word 'legendary' on its own would indicate a positive sentiment, but 'legendary fiasco' indicates a negative sentiment. There are also longer and more ambiguous word relations, like the 'expensive and professional look', which sounds positive on its own, but is referred to by 'even more difficult to bare' which makes the whole sentence negative (examples from figure 1b).

Even though it might seem advantageous for the features to be as large as possible to capture these contexts, the likelihood of observing such a word combination in a new piece of text diminishes as the number of words in one feature increases. For this reason, each word is considered one feature for the models used in this thesis.

3.4 Bag of Words

To represent the reviews using these features in such a way that machine learning algorithms can process the reviews, the so called bag of words representation is often used. The bag of words representation represents texts in terms of the vocabulary, which is visualized in figure 2. The frequency with which each word in the vocabulary is encountered in review is logged. These frequency values are listed in a vector, where each dimension represents a word in the vocabulary. This vector is the bag of words representation of the

text. A more elaborate buildup can be found in appendix A.1.2, where this representation is used in context of a bayesian classifier.

The bag of words representation has a few important disadvantages, despite being used often for several types of machine learning and neural network classifiers.

3.4.1 Loss of word arrangement. Because each dimension in a bag of words vector corresponds to a word in the vocabulary, the original arrangement of the words in the review is lost. The reviews are treated as a bag of words, without taking into consideration how these words are arranged with regards to each other. This loss of word arrangement makes it impossible for a machine learning model to use contextual information to differentiate between the meaning of the word legendary on its own and the word legendary in 'legendary fiasco'. The bag of words representation, as the name implies, functions as if the words are independent of each other and bear no relation to one another. However, real sentences are not merely bags of words. The order in which words are arranged are important, for example the sentences *"The quick brown fox jumps over the lazy dog."* and *"The quick brown dog jumps over the lazy fox."* obviously mean different things but

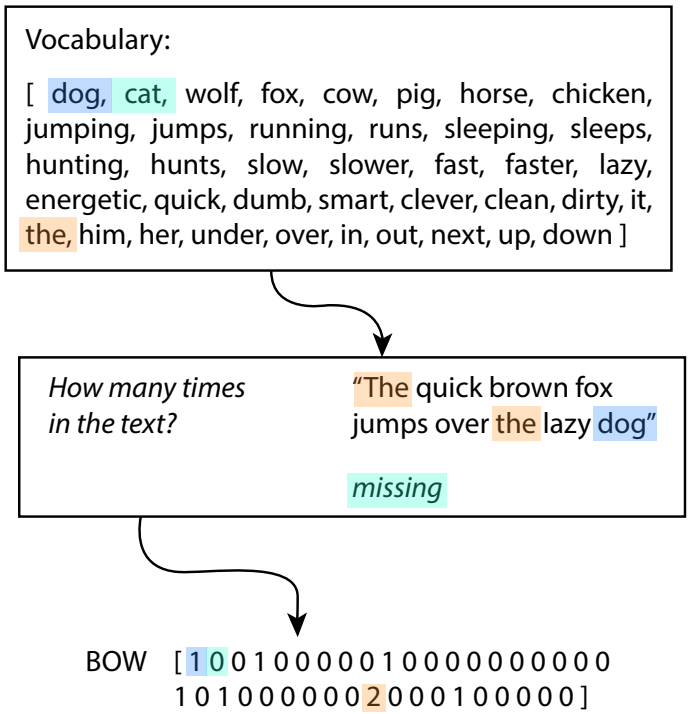


Figure 2. Example of the bag of word (BOW) representation. The BOW representation consists of frequency values for each word in vocabulary in the represented review.

these two sentences would be represented in exactly the same way in the bag of words representation.

In this example, if the words known to the model are:

[dog, cat, wolf, fox, cow, pig, horse, chicken, jumping, jumps, running, runs, sleeping, sleeps, hunting, hunts, slow, slower, fast, faster, lazy, energetic, quick, dumb, smart, clever, clean, dirty, it, the, him, her, under, over, in, out, next, up, down] (2)

then both the sentence *"The quick brown fox jumps over the lazy dog."* and *"The quick brown dog jumps over the lazy fox."* would be represented as:

```
[ 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 2 0 0
0 1 0 0 0 0 0 ]
```
(3)

The reason for this is that the bag of words representation is arranged in the order of the vocabulary of the model. The bag of words vector is a direct representation of the vocabulary, where each dimension in the vector represents the frequency with which each word in the vocabulary occurred in the review. The difference in word order makes a large difference to the interpretation of the sentence as this example shows. The impact of the absence of word order information on interpretability increases as texts get longer.

While the bag of words representation has yielded good results (Dreiseitl & Ohno-Machado, 2002), machine learning performance could be improved if the word representation keeps the arrangement of the words into account.

3.4.2 Sparsity of data. Apart from the loss of word arrangement information, the bag of words representation is very sparse, especially for shorter texts. Because each representation of a text has the length of the entire vocabulary, which could be thousands of words long, the representation of the text could be many times longer than the actual text. If, as most film reviews are, a text is only a few hundred words long, the vast majority of the bag of words representation will consist of zeros, as the vast majority of the words in the vocabulary will not be used.

These sparse representations are difficult and computationally expensive to train for machine learning models. Machine learning models train best on representations in which every dimension contributes to the meaning of what is represented, in other words, rep-

resentations with a high information density.

Embedding layers are often used to transform vectors into more useful representations by mapping each dimension of a vector to a vector itself, a word embedding vector. Lookup tables are used in which the word vectors are defined. These word vectors are created during training of the model and represent words that co-occur in the same contexts often as similar word vectors. The reason for this is that words that co-occur often are often similar in meaning. To give an example, if a review is represented by

$$[\ 1\ 0\ 2\ 0\ 0\ 1\ 0\ 0\ 0\ ] \tag{4}$$

and the following word vectors were generated during training:

$$\begin{array}{c|l} 0 & [\ 0.156784,\ 0.083149,\ 0.734812\ ] \\ 1 & [\ -0.006543,\ 0.0134820,\ 0.0370049\ ] \\ 2 & [\ -0.016843,\ -0.045467,\ 0.003598\ ] \end{array} \tag{5}$$

it would be mapped to the following embedding matrix:

$$\begin{array}{l} [\ [\ -0.006543,\ 0.0134820,\ 0.0370049\ ], \\ [\ 0.156784,\ 0.083149,\ 0.734812\ ], \\ [\ -0.016843,\ -0.045467,\ 0.003598\ ], \\ [\ 0.156784,\ 0.083149,\ 0.734812\ ], \\ [\ 0.156784,\ 0.083149,\ 0.734812\ ], \\ [\ -0.006543,\ 0.0134820,\ 0.0370049\ ], \\ [\ 0.156784,\ 0.083149,\ 0.734812\ ], \\ [\ 0.156784,\ 0.083149,\ 0.734812\ ], \\ [\ 0.156784,\ 0.083149,\ 0.734812\ ]\ ] \end{array} \tag{6}$$

For each value in an n-dimensional vector, n being 9 the example above, an n x m matrix is created. m depends on the output dimension parameter that is given for the embedding layer and usually vary between 32 and 128, although they may be larger as is the case in the Word2Vec embedding which has an output dimension of 300 (Abadi et al., 2015). The point of an embedding layer is to map words into a vector space in which words

that appear in close proximity to each other, appear in close proximity in the vector space. This increases the information that is captured in sparse representations. More information how this process works can be found in section 4.4.

Pre-trained machine learning algorithms such as Word2Vec are in essence a pre-made word embeddings and therefore yield a similar result to using an embedding layer. If, like Word2Vec, this embedding is trained on millions of sentences, it can become quite capable of capturing closeness in meaning in terms of closeness in vector space.

While these methods do improve the performance of machine learning models, the amount of data that is used to represent small texts is disproportionate to the size of the text. For example if a vocabulary contains 1000 words, a representation using the Word2Vec embedding would yield a 1000 by 300 matrix to represent a text, as the Word2Vec model uses 300 features per word. If a text is only a few hundred words long, as is the case with the texts from figure 1 that are used in this thesis, the representation is not very efficient. On top of that, making the representation less sparse does not solve the initial problem: the word arrangement information is still lost. Embedding layers and Word2Vec therefore do not offer a solution to the problem at hand, which is the inability of machine learning algorithms to effectively use word order to determine contextual information.

There have been attempts to retain the original word arrangement when processing natural language. They turn out to be either not very generalizable, not very effective, or not very efficient. Two general types of proposed solutions are elaborated upon in appendix B.

3.5 String of words

In order to enable machine learning models to utilize contextual information, a text representation which retains the original arrangement of the words in the reviews is developed in this thesis. Rather than indicating what the frequency of each word in the vocabulary is, it indicates the corresponding vocabulary item for each word in the review. This process is visualized in figure 3. The 'string of words' (SOW) representation captures what words were mentioned *where* in the text, which enables machine learning models to make use of contextual cues. The string of words approach is more similar to how humans process text, regarding it as a string of words belonging together in a certain arrangement, rather than a bag of individual words. In the string of words representation,

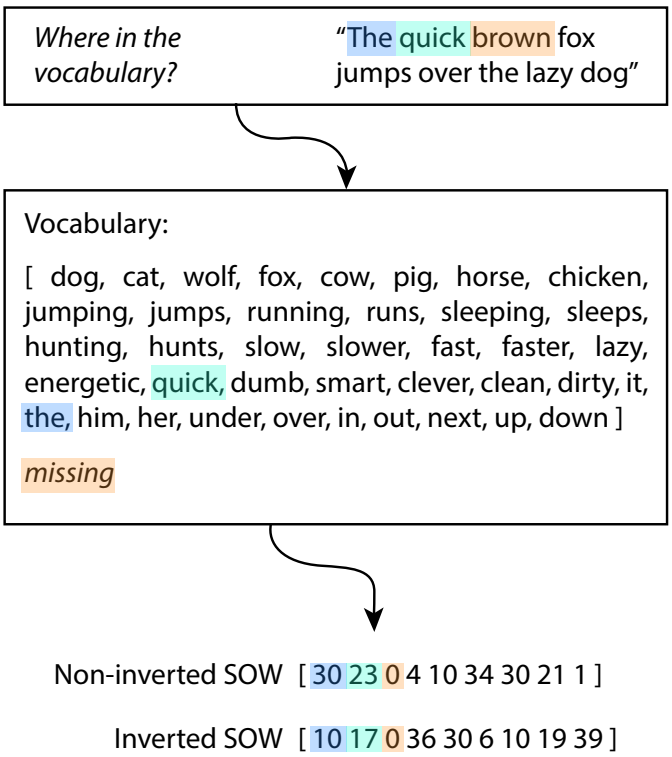


Figure 3. Example of the string of words representation (SOW). The SOW representation consists of references to the vocabulary in the original order of the review. These references are the index (starting at 0) for the non-inverted SOW representation and the vocabulary size minus the index for the inverted SOW representation.

each review is represented by one vector, just like with the bag of words representation, but each dimension in the vector corresponds to the word in the piece of text instead of a word in the vocabulary.

The value of each dimension is defined by the index of the represented word in the vocabulary, instead of the frequency of each word from the vocabulary as is the case in the bag of words representation. Figure 3 shows this (non-inverted) string of words method. The index starts at 1 (*dog* in figure 3) and ends at 39 (*down*). The number 0 is reserved for missing words (*brown*). Since words ranked higher in the vocabulary have a lower value for their index, this means that words that are missing (like *brown*) are represented in the representation by a value lower than the words highest in the vocabulary. This implies that the words that are missing from the vocabulary were more distinguishing (if the chi square vocabulary is used) or more frequent (if the frequency vocabulary is used) than the most distinguishing or frequent words.

To solve this issue, the vector is not build up following the index from the vocabulary

directly but using the inverse. This is achieved by using the length of the vocabulary minus the index of the word as values. If the index is taken to start at 0 instead of 1, this results in the very first word in the vocabulary being represented by the length of vocabulary and the very last word by the value 1. The value 0 can still be used for words that are not featured in the vocabulary but it now implies that missing words are less distinguishing or frequent than the least distinguishing or frequent words in the vocabulary. This inverted SOW representation is also shown in figure 3. Since this inverted representation captures the meaning of missing words better than the non-inverted representation, the string of words representation refers to the inverted variant throughout this thesis.

If we take the same vocabulary from example 2 on page 16, then the sentence *"The quick brown fox jumps over the lazy dog."* would be represented as

$$[\ 10\ 17\ 0\ 36\ 30\ 6\ 10\ 19\ 39\ ] \tag{7}$$

which is a much denser representation of the same text than the bag of words representation as seen in example 3, which is not only much longer, but consists mostly of zeros. Additionally, the sentence *"The quick brown dog jumps over the lazy fox."* would be represented differently, namely as

$$[\ 10\ 17\ 0\ 39\ 30\ 6\ 10\ 19\ 36\ ] \tag{8}$$

Figure 4 shows an overview of the difference in buildup of the two representations. The main difference is that the bag of words representation is vocabulary based, while the string of words is based on the input text. The numbers in the string of words representation represent indices of the vocabulary that is used.

3.5.1 Advantages of SOW. Apart from being a denser representation than the bag of words representation, the string of words representation retains the original arrangement of the words from the processed texts. This enables a machine learning model that is processing it to use contextual information in the classification of the text. For example, because the original word arrangement is still present, negations can be processed more accurately. In a text containing for example the words ‘not bad’, the negation by ‘not’ can be linked to the negative ‘bad’, resulting in an altogether positive evaluation of

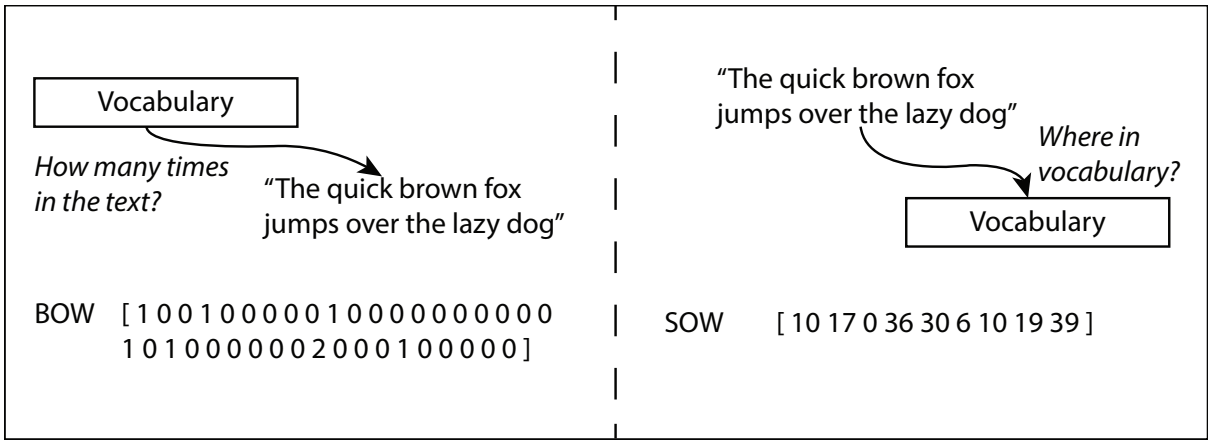


Figure 4. Difference between the bag of words (left, BOW) and string of words (right, SOW) representations.

the text. This could not be done in a bag of word representation, as it is unknown if the word 'not' applies or 'bad' or to another word.

The explicit value reserved for unknown words provides another advantage for the string of words representation. At one point or another, every natural language classifier will encounter words that it has not seen before. This is a problem that can be dealt with in a number of ways. The word in question can be ignored and skipped, or it can be taken into account when classifying the text. In the bag of words representation, words that were not previously encountered will not be in the vocabulary and will therefore not be represented in the vector representation of the text. An argument for doing this may be that it is not an important word if it did not feature in the training data. This may be a valid point in for example a very specialized interpreting algorithm made for a very specific field. Here one could claim to know every important word for that domain. However, this does not hold in general, as the language we use is creative and a similar sentiment can be expressed in a number of different ways (Jurafsky & Martin, 2017).

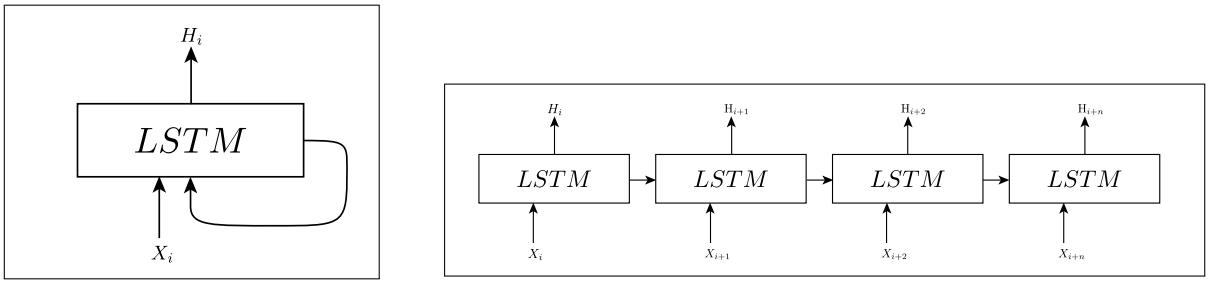
Another approach is to explicitly define an 'unknown word' word feature. The string of words representation has reserved the value 0 for words that do not occur in the vocabulary which gives a natural place in the representation for new words. The unknown words will not give decisive information about the classes but will help with updating the contextual information and also the confidence of the model in its predictions. For example, if a negation is followed by a number of unknown words, the negation might not apply on a known word later on. If the unknown words were simply not represented, there would be no way of determining this. Similarly, the confidence of a certain classification

might be lower if a lot of unknown words are encountered, which requires the model to keep track of the number of unknown words.

