Opinion Mining of Danish Social Media

Joachim Klokkervoll, Emil Kongsgaard, Samuel Nygaard,
Mike Pedersen, Lynge Poulsgaard, and Christian Slot
{jklokk12, ekongs12, snpe12, mipede12, lkpo12, cslot12}@student.aau.dk

May 31, 2016

Aalborg University
Faculty of Engineering and Science
Department of Computer Science
8th Semester project
Supervisor: Peter Dolog

The report is freely accessible, but publication (with references) is only allowed with permission from the authors.

Abstract

Opinion mining of written text is useful in many areas, e.g. in the retail industry, but is not a trivial problem. Sentiment analysis using different lexicon- and machine learning-based method for English text have been widely researched and accepted. However, only a few studies are concerned with sentiment analysis for text written in Danish, whereof none have explored a machine-learning based approach. Two labeled datasets containing reviews written in Danish have been investigated in this project. The first dataset contains 2,611,777 Trustpilot reviews concerned with the general consumer experiences of various Danish companies. The second dataset contains 11,329 product reviews from the webshop of the Danish electronics retailer Elgiganten. Both datasets were filtered to exclude non-Danish reviews and hereafter converted to paragraph vectors using the Doc2Vec model. As the datasets are both multiclass labeled and imbalanced we propose a modified and balanced root-mean-square error (BRMSE) for evaluating classification. The reviews from the first dataset were used to train various machine-learning models, while the reviews from the second training set were used for evaluation. Several classification methods have been used in this project including Support Vector Machines (SVMs), logistic regression, and perceptron. All of these methods where optimized using Stochastic Gradient Descent (SGD). The final result shows that cross-validating the Trustpilot dataset gives a BRMSE of 0.95 and a BRMSE of 1.32 when applying the best model to the Elgiganten dataset. We found that BRMSE is a suitable metric for evaluating classification of imbalanced multiclass labeled data. We also conclude that using a machine learning-based approach to sentiment analysis of Danish text is viable.

Contents

1 Introduction

Opinion mining is an invaluable tool in observing trends and identifying social media movements. Ever since the inception of media, certain people have been able to express their personal opinions about anything, be it politics, fashion, consumer wares, and everything in between. Before the introduction of the World Wide Web (WWW) in 1989 it was required to be well-educated, of good social status, or otherwise influential to have your opinion expressed to the masses through television, radio, and newspapers. With the digitalization of media and the subsequent creation of social media, everyone could now express their opinion on equal terms. Even more interestingly, these opinions are now available for gathering and aggregation in order to inform political parties and private companies about the public opinions of voters and consumers. Processing and understanding public opinion has thus gained a lot of attention within the field of machine intelligence under the term *opinion mining*.

As social networks grew larger, they quickly became the main source of opinion data, as they provided a platform for sharing information, ideas, and opinions with friends, coworkers, and even strangers. Every day countless texts containing sentiment is posted online, each sentiment directed towards a subject. Identifying the sentiments and their subjects in natural text is an ongoing field of study with a large interest to political and private actors. Most research in *sentiment analysis* have focused on the English language, but we will look into extending or applying existing work to Danish text, in the interest of the Danish business intelligence company NORRIQ.

In short, we will look into possible approaches to Danish sentiment analysis within the field of machine intelligence. We will explore existing techniques and their application in the Danish domain while considering areas of possible improvement.

1.1 NORRIQ collaboration

As a part of this project, a collaboration with the Danish IT consultant company NORRIQ was established. NORRIQ provides several different IT solutions for businesses in most industries e.g. CRM, ERP, and BI solutions [1]. NORRIQ's customers, mainly in the retail industry, sees potential in opinion mining of social media data on individual products and brands to improve marketing and furthermore to identify social media movements.

NORRIQ has provided a product catalog from three Danish retail companies for possible detection of product opinions in social media data.

1.2 Social media data

In collaboration with NORRIQ, the project is focused on opinion mining of social media data, with a secondary priority on identifying public opinion of specific products from the provided product catalog. Figure 1.1 shows the top six social networks in Denmark and the number of registered Danish users as of April 2015.

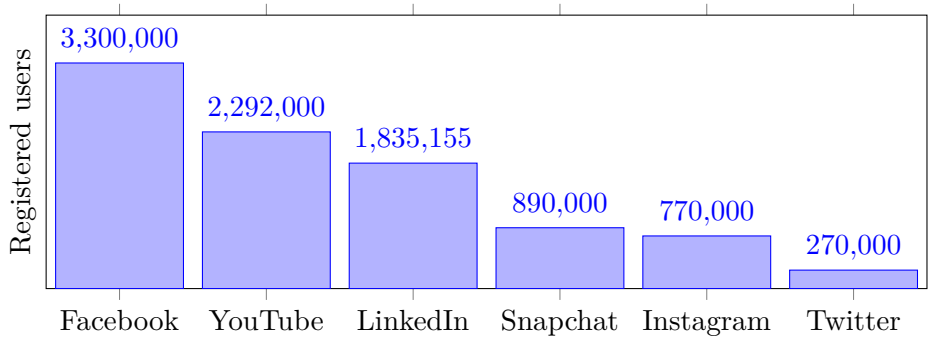


Figure 1.1: Use of social media in Denmark, April 2015 [2]

Because YouTube, Instagram, and Snapchat mainly focus on video or image data, they are not relevant for the scope of this project. LinkedIn is also not interesting since it is a business-oriented social network mainly used for professional networking and therefore unlikely to contain opinions in the interest of NORRIQ's customers.

As Facebook is the social network with most registered users in Denmark, it would be very interesting for this project to investigate data from Facebook. However, due to limited resources and privacy rules, buying or scraping data from Facebook is not an option for this project. These problems led us to research one of the less popular social networks in Denmark, namely Twitter.

Twitter has a free API for retrieving tweets and can be searched with keywords, e.g. the name of a company, brand, or product, but with several limitations for non-paying

users. The biggest limitation is that tweets older than 14 days cannot be retrieved. After collecting tweets for some time, we realized that the data from Twitter would need to be filtered intensely due to many non-Danish results in the API and the high amount of spambots. When considering these downsides, combined with the fact that Twitter only has half a million users in Denmark, we chose not use Twitter data for this project.

Due to the trouble with collecting data from social networks, we directed our focus towards other social media sources such as consumer review platforms with the biggest in Denmark being Trustpilot. Trustpilot reviews are interesting for this project scope since the data is labeled with a rating (user defined satisfaction-score) related to a company. As the customers of NORRIQ are also interested in product specific reviews, we also examined the Danish retail chain Elgiganten's webshop, which allows customers to review individual products.

1.3 Problem statement

The central problem explored in this project is partly provided by NORRIQ:

How well can we determine sentiment of Danish social media data?

More specifically, it aims to answer whether a model can be constructed to algorithmically predict the sentiment of reviews to aid manufacturers and retailers in their decision making, and whether this generalizes to other Danish textual domains. Positive feedback through product reviews could inform the product manufacturer that the product is being well received without the need of an in-depth and costly manual analysis. In the same manner, overwhelmingly negative feedback through product reviews could be used to decide whether it would be beneficial to change the design or even stop production of a product.

1.4 Related work

In this section we will explore some of the related work that has been done within the topic of sentiment analysis. We will look at different methodologies, namely lexicon oriented and machine learning oriented sentiment analysis. Articles looking into the issues with imbalanced datasets for sentiment analysis will also be explored. Paragraph vectors present us with new ways of constructing features for sentiment analysis and we will thus examine approaches using it for feature construction. Lastly, the paper explaining the Danish sentiment measure AFINN score will be described.

Machine learning-based methods can be used for sentiment classification as described by Pang et al. [3] who focus on three machine learning methods for classifying movie reviews written in English: naive Bayes, Maximum Entropy, and Support Vector Machines (SVMs). They use a movie review dataset from IMDB containing reviews from several different IMDB users. The total dataset contains only 752 negative and 1,301 positive reviews due to a restriction of 20 reviews per review

author to avoid a small number of prolific reviewers to dominate the corpus. The features are selected to try out different combinations of frequency versus presence of words, unigrams versus bigrams, using positional information, and using only adjectives. Notably it is found that presence outperforms frequency, and that using unigram and position information together yields results slightly worse than only using unigram features, thus indicating that the positional information is redundant and leads to overfitting. Depending on the configuration of features and choice of classifier, the results vary, but it is clear that SVM performs best in general with a recall of 82.9 % for the best configuration. The biggest problem with this research paper is the size of the dataset. The fact that it only contains 2,053 reviews could have influenced the final result. Having that downside in mind, this paper is still very interesting for this project because it shows that SVMs are relatively good for classifying sentiment in reviews.

Lexicon-based methods are alternative methods for sentiment analysis. Musto et al. [4] compares different state of the art lexicon-based methods tested on two different datasets, both containing twitter data. The first dataset is the Stanford Twitter Sentiment which contains 1,600,000 tweets. The dataset is split into a test and training set by the source, with a test set size of only 359 tweets. The tweets are automatically labeled according to the emoticons they contained. The other dataset, SemEval-2013, used in the article contains 14,435 manually labeled tweets and is split into a 66/33 % training and test set. The tweets are randomly selected and has no specific domain. The negative and positive weights of words in the lexicon was learned in a greedy manner. For the Stanford dataset, 10,000 random tweets were used to learn these weights, whereas the entire dataset was used for SemEval. The various lexicon-based methods performed differently, but our outtake is that for the SemEval dataset, the method performing best had an accuracy of 58.99 %. For the Stanford dataset, the method performing best had an accuracy of 74.65 %. This leads us to believe than a lexicon-based approach could be viable.

Lexicon- and machine learning-based methods in combination is described by Nakov et al. [5] who analyzed a competition with two tasks in finding sentiments, both of which had results that are interesting for the topic of this project. The dataset for the competition consisted of data gathered from Twitter and SMS data. The datasets are fairly balanced, slightly leaning towards positive data instances. The first task is a classic sentiment analysis of positive versus negative sentiment in a piece of text. The competition accepted both machine learning-based methods and lexicon-based methods. Out of the top 20 entries, 19 of them were machine learning-based, and only one was lexicon-based. This indicates that for state of the art for machine learning-based and lexicon-based methods, machine learning-based methods seems to be outperforming the other. The second task was a classification task where, given a sentence, the goal was to classify a given word or phrase in that sentence to be either positive, negative, or neutral. The best ranking entry for this task had an F1-measure of 88.93 %. The paper is interesting for this

project because it indicates that while the machine learning-based approach yield better results in terms of accuracy, the lexicon-based approach might still have its merits. Often the lexicon-based approach is entirely unsupervised, or semi-supervised, resulting in less need for manual labeling.