4 Neural network

4.1 Overview

The neural network used in this thesis is a recurrent neural network (RNN), which means it has the ability to let previously encountered data weigh in on the parameter values of certain layers. Recurrent neural networks are networks that can feed back on its own input between steps. One set of layers is repeated a number of times, where the output of a previous step is (part of) the input of the next step. The recurrent layers can be seen as a linear series of layers like in non-recurrent models with the difference being that many of the layers share their weight parameters with each other. Figure 5 visualizes the feedback element in the specific RNN used in this thesis, which is called an Long Short Term Memory (LSTM) RNN. Figure 5 also shows how this model with a feedback element is essentially the same as a model model without feedback which has a repetitive layer.



(a) A simplified representation of the LSTM layer, which uses its own output as an input for a next iteration.

(b) The same LSTM network represented as a series of feedforward layers. It is important to note that all feedforward layers share weights.

Figure 5. The feedback in the LSTM layer (as indicated in 5a) can be seen as a series of feedforward layers where all layers share one set of internal weights, as shown in 5b.

Because of this feedback, previously encountered data can be utilized by the machine learning model. The incorporation of previously encountered data in the processing of later data makes sense if the functioning of neural networks is compared to how humans interpret language. We do not assess each word individually without remembering what the last word meant. The same is true for entire sentences; we update our beliefs as we go instead of starting with a blank slate with every sentence we read.

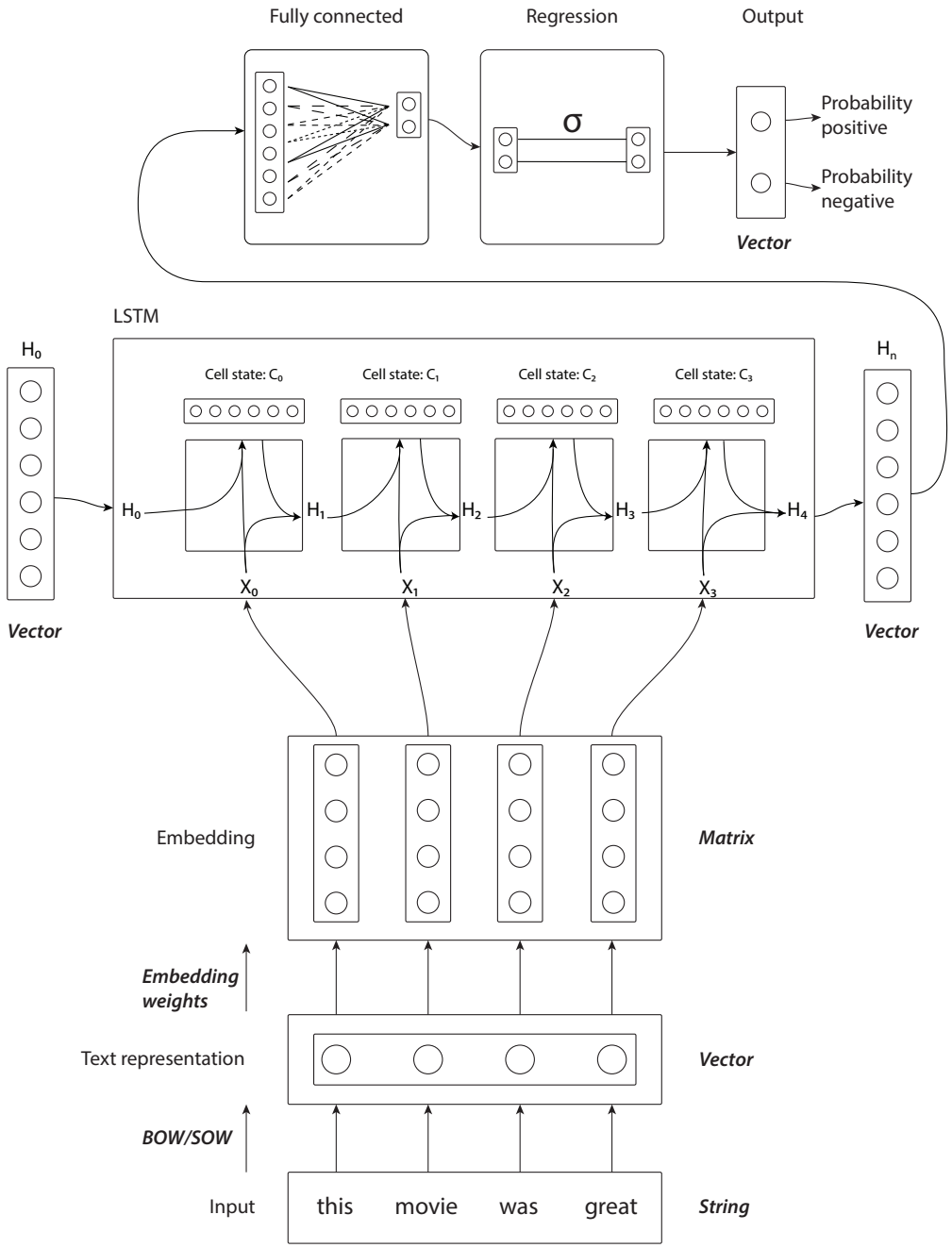


Figure 6. Flow of the neural network as used in this thesis. The flow starts at the bottom with the input layer where text is ingested. The word representation (be it bag of words or string of words) transforms it into a vector, which is in turn transformed into a matrix in the embedding layer. The LSTM layer outputs a vector, which is made into two weighted averages in the form of a two dimensional vector. The regression layer transforms these two dimensions into valid probabilities for each class.

The specific type of recurrent layer for the classifier used in this thesis is the so-called long short term memory (LSTM) layer, which has the ability to selectively 'remember' or 'forget' information on a short or longer term using the so called 'cell state'. This enables it to use information such as negations of a certain word further along in the text.

Figure 6 shows an overview of how the model in its entirety is built up. It starts with a movie review: *This movie was great!*. The review is imported and normalized (see section 4.2) to become *this movie was great*. This is how it enters the model, which can be seen at the bottom of figure 6. This sentence is processed into a vector using either the Bag of Words or String of Words representation (see section 4.3).

The embedding layer turns every word into a word vector, which yields a matrix (section 4.4.1). In the LSTM layer, each word vector is processed in one timestep. The LSTM layer keeps certain pieces of information saved in the cell state, which it uses to retain long term contextual information. How this is done exactly is visualized in figure 9 and explained in section 4.4.2. The output of the LSTM layer is summarized in a twodimensional vector in the fully connected layer (section 4.4.3). This vector is then squashed to values between 0 and 1 to give probabilities of the review being positive or negative (section 4.4.4).

Appendix C gives an example of the model worked out with numerical values instead of the node representation of figure 6. The code for the machine learning layers can be found in figure 7, the code for the model in its entirety can be found online[4].

4.2 Importing and normalization

The data used to train and test the models in this thesis were obtained from Maas et al. (2011). They have collected 50.000 reviews from the Internet Movie Database (IMDB), with the restriction of at most thirty reviews per movie to avoid a systematic bias. Some examples of these reviews are shown in figure 1. The reviews are split into two classes, positive and negative reviews, based on the star rating that the reviewer on IMDB gave the review. The classes are relatively polarized, as positively rated reviews needed to be rated seven out of ten or more and negatively rated reviews four out of ten or less for the study in Maas et al. (2011). As the same dataset is used in this thesis, the same restrictions apply. The dataset is balanced, with exactly 25.000 reviews for each class.

[4] See https://lucschootuiterkamp.nl#ml for the full program, see section 5.3.2 for more information about versions

After the training data is imported, some normalization steps need to be performed on it. Even though the neural network learns implicitly while training, the application of some rules is required to make the training process efficient. Tokenisation, chunking and part of speech tagging are often used to structure the data so it can be processed. When a sentence is tokenised, it is formatted as a list of separate words called tokens. For example, the sentence: *"The quick brown fox jumps over the lazy dog."* would be tokenised into a list of items as follows:

$$\text{["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog."]} \tag{9}$$

In this thesis, the NLTK tokenization module was used to tokenize the review texts (Bird, Klein, & Loper, 2009)(see source code[4], lines 57 and 77).

Capitalization and punctuation are removed from the words, to homogenize the text and to enable capitalized words to be recognized as the same word as its lowercase counterpart (see source code[4], lines 58, 59, 78 and 79). Punctuation marks by themselves were not considered words and were discarded. In general, punctuation marks may provide useful information such as the beginning and ending of sentence chunks. They may also provide direct semantical insight in the form of emoticons such as ':)'. However, sentences are not evaluated in chunks in the model used in this thesis and since punctuation does not carry much semantic meaning in itself apart from its use as emoticon, it was removed altogether.

Chunking and part of speech tagging are labeling techniques that provide the sentence chunks and parts of speech of the words with their types. These algorithms can be either rule based, where words are looked up in a manually labeled corpus or be a machine learning algorithm itself, trained specifically to recognize parts of speech (Hardeniya, 2015). No sentence chunking was performed because the goal of this thesis is to retain dependencies between words even if relatively many words come in between the dependencies. Because dependencies might get split up into different chunks if chunking were to be applied, it was not applied. During part of speech tagging, the list of words that

made up the piece of text is made into a dictionary[5], and looks as follows:

$$\{ \text{"the"} : \text{"DT"}, \text{"quick"} : \text{"JJ"}, \text{"brown"} : \text{"JJ"}, \text{"fox"} : \text{"NN"}, \text{"jumps"} : \text{"VBZ"}, \text{"over"} : \text{"RP"}, \text{"the"} : \text{"DT"}, \text{"lazy"} : \text{"JJ"}, \text{"dog"} : \text{"NN"} \} \tag{10}$$

The part of speech tags associated with the words are part of the NLTK standard. An overview of all the tags used in the IMDB dataset, which includes the tags used here, can be found in appendix F.

The NLTK part of speech tagger, which is a machine learning based tagger, was used in this thesis because its performance is generally good and part of speech information mainly used to gain insight in the type of words used in the dataset. As currently implemented, it is possible to filter out certain parts of speech, but this was not done in this study as this would require a priori knowledge of the influence of word types on classification performance and this was not present for the model using the string of words representation. This filtering may be used to delete words types that may be uninformative or are prone to cause overfitting of the model. An example of certain parts of speech causing overfitting for the IMDB dataset used in this thesis could mentioning of the score in the review. If reviews consistently include the corresponding score, this might cause a model to train on only this part of the review as it yields high accuracy. However, such overfitted models are not able to classify reviews that do not include the score. Part of speech information could be used to remove all cardinal numbers in such case, which would cause some information loss but (in the case of overfitting) generally improve performance on new data.

Part of speech tags can also be included as input for a machine learning model in addition to the words they are derived from. Words with specific parts of speech tags may for example be given different weights in the text representation. In the BOW representation, this can be implemented by multiplying the frequency of a word with a set value for each POS tag but this ability does not make sense in the SOW representation. Doing this requires a priori knowledge about which word groups may need to weigh in more and since such knowledge was not present while making the model used in this thesis and this functionality is not easily transferred to the SOW representation, it was not applied in the model used. Additionally, when using neural networks, features are grouped implicitly

[5] A dictionary as it is used in programming languages, which is Python in this case. This dictionary is not to be confused with the list of words known to a model, the vocabulary.

which decreases the need to specifically include part of speech tags as input nodes. This implicit feature selection occurs mainly in the embedding layer, where co-occurring words are grouped in the vector representation.

Part of speech tags provide much information about the buildup of the natural language that is evaluated. In the model used in this thesis, it is only deployed as an optional filter for filtering out certain parts of speech and to provide an insight in the buildup of the language used in the IMDB reviews. Additional information on part of speech tags can be found in chapter five of Bird et al. (2009)[6] and in Hardeniya (2015). A list of part of speech tags used in the corpus used in this thesis can be found in appendix F.

In the preprocessing of natural language for machine learning applications, there exist more advanced text manipulation options such as word normalization or numerical transliteration correction to compensate misspelled words and different word forms of the same word. These functions were not applied because these steps were not the main focus of this thesis and more time was spend on the development of the word representation.

4.3 Processing

During the processing of the text for the part of speech analysis, a list of all encountered words is made in which each word and its corresponding class is saved. The vocabulary is constructed using this list of words, using either the information gain or frequency approach to building a vocabulary as described in section 3.1. In this thesis, both vocabulary types will be tested separately as their impact on bag of words an string of words word representations is not clear.

After the vocabulary is defined, the reviews are processed into the vectors that can be interpreted by the machine learning model. Here, each entire review is represented as a single vector representation. This is the step where many neural networks would use a bag of words representation for their data, and where the string of words representation is tested.

Bag of words. The bag of words representation consists of the frequencies of every word in the vocabulary in the to-be-represented text. This results in a vector for each review in the dataset. Each vector is n dimensions long, where n is equal to the vocabulary length. The value of each dimension in the vector is defined by the number of times that

[6] A python3 compatible version is available at https://www.nltk.org/book/ch05.html

the word with the same index in the vocabulary is used in the text. A more thorough explanation of the bag of words representation can be found in section A.1.2. Each vector is saved in a list and every time a review is added to the list, the class of that review is saved at the same index in a different list.

String of words model. The string of words representation consists of indices in the vocabulary for each word in the to-be-represented review. These indices are meaningful, as the vocabulary is ordered in terms of distinguishability or frequency, for the chi square and frequency vocabularies respectively (see section 3.1). Instead of indicating the frequency of each word in the vocabulary, the string of words representation indicates where in the vocabulary each word in the review is located[7]. The process of indicating where each word is in the vocabulary is done with the inverted string of words method, as explained in section 3.5, and is repeated for each word, resulting in a vector with as many dimensions as there are words in the review. There is an additional step in the string of words model, which is to make the lengths of each vector uniform. In the string of words representation, each vector has the length of the review it was based on instead of the length of the vocabulary as is the case with the bag of words representation. Because the neural network expects uniform vector sizes within one training session, the vectors need to be either extended with zeros or truncated to fit the expected length. This is not desirable, as extending the vector unnecessarily with zeros will diminish the beneficial effects of a denser representation, while truncating the vector will result in a loss of information from the text, but is unavoidable when using the Tensorflow neural network API. The number of dimensions in the final string of words representation is manually defined and expected to be optimal if it is close to the mean review length, as both truncation and extension of the vector is minimal at this point. The resulting vector is saved in a list along with their respective classes in a separate list, as was done for the bag of words model.

Data organization. To ensure that the model is trained with heterogeneous data and not exclusively with data from the category that was processed first, the lists with vectors and the list with labels are shuffled randomly in such a way that the indexes from both lists still match. In other words, the order is shuffled, but review and label match. This is done both in the bag of words an string of words condition. Ten percent of the processed

[7] Figure 4 shows this distinction.

```
1 if mode == "bow": # bag of words
2    net = tflearn.input_data([None, vocab_size])
3 elif mode == "sow": # string of words
4    net = tflearn.input_data([None, max_len])
5 net = tflearn.embedding(net, input_dim=(vocab_size + 1),
     output_dim=128)
6 net = tflearn.lstm(net, 128, dropout=0.8)
7 net = tflearn.fully_connected(net, 2, activation='softmax')
8 net = tflearn.regression(net, optimizer='adam', learning_rate
     =0.001, loss='categorical_crossentropy')
9 model = tflearn.DNN(net, tensorboard_verbose=2,
     tensorboard_dir=directory)
```

Figure 7. A code snippet showing the buildup of the tflearn layers in the machine learning model as used in this thesis. The layers are: 1. input layer, 2. embedding layer, 3. lstm layer, 4. fully connected layer, 5. regression layer. The tflearn API is used for this model construction (Damien et al., 2016). Note that the data shape is dependent on the representation type that is used.

data is kept separate to be used as validation set and is not used in the training of the model.

4.4 Neural network layers

Because the emphasis of this thesis is on the word representation and not mathematical innovations, an API was used for the implementation of the neural network. The TFLearn API for Tensorflow (Abadi et al., 2015; Damien et al., 2016) was used to construct the neural network in the form of high-level commands, which can be seen in figure 7. The specific lines of code are referenced as each layer in the network is discussed.

The long short term memory (LSTM) recurrent neural network (RNN) model contains, like all RNN's, a circularity which is dependent on its own outcome of a previous iteration for the current iteration. This circularity is interrupted after a certain number of iterations. The entirety of the recurrent layer can be viewed as a convolutional network that shares weights across the entire network. The output of the first iteration is used as an input in the second iteration. This process continues until a certain number of iterations has passed. The output of the final iteration forms the output of the LSTM layer. An 'unfolded' view of a recurrent layer is shown in figure 5.

Parameters for the operations in each iteration are shared between the iterations, making updating the weights much easier than if each iteration had their own weights. The shar-

ing of weights is what differentiates a recurrent neural network from a large feedforward network.

Lines 1 through 4 in figure 7 define an 'input layer', which is not a real neural network layer but a TFLearn initialization step. It forms a placeholder that is used to build up the model. The placeholder should be the same shape as the actual data will be, which for the model used in this thesis is dependent on the word representation used. The data represented using the bag of words model are by definition all exactly as long the vocabulary is (figure 7, line 2). In contrast, the length is defined manually for string of words representation and all data is either extended or truncated to be this length (figure 7, line 4). Other layers use the input shape given in the input data layer to determine what the incoming and outgoing shape will be in their initialization.

4.4.1 Embedding. Two words that are close to each other in the vocabulary, and therefore also have closely resembling values in the vector representation, do not necessarily have similar meanings. It may be that they are both very differentiating for their class but this may be for different classes. The embedding layer will group features that co-occur in classes and therefore distinguish between words that are differentiating for either the positive or negative class.

The function of the embedding layer is to transform the vector representation in such a way that words that co-occur in the reviews resemble each other in the vector representation. This groups certain features together, which increases the amount of useful information when representations are sparse, as the information that *is* present will say something about a category of words instead of only one word. Embedding layers transform each integer from the word representation - be it bag of words or string of words - into a vector, thereby transforming the word representation into an n by m dimensional matrix. The product of each dimension of the word representation vector and each embedding weight vector form each word vector. This is represented in figure 8 and the lower part of figure 6. The input *this movie was great* is represented as a vector by means of either the BOW or SOW method. A vector of embedding weights is trained for each dimension of the text representation and embedding vectors are created for each word by taking the product of the embedding weights and each dimension of the text representation.

The values of the weight vectors are updated during training and are meant to bring

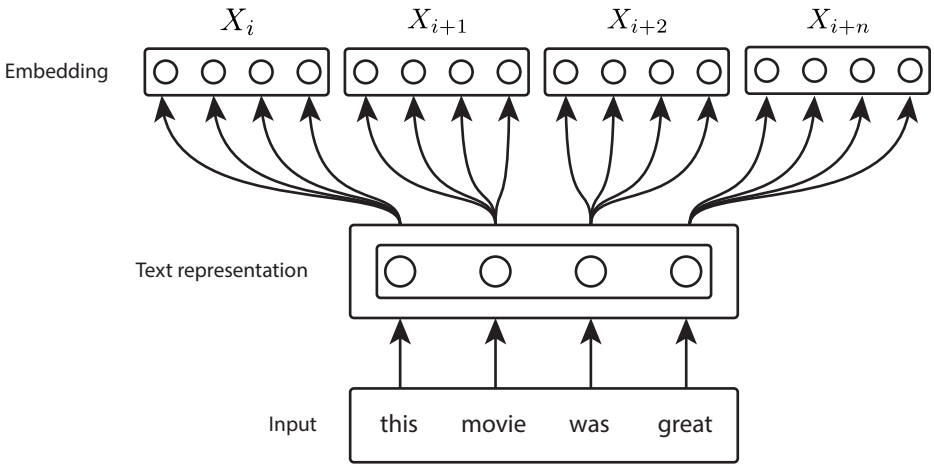


Figure 8. Flow of input *this movie is great* through the text representation into word embeddings.

words that are close in meaning close to each other in representation. This is achieved by updating the weights such that the output (word) vectors are similar if the input values co-occur frequently.

It might for example transform the string of words representation of *"The quick brown fox jumps over the lazy dog."* that we have seen before in examples 7 and 8 into:

$$[\ [22.0\ \ 22.8]\ [28.6\ \ 26.7]\ [20.2\ \ 40.2]\ [8.8\ \ 7.3]\ [30.1\ \ 33.3]\ [11.0\ \ 12.9]\ [22.0\ \ 22.8]\ [17.1\ 16.4]\ [8.0\ 15.4]\ ] \tag{11}$$

like the example in section 3.4.2. This matrix contains the same information but as a list of vectors instead of a list of integers. The vector has transformed from a 9-dimensional vector to a 9 by 2 dimensional matrix. The embedding layer increases the dimensionality of the data, which increases the separability of the categories. This means that a simpler function may be used to separate the classes (Bishop, 2011). Line 5 in figure 7 shows the implementation of the embedding layer in the program used in this thesis.

4.4.2 LSTM. The LSTM layer is the third layer in the model and is the core of the recurrent neural network. It takes the n by m-dimensional output of the embedding layer and processes it iteratively, one word vector at the time. The output of the LSTM layer is again an n dimensional vector where each dimension represents one word.

The major advantage of the LSTM layer versus a regular non-recurrent layer is that it takes into account the previously processed words. This feature is put to good use in the string of words word representation, as the words are in the original order. This means

that negating words influence subsequent words, something that cannot be done with the bag of words model. Before the background of the LSTM layer is touched upon, it is useful to name its different parts. The terminology correlates largely with Olah (2015), whose guide to LSTM networks provides additional background information.

Naming of LSTM parts. The terminology explained here is the same as used in figure 9 (and appendix C), which can be used to follow the steps in the LSTM layer. The LSTM layer is a recurrent layer, the same set of formulas is executed iteratively with a new input at each step. This input is one word vector from the word embedding matrix, named X_i (see figure 8). Every step, the output of the previous state is used as an input. The output of each step in general is referred to as H_i, which is the hidden state. By extend, the previous output will be H_{i-1}. There is also a cell state, C_i, which can carry information over separate steps and which is updated in each step. The cell state is what sets LSTM layers apart from normal recurrent neural networks (RNN's). In normal RNN's, each previous term is considered as an input for each subsequent term (as seen in figure 5). While this also happens in LSTM layers, there is another input to each step in the LSTM layer. The cell state enables information to be retained for a longer period of time than only one step. During each step, information is removed from and added to the cell state.

Figure 9 shows one iteration of the LSTM layer, with dashed lines indicating the flow to the next iteration and the word vector X_i, the previous hidden state H_{i-1} and the previous cell state C_{i-1} as inputs. Each iteration has an updated hidden state H_i as output. The several steps in this figure are discussed in the next sections, which are divided into *deletion from the cell state*, *insertion into the cell state* and *generating the output*.

Deletion from cell state. Every step, previously added information is evaluated and kept in or discarded from the cell state based on the word vector X_i combined with the output of the previous step H_{i-1}. A sigmoid function of the trainable weights W_{del} times concatenated vectors X_i and H_{i-1} plus a trainable bias B_{del} defines what should be deleted from the cell state during each step. This sigmoid function is called the deletion formula:

$$f_{del} = \sigma(W_{del} \cdot [H_{i-1}, X_i] + B_{del}) \tag{12}$$

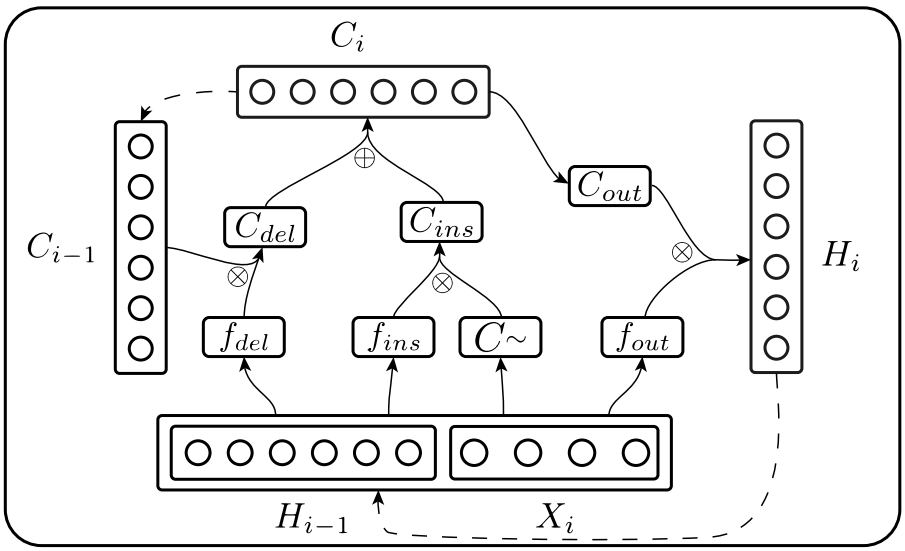


Figure 9. Flow of one word vector within one iteration of the LSTM layer. Solid lines indicate flow of vectors within one iteration of the LSTM layer, dashed line indicate feedback between iterations. Merges marked $\oplus$ are component-wise additions and merges marked $\otimes$ are inner products.

The deletion formula is multiplied with the cell state of the previous iteration to form C_{del}, the elements of the previous cell state that are to be deleted:

$$C_{del} = f_{del} \cdot C_{i-1} \tag{13}$$

Insertion in cell state. After deciding what information should be deleted from the cell state, new information is added to the cell state. Firstly, information to be inserted into the cell state should be selected. The candidate values $C \sim$ are defined by a tangent function of weights W_c times the concatenated vectors X_i and H_{i-1} plus a bias B_c.

$$C \sim \ = tanh(W_c[H_{i-1}, X_i] + B_c) \tag{14}$$

The cell state values that should be updated with this information are defined by a sigmoid function again based on weights, a bias and the concatenated vectors X_i and H_{i-1}. This insertion function resembles the deletion function but serves to indicate which values should be placed where. It has its own distinct weights W_{ins} and bias B_{ins} values from the deletion function for this reason.

$$f_{ins} = \sigma(W_{ins} \cdot [H_{i-1}, X_i] + B_{ins}) \tag{15}$$

The product of $C \sim$ and f_{ins} defines what should be added in place of the data deleted from the cell state.

$$C_{ins} = f_{ins} \cdot C \sim \tag{16}$$

An element wise addition defines the current cell state.

$$C_i = C_{del} + C_{ins} \tag{17}$$

Generating the output. One final step in the LSTM iteration remains, which is to determine the output of the current step: hidden state H_i. First, a tangent function of the current cell state C_i squashes information from the cell state to be incorporated into the hidden state.

$$C_{out} = tanh(C_i) \tag{18}$$

A sigmoid function weights W_o, the concatenated inputs X_i and H_{i-1} and bias B_o defines how what parts of this squashed information should be used.

$$f_{out} = \sigma(W_o[H_{i-1}, X_i] + B_o) \tag{19}$$

The product of f_{out} and C_{out} defines the output of the current iteration, H_i

$$H_i = f_{out} \cdot tanh(C_i) \tag{20}$$

This output forms the input of the next step or the final output of the LSTM layer if the current step is the last step.

4.4.3 Fully connected. The next layer is the fully connected layer, which maps each dimension of the output vector of the LSTM layer to each dimension of the fully connected output vector, making both layers 'fully connected'. This fully connected mapping is visualized in figure 10. The connections are not one to one, but are weighted. The weights differ for each dimension of the input vector and the output vector hence the two dimensional weight and bias vectors $Weight_{i,j}$ and $Bias_{i,j}$ as shown in figure 10. These weights are updated during training. The fully connected layer decreases the size of the vector to two dimensions, which represent the two classes that the neural network

classifies. These two dimensions essentially form weighted sums of the input vector, but both with different weights so as to capture the different classes they represent.

4.4.4 Regression. The last layer is the regression layer, in which a logistic regression is performed on the vector from the fully connected layer. This process resembles the logistic regression classifier discussed in appendix A.2, the distribution is fitted to a sigmoid function. The weights associated with this function can be trained like the other parameters in the model. This step yields a probability for each class in the form of a vector with two floats between 0 and 1, each float represent the probability for one class. The output of the model is a vector in which the first value represents the probability of the input being positive and the second value represents the probability of the input being negative.

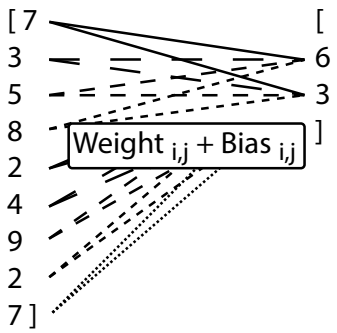


Figure 10. Fully connected layer, producing two different weighted sums of the input, using different weights for each sum.

4.4.5 Updating weights. The weights in the neural network are updated in a way similar to the logistic regression classifier discussed in section A.2. The loss function for the neural network is categorical cross entropy (see equation 37 and 38), which is minimized using adaptive moment estimation (ADAM), an adaptive gradient descent optimizer (Kingma & Ba, 2014). This optimizer performs gradient descent as the discriminative classifier in appendix A.2 but also uses the loss value to determine the weight updates, as well as the usual learning rate η and gradient ∇ values.

5 Research

5.1 Hypotheses

The focus of this thesis is the development of the string of words (SOW) representation for natural language for use in natural language processing machine learning models. To test whether this representation outperforms the bag of words (BOW) representation, both time measures and performance measures are taken of models that use different representations.

Because the SOW representation enables the use of contextual information and more

information about word relations is retained in the SOW representation, **models using the SOW representation are expected to have higher performance scores than models using the bag of words representation**.

In the string of words representation, the text vectors are truncated or extended to a predefined length regardless of the vocabulary length and are therefore much shorter. As the vocabulary size of the model increases, the processing time before and during training is expected to increase less sharply for the SOW representation than for the BOW representation, which would make SOW more suitable for scaling to larger models because of the smaller processing time penalty. For this reason, **the SOW representation is expected to perform better in time based measures than the BOW representation**.

5.2 Dataset

A set of reviews of movies and tv-shows originally posted to IMDB, composed by Maas et al. (2011) was used as dataset for training and testing the different word representations. The dataset contained 50.000 reviews which were classified into the classes 'positive' and 'negative' according to the star rating associated with each review. Reviews were classified as positive if the star rating was seven out of ten or more and negative if the star rating was four out of ten or less. At most thirty reviews per movie were allowed, to avoid a systematic bias. The dataset is balanced, with exactly equal 25.000 reviews for each class.

Thirty-eight reviews were removed from the dataset because they contained non-utf8 characters and could therefore not be processed by the data preparation code. Specifically, tokenisation of the words failed and yielded an error. Since this an integral part of the data processing, either the non-utf8 characters or the entire reviews containing the characters should be deleted. Because the dataset is amply big, and because it concerns only a small portion of the data and to avoid altering the data, the entire reviews containing non-utf8 characters were removed. The names of the reviews that were not used can be found on page 80 in appendix E, the corresponding reviews can be found via the dataset provided in the original article of Maas et al. (2011). The omissions from the dataset caused it to be slightly imbalanced as more negative than positive reviews were removed but the methods used are robust enough to handle this.

5.3 Model implementation

5.3.1 Training variables. As mentioned in section 3.5, the length of the vector is expected to be uniform. For the string of words representation, a vector length should be defined such that a balance exists between performance and computational efficiency. The average length of the film reviews used in this thesis was 250 words, so the vector length was chosen to be 250. This value was chosen because the effects of extending and truncation are not known, so they cannot be balanced by effect size on performance. Using the average word length results in a minimal amount of detrimental effects assuming the extending and truncating have equal negative effects on performance and assuming the distribution of review length is symmetrical around the mean.

As can be seen in the code snippet in figure 7, the input shape differs between the word representations, as the input length of the bag of words vector is defined as the vocabulary size while the length of the string of words vector is manually set.

5.3.2 Program versions. The program used in this thesis is split into two distinct versions, both of which can be downloaded on https://lucschootuiterkamp.nl#ml. The standalone full program features a command line interface and can be used to easily change variables used in the program. This environment is useful for optimizing the model and testing different parameter settings. The command line interface is shown in figure 11, the options are as follows:

1. Train and save classifier with current settings

   Train and save a new model with the current settings, even if a model is already saved.

2. Test classifiers

   Perform selected statistics (as described in section 5.5) and save results. Attempts to load model with the current settings and trains a new model is no model is available.

3. Save most useful and most frequent words for current settings

   Saves the the frequency and chi-square vocabulary with current vocabulary size.

4. Fill in movie review to classify manually

5. Change rnn directory (default: homedir)

Change the base directory. In this directory, the IMDB *reviews are expected, and here the models, statistics and wordlists are saved.*

6. Change vocabulary size
7. Change max SOW vocabulary length
8. Change allowed word types
9. Change vocabulary type
10. Change vocabulary mode
11. Raw python - for debugging

    Enables a python prompt which a model is loaded, which is useful for testing the program.

There is also a batch optimized version, which takes arguments when the program is called in the form of commandline flags which set the parameters in the model. This version can be used to train different models with different parameters back to back without supervision and takes the following arguments:

```
Select option:

        1. Train and save classifier with current settings
        2. Test classifiers
        3. Save most useful and most frequent words for current settings
        4. Fill in movie review to classify manually
        5. Change rnn directory (default: homedir)
        6. Change vocabulary size
        7. Change max SOW vocabulary length
        8. Change allowed word types
        9. Change vocabulary type
        10. Change vocabulary mode
        11. Raw python - for debugging

Current settings:
        Directory: /media/lucas/95879cd0-c7bc-4efa-97b7-393b20df88a83
        Vocabulary size: 500
        Max SOW vocabulary length: 500
        Allowed word types: all
        Vocabulary type: chi
        Process mode: sow
        sow mode: inverted<-- this should always be 'inverted'

Press ctrl + C to exit.
```

Figure 11. Command line interface for tuning parameters in the machine learning models used in this thesis.

```
python /path/to/file/test_classifiers.py $vocabulary_size $mode
$type $base_directory $batch
```
(21)

The $vocabulary_size argument defines the size of vocabulary that should be used, and should be an integer. $mode is used to distinguish between bag of words and string of words models, and should be either "bow" or "sow". $type defines the vocabulary type - either frequency or chi square - that is used for the current model, and should be either "chi" or "freq". The $base_directory variable defines the directory where the IMDB reviews are expected, and where the models, statistics and wordlists are saved. The current batch number is given in $batch, which should be an integer. The batch number is useful to distinguish series of runs from each other. Lastly, there is one optional parameter that can be used to set the program to use the non-inverted string of words representation by adding "nonInverted" to the arguments. However, since the inverted mode is always superior this should not be used. An example batch file that may be used to batch test a model is provided at https://lucschootuiterkamp.nl#ml.

5.3.3 Design. For both word representations, the neural network was trained along a range of vocabulary sizes. The models were trained with vocabulary sizes 100 to 500 with intervals of 100 and from 1000 to 2500 with intervals of 500. This yielded nine different vocabulary sizes for each of the word representations.

Both the bag of words and the string of words model were trained with varying vocabulary type - chi square and frequency vocabulary, see section 3.1 - and vocabulary size which resulted in a 2x2x9 setup[8]. The frequency approach and chi-square approach are both tested because the word representations are closely related to the buildup of the vocabulary and the string of words representation might respond differently on different vocabulary types.