Imbalanced data occurs in many domains such as customer reviews. A paper by Burns et al. [6] examines a naive Bayes classifier for sentiment analysis, with an imbalanced dataset. The dataset contains a total of 41,714 reviews, of which 7,294 are negative and 34,420 are positive. The dataset consists of customer reviews within three product categories: TVs, cameras, and kitchens. The researchers performed two types of experiments using naive Bayes on the dataset: one with all the data, and one where the data was resampled to balance negative and positive reviews. They performed several splits using different percentages of the data in both categories. The results are given in averages of accuracy on these different splits. For the balanced category, the results show a 88.34 % accuracy of negative, and 92 % accuracy of the positive reviews, yielding an overall accuracy of 90.17 %. For the imbalanced category, the results show 49.57 % accuracy for the negative, and 98.99 % accuracy for the positive reviews, yielding an overall accuracy of 91.89 %. The outtake form the article is that while the overall accuracy indicates that the imbalanced dataset yields the best results, learning with the entire imbalanced dataset might be a bad idea, depending on the cost associated with falsely classifying a positive or negative review.

Feature construction is the process of creating a feature vector from a given input, in our case a textual input. The typical feature vector for sentiment analysis using machine learning methods, is the bag-of-words representation. Le and Mikolov [7] proposes a way of condensing variable length text reviews into fixed length vectors called paragraph vectors. In the article they compare several machine learning techniques such as Support Vector Machines and naive Bayes using the bag-of-words feature space, with a logistic regression classifier using the paragraph vector feature space. An IMDB movie review dataset is used, containing a total of 100,000 reviews: 25,000 labeled and 50,000 unlabeled for training, and 25,000 labeled for testing. The results indicate that paragraph vector feature space performs well, and yields a 2 % improvement in terms of error rate compared to the state of the art machine learning techniques using the bag-of-words feature space. This indicates that paragraph vectors might also be interesting to explore in this project.

AFINN score is invented and described by Nielsen [8] who explores Danish sentiment analysis based on a lexicon of labeled words created from the valence of each word. The AFINN score is very simple and might not describe the sentiment accurately enough for our problem, however it might be interesting to look into the method used as it is easily applicable and could be used as a baseline. Section 2.4.1 describes the AFINN score in more detail.

In general, a lot of work has been done within sentiment analysis of social data. As mentioned, there are two main methodologies: lexicon-based and machine learning-based. As the research indicate that machine learning-based methods yield better results

than lexicon-based methods, the project will focus on machine learning-based methods.

We also looked at a paper looking into the effects of balanced versus imbalanced datasets. It is the case in many real world domains that there are more positive reviews than negative ones.

For the machine learning-based approach, the traditional way of constructing features has been taking either the unigram or bigram of all the words in the vocabulary, and either using a frequency value or a presence indicator. The article by Le and Mikolov [7] had success in using paragraph vectors for feature construction and will therefore take a similar approach.

1.5 Machine learning framework

To approach the problem of predicting sentiments in Danish text reviews using machine learning techniques, one often go through several stages to ensure the most accurate classification. Typically the stages are as follows: data collection, data filtering, feature extraction, classification, and evaluation. Each of these stages can be broken down into smaller components, typically thought of as building blocks. A building block is an algorithm with an input and an output. For two building blocks to work in succession, the output of one block has to match the input of the next building block.

This can be considered as a general framework for machine learning problems - a sort of pipeline that is set up to ensure that the flow of information in the system works together, and that everything can happen automatically. As there are many ways of configuring the building blocks at your disposal and no analytical way of knowing the accuracy of a configuration, one has to take an experimental approach towards choosing a suitable configuration. In the following chapters we will focus on describing the algorithms that we use as different building blocks in the implementation chapter. This project proposes a pipeline as illustrated in Figure 1.2.

Figure 1.2: Pipeline flow

2
Background

Many theories and algorithms relate to the problem described in Section 1.3. This chapter serves as a description of existing algorithms and theories to be expanded on in Chapter 3 and/or implemented in Chapter 4. This chapter is concerned with three main topics: data filtering, feature extraction, and classification. These topics correspond to the middle three steps of the machine learning framework pipeline from Figure 1.2. Each of these sections will explain existing knowledge within each subfield relating to our problem. Additionally, Section 2.4 contains information relevant to the report as a whole that does not fall into the machine learning framework.

2.1 Data filtering

This section describes methods of language detection related to filtering out irrelevant entries from the training data.

Language detection

The scope of this report is Danish text, but as review data might contain some non-Danish entries, it would be necessary to filter out these foreign reviews from our training data. Due to the size of the training data, it would be infeasible to filter manually and thus automatic language detection is needed. Several different methods for language detection exist, such as short word-based comparison, frequent word-based comparison, and n-gram-based comparison [9], and naive Bayes classifiers. In general, each of the different methods work by creating a profile for a language using a set of text fragments with a known language. To classify a new text fragment its profile is computed and compared to the language profiles. The language profile most similar to the text profile will decide the outcome of the algorithm. Following is a brief overview of n-grams, the most used technique to create language profiles.

The distribution of different n-grams varies from language to language. Even though many languages are related and similar in some way, these differences can be detected algorithmically. The frequency of n-grams constructed from a set of documents of the same language roughly follows a power-law distribution, when sorted according to the n-gram rank[1]. With this knowledge, it is possible to create a profile of a language, by finding the top k ranking n-grams from a set of documents for that language. If the number k for the created profiles equates roughly to the number of n-grams that dominate the power-law distribution, the profiles can be used to distinguish between languages as the few n-grams that dominate, and their rank, will be different across languages [10].

Naive Bayes classifiers have also been extensively used for language detection. An overview of naive Bayes is given in Section 2.3.1. In language detection using naive Bayes, the languages the classifier is built for is defined as the target classes. The feature space is typically n-grams or word frequency, but vowel/consonant or average word length has also been used [11]. Given a piece of text, the classifier returns with a probability for the text to be each of the different possible languages.

2.2 Feature extraction

This section describes general theories of feature extraction and specific algorithms that have been explored. The theory of tokenization and canonicalization is explained in detail. Additionally, two approaches to word embedding, Word2Vec and Doc2Vec, are also described.

2.2.1 Tokenization

Tokenization is the process of splitting text into smaller sections, e.g. splitting a text into sentences or words. Tokenization is typically done prior to stemming or lemmatization. There are numerous ways of tokenizing text, the simplest of which is treating each word as a token. As punctuation might be significant in determining the sentiment of a review, it might be beneficial to include these as well, e.g. negative reviews may have a larger chance of containing exclamation marks. If certain punctuations are included in the tokenization it would also be required to differentiate between abbreviations or numbers containing punctuation and punctuation used as part of sentence structure.

2.2.2 Canonicalization

Canonicalization is the process of transforming data with multiple possible representations into its canonical form, also called normal form. In the context of natural languages, the task is concerned with representing words without endings and inflections.

[1]The n-gram with rank 1 is the most occurring n-gram, rank 2 the second most occurring n-gram and so on.

Stemming and lemmatization are both methods for finding the root form of a word, e.g. by removing the ending of the different conjugations of the word. This is done to make it easier for machine learning algorithms by cutting down on the number of different inputs that they should learn to distinguish. The reasoning behind this idea is that inflections provide no or little additional information about its root form seen from a machine learning perspective.

Stemming strips inflectional endings of words to create the root form of the word, e.g. by removing "ing" endings from words like "sprinting" [12]. This is simpler compared to lemmatization, as it can be implemented using a few rules that define the possible suffixes that should be removed from words. However, canonicalization using stemming is susceptible to over- and under-stemming, where too much or too little is removed from a word respectively. Good stemming methods try to balance over- and under-stemming but it cannot be avoided completely. Additionally stemming does not work with irregular inflections such as "ran" because even though "ran" is an inflection of "run" they each get their own stem or root. Irregular inflections can also pose the opposite problem where different word forms share a common inflectional form, e.g. "axes" being the plural form of both "axe" and "axis". A simple stemming method based on rules would always stem "axes" to "axe" because of the irregularities of "axis". Some context-information is thus lost when stemming is performed.

Lemmatization on the other hand looks each word up in a dictionary and returns the lemma or base form of the word. As lemmatization does not manipulate the word itself but instead returns the lemma by looking it up, irregular inflections such as "ran" can be handled just as well as regular inflections. But even when using a lemmatizer there are problems with words that are spelled the same but have different meaning. For example, the word "meeting" is a noun as in "we are having a meeting", but is also an inflection of "to meet" as in "we are meeting again tomorrow". Thus, there exist more than one valid lemma for certain words and picking the right one requires further contextual analysis which can be a difficult task.

2.2.3 Word embedding

Word embedding is a mapping from a vocabulary to a dense, low-dimensional vector-space, e.g. 100 - 500 dimensions. This mapping is performed in such a way that semantically similar words are placed together. This is contrary to typical one-hot encoding, where the number of dimensions is equal to the size of the vocabulary, and has been shown to have a number of benefits as input for machine learning methods.

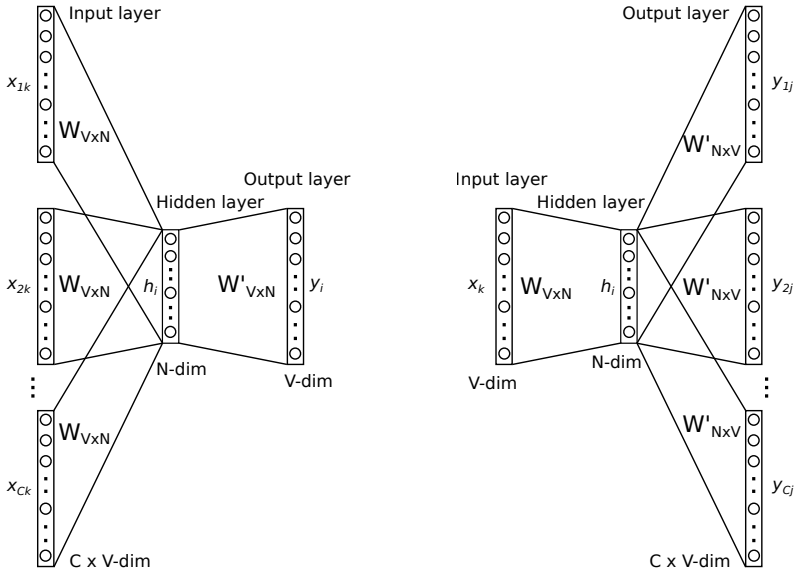
Word2Vec

Word2Vec is a model proposed by Mikolov et al. [13] for generating word embeddings from a collection of text.

Word2Vec generates these word embeddings using a two-layer neural net with one hidden layer. The basic idea is to create a neural net that can predict words based on its context, and then use the representation of the word in the hidden layer as a word embedding vector. Thus, the neural net of the hidden layer is the actual result of the Word2Vec word embedding.

Word2Vec is an iterative model that starts out with randomized vectors for each word and gradually adjusts them by backpropagation using training data. This is done using either the continuous bag-of-words model (CBOW) or the skip-gram model:

- The CBOW model (illustrated in Figure 2.1a) tries to predict words based on the average vector of the preceding and succeeding words. This averaging is why it is called bag-of-words as it discards word order.
- The skip-gram model (illustrated in Figure 2.1b) is much like the former, but in reverse. Based on a single word, the preceding and succeeding words are predicted. This tends to be more expensive than CBOW, but unlike the CBOW model it does not discard word order.



(a) CBOW model **(b)** Skip-gram model

Figure 2.1: Word2Vec models

Simply training these models as described above is unfortunately very slow. The output layer is the size of the vocabulary and evaluating this for every word will be computationally expensive. To improve this, Mikolov et al. [13] uses either negative sampling or hierarchical softmax to improve the training performance of the model. We will not

explain these optimization strategies in this paper as they have little influence on the generated models and as such falls outside the scope of this report.

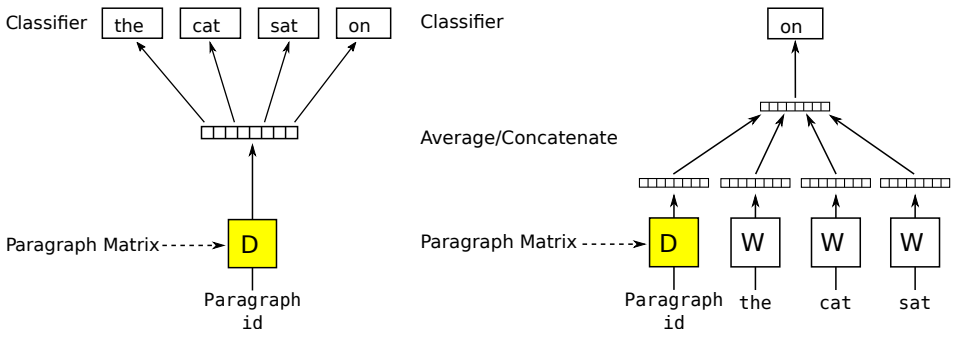
The end result of Word2Vec is a mapping from words to vectors, where the vector of each word will be close to the words that are often used in the same context. According to Firth [14], words commonly used together tend to be semantically similar and thus the Word2Vec vectors will also tend to be grouped semantically. Mikolov et al. [15] also found that semantic meaning weakly composes under vector arithmetic, e.g. "king" − "man" + "woman" yields a vector that is very close to "queen".

Doc2Vec

Doc2Vec is an extension to the original Word2Vec model [7]. The Doc2Vec model aims to find vectors for entire paragraphs instead of individual words, i.e. *paragraph vectors*. This is done by introducing a virtual word for each document and adding this to the vocabulary. This virtual word represents an entire document. The context of each word now becomes the surrounding words, as seen in the Word2Vec model, and the additional virtual document word.

Like the Word2Vec model, the Doc2Vec model has two different prediction models, but augmented for the virtual document words: distributed bag of words (DBOW) similar to skip-grams, and distributed memory (DM) similar to CBOW.

- In the DBOW model (illustrated in Figure 2.2a) word ordering is discarded. Training is done by selecting a text window randomly, then sampling a random word from the window and forming a classification task. As such, the paragraph vector is trained to predict words in small windows of text.
- In the DM model (illustrated in Figure 2.2b) the paragraph vectors are trained with a selection of words and used to define a context. The model then tries to predict the missing words from the defined contexts by updating them based on the error of the prediction.



(a) DBOW model **(b)** DM model

Figure 2.2: Doc2Vec models [7]

2.3 Classification

This section describes classification algorithms relevant to the scope of this report. These algorithms include naive Bayes, logistic regression, support vector machines, and the optimization strategy stochastic gradient descent.

2.3.1 Naive Bayes

Naive Bayes is a probabilistic model for classification [16]. It is known for its speed, as model training is computationally cheap compared to other methods. Naive Bayes relies on Bayes theorem and the assumption that each feature is conditionally independent of the other features. Bayes theorem states:

$$p(C_k|\boldsymbol{x}) = \frac{p(C_k)p(\boldsymbol{x}|C_k)}{p(\boldsymbol{x})} \tag{2.1}$$

where $p(a|b)$ denotes the probability of a given b, C_k is a target classification class, and $\boldsymbol{x} = (x_1 \dots x_n)$ is a set of features. When predicting a new data sample, $\boldsymbol{x}$ is known and the probability of the class C_k given those features can be calculated. The complexity of directly calculating the conditional probability using a probability table is exponential to the number of features.

Given the assumption of conditional independence, and that the denominator does not depend on C_k, equation (2.1) can be rewritten to the following:

$$p(C_k|\boldsymbol{x}) = p(C_k)p(x_1|C_k)p(x_2|C_k)\dots p(x_n|C_k) \tag{2.2}$$

The parameters of naive Bayes are thus $p(C_k)$ and $p(\boldsymbol{x}|C_k)$. These probabilities are obtained by counting the number of occurrences in the training set, and assigning the maximum likelihood probability. This can be done in linear time to the number of samples in he training set, making the learning time short.

When classifying a new data instance with naive Bayes, the value $p(C_k|\boldsymbol{x})$ is calculated for each of the k classes. This is typically done in log space to avoid numerical instability problems, which converts equation (2.2) to the following.

$$log(p(C_k|\boldsymbol{x})) = \sum_{i=1}^{n} log(p(x_i|C_k)) + log(p(C_k)) \tag{2.3}$$

This is simply a summation with a linear complexity to the number of features, which makes for a faster classification. The probability of each class is calculated and the class with the highest probability is picked as the classified class.

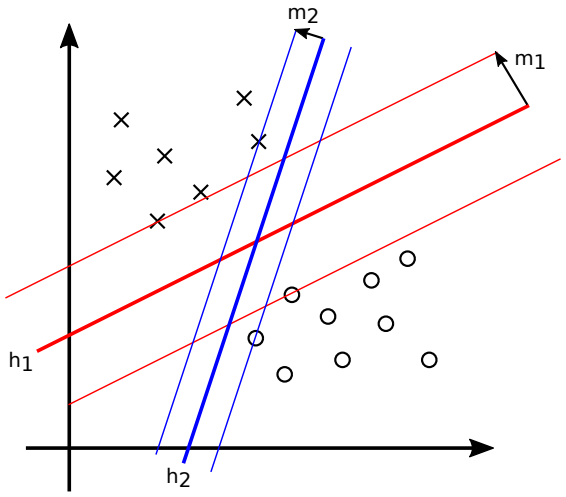


Figure 2.3: Maximum Margin Hyperplanes example

2.3.2 Support Vector Machines

Support Vector Machines (SVMs) are classification and regression models based on statistical learning theory. It is known for performing well on many classification problems, such as recognition of handwritten digits and text categorization [16]. Furthermore SVMs does not depend on the dimensionality of the dataspace and therefore works great with highly dimensional data. In this section the background of SVMs will be explained based on Tan et al. [16].

As seen in Figure 2.3 the plot contains a dataset of two different classes represented as circles and crosses. The datasets are linearly separable as a hyperplane can separate each class. Yet one needs to find the best hyperplane for the best categorization out of infinitely many possible hyperplanes. Consider the hyperplanes h_1 and h_2 which both separate the dataset without any misclassifications. Each hyperplane has a margin to the closest point on each side. In Figure 2.3, it is clear that the margin m_1 for h_1 is larger than the margin m_2 for h_2, and thereby h_1 is the hyperplane closest to the maximum margin hyperplane. The hyperplane with the largest possible margin is the maximum margin hyperplane.

Intuitively a larger margin is better for classification as a smaller margin would be more likely to wrongly classify instances if noise was introduced to the data.

For linearly separable binary classification, the training dataset consists of (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$ where x_i is a vector and y_i is either -1 or 1. The task is to find a maximal separating hyperplane $w \cdot x - b = 0$, where w is the normal vector of the hyperplane and b is the offset of the hyperplane, such that for all points (x_i, y_i) the expression $w \cdot x_i - b \geq 1$ is true if $y_i = 1$ and $w \cdot x_i - b \leq -1$ is true if $y_i = -1$.

This describes the hyperplane, but for it to be maximal the margin of the hyperplane should be maximized. As the distance from the hyperplane to a point x is $\frac{|w \cdot x + b|}{\|w\|}$, $\|w\|$ should be minimized subject to the previous conditions.

However, if the data is not linearly separable, it is not immediately possible to find a perfectly separating hyperplane. In this case, it is common to use a soft-margin, where some crossing of the margin is allowed to construct a suitable hyperplane. Another method is the use of kernel functions which aims to transform the non-linearly separable dataspace into a linearly separable dataspace. Kernel functions may also change the dimensionality of a function to achieve this. Kernel functions are basically functions taking the values of each point in the original dataspace and mapping it to a new dataspace that, ideally, is linearly separable.