5.4 Data visualization

To get an understanding of how the data is distributed in terms of length of reviews and distribution of part of speech tags, frequency graphs were made to indicate the frequency of different review lengths and the frequency of different part of speech tags.

[8] bag of words/string of words x chi-square/frequency x vocabulary size: 100 200 300 400 500 1000 1500 2000 2500

5.5 Model comparison methods

A multitude of methods exist that have been used to compare machine learning models. An often-used measure is the the k-fold (or k-times) cross validation test, which uses a subset of the training data to test performance with error functions such as the least mean square error function. To prevent overfitting of the model, this process is repeated k times with different parts of the training data. Common k values lie between 4 and 8, less than four being prone to overfitting and more than eight often being too computationally expensive to justify. Dietterich (1998) advice running 2-fold cross validation test five times if it is feasible to run all the to-be-compared models ten times. However, this is not the case in the models tested in this thesis as they are already tested across a number of variables and therefore already run a large number of times. Using the 5x2 cross validation would increase the number of runs tenfold which was not feasible for this project.

Dietterich (1998) recommend McNemar's test to compare models if it is preferable to run the algorithm only once, however this test is only suitable to compare two models and is unsuitable for the 2x2x9 design of this study.

The area under a receiver operating characteristic (ROC) curve is a model performance measure and indicates the balance between true and false positives. This means that it can used to compare any number of models. The ROC curve specifies the amount of false positives found per true positive, a higher area under the curve (AUC) points to a high true positive to false positive ratio and seems therefore an appropriate measure to assess classifier performance.

It is a widespread measure (Bradley, 1997; Fawcett, 2006; Lobo, Jiménez-Valverde, & Real, 2008) and is recommended by some (Bradley, 1997) but found misleading by others like Lobo et al. (2008), who describe a number of shortcomings in their paper. The ROC curve only takes the positive condition into account. For this thesis, that would mean that it can measure only the performance of the model for one of the two classes. The confusion matrix contains a second condition, the true negatives and false negatives. This would translate to true 'class 2' and false 'class 2' for this thesis. It makes more sense to incorporate these measures as well, as Chicco (2017) also explains.

The Matthews Correlation Coefficient (MCC) measure does exactly this. The MCC measure is a single number interpretation of the confusion matrix. Unlike other measures

mentioned above, it accounts for true and false positives and true and false negatives. The MCC measure ranges from -1 to 1. A classifier that falsely classifies both classes is scored a -1, a score of 0 represents chance level classifications and 1 represents perfect classifications. The formula for the MCC score is as follows (TP = true positive, TN = true negative, FP = false positive, FN = false negative)

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{22}$$

Jurman, Riccadonna, and Furlanello (2012) compared MCC to confusion entropy (CEN), a method based on a probabilistic confusion matrix capable of measuring accuracy and discriminative power. Jurman et al. (2012) concluded that MCC is *"a good compromise among discriminancy, consistency and coherent behaviors with varying number of classes, unbalanced datasets, and randomization"* (Jurman et al., 2012). They also found that CEN and MCC have a strong linear relation and that they therefore can be used interchangeably in most cases. However, MCC results are consistent between binary and multi-class classifiers whereas CEN cannot be reliably used in binary classification cases such as the classifier in this thesis. They conclude that MCC is a good off-the-shelf tool suitable for most practical tasks (Jurman et al., 2012). For these reasons, the Matthew's correlation coefficient is used as performance measure to compare the different models.

5.6 Comparison of measures

To compare the performance scores of the models, Bayesian generalized linear models were used. Resources from Schmettow (2016) and other resources mentioned in Schmettow (2015) were used to construct the R code and to conduct these analyses. All analyses were done in the statistical program R, version 3.5.0 (R Core Team, 2014), for which the graphical user interface package RStudio version 1.1.453 (RStudio Team, 2015) was used. An R markdown document containing code and results from all the analyses that were performed can be found in appendix D. The prior distribution used for the Bayesian analyses was a normal (Gaussian) distribution as this was the closest fit to how the MCC score would be distributed if the classifier operated purely on chance level (i.e. if the probability of predicting a class correctly is 0.5). There is one violation of the normal distribution in the MCC data, which is that they will never be larger than 1 or smaller than -1.

To estimate the difference between the frequency and chi square vocabulary buildup, a Bayesian statistical model was constructed in which the mean MCC value is predicted by the vocabulary buildup type. This was done to be able to take this effect in consideration in the interpretation of the rest of the analyses should one vocabulary buildup perform better than the other.

To discern whether the vocabulary buildup technique influenced performance of either the bag of words model or the string of words model individually as opposed to the word representations combined, the MCC values of the two word representations were predicted by the vocabulary buildup technique separately.

The means of the two word representations were compared, again using a Bayesian linear model. This enabled a broad comparison of the word representations by comparing the average MCC scores over all the training parameters. To be able to distinguish how the vocabulary size affects the different representations, a random effects Bayesian model was constructed in which the MCC value was predicted by the word representation with the vocabulary size as a random effect.

5.7 Exploratory analysis: Truncation length and text length

The SOW representation truncates film reviews after a certain predefined length, which is why it is hypothesized to be more efficient. As the consequences of truncation and extension to this predefined length were unknown at the time of writing and assumed to be similar, the average text length was taken as a cutoff point.

To explore the effect of different cutoff points on MCC scores of the resulting models, a range of cutoff points were tested. A string of words model with a chi square vocabulary with a size of 1000 words was used to test these different cutoff points. The MCC values were tested for a number of ranges of text lengths. Shorter text lengths were expected to perform optimal even if low cutoff points are used, while longer texts may benefit from higher cutoff points. However, the increase in performance is expected to slope off to a constant at some point, as one often does not need to read the entirety of especially long film reviews to get a sense of whether it is positive or negative. Therefore, the model was expected to be able to extract enough information from truncated texts to accurately determine whether it is a positive or negative review. This implies that there is a ceiling effect: if the first few lines of text are enough to determine a class, the classifier should

not be much more effective at determining texts with a larger truncation length than it would be for texts with similar lengths truncated shorter. Where this ceiling effect starts was expected to be dependant on the review length, as the first sentence in a 100 word review will likely be more informative than the first sentence of a 500 word review.

To determine if and where this effect occurs, the average MCC values of different ranges of text length were compared. Ranges of text lengths were used as the MCC measure only works as an average measure, as individual measures will only be 1 or -1. Ranges of size 50 were used to ensure enough samples to provide a valid mean while preserving a high enough resolution to estimate a target truncation length. A new SOW model was trained for each truncation length and tested for each text length range with a small subsection of the the IMDB data. Because this subsection of data was not very large and review sizes of 850 words and larger are infrequent (see figure 12), review lengths above 850 words were not taken into account, as the MCC scores became unreliable because of the small sample size at this point.

6 Results

6.1 Data properties

6.1.1 Review lengths. The IMDB film reviews had an average length of 238 words after removal of the 38 reviews that were not processable and after standardization steps (see section 5.2). Positive reviews were slightly longer than negative reviews, with averages of 240 and 236 respectively. The length ranged from 2 words to 2511, with 90% between 62 and 626 for the positive reviews and from 6 to 1580, with 90% between 69 and 539 for the negative reviews. The high density around the mean resulted in 14.1% of the reviews being extended by more than 150 dimensions in the string of words representation and 13.9% of the reviews being truncated by more than 150 dimensions. Figure 12 shows the size distribution for the review lengths.

6.1.2 Part of speech tags. Figure 13 shows the frequency distribution of the most used part of speech tags as encountered in the corpus used in this thesis. The least used are omitted in the figure but can be found in the full list of part of speech tags in appendix F. This list includes the definitions of the tags and is listed in order of high to low frequency in the used reviews.

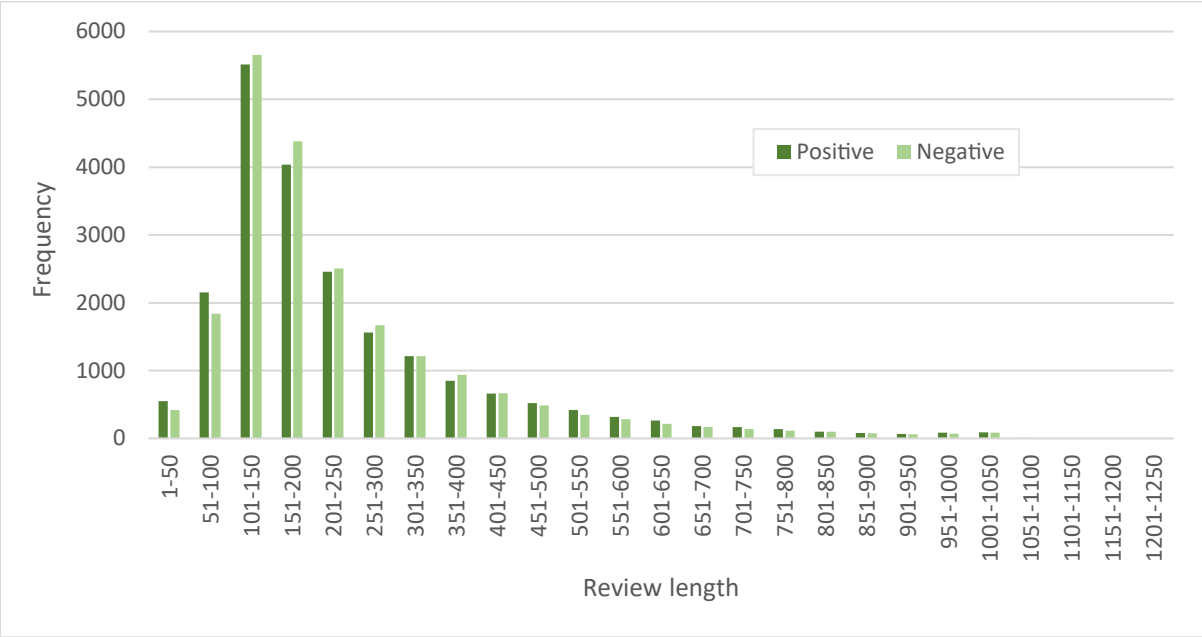


Figure 12. Frequency graph of the number of words in the film reviews used as texts. Positive and negative reviews are indicated separately.

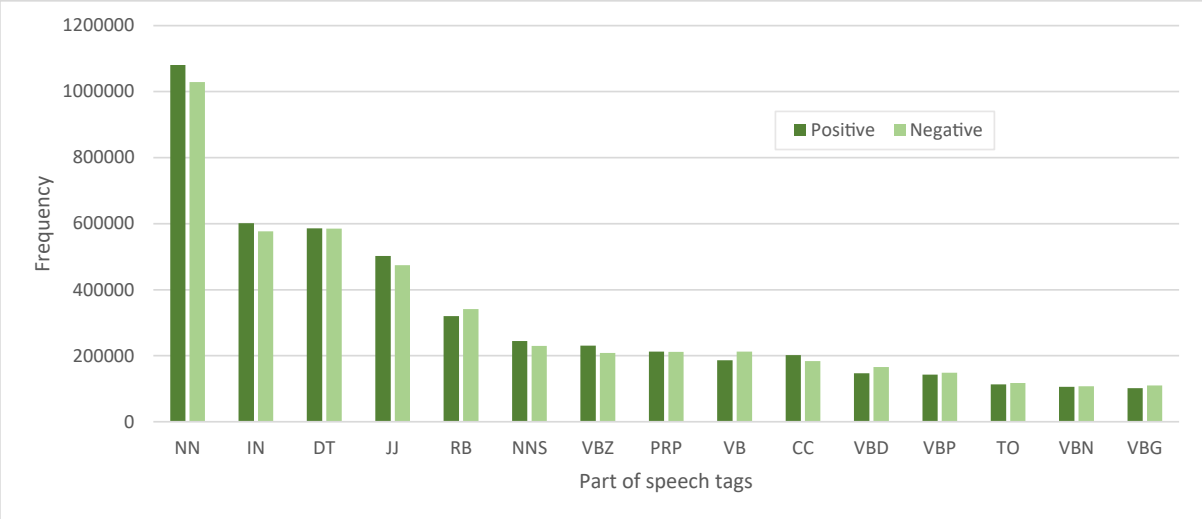


Figure 13. Frequency graph of the most used part of speech tags as occurred in the corpus used. Definitions for POS tags can be found in appendix F, as can the least used and here omitted tags.

6.2 Analysis results

6.2.1 Vocabulary type. There is a small difference in average MCC scores when comparing the two vocabulary building techniques without differentiating between the bag of words and the string of word representation. The frequency vocabulary building technique averages 0.058 MCC lower than the chi square technique, which averages a score of 0.426. The difference is rather small and the credibility interval associated with

the difference $(-0.24; 0.13)$ [9] is rather large.

If vocabulary building techniques are considered separately for both word representations, the chi square technique performs marginally better for the bag of words representation as can be seen in table 1. For the string of words representation there is no difference between the chi square and frequency vocabulary building technique.

Table 1
Mean MCC values (center) and 95% credibility intervals (lower, upper) of bayesian generalized linear models in which the MCC value was predicted by the vocabulary type for the bag of words representation and the string of words representation separately.

Representation	Vocabulary type	Center	Lower	Upper
Bag of words	Intercept (Chi)	0.19	0.12	0.27
	Freq	−0.11	−0.21	0.00
String of words	Intercept (Chi)	0.66	0.61	0.70
	Freq	−0.00	−0.07	0.06

6.2.2 MCC values. Table 2 shows the mean MCC values and credibility intervals of both word representations for differing vocabulary sizes. In table 1, both chi square and frequency vocabulary types were used. The BOW and SOW representations are taken into account separately in table 2 and table 3 shows the mean MCC values and credibility intervals for the different word representations split up over both the vocabulary type and size.

Figure 14 shows the MCC value of both the bag of words and string of words representations for a range of vocabulary sizes for both chi square and frequency vocabulary types. The dashed vertical line denotes a change of scale in the x-axis.

6.2.3 Time measures. The preparation time increased mostly linearly for the bag of words representation at a rate of about 80 seconds per hundred words. The preparation time for the string of words representation remained constant until about 500 words and after that increased more slowly than the preparation time for the bag of words representation, at a rate of about 40 seconds per hundred words. The additional preparation time that the bag of words representation requires is relatively small compared to the extra training time the bag of words representation requires, which scales linearly with about 27 minutes per 100 words. The string of words representation requires around 67 minutes

[9] The credibility interval is an interval in which the estimated value lies with a certain predefined probability, 95% in this case. For example, the credible interval $-0.24; 0.13$ should be interpreted as $P(-0.24 < MCC < 0.13) = 0.95$. See Schmettow (2016) or van de Schoot and Depaoli (2014) for more information on this subject.

Table 2
Mean MCC values (center) and 95% credibility intervals (lower, upper) of the bag of words (BOW) and string of words (SOW) representation for different vocabulary sizes.

	BOW			SOW		
Vocabulary size	center	lower	upper	center	lower	upper
Intercept	0.41	0.31	0.50	0.41	0.31	0.50
100	−0.09	−0.19	0.01	0.15	0.03	0.28
200	−0.21	−0.33	−0.10	0.35	0.23	0.47
300	−0.22	−0.33	−0.10	0.41	0.29	0.53
400	−0.23	−0.35	−0.11	0.44	0.32	0.55
500	−0.30	−0.42	−0.17	0.53	0.41	0.65
1000	−0.33	−0.47	−0.20	0.60	0.48	0.71
1500	−0.33	−0.46	−0.19	0.59	0.47	0.71
2000	−0.35	−0.48	−0.21	0.63	0.51	0.75
2500	−0.38	−0.52	−0.23	0.66	0.55	0.79

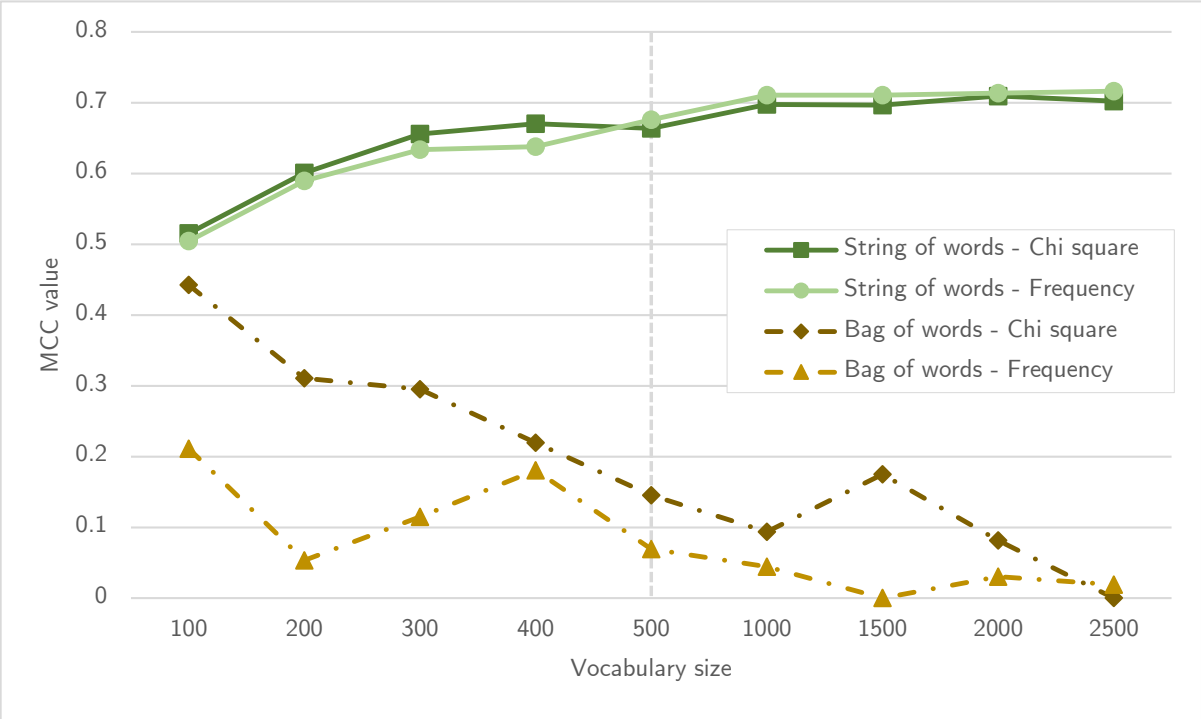


Figure 14. Graph of MCC scores along increasing vocabulary sizes for both string of words and bag of word representations and both chi square and frequency vocabulary buildup. The dashed vertical line denotes a change of scale in the x-axis.

regardless of vocabulary size as the vector length stays 250 regardless of vocabulary size. The crossover point at which the string of words representation is quicker in total lies around a vocabulary size of 250 words.