2.3.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [17] is a popular optimization method for machine learning methods such as SVMs. SGD approximates the true gradient of the optimization problem for SVM by considering only a single randomly selected training sample per iteration.

For SVMs this means minimizing the regularized training error E given by:

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w) \tag{2.4}$$

where L is a loss function measuring how well the model fits the data given the predicted value and the actual target value, R is a penalty term penalizing model complexity, and α is a non-negative hyperparameter.

An example of a widely used penalty function is the L2 penalty function, which is also the one we refer to in this report when talking about a penalty function. L2 is defined as:

$$R_{L2}(w) = \frac{1}{2} \sum_{i=1}^{n} w_i^2 \tag{2.5}$$

Based on this, the model can be updated by iterating over the training samples and in each iteration update the model based on the following rule:

$$w \leftarrow w - \eta \left(\alpha \frac{\delta R\left(w\right)}{\delta w} + \frac{\delta L\left(w^{\intercal} x_i + b, y_i\right)}{\delta w} \right) \tag{2.6}$$

where η is the learning rate controlling how much the model changes per iteration. The learning rate can either be constant or decaying, where decaying means that the learning rate will get smaller as the training iterates to allow fine tuning of the model.

The training time of SGD is significantly lower when compared to other training methods, while the trained model still performs reasonably well [17]. As such, SGD is interesting when looking at problems with large amounts of data. This also makes it interesting for this project, as the short training time allows for quick exploration of different configurations.

2.3.4 Logistic regression

Logistic regression is a classic prediction method for binary classification, the following section is based on the paper by McCullagh [18]. The logistic regression method creates a sigmoid function based on the data, giving the probability at each input point. Logistic regression is meant for calculating the probability of an instance having one of two different outcomes, but can be used to classify multiple classes by making one versus rest logistic regression, e.g.:

$$log\frac{p_1}{p_2+p_3+p_4+p_5}, log\frac{p_2}{p_1+p_3+p_4+p_5}, etc. \tag{2.7}$$

This is the default method of classifying multiple classes which is unordered. Instead of getting the probability of each of the value classes in an unordered way, one can use ordered logistic regression to make the classification ordered, e.g.:

$$log\frac{p_1}{p_2+p_3+p_4+p_5}, log\frac{p_1+p_2}{p_3+p_4+p_5}, log\frac{p_1+p_2+p_3}{p_4+p_5}, etc. \tag{2.8}$$

Logistic regression can also be used as a loss function in SGD, as defined by:

$$L(p,y) = \frac{1}{\ln 2}\ln(1+e^{-yp}) \tag{2.9}$$

where p is the predicted class returned by the classifier, and y is the correct class.

2.3.5 Modified Huber loss function

The modified Huber loss function is defined by Zhang [19] and is based on the Huber loss function. Modified Huber loss is made for binary classification problems with classification domain $y \in \{\pm 1\}$. This loss function is often used with SGD. The definition of modified Huber loss can be seen below:

$$L(p,y) = \begin{cases} \max(0, 1-py)^2 & \text{for } py \geq -1 \\ -4py & \text{otherwise} \end{cases} \tag{2.10}$$

where p is the predicted class returned by the classifier, and y is the correct class.

2.3.6 Perceptron loss function

The perceptron loss function is used in neural networks and is defined by Rosenblatt [20]. The perceptron loss function is also used with SGD and is defined as a sum of data pairs (x_n, y_n) for all n:

$$E(w,b) = \sum_{n=1}^{N} \max\left\{0, -y_n\left(w^\intercal x_n + b\right)\right\} \tag{2.11}$$

where w is a vector defining the separating hyperplane and b is the offset.

2.4 Additional background

This section contains information about data and methods not directly related to the machine learning framework, but relevant in respect to our study.

2.4.1 AFINN score

The AFINN score described by Nielsen [8] is a very simple sentiment analyzer for both English and Danish words which makes it suitable as a baseline method for sentiment classification.

The AFINN score uses a list of words where each words is associated with an AFINN word score, an integer between -5 (very negative words) and 5 (very positive words). The AFINN sentiment score for a text is the sum of the AFINN scores of the words contained in the text. The Danish word lists contains 3,552 words manually labeled by the author.

2.4.2 KorpusDK

KorpusDK is a digital collection of written Danish text from different sources, such as books, news paper articles, and blog posts, ranging from the beginning of the 1980s until the end of 2010 [21]. The total number of words from the subset of KorpusDK obtained for this project is 45.9 millions words, each word contains additional information about the lemmatized form, stemmed form and word class.

The purpose of KorpusDK is to give a wide and varied view of the diversity in Danish text. The large size of the dataset makes it possible to try out sentiment analysis techniques that requires extensive language analysis of Danish text.

2.4.3 Named entity recognition

Named entity recognition is the task of annotating a text string with entities. The entities are predefined and categorized into organizations, locations, persons, products, etc. Given a text string, the task is then to annotate part of the text with entities such that they match the intended target. Typically, blocks of text are annotated such that entities containing several words are categorized as one entity. As an example, "Andrew just bought a new pair of Nike Free from Salling" should be categorized into "(Andrew)[Person], just bought a new pair of (Nike Free)[Product] from (Salling)[Organization]".

Many classifiers can be applied to recognize named entities, where models such as hidden Markov models and conditional random fields have been applied with success to structured text. Such models require annotated training data to build the model, which is typically done manually. It is also possible to create hand-crafted models by defining a grammar to parse the text. This can be a time consuming task, given the fact that models developed for one domain typically do not work well for other domains [22].

Current state of the art models perform with an F1-measure of 93 %, in the MUC-7 conference. This number might vary depending on the complexity of the data. As an example, a state of the art model in [23] perform with an F-measure of 80 % on search queries.

2.4.4 Resampling

Resampling covers different methods for training with imbalanced datasets. The default choice would be to use the training data as is (full training), but this incentivizes the machine learning algorithm to be biased towards majority classes. For example, the Trustpilot data contains mostly 5 star reviews, making a classifier more likely to choose 5 stars as the rating based on this imbalanced frequency. The most common ways to deal with an imbalanced dataset are [24]:

- Cost-sensitive training: examples are weighted such that the total weight of each class is equal.
- Undersampling: examples of the majority classes are removed to match minority.
- Oversampling: examples of the minority classes are added to match majority.

Availability of cost-sensitive training depends on the machine learning algorithm used. For SVMs this is generally done by creating larger soft-margin misclassification penalties for highly weighted observations. Cost-sensitive training has been observed to perform better than full training, but less so than resampling techniques [24].

For undersampling and oversampling there are a number of methods of varying complexity. The simplest methods are random undersampling and random oversampling. In random undersampling random majority observations are discarded, while in random oversampling random minority observations are duplicated.

A more complex method of oversampling is the "Synthetic Minority Over-sampling Technique" (SMOTE) [25]. SMOTE is inspired by character recognition where it is common to distort examples to generate more training examples, e.g. by translation, rotation, or skew, in order to create a more resilient classifier [25, 26]. SMOTE works in feature space and directly generates the input vectors, making it more versatile. SMOTE increases the representation of minority class by randomly selecting minority examples and generating new examples from a randomly weighted average of the k-nearest neighbors of the same class.

For undersampling, there are also more complex techniques. Identifying and removing *Tomek links* is a common method of undersampling. A Tomek link is an example pair (a, b) of different classes for which there are no example c such that $d(a, c) < d(a, b)$

or $d(b, c) < d(a, b)$ where d is the distance function. If two examples form such a link, either they are both noise or both borderline [27]. A combination of oversampling using SMOTE followed by undersampling using Tomek links has also been found to be effective in some cases [27].

3
Contributions

Many different theories and methods can be used to solve the problem described in Section 1.3. Some relevant theories that have been studied in this project are described in Chapter 2. In this chapter we present our own contributions to solving some of the problems where we found no existing solutions to be satisfactory.

In the following sections a metric and an analysis method is described. The metric is defined to evaluate machine learning methods in a way that accounts for multiclass labeled data, such as the data available for this project. The analysis method was created to explore the possibility to accommodate the requirements by NORRIQ, that the sentiment of individual products should be traceable as explained in Section 1.1.

3.1 BRMSE Metric

In order to evaluate different machine learning methods, a fair evaluation metric is needed. The area under curve (AUC) metric, which is the area under the receiver operating characteristics curve, is commonly used for binary classification problems. However, AUC does not extend into multiclass problems in an obvious way. Some research has been done by Landgrebe and Duin [28], wherein they define a volume under hypersurface (VUS) metric for multiclass classification, but this results in a significantly more complex and computationally expensive metric than AUC. Due to this and VUS not being widely used, we instead focus on fitting more common metrics to the specifics of our problem.

A way of presenting the results of a classification model is by using a *confusion matrix*. A confusion matrix M of a classifier is an $n \times n$ matrix, where n is the number of classes and $M_{i,j}$ is the number of observations with labels belonging in class i, but classified as class j. We wish to find an error metric $e(M)$ that can be used to rank the confusion matrices of different classifiers. This section will deal with error metrics, i.e. a metric that is better the lower it is.

We start by considering the accuracy metric. This metric is simply the fraction of correctly classified instances. As the correct classifications are all along the diagonal (since $M_{i,i}$ will be the number of elements known to be i and classified as i), this is simply the fraction of the confusion matrix that lies on the diagonal (i.e. the fraction of the trace of M):

$$e_{accuracy}(M) = 1 - \frac{\mathrm{tr}(M)}{\sum_i \sum_j M_{i,j}} \tag{3.1}$$

This metric works, but it presents the problem that a misclassification is equally bad if it is close or far. For example, a 1 star review classified as 2 stars is just as bad as if it was misclassified as 5 stars. We want to capture the idea that the numerical distance of a misclassification influences the error metric.

For this the root-mean-square error (RMSE) [29], a common regression metric, can be used. Although we are doing classification rather than regression, this metric still applies to classification as the class labels in our case are integers. This can be written in terms of a confusion matrix as follows:

$$e_{RMSE}(M) = \sqrt{\frac{\sum_i \sum_j M_{i,j} (i-j)^2}{\sum_i \sum_j M_{i,j}}} \tag{3.2}$$

But this presents some problems due to the unevenly distributed dataset from Trustpilot, as described in Section 4.1.2. When applying the RMSE metric above to our data, machine learning methods are encouraged to classify 4 and 5 star reviews much better than 1-3 star reviews. This is problematic, as it is not known if this distribution of reviews matches that of the application domain. Instead of assuming that the distribution is the same, we assume that the distribution of the application domain is uniform. This means that the classifier should perform equally well across all ratings. Therefore, every class (row in the confusion matrix) will be weighed equally.

We do this by normalizing each row of the confusion matrix such that the sums of the rows are equal. For each row i in the confusion matrix, we define the weight as

$$w_i = \frac{1}{\sum_j M_{i,j}} \tag{3.3}$$

To apply this weight, it can be noted that the inside of the root in equation (3.2) is a kind of *weighted average*, the definition of which is displayed here:

$$\overline{x} = \frac{\sum_i w_i x_i}{\sum_i w_i} \tag{3.4}$$

As can be seen by comparing equation (3.4) and equation (3.2), the confusion matrix M already acts as a kind of weight between the different error sizes. By replacing $M_{i,j}$ with $M_{i,j} w_i$ in equation (3.2), we obtain a new weighted average that applies the weighing from equation (3.3). The final result of this is the metric used in this paper, which we

		Predicted		
	Class	1	2	3
Actual	1	0	0	2
	2	0	0	2
	3	0	0	96

(a) Confusion matrix a

		Predicted		
	Class	1	2	3
Actual	1	2	0	0
	2	0	2	0
	3	0	20	76

(b) Confusion matrix b

Table 3.1: Example confusion matrices

Metric \ Matrix	a	b
RMSE	0.32	0.45
BRMSE	1.29	0.26

Table 3.2: Metric results. Lower numbers are better.

will call the balanced root-mean-square error (BRMSE):

$$e_{BRMSE}(M) = \sqrt{\frac{\sum_i \sum_j M_{i,j} w_i \left(i-j\right)^2}{\sum_i \sum_j M_{i,j} w_i}} \tag{3.5}$$

To our knowledge the BRMSE metric have not been utilized in other papers or research prior to this report, although it bears similarity to some of the concepts within dataspace weighting described by He and Garcia [30].

Example

Suppose two different machine learning methods a and b were used, and the resulting confusion matrices are as seen in Table 3.1. It can be seen that a only classifies results as 3, but attains a low RMSE as the data is highly imbalanced. Method b matches all of row 1 and 2, but incorrectly matches some of row 3 as 2. If it is assumed that the distribution of the reviews of the application domain is uniform, then b would be preferable to a. Intuitively it makes sense to trade the ~20 % decrease in classifying 3 in order to gain 100 % increase in classifying 1 and 2.

In Table 3.2, it can be seen that the RMSE score favors a while balanced BRMSE favors b. This demonstrates how the BRMSE metric values performance across different classes higher.

3.2 Entity Sentiment Detection

The problem presented by NORRIQ is concerned with finding sentiments for specific products in natural text. This would require finding references to a set of products from a defined product catalog as well as determining the immediate context of the product

in the text. If products and their textual context can be accurately identified, sentiment analysis can provide a fine grained analysis of many diverse sentiments in a single piece of text. A method for analyzing sentiment of text as a whole would miss what the positivism and/or negativity is directed at. A review of 4 stars could mention the great product that was received, but complain about a long delivery time. A more detailed analysis of entities such as products, price, delivery, and experience could provide a better decision-making foundation for the reviewed companies.

We have investigated how to approach this challenge with respect to the Trustpilot reviews that have been gathered. We designed an analyzer based on the recognition of grammatical word-classes, e.g. nouns and verbs etc., and using the word-class probability distributions of the KorpusDK data described in Section 2.4.2.

3.2.1 Preprocessing and data collection

The Trustpilot reviews were first tokenized, while preserving punctuation, as described in Section 2.2.1. Then custom entities were detected and marked as described in Section 3.2.2. The tokens were subsequently stemmed using a Danish snowball stemming algorithm [12] to reduce complexity by reducing the word space as inflections of words are removed.

A naive approach would be to tag each token with the most frequent word class for that token's word stem according to KorpusDK, the list of word classes and their abbreviations can be seen in Table 3.3. As the most frequent class is not always the correct class the tags are refined in an iterative 3-gram algorithm. This algorithm calculates the probability that the currently assigned tag of each token fits within the tag of the preceding and following token. These probability distributions are also calculated according to KorpusDK. The tags are then updated if the probability that they belong to a less frequent word-class is higher considering the word-classes that surround them, see example in Section 3.2.4. The iterative algorithm takes 3 parameters, number of iterations, learning rate α, and a learning rate decay value γ. The learning rate determines how much the model changes per iteration, and γ determines the change of the learning rate at every iteration, calculated by the formula $\alpha_{n+1} = \alpha_n \cdot \gamma$. The parameters were optimized by testing them on a balanced set of 25 manually tagged Trustpilot reviews. We found suitable parameters to be 2 iterations with an initial α of 1.0 and a γ of 1.0. The learning rate is constant through all iterations with γ set to 1.0.

3.2.2 Entity recognition

We decided to detect entities by looking at word classes in the review text and by using user defined custom entities and their inflectional forms. The detection of custom entities, applied in the preprocessing steps, allows for the definition of typically unrecognizable entities in text, e.g. product names. These custom entities were found and tagged by simple word comparison which would probably not scale to enormous product-catalogs where more sophisticated approaches would become necessary. Nouns

Abbreviation	Word class
A	adjective
C	conjunction
N	noun
V	verb
T	preposition
P	pronoun

Table 3.3: Abbreviations of common word classes

are relatively easy to detect and includes most important entities that an analyst would be interested in, e.g. “price”, “service”, “delivery”, etc. As this is the case each detected noun is simply marked, after the preprocessing steps, as being an entity that should be analyzed.

3.2.3 Context recognition

Each detected/tagged entity in a review has a context. It is important to accurately determine the context boundaries as a wrong context detection could potentially apply a wrong sentiment to an entity. In our approach, the context is determined by a cautious rule-based algorithm based on the detected word-classes. The context of each entity is expanded forwards and backwards as long as the words encountered belong to a certain word class. Inversely, this means that certain word-classes, e.g. conjunctions, act as context boundary markers. Additionally the punctuation symbols retained in the preprocessing steps also aid in the detection of boundaries.

After the context of each entity has been determined the words in each context is scored according to the AFINN method described in Section 2.4.1. The score is then aggregated to obtain the final sentiment score of each entity. As such, the word order within each context is disregarded. Negation is primitively handled by simply inversing the score of each word in a context if the context contains a negation word.

3.2.4 Example

As an example we will go through the analysis process step-by-step for a Danish text. The tokenized text seen in Table 3.4 is a fictitious review for a webshop containing both negative and positive sentiments. The translation is not a part of the analysis process but is just a word-by-word translation from Danish to English.

For this analysis a lot of custom entities (CE) relevant for webshops have been manually created. The list of custom entities included the words “hjemmeside” and “kundeservice” which were also correctly identified as indicated by the column labeled CE.

After the stemming process the iterative process of determining the word-classes are seen in column 5 through 7, where the naive column represents the results of a run with

0 iterations, followed by columns with results for a run with 1 iteration and 2 iterations respectively. There was only one difference in results between the initial naive word-class detection and the first iteration of the 3-gram iterative algorithm (marked in bold). The naive method classifies the last word "hjælp" as a verb (V) as it is mostly used as a verb in Korpus2010. The first iteration of the 3-gram algorithm takes the word-class of the preceding word into account which increases the probability of "hjælp" being a noun (N). This increase is enough to convince the algorithm that "hjælp" is a noun and its tag is changed. There are no changes from the first to the second iteration. In this example all the words were tagged with the correct word-class.

After the word-class detection algorithm has completed the context is calculated for every token tagged as a noun or marked as a custom entity. For our example 5 entities were detected with their contexts visualized in column 8 (Ctx.). Each context spans a continuous list of tokens and the entity of the context is marked in bold. Contexts are allowed to overlap as seen with context 4 and 5.

The sentiment score of each context is then calculated by summing up the AFINN scores (AS) of the tokens in the context giving the results seen in Table 3.5.

3.2.5 Conclusion

While our preliminary exploration of this challenge returned meaningful results for simple texts, as seen in Section 3.2.4, we have been unable to find any datasets that can be used for validation and evaluation. Such a dataset should consist of natural Danish text where each entity have been identified and scored according to sentiment polarity. Without creating such a dataset ourselves we have no reliable way of telling whether our approach yields valuable and correct results. Producing such a dataset of any usable size would require extensive amounts of time, which is not available in this project, as such we will not pursue this approach any further.

Even without a proper way to evaluate this approach, we have identified several aspects that could be improved. Using lemmatization instead of stemming would be slower, but would also yield a better basis for our word-class analysis. Many words carry no sentiment individually, but may form a sentiment when combined with other words. Detecting these combinations and their sentiment would require a comprehensive analysis of the Danish language but would greatly improve sentiment detection.

Many words also carry sentiment differently in certain contexts, e.g. the word *"hurtig"* which is Danish for *"fast"* or *"quick"* would in most contexts be positive but in the context of already negative words like *"problem"*, it would affect the sentiment negatively. The ability to handle sentiment of context sensitive words would also require further language analysis. The complexity of negation is also not handled optimally in our preliminary approach and could be improved.

It would also strengthen the analysis method if entities were detected in a more sophisticated and categorical way, e.g. by using hidden Markov models or conditional random fields as described in Section 2.4.3. But as the section also describes, constructing such models usually takes lots of time and annotated data.