Figures 15a, 15b and 15c show the preparation time, training time and total time of both word representations for a range of vocabulary sizes for both vocabulary buildup

Table 3
Mean MCC scores (center) and credibility intervals (lower, upper) of the bag of words (BOW) and string of words (SOW) representations for both chi square (chi and frequency vocabulary (freq) building techniques for a range of vocabulary sizes.

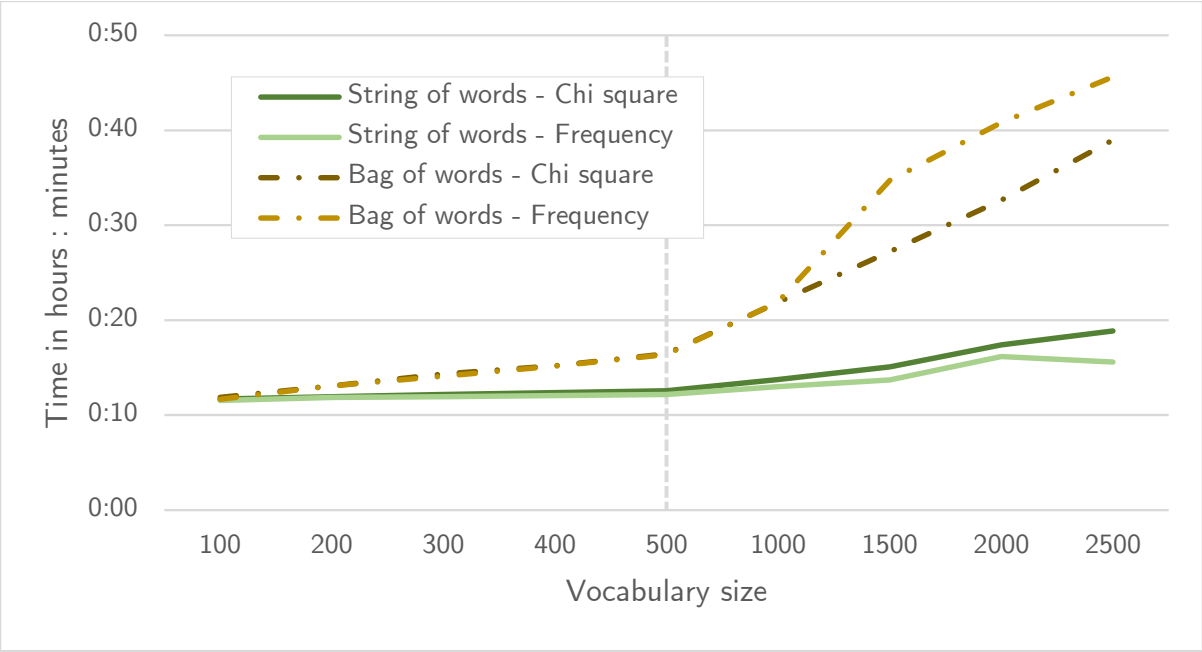
	BOW					
		chi			freq	
Vocabulary size	center	lower	upper	center	lower	upper
Intercept	0.12	0.02	0.22	0.12	0.02	0.22
100	0.20	−0.02	0.43	−0.11	−0.32	0.08
200	0.10	−0.06	0.29	−0.10	−0.28	0.05
300	0.11	−0.06	0.29	−0.08	−0.25	0.07
400	0.07	−0.08	0.24	−0.03	−0.18	0.11
500	0.02	−0.13	0.15	−0.03	−0.18	0.10
1000	−0.01	−0.16	0.12	−0.02	−0.16	0.10
1500	0.03	−0.13	0.18	−0.07	−0.24	0.07
2000	−0.02	−0.16	0.12	−0.02	−0.16	0.10
2500	−0.07	−0.22	0.05	0.01	−0.14	0.16
	SOW					
		chi			freq	
Vocabulary size	center	lower	upper	center	lower	upper
Intercept	0.63	0.57	0.69	0.63	0.57	0.69
100	−0.12	−0.21	−0.04	−0.02	−0.10	0.03
200	−0.06	−0.14	0.01	−0.01	−0.08	0.03
300	−0.02	−0.09	0.06	−0.01	−0.06	0.03
400	−0.00	−0.07	0.07	−0.00	−0.05	0.04
500	0.01	−0.06	0.10	0.00	−0.04	0.05
1000	0.03	−0.03	0.12	0.01	−0.03	0.06
1500	0.04	−0.03	0.11	0.01	−0.03	0.06
2000	0.05	−0.02	0.13	0.01	−0.03	0.06
2500	0.05	−0.02	0.13	0.01	−0.03	0.07

techniques. In all subfigures of figures 15 the dashed vertical line indicates a change of scale of the x-axis. The different time plots therefore appear more non-linear than they are.

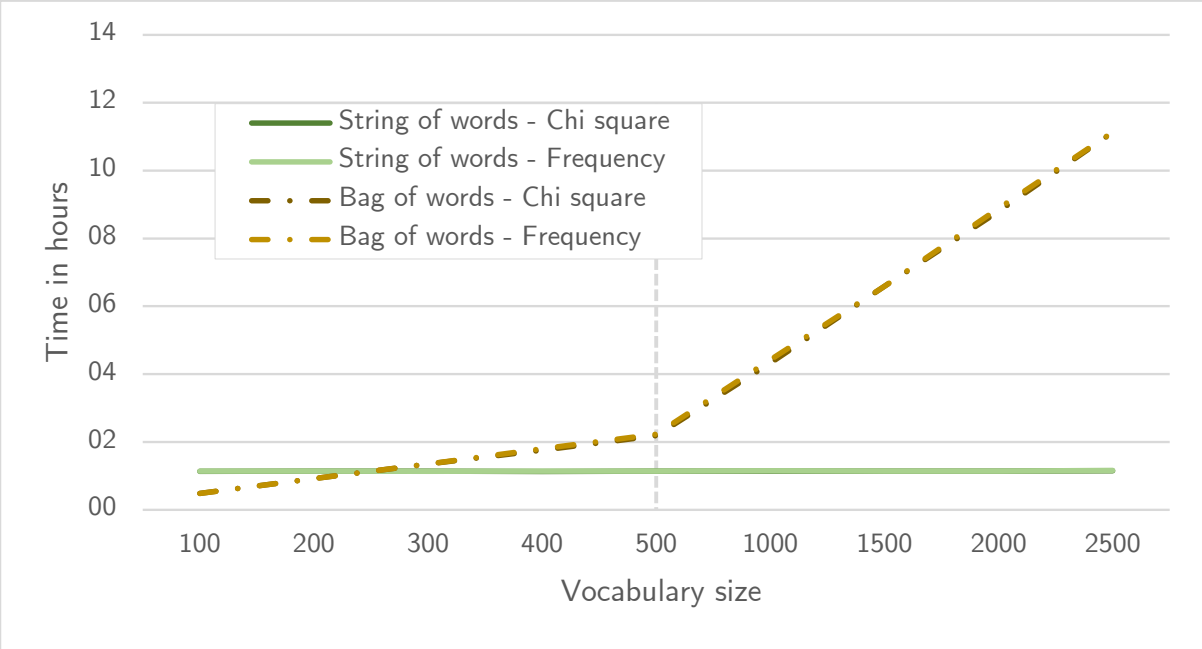
6.3 Truncation length and text length

Figure 16 shows MCC scores of SOW models for a range of text lengths and a number of truncation lengths. A red line indicates at which points the review lengths and truncation lengths are the same. The mean MCC value of the SOW models in which the review length was equal to or lower than the truncation length was 0.79 ($\sigma = 0.05$), the mean MCC score for model where the review length was higher than the truncation length was 0.61 ($\sigma = 0.11$).

(a) Preparation time.



(b) Training time.



(c) Total time.

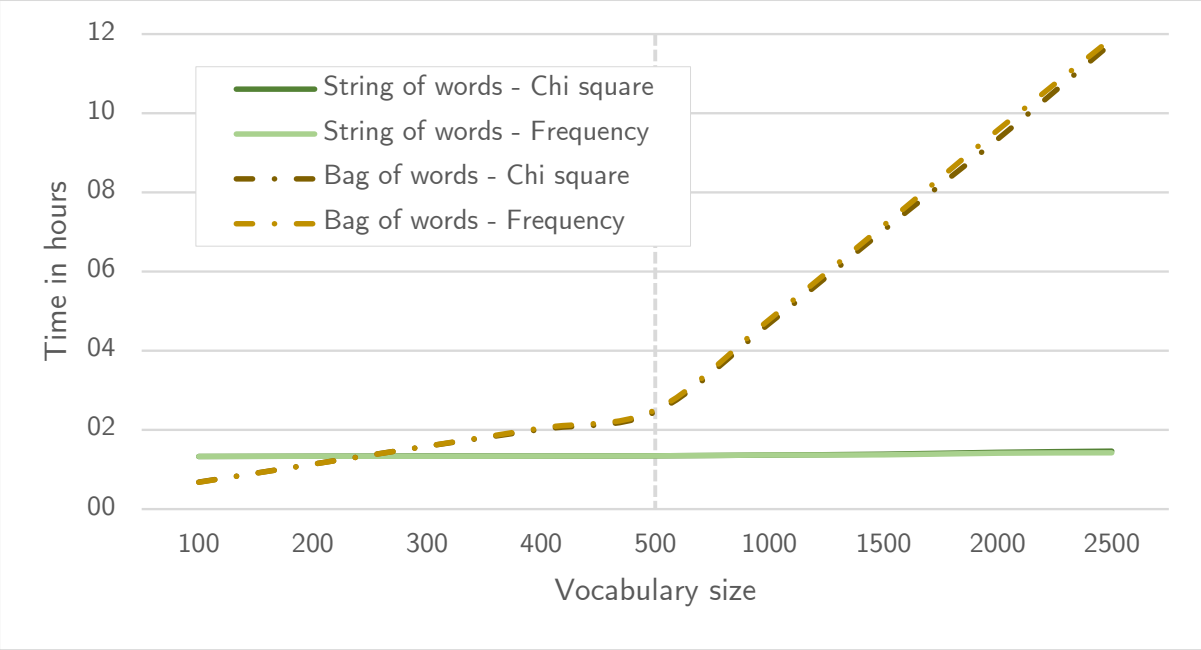


Figure 15. Preparation time (a), training time (b) and total time (c) for the bag of words and the string of words representation for both chi squared and frequency vocabulary buildup. The dashed vertical line denotes a change of scale in the x-axis.

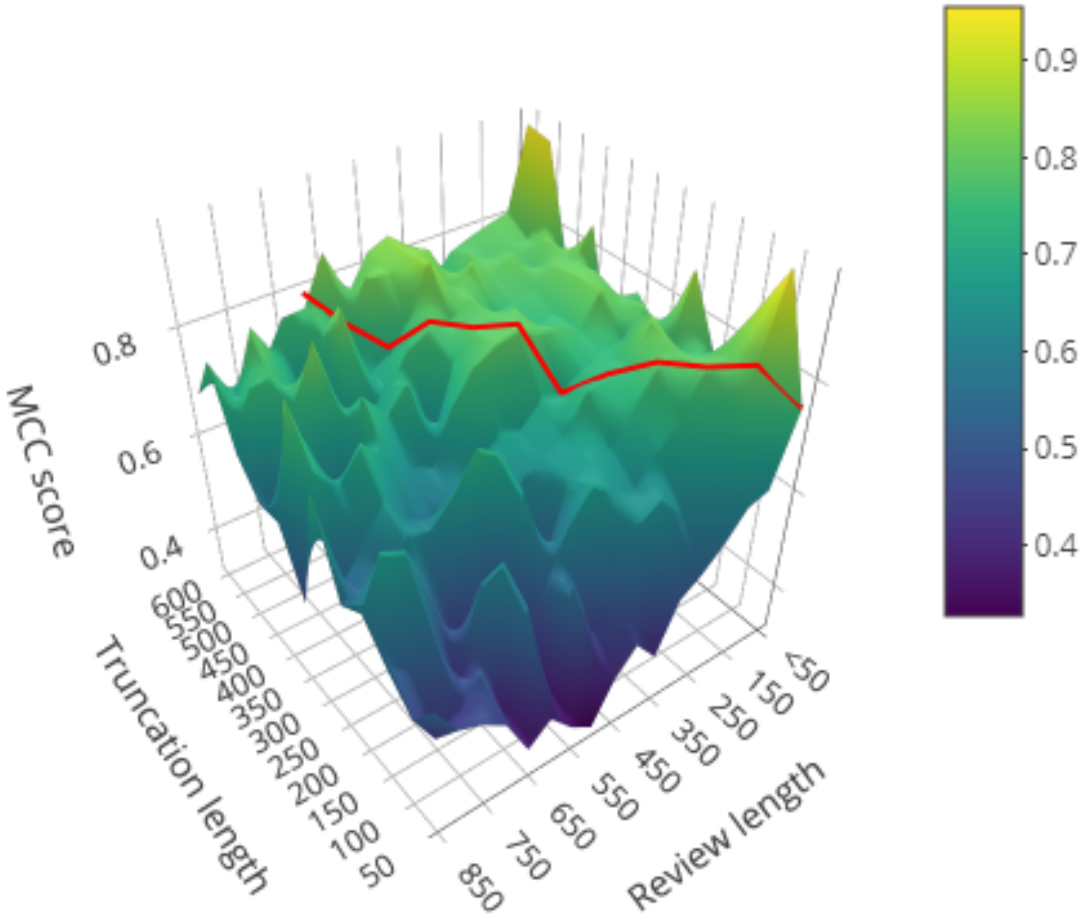


Figure 16. Plot of MCC scores of reviews of different lengths for different truncation lengths (the length at which point the SOW vector is truncated or until which point the vector is extended). The red line denotes the point at which the cutoff length and review lengths are the same.

7 Discussion

This thesis started out with the observation that the bag of words word representation does not afford word order to be used when it is used to represent text for machine learning models. Since humans do use word order as contextual cue to derive meaning from a sentence, natural language processing can be improved by observing human information processing. This notion is the very foundation of neural networks and enables us to make computers think more like we do. Using a word representation that lies closer to how humans deal with natural language, it is possible to use contextual information within sentences like humans do. The development of the string of words representation was meant to do just that, and from a human perspective comes closer to how we deal with words than the bag of words representation does. The main reasons for this are the retention of the original word order and the representation of the text in its original terms instead of in terms of all the words that are known.

7.1 Vocabulary types

The vocabulary building techniques differ only slightly and the small difference compared to the large credibility interval indicates that there is not enough evidence to conclude that there is a difference when averaged over both the bag of words and string of words word representations. If the two word representations are taken into account however, there appears to be a larger difference between frequency and chi square approach for the bag of words representation than there is for the string of words representation. The frequency approach averages 0.11 MCC lower than the chi square approach for the bag of words word representation. Here, the credibility interval of (-0.21;0.00) associated with the difference is large compared to the size of the difference, but does indicate that there is at least a 97.5% probability that models using the frequency approach yield worse MCC values than models using the chi square approach. There is no discernible difference between vocabulary building techniques for the string of words representation.

That being said, the chi square vocabulary resembles the interpretation humans give to language more than the frequency approach, as the information carried in the words that are used is more important than the frequency with which they are used. Furthermore, the implementation of the chi square vocabulary can be improved upon. Rare words may get a high chi-square score assigned to them if they are featured a small number of times

in one class and never in the other because the proportion of appearance between classes is high. This leads to false characteristicness of features and a disproportionately high placement in the chi-square vocabulary. The performance of the chi square vocabulary building technique can be improved by applying a filter that only allows words with a minimum amount of appearances in both classes to be used in the chi-square vocabulary.

7.2 BOW vs SOW

Both word representations and both vocabulary building techniques were trained with a number of different vocabulary sizes ranging from relatively large to relatively small. As the vocabulary size increases, the average MCC score increases for the string of words method but decreases for the bag of words method. Increasing the vocabulary size yielded a more effective classifier if the string of words representation was used but performance decreased if the bag of words representation was used.

Taking the vocabulary type into account shows that there is no interaction effect between vocabulary type and size, and the BOW representation consistently performs worse than the SOW representation across all tested vocabulary sizes regardless of vocabulary type. Figure 14, which shows MCC values of both bag of words and string of words approaches using both frequency and chi square vocabulary building techniques, reflects this observation as well: changing vocabulary size does not change the influence of vocabulary building technique on MCC score for either of the word representations.

The string of words method performs better overall and performs slightly better with larger vocabularies than with smaller ones. This makes the string of words more scalable than the bag of words methods the performance of which deteriorates as vocabulary size increases. Even for smaller vocabularies, the string of words representation outperforms the bag of words approach which indicates its usefulness in highly specialized language models as well. This is significant, as word embeddings produced by means of especially large scale bag of word models such as Word2Vec or GloVe are too generalized to be used in specialized language models.