Original text	Translation	CE	Stemmed	Naive	1st	2nd	Ctx.	AS
Hurtig	Fast		hurt	A	A	A		**2**
og	and		og	C	C	C		
korrekt	correct		korrek	A	A	A	1	**2**
levering	delivery		levering	N	N	N	**1**	
af	of		af	T	T	T		
det	what		det	P	P	P		
,			,					
jeg	I		jeg	P	P	P		
havde	had		havd	V	V	V		
bestilt	ordered		bestilt	V	V	V		
.	.		.					
Godt	Great		godt	A	A	A	2	**3**
udvalg	selection		udvalg	N	N	N	**2**	
hvis	if		hvis	C	C	C		
man	you		man	P	P	P		
kan	can		kan	V	V	V		
udholde	tolerate		udhold	V	V	V		
den	the		den	P	P	P		
forfærdelige	horrible		forfærd	A	A	A	3	**-3**
hjemmeside	website	X	hjemmesid	N	N	N	**3**	
men	but		men	C	C	C		
kundeservice	customer service	X	kundeservic	N	N	N	**4**	
gav	gave		gav	V	V	V	4	
god	great		god	A	A	A	4, 5	**3**
hjælp	help		hjælp	V	**N**	N	4, **5**	**2**
.	.		.					

Table 3.4: Entity detection example

Entity	English translation	Score
levering	delivery	**2**
udvalg	selection	**3**
hjemmeside	website	**-3**
kundeservice	customer service	**5**
hjælp	help	**5**

Table 3.5: Resulting sentiments of the example in Table 3.4

4 Implementation

This chapter describes in detail our implementation of the algorithms and theories discussed in Chapter 2 and Chapter 3. As explained in Section 1.5, we generalize our machine learning framework as a five-step pipeline, this pipeline process is detailed in Figure 4.1 where the implementation tasks in each stage is listed. This chapter follows the flow of the detailed pipeline, with the exception of Evaluation which will be described in Chapter 5.

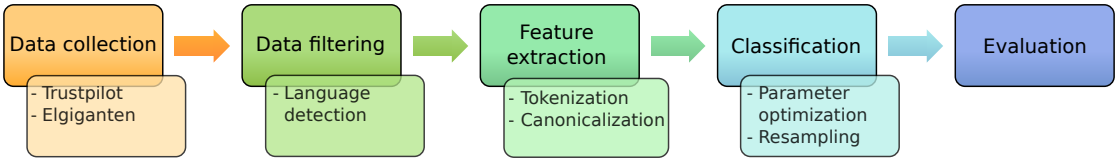


Figure 4.1: Detailed pipeline

4.1 Data collection

As described in Section 1.2 the data sources for this project are reviews from Trustpilot and Elgiganten. This section describes how the data was collected, along with the data characteristics. The process of the data collection is illustrated in Figure 4.2 and is explained further below.

4.1.1 Web crawling

In order to collect the review data, we used a simple web crawler, *grab-site* [31], to recursively crawl the webpages of the review platforms. Grab-site stores the HTML websites in the Web ARChive format (WARC) [32], which makes it convenient to convert the stored data into CSV format using a custom-built WARC/HTML parser.

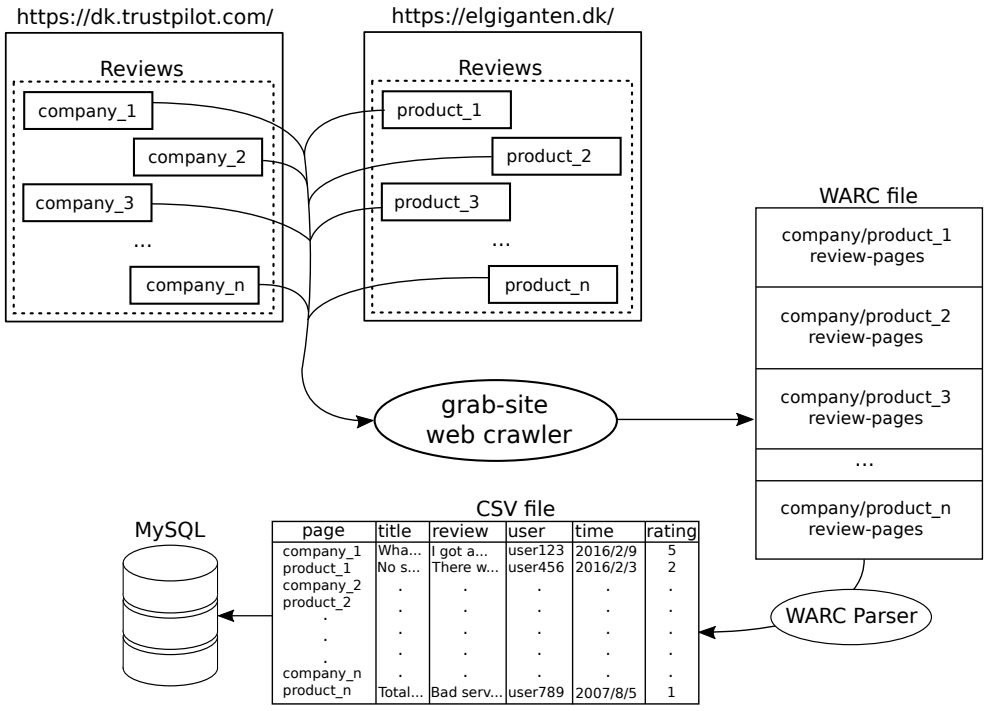


Figure 4.2: Process of review collection

4.1.2 Data sources

Trustpilot [33] is a large consumer review platform with language specific sub-pages, which means that the Danish Trustpilot page mainly contains reviews written in Danish. The reviews are mostly concerned with general consumer experiences related to companies, meaning that only few reviews are product specific. Each review consists of a title, a review text, a rating from 1 to 5 stars, the user-id of the user who wrote the review, the company the review was written about, and the submission time of the review.

Using web crawling, we gathered 2,611,777 reviews made between 2007-07-25 and 2016-02-16.

There are some problems in the Trustpilot review data. The review ratings are not equally distributed in the collected data as illustrated in Figure 4.3. Of the 2,611,777 reviews, 1,915,491 are labeled with a perfect rating of 5 stars (73 %). This imbalance is likely to influence the result of the classification model to favor the majority class as explained in Section 1.4. This behavior is not desirable as we do not know the distribution of the application domain and thus would like to predict all classes equally well. Different methods of handling this kind of imbalanced data is discussed in Section 2.4.4.

Another problem in the Trustpilot dataset is that it contains reviews written in other languages than Danish which increases noise in the dataset, means to remove this is described in Section 4.2.1.

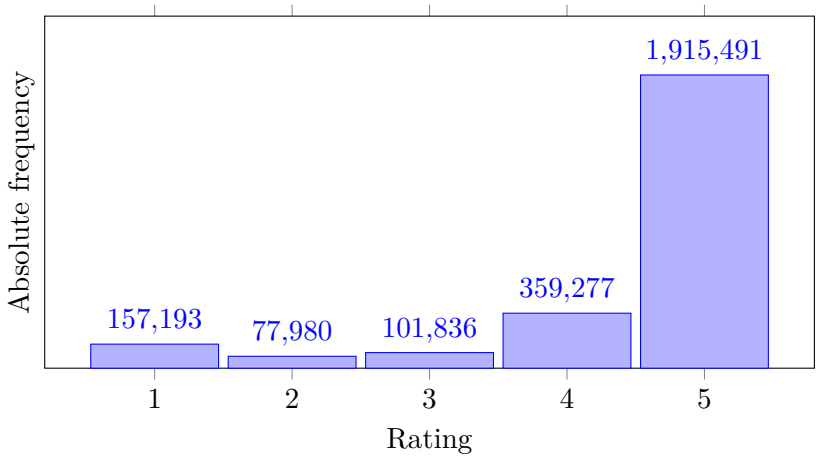


Figure 4.3: Distribution of Trustpilot reviews

Elgiganten [34] is a Danish webshop and chain of consumer electronic stores. Their webshop contains reviews of some of their products. These reviews are indicative of the public opinion on each individual product which makes them relevant for this project as it is relevant to the problem proposed by NORRIQ. Each product review consists of a product description, a title, a review text, and a rating from 1 to 5 stars.

Using web crawling we gathered 11,329 reviews.

There are also some problems with the Elgiganten dataset similar those in the Trustpilot dataset. Labels are distributed in a much similar way, as illustrated in Figure 4.4. Of the 11,329 reviews 6,123 are labeled with the rating 5 (54 %). Another problem arises from the fact that the Elgiganten dataset does not contain as much data compared to Trustpilot as only 11,329 individual reviews were found. We briefly tested for non-Danish reviews and found no indication of such a problem.

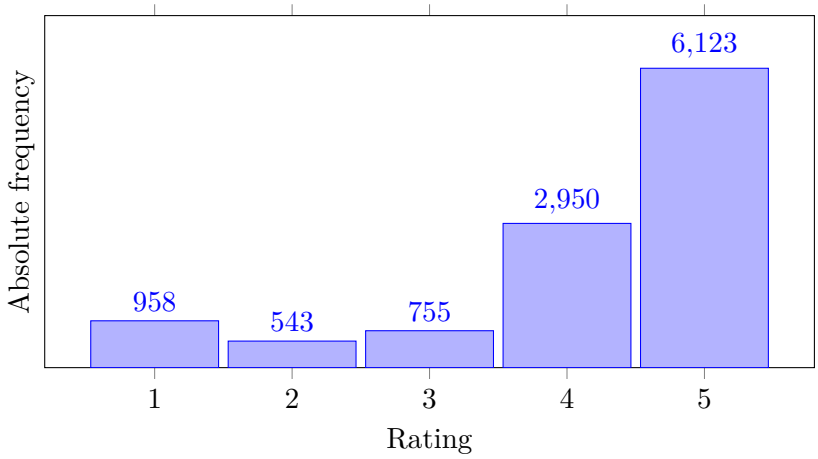


Figure 4.4: Distribution of Elgiganten reviews

4.2 Data filtering

This section describes how we deal with the problem of non-Danish reviews, mentioned in Section 4.1.2. This section will include a description of the filtering processes used in the project.

4.2.1 Language detection

With 2,611,777 Trustpilot reviews from the danish Trustpilot site we found a large number of non-Danish reviews. As the aim of the project is to classify sentiment in Danish reviews, it is sensible to remove these entries.

To filter out the non-Danish reviews, we use three Java libraries based on either naive Bayes classifiers or n-grams as described in Section 2.1 and Section 2.3.1 respectively:

language-detection by Shuyo [35] uses language profiles generated from Wikipedia text. Languages in text is detected using naive Bayes classifiers.

language-detector by Shuyo et al. [36] is a fork of the above mentioned *language-detection* library which uses the same language profiles, but uses n-grams for detection.

JLangDetect by Champeau [37] implements n-gram language profiles generated from text from the European Parliament. This means that only languages from member countries in the European Parliament can be detected.

The shorter the text is, the harder it is for the languages to be recognized by the language profiles, this is due to the language detection algorithms uses both the ordering of the words and the likelihood of each word being part of a certain language. This also means that the language detection is more difficult in reviews with multiple languages as the different words pulls the prediction in different directions. This is amplified for similar languages like Danish, Swedish and Norwegian as they share a lot of common words and language profiles only store the most used words in a language.

To avoid these problems we implemented all three languages detection libraries and letting each method label the reviews with the language with the highest certainty. We label reviews as Danish where at least two out of three detectors labeled the review as Danish, as we are only interested in knowing if the review is Danish or not. We found 75,839 non-Danish reviews, 3 % of the entire dataset, using this implementation.

4.3 Feature extraction

The following section describes the extraction of features from textual reviews, these features can then be used in the training of various machine learning algorithms. The process of tokenization is described and hereafter a brief discussion on the use of canonicalization will follow. Lastly, the Doc2Vec implementation is explained which is used to build the paragraph vectors on which we can train the machine learning algorithms.

4.3.1 Tokenization

For tokenization we decided to use the Natural Language Toolkit (NLTK) developed by Bird et al. [38], which is a platform for Natural Language Processing including tools such as stemming, tokenization and lemmatization. First, we identify individual sentences in the review using the NLTK Punkt sentence tokenizer. As NLTK has no built-in support for Danish, we have trained our own Punkt tokenizer using the KorpusDK data that is described in Section 2.4.2. The Punkt sentence tokenizer is ideal for our purposes, as it has been shown to perform well with most European languages and only requires unlabeled text to learn abbreviations, structure, etc. [39].

Hereafter, we divide the sentences into tokens using the Penn Treebank tokenizer. The tokenizer is built for English sentence structures, but works well for Danish as there are only small differences in structure. The final result is a list of word tokens of the review text.

4.3.2 Canonicalization

For canonicalization we decided to use stemming in favor of lemmatization due to the complexity of Danish compound words and lack of Danish training data.

Stemming was implemented using the Danish Snowball stemmer written by Porter [12]. The algorithm is implemented in Java based on the stemming rules documented in [40].

4.3.3 Doc2Vec

To find word embeddings in our text reviews we use Gensim, a software library developed by Řehůřek and Sojka [41], which is a collection of machine learning tools for Natural Language Processing implemented in python. We use the Word2Vec and Doc2Vec methods from Gensim, which are implementations of algorithms described in Section 2.2.3 and Section 2.2.3 respectively.

In this paper, we will try three different Doc2Vec models as listed in Table 4.1. The two initial models are the DBOW and DM models described in Section 2.2.3. In addition to these, we also include a DM model that uses vector concatenation instead of mean. This is significantly more expensive to compute, and we therefore reduce the window size to 5 in order to achieve a manageable execution time.

Name	Method	Window size
dbow	Distributed bag of words	10
dm-mean	Distributed memory - window mean	10
dm-concat	Distributed memory - window concatenation	5

Table 4.1: Doc2Vec models tested

Parameter	Value
Vector size	100
Evaluation method	Negative sampling
Epochs	40
Initial alpha	0.025
Minimum alpha	0.001

Table 4.2: Common Doc2Vec parameters

Other than the window size and method, the remaining parameters are identical and can be seen in Table 4.2. The vector size has been set to 100 - a reduction from the library default size of 300. We did this to reduce memory consumption, as a size of 300 resulted in too large memory consumption for our hardware. Like Mikolov et al. [15] we use negative sampling over hierarchical softmax for better performance.

We ran the Doc2Vec tool for 40 epochs (iterations over the dataset). This is a conservative estimate of the number of epochs necessary and could likely be lowered without much loss in accuracy. After each epoch, we shuffle the entire dataset and gradually decrease the alpha (learning rate) until the minimum alpha has been reached. Shuffling was a recommendation by the authors of Gensim.

For each method in Table 4.1, we generate two sets of vectors: a set of trained vectors, and a set of inferred vectors. The trained vectors are part of the training process - they influence the representation of other vectors and also needs to be present under the training. Thus, the trained vectors are computed offline in batches.

Inferred vectors, on the other hand, can be computed online without influencing the model. This makes them more practical to use in a lot of applications, but they are generally not as good as the trained vectors.

4.4 Classification

This section describes the classification methods and techniques utilized in this project. Most methods and techniques in this section are from the library Scikit-learn, developed by Pedregosa et al. [42], which is a collection of machine learning tools implemented in python.

4.4.1 Resampling

While we tried to implement the resampling methods mentioned in Section 2.4.4, we quickly found the performance to be a significant problem because many of the resampling methods relies on the nearest neighbor of each point. In Scikit-learn, the KD-tree or ball-tree used for nearest neighbor computation of a single point has a worst case $O(n)$ complexity. With this method, the worst case for finding nearest neighbor of all points would be $O(n^2)$. In addition to this, another problem is that spatial data structures

like KD-trees perform worse as the number of dimensions go up. As there is 100 dimensions in the output from Doc2Vec (see Section 2.2.3), this also decreases performance significantly.

Instead of repeatedly finding the nearest neighbor for a single point, there are more advanced algorithms for finding all nearest neighbors that runs in $O(n \log n)$ time. However, we could not find any such implementation already available and we did not have the necessary time to implement it ourselves.

The effect of these problems is that any nearest neighbor-based resampling method becomes very slow. Therefore, we decided only to proceed with undersampling, cost-based training, and full training for control. The undersampling method consist of a simple random undersampling, whose complexity is $O(n)$ to resample the entire dataset. We used a library for python to perform the undersampling, using the `numpy.random` class to pick random samples to remove from the overrepresented classes. The cost-based training is directly implemented into the Scikit-learn, as parameters for the classification methods we use. The cost-based training is implemented by associating a weight to each data instance, where the normal weight is 1. The balanced mode sets a weight inversely proportional to the class frequency in the input data, thus a class representing $\frac{1}{10}$ of the input data would have a weight of 10.

4.4.2 Stochastic Gradient Descent

We use SGD (see Section 2.3.3) from Scikit-learn for parameter optimization of our classification algorithms. The Scikit-learn SGD implementation supports a number of different loss functions, or algorithms, that can be utilized for classification:

hinge is the same as in an SVM (see Section 2.3.2).

log is the same as in logistic regression (see Section 2.3.4).

modified_huber is a variant of SVM based on the more robust modified Huber loss (see Section 2.3.5).

perceptron is the same loss function as seen in the perception algorithm (see Section 2.3.6).

As SGD is fairly fast, this means that we can explore more hyperparameters than we otherwise would have time to. It also means we can more easily scale to larger datasets once we found the optimal hyperparameters.

4.4.3 Hyperparameter optimization

When a classification method has been chosen, it is important to select the most optimal hyperparameters for each classifier. Training and evaluating different models for various combinations of such parameters can be a long and tedious task as most classification methods offer many such hyperparameters. Therefore, we conduct an automated grid search over the possible hyperparameters.

We utilize the grid search algorithm in Scikit-learn. The algorithm takes a classifier, a grid of parameters to search through, and optionally a scoring function. In our case this included SGD with different loss functions and a range of different learning rates all evaluated using the BRMSE metric as scoring function (see Section 3.1).

Each point in the grid is then trained and evaluated on the training set using cross validation to prevent overtraining. An important detail is that the evaluation of the grid point is done on the training set, and not on the test set. If the evaluation is done on the test set instead, we risk overtraining the hyperparameters and lowering the ability for the classifier to generalize, which is what we wish to achieve.

The grid can be visualized as in Figure 4.5. We combine each sampling method (cost-based, full, or undersampling, as detailed in Section 2.4.4), with each set of Doc2Vec vectors (dbow, dm-mean, or dm-concat, as detailed in Section 4.3.3), and both the trained vectors or inferred vectors described in Section 4.3.3.

From the grid search, it seems that DBOW with undersampling performs best. It performs largely the same across different algorithms and alpha values. Cost-based DBOW also seems very stable, except for relatively large alpha values. Cost-based training performs worse than undersampling, but still significantly better than full training.

After the grid has been evaluated, the point with the lowest mean BRMSE across folds is selected as the best and trained with the entire training set. The selected hyperparameters for each combination of vectors and sampling function can be seen in Table 4.3.