7.3 Durations

For all but the smallest vocabulary sizes, the string of words representation was temporally more efficient for the dataset as used in this thesis. Using a corpus with a larger

average text size would, as long as the average text length is used as a cutoff point for vector length as was done in this thesis, result in a larger vector length of the string of words representation. This in turn increases the total time of preparing and training of the model and pushes the crossover point towards larger vocabulary sizes. This means that if a corpus with very large texts is used, it might be temporally more efficient to use the bag of words representation. Assuming the average lengths is used as a cutoff for vector size, the bag of words representation is expected to be more efficient when the average text length is larger than the vocabulary used. However, as larger average text sizes are usually accompanied by a larger vocabulary, the average text length will only be larger than the vocabulary size if exceptionally large texts or exceptionally sparse vocabularies are used. Therefore, the string of words representation can be considered, at least temporally, more efficient for the majority of applications.

7.4 Model performance

Even though the string of words models performed better than the bag of words one, MCC scores peaked around 0.725, which is not tremendously high. This lack of performance is expected to be largely due to the absence of optimizations in the preprocessing phase. Steps like normalizing spelling transformations, stemming, or lemmatization are often used to decrease the number of duplicate word features and increase performance.

Additionally, the lack of the expected significant performance difference between the frequency and chi-square vocabulary building techniques is expected to be due the lack of optimization of the chi-square method mentioned in 7.1.

The bag of words representation yielded poor results compared not only to the string of words representation but also to previous applications of the bag of words method. The embedding layer seems to have failed to capture closeness in meaning correctly, which means the data was not made into a denser vector representation properly. Failure of the embedding layer to properly transform the bag of words vectors into meaningful features may be caused by the relatively small dataset - at least small compared to for example Word2Vec, a word embedding trained on billions of words (Google, 2013). As vocabulary size, and with it BOW sparsity, increases, the performance deteriorates further (see figure 14).

The string of words did not suffer from this problem, because the word representation in

itself already is a dense representation containing word relation information. It can be directly used in a recurrent neural network and does not need to be transformed a feature vector in order to be effective. This direct usability can prove very useful if the dataset used is not large enough to train a vector representation based on closeness in meaning successfully, as was likely the case in the BOW model used in this thesis.

When comparing the embedding implementation as used in this thesis to a pretrained model such as the Word2Vec model developed by Google (2013), there are some notable differences. Models like Word2Vec are embedding layers trained using the bag of words or skip-gram (see Bishop (2011) and Google (2013)) representation. The main difference between the embedding layer used in this thesis and the Word2Vec model is that the Word2Vec model is trained on billions of words, whereas the embedding layer only had a few thousand words to train with.

However, that does not mean there is not a place for it in the world of deep machine learning. A disadvantage of a large generalized training dataset is that domain-specific classification problems are difficult to solve using datasets comprised of billions of words, because the features it was trained to recognize are too generic. For example, when making a natural language processing classifier specifically aimed at the vocabulary of the financial world, words like ‘buying’, ‘selling’, and ‘debt’ would be useful to differentiate. However, in an embedding like Word2Vec, these words will all closely relate to each other, as they are related to each other more closely than to most other words in the dataset. This results in all of these words being represented by a low amount of features, which would make distinguishing between them difficult. Additionally, unwanted relations such as that between the word ‘bank’ and the word ‘river’ might cause the classifier to make mistakes that would not occur if the classifier was trained on a domain specific corpus. Since these domain specific corpora might not be sufficiently large to train bag of words based models such as Word2Vec or the BOW model as used in this thesis to perform well, the SOW representation may be quite useful in these domain specific applications. As the SOW representation forms vector representations in which the dimensions correspond meaningfully to one another, it can be used as a feature vector without relying on an embedding layer.

7.5 Truncation length

The exploratory truncation length analysis appears to show a distinction between MCC scores of reviews that were equal to or shorter than the truncation length and MCC scores of reviews that were longer than the truncation length. The average MCC score of reviews shorter than the truncation length was 0.18 higher than the average MCC score of reviews longer than the truncation length, which is a significant difference.

The hypothesized ceiling effect of the review lengths whereby the texts truncated shorter were expected to perform equally well as the same length texts truncated longer was not found. The opposite seems to be the case, reviews shorter than the truncation length score within a small margin of 0.79 and scores decrease as review lengths increase further away from the truncation length. The truncation lengths used throughout this thesis was based on the average review length as consequences of truncation and extension were unknown.

This exploration seems to indicate that the effect of truncation is larger than the effect of extension, as the smaller reviews still perform well if the truncation length is large but MCC values drop quickly when reviews are truncated. The SOW representation is expected to work best if there is a clearly defined maximum text length that can be used as truncation length. Such clearly defined text lengths are found in many social media, even if they are not enforced by the medium itself. If no such clear maximum is available, the truncation length should at least be above the mean text length, so that the amount of truncated data will decrease and the amount of extensions will increase. To what extend the truncation length should be over the mean might be the subject of another study, as no clear conclusion can be drawn from the data obtained in this thesis.

7.6 Assumptions

As mentioned, the truncation length was defined by the average text length of the reviews because the detrimental effects of extending and truncating the text were assumed to be equal and because the distribution of review length was assumed to be symmetrical around the mean. Both these assumptions were violated. The effects of extending and truncating the reviews appear to not be equal as truncation leads to much lower MCC scores than extension. Additionally, text lengths were not symmetrical around the mean, but resembled a Poisson distribution which skewed the mean such that the influence

of extension was greater than that of truncation of the texts. In hindsight, with the conclusions drawn from the exploratory analysis of truncation lengths, this seems to have had a positive effect but should be taken into account when defining a truncation length for similar datasets.

7.7 Limitations

The study of sentiment analysis in general, this particular study and the representation proposed in this study all suffer some shortcomings. In this section, they will be discussed in turn.

7.7.1 Limitations of sentiment analysis in general. One problem with interpreting texts in an automated fashion is that, even if an algorithm succeeds in identifying a certain tone or sentiment in a piece of text, there might be more than one subject in the text that is not discernible from the main focus. Alternatively, the main focus of the text might not be what the algorithm expects. The subject of for example an IMDB review might contain some thoughts on a director or a certain actor. Phrases like 'This director is usually great and produces spectacular films, except this one.' contain a positive sentiment, about the director, but contain a negative appraisal of the particular film that is reviewed. This distinction is very difficult to identify, even with neural networks such as this one. In the introduction of this thesis, sentiment analysis was defined as 'the extraction of the author's feeling towards what is described in the text', but the insight of what exactly is described in the text remains, for now, very difficult to identify.

7.7.2 Limitations to this study. There are some practical improvements to be made to the model as it was used in this thesis. Some common natural language processing optimizations, such as filtering out certain part of speech (POS) tags or the aforementioned lack of filtering of the chi square vocabulary, were not implemented to their full potential. While the possibility of filtering based on part of speech tags was built in to the program used, no filtering was applied. This was partly due to a lack of insight into how different parts of speech influence the classification in specific setting of film reviews and partly because the entire philosophy of the string of words model dictates that words should be viewed in their proper context, which requires a minimal amount of filtering. That being said, there may very well be some POS filters that would make the model more efficient while retaining the proper context.

In other areas there might have been too much filtering. Punctuation and capitalization were removed in order to streamline the vocabulary building and implementation but this inevitably leads to a loss of information, both in the form of explicit information such as smileys and emotes as well as contextual information such as at which point sentences start and stop.

7.7.3 Limitations of the string of words model. The string of words representation does bring some limitations with it. Firstly, it requires a predefined vector length, which requires a priori information about the texts to be evaluated. If the truncation length is set too low, a lot of text would be discarded which has a detrimental effect on the classification performance. The truncation length can also not be set too high, as the efficiency advantage of the SOW representation would be lost. In this thesis, 13.9% of the reviews was truncated by more than 150 dimensions, which, as figure 16 shows, has a large impact on the MCC scores. If more heterogeneously sized texts were to be classified using the SOW method, either more information would be lost because of truncation of the reviews or the density advantage that the string of words representation has would be lost because of the large extensions that are necessary to fill a large vector size with a small text. Either option will decrease the added value that the string of words representation has over the bag of words representation, which does not suffer from these limitations. The string of words representation, therefore, works better on texts of which the lengths are known beforehand and have a small range of lengths. This sounds like a big limitation but such text sources are plentiful. For example, data streams from social media are often small texts, for some media this is guaranteed by a fixed character limitation.

7.8 Future research

For the corpus used in this thesis, the string of words representation performs better overall and is mostly faster to train than the bag of words representation. This is especially true if texts are on average smaller than the vocabularies used. Generally, corpora with very large texts make use of and benefit from a large vocabulary, which would make the string of words method preferable as the time- and performance penalty of the bag of words representation are much larger in this case. However, if the vocabulary is small and texts are large, the bag of words might be temporally more efficient. Looking at

figure 14, the performance may also converge if vocabulary sizes are kept down. It would be interesting to explore how text size and vocabulary size influence both performance and temporal efficiency in a similar fashion to the exploratory analysis of text size and truncation length as performed in this thesis. Defining ranges of text size and vocabulary size for which each text representation is optimal will be helpful for future research and practical applications.

The exploratory analysis of the effect of truncation length and review length on performance indicated that the effect of truncation seems larger than the effect of extension. This result indicates that the truncation length should be above the mean text length so as to minimize its impact. However, although no classification performance effect was found, the model becomes temporally less efficient as the truncation length increases. From the exploratory analysis, no definitive optimum truncation length could be found. A study of the extend of the impact on temporal performance as well as classification performance as truncation lengths increase may give insight into an optimal truncation length.

The tree LSTM model proposed by Socher et al. (2013) has proven to be suitable especially for negated sentences because of its ability to flip the entire estimated class at the occurrence of a negation. Since the string of words representation in combination with an LSTM recurrent neural network should be able to do the same, it is worth comparing the specific ability of the string of words model to correctly classify negated texts as was done in Socher et al. (2013) using their specific data of negated sentences. Since the structure with which the model evaluates texts differs entirely but both models use LSTM neural networks to classify the text, it should be interesting to compare performance and preparation and training times. Because of the different structures of the models and the very different way the models cope with word representation and processing, it is difficult to make predictions about how they will compare.

To keep as many variables as similar as possible when comparing the two word representations, the same neural network was used for both bag of words and string of words in this thesis. This means that the embedding layer that was present for the bag of words representation was also present in the string of words representation. The effect of the embedding layer on the string of words method is not known, but the additional step might have influenced the performance, positively or negatively. One of the major

positive aspects of the string of words representation is that it is not necessary to use an embedding layer, as the string of words representation forms a meaningful embedding of itself. A logical next step in the development of this representation would be to compare the performance of the SOW model without embedding layer. This will help make the model even more efficient while, it is expected, retaining similar performance.

8 Summary

In this thesis, a text representation for machine learning models was developed with the aim of improving the ability of such models to use contextual information from pieces of natural language. This representation is inspired by how humans process text: not as a collection of words but as words that relate to each other if put in a specific order. These word relations are lost if the often-used bag of words (BOW) representation is used but are retained in the string of words (SOW) representation developed in this thesis, as the SOW representation retains the original arrangement of the words in a text.

To test whether the SOW representation outperforms the BOW representation, machine learning models using both representations were tested. For both conditions, a Long Short Term Memory (LSTM) Recurrent Neural Network (RNN) was used. Performance and time measures were taken of the different models for two different vocabulary buildup types (chi square and frequency) and a range of vocabulary lengths (100 to 2500 words). Additionally, an exploratory analysis gave an impression of the effects that vector truncation length in the SOW representation and text length of the film reviews had on classification performance.

8.1 Conclusions

There was little difference between the chi square and frequency vocabulary types, although the chi square type performed marginally better. Because of the the theoretical advantages of the chi square vocabulary and the limitations in its application in this study, the use of the chi square vocabulary is advised over the frequency application.

The SOW representation outperformed the BOW representation both in performance and time measures independently of vocabulary type. For both temporal as well as classification performance, the gap between the BOW and SOW representation increased as vocabulary sizes increased. Because the SOW representation outperforms the BOW

one in this wide range of vocabulary sizes, it can be used for a very wide range of texts. Only when texts significantly exceed the size of the vocabulary can the BOW representation be computationally more efficient. However, as the average text length increases, the vocabulary that should be used will also increase in size. For this reason, SOW will outperform BOW for small texts lengths as well as longer ones, to the extend tested in this thesis, which is up to around one thousand words.

The exploratory analysis of the truncation length and the review length indicated that the effect of truncation seems larger than the effect of extension. Therefore, the SOW representation is expected to work best if there is a clearly defined maximum text length, as often found in many data streams that natural language programs are applied to, that can be used as truncation length. The truncation length should at least be above the mean text length, but the extend with which the truncation length should be over the mean is cannot be concluded from this study.

Additional future study directions include comparisons of the neural network as was used in this thesis with other types of models. One such model should be without embedding layer to see how well the SOW representation holds up as a meaningful embedding on its own and another should use a Word2Vec embedding to see how it compares to a properly trained embedding.

9 References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Bishop, C. M. (2011). *Pattern recognition and machine learning (information science and statistics).* Springer. Retrieved from `https://www.springer.com/us/book/9780387310732`

Bradley, A. P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, *30*(7), 1145–1159.

Chicco, D. (2017, dec). Ten quick tips for machine learning in computational biology. *BioData Mining*, *10*(1). Retrieved from `https://doi.org/10.1186/s13040-017-0155-3` doi: 10.1186/s13040-017-0155-3

Damien, A., et al. (2016). *Tflearn.* `https://github.com/tflearn/tflearn`. GitHub.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, *10*(7), 1895–1923.

Dreiseitl, S., & Ohno-Machado, L. (2002, oct). Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, *35*(5-6), 352–359. Retrieved from `https://doi.org/10.1016/s1532-0464(03)00034-0` doi: 10.1016/s1532-0464(03)00034-0

Englebienne, G. (2016). *Machine learning pattern recognition.* (Lecture notes from 'Basic machine learning and pattern recognition')

Fawcett, T. (2006, jun). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. Retrieved from `https://doi.org/10.1016/j.patrec.2005.10.010` doi: 10.1016/j.patrec.2005.10.010

Google. (2013). *word2vec: A tool for computing continuous distributed representations of words.* Retrieved from `https://code.google.com/archive/p/word2vec/`

Hardeniya, N. (2015). *Nltk essentials.* Packt Publishing Ltd.

Jurafsky, D., & Martin, J. H. (2017). *Speech and language processing.* (Draft of third edition. Available on: https://web.stanford.edu/ jurafsky/slp3/)

Jurman, G., Riccadonna, S., & Furlanello, C. (2012, aug). A comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE*, *7*(8), e41882. Retrieved from `https://doi.org/10.1371/journal.pone.0041882` doi: 10.1371/journal.pone.0041882

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lobo, J. M., Jiménez-Valverde, A., & Real, R. (2008, mar). AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography*, *17*(2), 145–151. Retrieved from `https://doi.org/10.1111/j.1466-8238.2007.00358.x` doi: 10.1111/j.1466-8238.2007.00358.x

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1* (pp. 142–150).

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval* (Vol. 1) (No. 1). Cambridge university press Cambridge.

McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *Aaai-98 workshop on learning for text categorization* (Vol. 752, pp. 41–48).

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems 14* (pp. 841–848). MIT Press. Retrieved from `http://papers.nips.cc/paper/2020-on-discriminative-vs-generative-classifiers-a-comparison-of-logistic-regression-and-naive-bayes.pdf`

Olah, C. (2015, 8). *Understanding lstm networks.* Retrieved from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

R Core Team. (2014). R: A language and environment for statistical com-

puting [Computer software manual]. Vienna, Austria. Retrieved from http://www.R-project.org/

RStudio Team. (2015). Rstudio: Integrated development environment for r [Computer software manual]. Boston, MA. Retrieved from http://www.rstudio.com/

Schmettow, M. (2015). Tutorial: Modern regression techniques for hci researchers. In J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, & M. Winckler (Eds.), *Human-computer interaction – interact 2015* (pp. 651–654). Cham: Springer International Publishing.

Schmettow, M. (2016). *Linear models.* http://rpubs.com/schmettow/lm.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631–1642).

van de Schoot, R., & Depaoli, S. (2014). Bayesian analyses: Where to start and what to report. *European Health Psychologist*, *16*(2), 75–84.

Appendix A

Background reading on probabilistic classifiers

Section 3 covers the buildup of neural networks, but this specific type of machine learning model is preceded by probabilistic models. These probabilistic models underlie many of the mathematical steps taken in neural networks, and these steps are often reflected in the neural network used in this thesis. Because of this reason, this appendix containing additional background information on some of the more popular probabilistic classifiers in included in this thesis.

Classification algorithms

Many different machine learning models exist, even specifically geared towards natural language processing. Since this thesis is centered around a classification problem, only classification algorithms are considered. Two types of machine learning classifiers, generative and discriminative, are discussed here. Examples of both types are given that will be relevant later on, as parts of these examples will be used in the model that is used to evaluate the bag of words and string of words text representations.

A.1 Generative classifiers

Generative models estimate a joint probability model $p(class, document)$ which is used to calculate the class that is most likely to have produced the document. The probability of a document given both the available classes is calculated and the class with the highest probability of producing that document is selected (Ng & Jordan, 2002).

$$Class = MAX(p(document|class_i)) \tag{23}$$

A.1.1 Bayesian classifiers. Bayesian classifiers are a type of generative model that, as the name implies, use Bayes' rule to calculate the most likely class given the document, from probabilities that can be calculated.

$$p(class_i|document) = \frac{p(document|class_i)p(class_i)}{p(document)} \tag{24}$$

The class i for which the probability is highest is the class that is selected. Because $p(document)$ is the same for both classes, the denominator is canceled out, simplifying

the formula to:

$$Class = MAX(p(document|class_i)p(class_i)) \tag{25}$$

The probability of $p(document|class_i)$ is given by $p(f_1, \ldots, f_n|class_i)$, where $f_1, \ldots, f_n$ refers to the words in a document. To make calculating this likelihood feasible, Bayesian models make strong assumptions about the data that is processed and not all of them are valid to presume in natural language data. The most prominent - and most violated - assumption is that of independence of features. This assumption is made because the likelihood $p(f_1, \ldots, f_n|class_i)$ in

$$Class = MAX(p(f_1, \ldots, f_n|class_i)p(class_i) \tag{26}$$

is computationally expensive to calculate, especially if texts, and therefore the set of words in a text $f_1, \ldots, f_n$, get larger. The independence assumption simplifies the formula as follows.

$$p(f_1, \ldots, f_n|class_i) = p(f_1|c) \cdot p(f_n|c) \cdot \ \ldots \ \cdot p(f_n|c) \tag{27}$$

The entire formula for the Bayesian classifier can then be simplified to:

$$Class = MAX(p(c) \prod_{1}^{f_n} p(f_i|c)) \tag{28}$$

This means that the model assumes different words in a sentence are independent of each other and that the probability of one word occurring does not relate at all with the probabilities of other words occurring in that sentence. This assumption does not hold, as words in a sentence must cohere somehow to form a semantically logical sentence. Despite the actual function approximation being poor, naive Bayesian models can classify natural language data relatively well (McCallum, Nigam, et al., 1998).

A.1.2 Training Bayesian classifiers. The Bayesian classifiers are trained based on maximum likelihood estimation. Frequency data is used to determine the probabilities of the documents given each class and the probabilities of the classes. This frequency data is obtained from a training dataset such as the collection of IMDB reviews used in this thesis.

The prior probability of each class is estimated by dividing the number of documents

of that class by the total number of documents.

$$p(class_i) = \frac{N(class_i)}{N(documents)} \tag{29}$$

The document probability given each class is given by $p(f_i|class_i)$. To estimate this, the number of times the word f_i is featured in each class is divided by the number of times each word in the vocabulary is featured in all documents of class i.

$$p(f_i|class_i) = \frac{N(f_i, class_i)}{V_{class_i}} \tag{30}$$

The Bayesian classifier uses frequency information from the texts to give a class prediction. This information can be neatly represented in an array of numbers, a vector, where each dimension represents a word in the vocabulary. As vocabularies consist of hundreds to thousands of words (see section 3.1), the vectors can get quite large. To illustrate how this representation works, consider the following mini-vocabulary:

$$[\text{ I, you, her, him, they, duck, goose, crouch, bend }] \tag{31}$$

If the sentence 'I saw her pet her duck' is represented in terms of frequencies of the words in the vocabulary, it would the yield the following vector:

$$[\ 1\ 0\ 2\ 0\ 0\ 1\ 0\ 0\ 0\ ] \tag{32}$$

In this vector, each dimension represents a word from the vocabulary, the values are determined by the frequencies in the text. The example vocabulary and text are both very small, but both can be of any length. However, the vector, and therefore the vocabulary, has practical limit as computation with longer vectors is computationally expensive. This way of representing natural language as frequencies of a vocabulary is called the 'bag of words' representation.

If a Bayesian classifier is trained with such a standard bag of words representation, it is called a multinomial Bayesian classifier. This is opposed to the Bernoulli classifier, which functions exactly the same but utilizes vectors of binary data in its training. Each piece of text is represented by a vector that indicates whether the words in that text are featured in a predefined set of words, the vocabulary. The example sentence 'I saw her

pet her duck' using the mini-vocabulary from formula 31 and the Bernoulli representation would look as follows:

$$[\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ ] \tag{33}$$

In this very small example, some information is already lost because only the presence is captured and the frequency is not. For longer examples, the loss of information due the binary nature of this representation is significant. The sparsity of information in the Bernoulli model causes it to mis-classify especially longer texts because of the presence of one word that the model has found indicative of a certain class. It can for example "[...] assign an entire book to the class China because of a single occurrence of the term China." (Manning, Raghavan, Schütze, et al., 2008).

A.2 Discriminative classifiers

Discriminative models directly estimate a the probability of the class given the document for both classes and select the class with the highest probability:

$$Class = MAX(p(class_i|document)) \tag{34}$$

A.2.1 Logistic regression classifier. The most-used discriminative classifier is the logistic regression classifier. Instead of using both the document likelihood for each class and the prior probability of the class, like the generative classifiers, it directly estimates the class given the document. This is done by taking the dot product of the bag of words vector V and a weights vector W and adding a bias B vector.

$$x = V \cdot W + B \tag{35}$$

To ensure that this formula yields a valid probability, it is squashed by a sigmoid function:

$$p(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{36}$$

The weights and bias vectors have the same shape as V and are initialized with small random initialization weights to prevent multiplication with a zero vector (Bishop,

2011). These weights and biases represent the slope and offset of x and therefore the shape of the sigmoid function in formula 36.

A.2.2 Training logistic regression classifiers. The logistic regression classifier also utilizes the bag of words representation like the bayesian classifier, the buildup of which was elaborated upon in section A.1.2. The weights vector is updated iteratively through gradient descent. At the start of training, the weight vector is filled with small random initiation weights. Functions 35 and 36 are applied and, using class with the highest probability, an expression of the amount of difference between the estimation and actual classification is made. This expression is known as a loss function.

To construct this loss function, we start with looking at our goal at this point in time: maximizing the probability that the classifier predicts the correct class given the document. The following expression can be used to express this probability in terms of $\hat{y}$, where $p(y|x)$ equals $\hat{y}$ if y equals 1, and $1 - \hat{y}$ if y equals 0. Because y and $\hat{y}$ can only have values 0 and 1, it follows[10] that if $\hat{y} = y$, $p(y|x)$ equals 1 and if $\hat{y} \neq y$, $p(y|x)$ equals 0 (Jurafsky & Martin, 2017).

$$p(y|x) = \hat{y}^y(1 - \hat{y})^{1-y} \tag{37}$$

To avoid computational underflow, the logarithm of function 37 is taken, which results in a log probability of the correct class. To make this function into a loss function, it is multiplied by -1. This finally yields the cross entropy loss function[11]:

$$Loss_{CE}(\hat{y}|y) = -log(p(y|x)) = -(y \cdot log(\hat{y}) + (1 - y)log(1 - \hat{y})) \tag{38}$$

Then, the gradient ∇ of the loss function for the used weights at time t (θ_t) is calculated, and weights are updated using the gradient, weighted by the learning rate parameter η,

---

[10]
$\hat{y} = y$:
$1^1(1-1)^{1-1} = 1$
$0^0(1-0)^{1-0} = 1$
$\hat{y} \neq y$:
$1^0(1-1)^{1-0} = 0$
$0^1(1-0)^{1-1} = 0$

[11] Some steps in the derivation of this formula have been skipped in the interest of brevity, and because the focus of this thesis lies with word representations and not loss functions. A more detailed walk-through can be found in Jurafsky and Martin (2017).

which yields the weights for time $t + 1$.

$$\theta_{t+1} = \theta_t - \eta \nabla Loss_{CE} \tag{39}$$

This process of updating weights is repeated until either a predefined number of steps has been taken or a predefined loss value has been reached. The weights can then be used in formula 35 substituted in formula 36 to produce the probability of the input vector V belonging to class 1.

Appendix B

Other word representation structures

Section 3 elaborates on the bag of words representation and its shortcomings. Alternatives to this representation have been developed, but have proven ineffective at solving the problems posed by the BOW representation without introducing more significant problems. Tree-like structures and n-gram structures are the most popular among those alternatives, and are elaborated upon here.

B.2 Tree-like structures

One such alternative was proposed by Socher et al. (2013), who have made a model that combines the tree like structure seen in decision trees[12] with modern recurrent neural networks. As the branches combine, the node evaluations are added up and a general evaluation is formed. An example of how this structure operates is shown in figure B1. The increased information that could be used from texts yielded a successful performance improvement, especially if entire sentence chunks were negated. In their model, each word in a sentence represents a branch on a decision tree and each node is evaluated individually. This method appears particularly suitable for classifying negated sentences as it is able to capture the negation and invert the meaning of a whole branch.

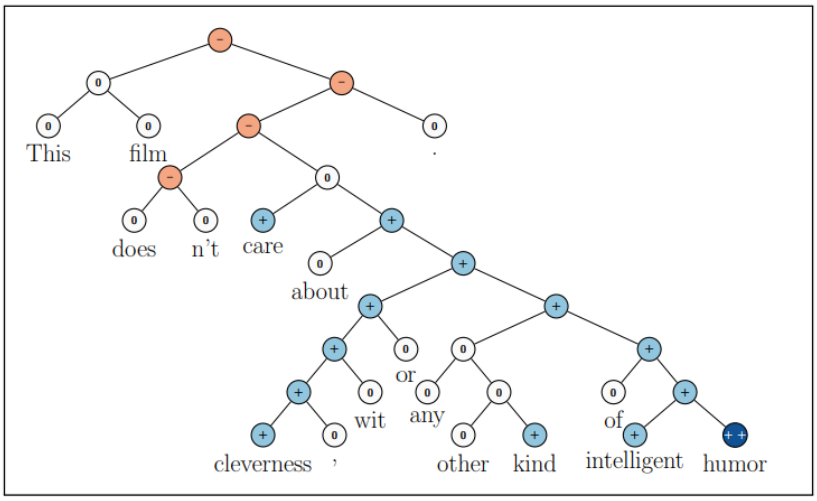


Figure B1. Example of the Recursive Neural Tensor Network accurately predicting 5 sentiment classes, very negative to very positive (- -, -, 0, +, + +), at every node of a parse tree and capturing the negation and its scope in this sentence. Figure from Socher et al. (2013).

[12] See Bishop (2011) or other machine learning literature.

Dreiseitl and Ohno-Machado (2002) name some disadvantages to decision tree structures that are also relevant to the solution to solving the word-arrangement problem as posed by Socher et al. (2013).

Firstly, the information is split per step; part of a vector goes to one downstream node, part of it goes to another. This may lead to a local optimum but globally deficient process. Socher et al. (2013) have attempted to remedy this by evaluating each node as it ascends through the tree structure in a bottom-up manner, and indicate that a looking-ahead algorithm might yield better results. Secondly, the classification of each node is binary, discretizing the data unnecessarily. Thirdly, the leaves of the model are assessed independently from one another, which means that despite the fact that the word order is kept in the representation of the text, the order is not taken into account when classifying the individual leaves.

Lastly, the fact that decision trees exponentially expand in size as text sizes increase is a significant problem. For one-sentence evaluations, the computational effort is acceptable but it rapidly becomes too computationally intensive to evaluate every branch on a tree. Because the tree is evaluated bottom-up and not top-down, methods like pruning that are normally used to make decision tree algorithms more efficient cannot be used.

B.3 N-grams and n-grams as bag of words

N-gram language models are developed with the same motivation as this paper: keeping context in mind when processing natural language. N-grams attempt this by applying maximum likelihood estimation to a set of word combinations instead of single words. N-grams are mainly used as a predictive algorithm, calculating the probability of a word given a history of a sentence. A Markov assumption is made here, which means that only a select number of words are used in the word prediction. The reason for this is that no corpus large enough exist to give the probability of for example the eleventh word in any ten word sentence. The number of words that is taken into account in n-grams is given by n. This includes the to-be-predicted word. Formula 40 shows the simplification of this Markov assumption, where the probability of a word given all previous words is approximated by $n + 1$ previous words.

$$p(w_k|w_1^{k-1}) \approx p(w_k|w_{k-n+1}^{k-1}) \tag{40}$$

As this thesis focuses on classifiers and not predictors, it is more useful to determine the probability of a word sequence, which can be achieved using the chain rule of probability:

$$p(w_1^k) \approx \prod_{k=1}^{n} p(w_k|w_{k-n+1}^{k-1}) \tag{41}$$

$p(w_k|w_{k-n+1}^{k-1})$ can be calculated by dividing the number of bigram occurrences by the number of occurrences of w_{k-n+1}^{k-1}.

$$p(w_k|w_{k-n+1}^{k-1}) = \frac{N(w_{k-n+1}^{k-1}w_k)}{N(w_{k-n+1}^{k-1})} \tag{42}$$

This sequence probability can then be used to calculate the probability of that sequence given a class, using counts from the appropriate trainingdata subset. The class that yields the highest sequence probability is the chosen class.

N-grams are able to take some context into account when evaluating classes, specifically n words of context. As n in $N(w_{k-n+1}^{k-1})$ gets larger, more context is used to evaluate the probability, but counts decrease tremendously. For this reason, n is rarely larger than three and even then requires a large corpus to capture often used expressions reliably. While n-grams are able to capture the difference between 'legendary' and 'legendary fiasco' as seen in figure 1b, it fails to capture more nuanced textual relations. Dependencies can easily be longer than a few words as can also be seen in figure 1b, where 'the expensive and professional look' caused the movie to be 'even more difficult to bear', even though 13 words came between the two expressions. N-grams therefore fail to capture many contextual relations that exist in our language.

B.4 N-grams as feature vector

It is possible to use n-grams as word features in a bag of words-like representation. This can be achieved by substituting a vocabulary for all known n-grams. A vocabulary, based on example 2 and made up of bigrams might then look as follows:

[[dog, jumps] [dog, sleeps] [dog, bites] [cat, sleeps] [wolf, howls] [fox, sports]
[cow, calf] [pig, pork] [jump, horse] [chicken, little] [jumping, over]
[jumps, over] [running, past] [runs, along] [sleeping, in] [sleeps, on]
[hunting, deer] [hunts, game] [slow, cooker] [drive, along] [fast, lion]] (43)

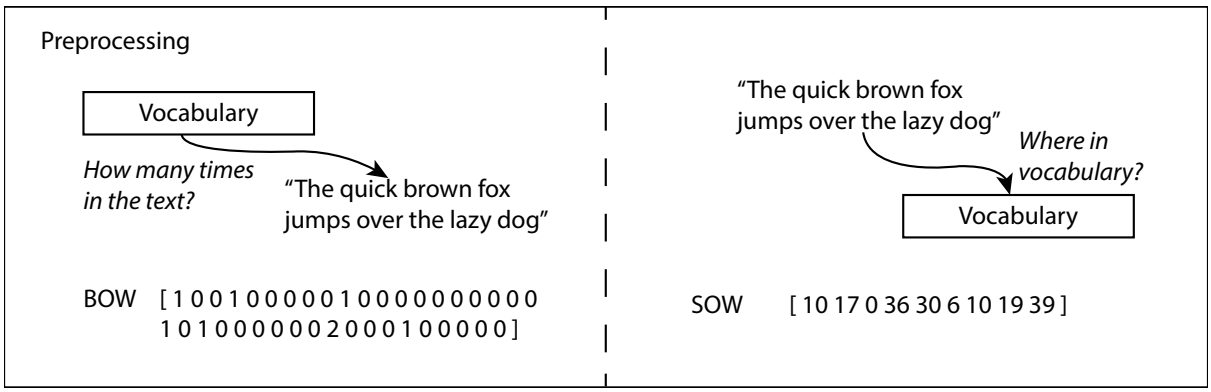
This quickly yields a number of problems, there are many more bigrams than unigrams, which yields in an even sparser representation than the regular bag of words would yield. This requires an extra filtering step to counter, such as only allowing bigrams that occur at least N amount of times in the training dataset to be added to the vocabulary.

Additionally, the probability of encountering the bigrams in new data is smaller than encountering unigrams, which means a larger vocabulary is required to capture a decent number of bigrams in a new text. These problems increase tremendously in size as n in the n-grams increases.

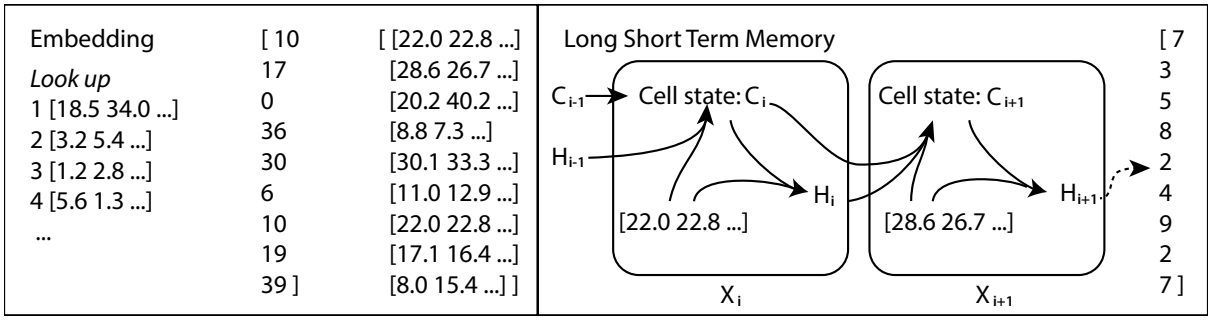
Lastly, the initial problem of n-grams of not being able to capture even moderately long dependencies in natural language is not resolved.

Appendix C

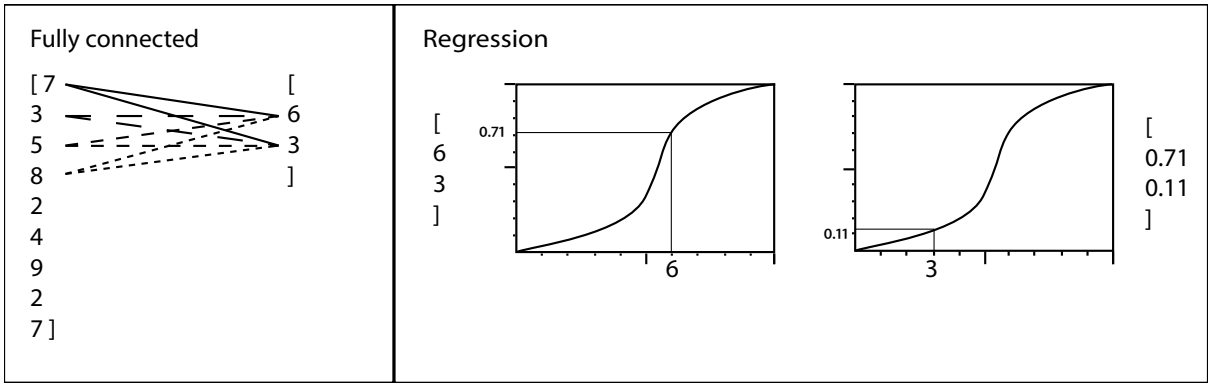
Worked out example of the model



(a) Preprocessing steps. The Bag of Words (BOW, left) indicates vocabulary frequencies of words from reviews. The String of Words (SOW, right) indicates vocabulary indices for each word in a review.



(b) The embedding layer (left) maps input dimensions to output vectors. The LSTM layer recursively iterates through its input (X_i). Each iteration feeds the next (H_i) while a cell state (C_i) provides long term memory.



(c) The fully connected layer (left) maps each input dimension to a weighted output dimension. The regression layer (right) maps this output to valid probabilities.

Figure C1. Flow of a single text vector through the preprocessing and neural network.

Appendix D

R markdown

From bags of words to strings of words

Luc Schoot Uiterkamp, 2019

This document shows, step by step, the process that was used to generate the results in the thesis this document accompanies. During training of the various models that are compared in the thesis, an MCC score (along with some other measures) was generated for each unique combination of parameters. This results in a dataframe containing the vocabulary size, mode (BOW or SOW), vocabulary type (chi square or information gain based), the truncation length for the SOW method, the matthews correlation coefficient and the time based measures: total time, training time and preparation time.

This data is imported and, of the SOW data, only the data labeled 'inverted' is selected. Other data is discarded as the inverted is always better.

```
load(file = "all_data.rda")

inverted_data <- all_data[(all_data$`sow_mode`=='inverted'|all_data$mode=="bow"),]
```

Performance measures other than MCC are discarded

```
MCC_data <- inverted_data %>%  select(vocab_size, mode, vocab_type, max_len, mat_co,  total_time, train_time, p
rep_time)
```

We see that for each condition there are nine vocabulary sizes. The truncation length is only appropriate for the BOW mode, hence they only vary in SOW conditions.

```
MCC_data %>%
  group_by(mode, vocab_type, max_len) %>%
  summarize(count = n())%>%
  kable() %>%
  kable_styling()
```

mode	vocab_type	max_len	count
bow	chi	100	9
bow	freq	100	9
sow	chi	100	9
sow	chi	250	9
sow	chi	400	9
sow	freq	100	9
sow	freq	250	9
sow	freq	400	9

We firstly look into the effect the vocabulary type has on the average MCC score for both bag of words and string of words methods.

```
gaussian_mcc_type_sow <-
  MCC_data[MCC_data$mode=="sow",] %>%
  brm(mat_co ~ vocab_type,
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )

gaussian_mcc_type_bow <-
  MCC_data[MCC_data$mode=="bow",] %>%
  brm(mat_co ~ vocab_type,
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )
```

Firstly for the effect of vocabulary type on MCC score of the SOW representation:

```
fixef(gaussian_mcc_type_sow)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.6317458	0.0191956	0.5934101	0.6694181
vocab_typefreq	-0.0043535	0.0276545	-0.0586410	0.0512611

Secondly for the effect of vocabulary type on MCC score of the BOW representation:

```
fixef(gaussian_mcc_type_bow)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.1948482	0.0399209	0.1139151	0.2686402
vocab_typefreq	-0.1148471	0.0561993	-0.2240577	-0.0014369

The main effect that this thesis measures is that of the representation of on the MCC scores, which the following model predicts.

```
gaussian_mcc_mode_fix <-
  MCC_data %>%
  brm(mat_co ~ mode,
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )
```

The effect of using BOW vs SOW method on the MCC score

```
fixef(gaussian_mcc_mode_fix)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.1382795	0.0245154	0.0907414	0.1864216
modesow	0.4916274	0.0281428	0.4370082	0.5459626

As vocabulary size impacts the amount of information that is available to the model, the effect of vocabulary size on MCC score is tested. As diminishing returns are expected, vocabulary size is considered as a random effect.

```
model_mcc_mode_ran <-
  MCC_data %>%
  brm(mat_co ~ (mode|vocab_size),
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )
```

```
fixef(model_mcc_mode_ran)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.4115169	0.0478219	0.3064865	0.4925836

```
ranef(model_mcc_mode_ran)%>%
  kable() %>%
  kable_styling()
```

	Estimate.Intercept	Est.Error.Intercept	Q2.5.Intercept	Q97.5.Intercept	Estimate.modesow	Est.Error.modesow	Q2.5.modesow	Q97.5.modesow
100	-0.0837835	0.0498432	-0.1811224	0.0182089	0.1544231	0.0603239	0.0387794	0.2719768
200	-0.2032876	0.0562924	-0.3194282	-0.0913361	0.3529425	0.0592173	0.2369163	0.4670209
300	-0.2194593	0.0582164	-0.3311695	-0.0991008	0.4093053	0.0593116	0.2929554	0.5254565
400	-0.2323449	0.0594319	-0.3415309	-0.1015043	0.4380537	0.0590083	0.3186462	0.5529144
500	-0.2941352	0.0634897	-0.4156406	-0.1571016	0.5298762	0.0592025	0.4150343	0.6493287
1000	-0.3321298	0.0668179	-0.4614832	-0.1939783	0.5966410	0.0586873	0.4806105	0.7107325
1500	-0.3224811	0.0664552	-0.4464431	-0.1816974	0.5863885	0.0592115	0.4718113	0.7065519
2000	-0.3477035	0.0706594	-0.4820157	-0.1991689	0.6274227	0.0606668	0.5084321	0.7526425
2500	-0.3750828	0.0723196	-0.5157052	-0.2198012	0.6635431	0.0595827	0.5455775	0.7828722

To detect a possible interaction between vocabulary type and vocabulary size, they are both taken into account, again separate for BOW and SOW methods:

```
gaussian_mcc_sow_vocab_type_vocab_size <-
  MCC_data[MCC_data$mode=="sow",] %>%
  brm(mat_co ~ (vocab_type|vocab_size),
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )

gaussian_mcc_bow_vocab_type_vocab_size <-
  MCC_data[MCC_data$mode=="bow",] %>%
  brm(mat_co ~ (vocab_type|vocab_size),
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )
```

The fixed and random effects of the vocabulary type taking into account the vocabulary size on MCC score for the SOW representation:

```
fixef(gaussian_mcc_sow_vocab_type_vocab_size)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.6297374	0.0310841	0.5697546	0.6886854

```
ranef(gaussian_mcc_sow_vocab_type_vocab_size)%>%
  kable() %>%
  kable_styling()
```

	Estimate.Intercept	Est.Error.Intercept	Q2.5.Intercept	Q97.5.Intercept	Estimate.vocab_typefreq	Est.Error.vocab_typefreq	Q2.5.vocab_typefreq	Q97.5.vocab_typefreq
100	-0.1168569	0.0439946	-0.2072627	-0.0337760	-0.0193655	0.0334452	-0.1008120	0.0345114
200	-0.0596405	0.0391630	-0.1408846	0.0126677	-0.0134996	0.0254472	-0.0770526	0.0268767
300	-0.0155286	0.0381227	-0.0917731	0.0617925	-0.0055948	0.0222190	-0.0599457	0.0349546
400	0.0002890	0.0382802	-0.0728212	0.0771593	-0.0020053	0.0208851	-0.0527346	0.0409876
500	0.0153396	0.0386800	-0.0582595	0.0932562	0.0015555	0.0211042	-0.0428146	0.0503739
1000	0.0372505	0.0392906	-0.0354418	0.1196766	0.0065854	0.0230472	-0.0360037	0.0615751
1500	0.0409547	0.0398822	-0.0315129	0.1206821	0.0078292	0.0227912	-0.0317030	0.0642037
2000	0.0512184	0.0396869	-0.0212583	0.1315761	0.0082323	0.0235779	-0.0365278	0.0650184
2500	0.0517468	0.0400298	-0.0202404	0.1331941	0.0096091	0.0238253	-0.0320111	0.0666688

The fixed and random effects of the vocabulary type taking into account the vocabulary size on MCC score for the BOW representation:

```
fixef(gaussian_mcc_bow_vocab_type_vocab_size)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.1074378	0.0519667	0.0151028	0.2110958

```
ranef(gaussian_mcc_bow_vocab_type_vocab_size)%>%
  kable() %>%
  kable_styling()
```

	Estimate.Intercept	Est.Error.Intercept	Q2.5.Intercept	Q97.5.Intercept	Estimate.vocab_typefreq	Est.Error.vocab_typefreq	Q2.5.vocab_typefreq	Q97.5.vocab_typefreq
100	0.2410035	0.1394582	-0.0094605	0.4309154	-0.1404884	0.1091457	-0.2950383	0.0676543
200	0.1384877	0.1051692	-0.0561670	0.2928692	-0.1449133	0.1089585	-0.2860082	0.0461262
300	0.1336231	0.1010675	-0.0507513	0.2877911	-0.1078625	0.0895739	-0.2403265	0.0603643
400	0.0873827	0.0803999	-0.0628536	0.2287272	-0.0291441	0.0578495	-0.1454859	0.0907440
500	0.0260060	0.0675546	-0.1165024	0.1494469	-0.0434871	0.0613095	-0.1576846	0.0885022
1000	-0.0052312	0.0657204	-0.1509354	0.1167945	-0.0318308	0.0589219	-0.1384951	0.0999261
1500	0.0471597	0.0780234	-0.1163695	0.1637549	-0.1012167	0.0917799	-0.2616405	0.0583830
2000	-0.0131524	0.0637395	-0.1555681	0.0986335	-0.0320742	0.0578307	-0.1443953	0.0950755
2500	-0.0717901	0.0661412	-0.2125659	0.0485102	0.0113607	0.0578623	-0.1171554	0.1334999

The amount of information that the model is able to use, as well as the dilution of this information, is determined by the truncation length of the SOW representation. Therefore the effect of the truncation length on the outcome scores is tested:

```
gaussian_max_len <-
  MCC_data[MCC_data$mode=="sow",] %>%
  brm(mat_co ~ (1|max_len),
      family= gaussian(),
      prior = NULL,
      autocor = NULL,
      data = .
      )
```

```
fixef(gaussian_max_len)%>%
  kable() %>%
  kable_styling()
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	0.635525	0.1041829	0.400142	0.8642575

```
ranef(gaussian_max_len)%>%
  kable() %>%
  kable_styling()
```

	Estimate.Intercept	Est.Error.Intercept	Q2.5.Intercept	Q97.5.Intercept
100	-0.0872745	0.1049929	-0.3199207	0.1461327
250	0.0195914	0.1048579	-0.2030699	0.2534364
400	0.0507097	0.1047092	-0.1769586	0.2886878

To visualize the relationship between MCC values, the truncation length and the review lengths, a three-dimensional plot is made. Here, the MCC scores are given for intervals of review lengths for 12 values of truncation lengths. A red line is traced to indicate the line in the graph where review length and truncation length are equal.

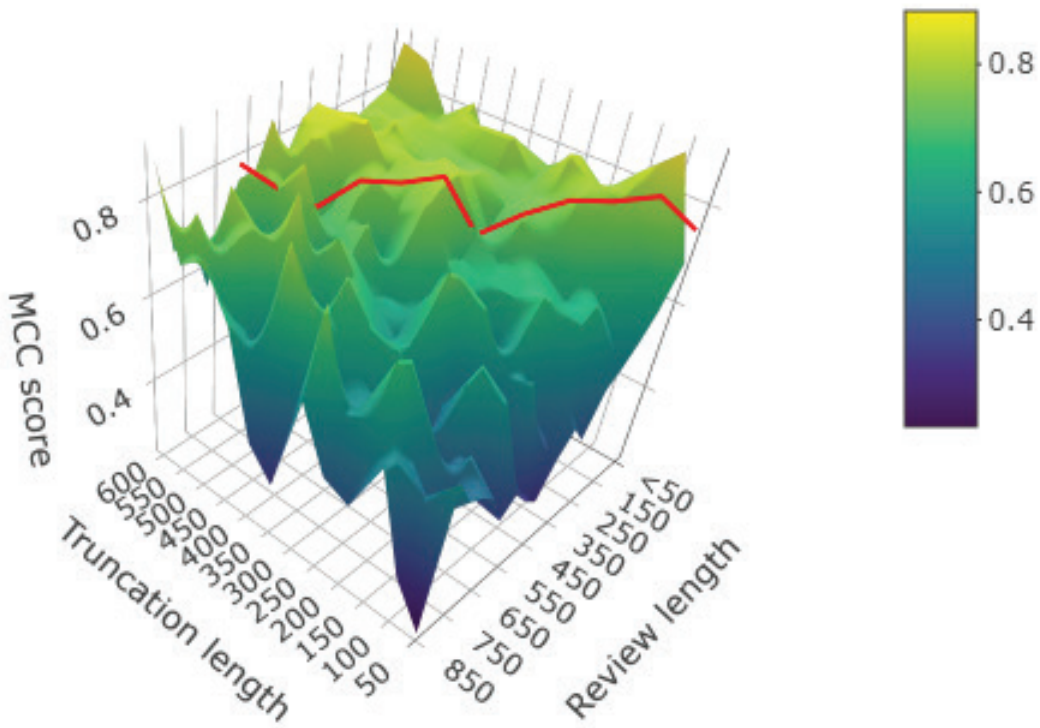
```
load(file = "3dplotdata.rda")
library(plotly)
```

define labels for axes

```
ticklabelsX = c("50", "100", "150", "200", "250", "300", "350", "400", "450", "500","550","600")
ticklabelsY = c("<50","150","250","350","450","550","650","750","850")
```

```
plot <- plot_ly(z = data.matrix(X3dplot, rownames.force = NA)) %>% add_surface() %>%
  layout(
    scene = list(
      xaxis = list(title = "Truncation length", tickvals=c(0,1,2,3,4,5,6,7,8,9,10,11), ticktext = ticklabelsX),
      yaxis = list(title = "Review length", tickvals=c(0,2,4,6,8,10,12,14,16), ticktext = ticklabelsY),
      zaxis = list(title = "MCC score"),connectgaps = TRUE
    )) %>%
  add_trace(x = c(0,1,2,3,4,5,6,7,8,9,10,11), y = c(0,1,2,3,4,5,6,7,8,9,10,11),
            z = 0.008+c(0.750390977,0.80709865,0.788803939,0.78190150,0.745792762,0.694842112,0.802713866,0.782
407223,0.77841065,0.717059754,0.745244273,0.787335271),
            type = 'scatter3d', mode = 'lines',
            line = list(width = 4, color = 'red', shape='spline'))

plot
```



Appendix E

List of unused reviews

```
/train/pos/10561_8.txt
/train/pos/1255_10.txt
/train/neg/7547_1.txt
/train/neg/1218_4.txt
/train/neg/7226_3.txt
/train/neg/505_2.txt
/train/neg/9545_4.txt
/train/neg/12484_4.txt
/train/neg/10100_1.txt
/train/neg/5494_3.txt
/train/neg/9611_3.txt
/train/neg/6697_2.txt
/train/neg/1399_3.txt
/test/neg/2298_4.txt
/test/neg/4970_3.txt
/test/neg/6224_1.txt
/test/neg/11175_2.txt
/test/neg/8360_3.txt
/test/neg/5914_4.txt
/test/neg/1035_3.txt
/test/neg/6835_1.txt
/test/neg/2237_1.txt
/test/neg/9909_1.txt
/test/neg/7683_4.txt
/test/neg/6414_2.txt
/test/neg/5420_4.txt
/test/neg/3422_4.txt
/test/neg/1409_4.txt
/test/pos/10320_10.txt
/test/pos/9967_10.txt
/test/pos/3569_10.txt
/test/pos/8174_9.txt
/test/pos/3959_10.txt
/test/pos/11494_8.txt
/test/pos/6809_8.txt
/test/pos/1707_8.txt
/test/pos/9499_7.txt
```

Appendix F

POS tags

All POS tags found in the reviews used in this thesis after deletion of punctuation and reviews with non-utf8 encoded characters in order of prevalence.

```
NN: noun, common, singular or mass
IN: preposition or conjunction, subordinating
DT: determiner
JJ: adjective or numeral, ordinal
RB: adverb
NNS: noun, common, plural
VBZ: verb, present tense, 3rd person singular
PRP: pronoun, personal
VB: verb, base form
CC: conjunction, coordinating
VBD: verb, past tense
VBP: verb, present tense, not 3rd person singular
TO: "to" as preposition or infinitive marker
VBN: verb, past participle
VBG: verb, present participle or gerund
PRP$: pronoun, possessive
CD: numeral, cardinal
MD: modal auxiliary
WP: WH-pronoun
WRB: Wh-adverb
WDT: WH-determiner
RP: particle
JJS: adjective, superlative
JJR: adjective, comparative
EX: existential there (there is)
RBR: adverb, comparative
PDT: pre-determiner
RBS: adverb, superlative
FW: foreign word
NNP: noun, proper, singular
WP$: WH-pronoun, possessive
UH: interjection
$: currency indicator
NNPS: noun, proper, plural
SYM: symbol
POS: genitive marker
```