Vectors	Sampling	Algorithm	Alpha	BRMSE
dm-concat-inferred	Full training	hinge	10^{-8}	1.63
dm-mean-inferred	Full training	hinge	10^{-9}	1.59
dm-concat	Full training	perceptron	10^{-4}	1.51
dbow-inferred	Full training	perception	10^{-7}	1.45
dm-mean	Full training	log	10^{-9}	1.40
dm-mean-inferred	Cost-based	perceptron	10^{-2}	1.40
dbow	Full training	log	10^{-2}	1.38
dm-concat-inferred	Cost-based	modified_huber	10^{-7}	1.32
dm-mean-inferred	Undersampling	log	10^{-3}	1.26
dm-concat	Cost-based	hinge	10^{-4}	1.21
dm-concat-inferred	Undersampling	log	10^{-3}	1.21
dbow-inferred	Cost-based	modified_huber	10^{-4}	1.17
dm-mean	Cost-based	log	10^{-4}	1.14
dm-concat	Undersampling	log	10^{-5}	1.09
dbow-inferred	Undersampling	log	10^{-3}	1.07
dm-mean	Undersampling	log	10^{-3}	1.03
dbow	Cost-based	log	10^{-4}	1.01
dbow	Undersampling	log	10^{-4}	0.95

Table 4.3: Selected hyperparameters in grid search

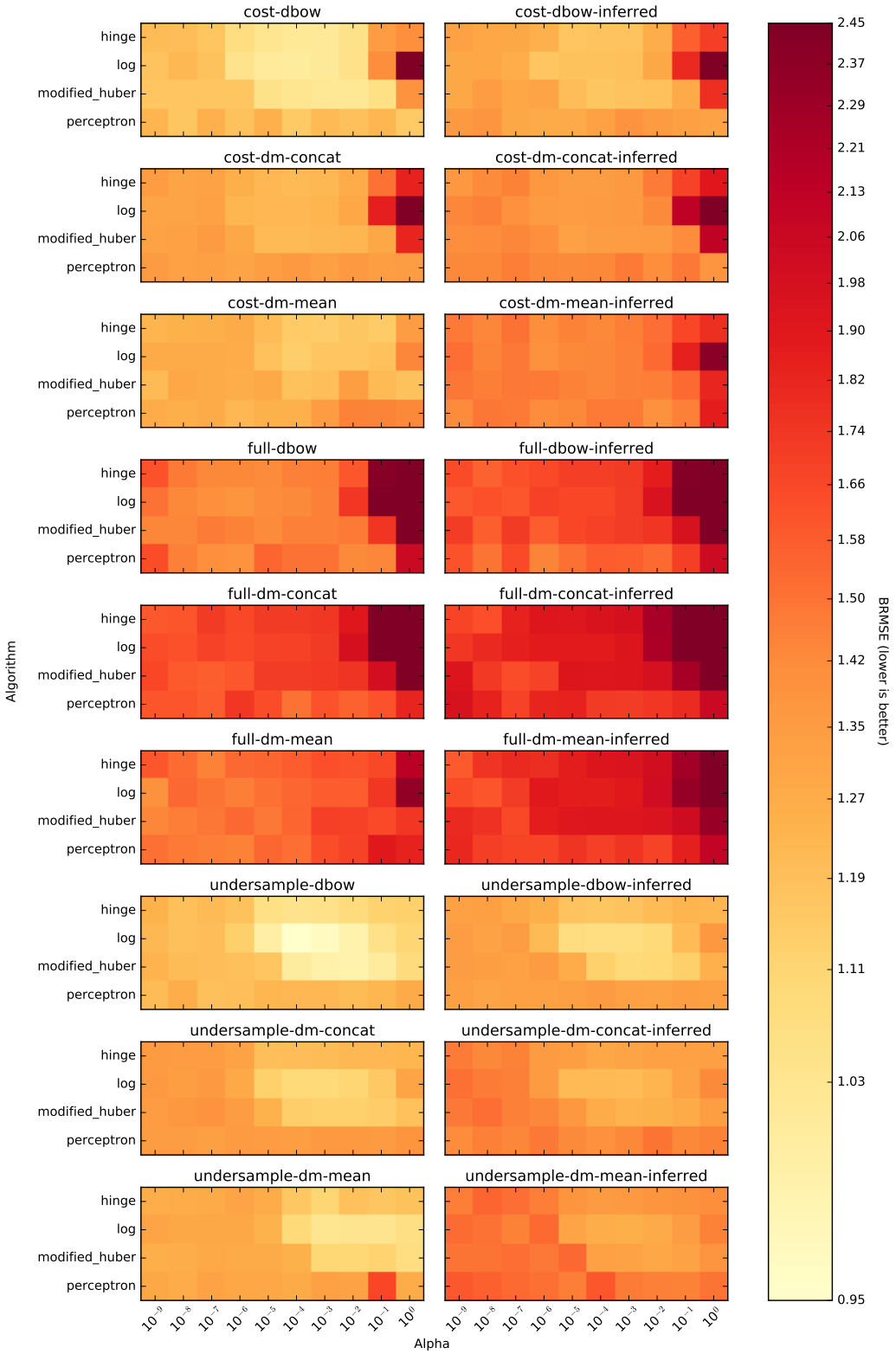


Figure 4.5: BRMSE of different Doc2Vec and classification methods

5 Evaluation

In order to evaluate the sentiment analyzer, we train it using the Trustpilot dataset and test it on the Elgiganten dataset, for more details on data sources see Section 4.1. This ensures that the sentiment analyzer can generalize to other kinds of reviews and as the parameter tuning is exclusively done on the Trustpilot dataset it also avoids overtraining the hyperparameters.

Only the inferred vectors are tested, as we do not wish the test data to have an effect on the training data, even though it is only through the vectorization of Doc2Vec. For practical applications the inferred vectors are also the ones most likely to be used, as they can be constructed online unlike the offline trained vectors which needs to be constructed in batches. The chosen configuration is based on the best BRMSE result from Table 4.3, given the vector and sampling method.

For comparison, we provide a random baseline which simply chooses a random class. The results of this is presented in Table 5.1.

We can see that it performs worse than the Trustpilot dataset alone, which can be seen in Table 4.3. While there is usually a decrease in performance of the chosen metric when

Vectors	Sampling	Algorithm	Alpha	Train BRMSE	Accuracy	BRMSE
Random baseline					20.0%	2.00
dm-mean-inferred	Full	hinge	10^{-9}	1.59	52.0%	1.94
dm-concat-inferred	Full	hinge	10^{-8}	1.63	48.4%	1.75
dbow-inferred	Full	perceptron	10^{-7}	1.45	46.6%	1.65
dm-concat-inferred	Cost-based	modified_huber	10^{-7}	1.32	48.4%	1.58
dm-mean-inferred	Cost-based	perceptron	10^{-2}	1.40	40.7%	1.53
dm-mean-inferred	Undersampling	log	10^{-3}	1.26	49.0%	1.46
dm-concat-inferred	Undersampling	log	10^{-3}	1.21	41.8%	1.45
dbow-inferred	Cost-based	modified_huber	10^{-4}	1.17	53.1%	1.44
dbow-inferred	Undersampling	log	10^{-3}	1.07	46.6%	1.32

Table 5.1: BRMSE of Elgiganten data

		Predicted				
	Class	1	2	3	4	5
Actual	1	356	215	238	74	75
	2	97	123	234	67	22
	3	71	83	393	145	63
	4	144	147	787	1095	777
	5	360	214	722	1518	3309

Table 5.2: Confusion matrix of dbow-inferred with undersampling on Elgiganten dataset

		Predicted				
	Class	1	2	3	4	5
Actual	1	140	232	27	82	477
	2	52	168	19	76	228
	3	58	170	24	144	359
	4	119	347	46	417	2021
	5	157	451	36	333	5146

Table 5.3: Confusion matrix of dm-mean-inferred with full training on Elgiganten dataset

comparing the result of the training and test set, it is difficult from our experiements to pin-point the cause. Traditionally one would attribute this difference to overfitting to the training set, but since the test set is not drawn from the same population as the training set we can not necessarily say it is caused by overfitting.

We also see that random undersampling is the best resampling method followed by cost-based training and lastly full training when compared on the BRMSE metric. Random undersampling is better than cost-based training, this result is in line with the results by Li et al. [43] which found random undersampling to generally perform well. A more concrete discussion on the cause of this is found in Chapter 6.

The confusion matrix of the best estimator, dbow-inferred with random undersampling, can be seen in Table 5.2. We can see that misclassifications are often local, i.e. it is more common to misclassify by one off than by two off or three off, this is an indication that BRMSE as a metric is working as intended. For comparison we also provide the confusion matrix from the worst estimator, besides the baseline, dm-mean-inferred with full training set, which can be seen in Table 5.3. While the accuracy of the latter is one of the best in the set of configurations, its BRMSE is almost equal to the random baseline. By analysing the confusion matrix we see that this is caused by the underrepresented classes being classified very poorly, thus driving the BRMSE up. This is caused by the issue of the learnt model favoring the majority class. Analyzing the three confusion matrixes for the configurations using the full training set, we see the same behavior.

We also provide a comparison of the AFINN algorithm compared to dbow-inferred with undersampling and logistic regression, on the test set. As the AFINN algorithm is only designed to classify as either {positive, neutral, negative} or {positive, negative}, we have

Negative	Positive	tp	fp	tn	fn	Accuracy	Balanced accuracy
-	1-5	10290	0	0	1039	0.908	-
1-1	2-5	9696	594	364	675	0.888	0.657
1-2	3-5	9298	992	509	530	0.866	0.643
1-3	4-5	8685	1605	651	388	0.824	0.623
1-4	5-5	5902	4388	818	221	0.593	0.561
1-5	-	0	10290	1039	0	0.092	-

(a) Results of AFINN.

Negative	Positive	tp	fp	tn	fn	Accuracy	Balanced accuracy
-	1-5	13329	0	0	0	1.000	-
1-1	2-5	6269	2814	3309	937	0.719	0.705
1-2	3-5	3810	2374	6699	448	0.788	0.817
1-3	4-5	2791	1019	8809	710	0.870	0.847
1-4	5-5	356	672	9699	2602	0.754	0.528
1-5	-	0	0	13329	0	1.000	-

(b) Results of dbow-inferred with undersampling using logistic regression.

Table 5.4: Metrics for different splits of Elgiganten data based on positive and negative ranges. tp is true positives, fp is false positives, tn is true negatives, fn is false negatives.

to do a mapping of our data onto the representation. We chose the binary classification of {positive, negative}. We do this by choosing ranges for which reviews are interpreted as positive and which are negatives on the 5-rating scale. It is not immediately obvious which value to split on to get the true division into positive and negative reviews, as different people might value different rating values differently. We provide the results for all different splits, but argue that the most reasonable are when 1-2 and 1-3 are the negative ranges. The tables of the different splits can be seen in Table 5.4a for the AFINN, and Table 5.4b for dbow-inferred with undersampling using logistic regression. As we can not calculate the BRMSE for this, we provide the accuracy and the balanced accuracy from Brodersen et al. [44], which is the average accuracy of each class. We only talk about the results for negative ranges of 1-2 and 1-3, as they seem like the most reasonable split values. While the accuracy for AFINN is high, the balanced accuracy suffer from its ability to classify the negative class, an indication that the accuracy is a bad metric for the imbalanced dataset. Overall we see the machine learning approach yields better results on the chosen splits compared to AFINN for the balanced accuracy. While this gives an indication that the machine learning based approach is best, the AFINN method has not been optimized for our dataset domain, which might explain why it does not perform very well.

6
Discussion

In this chapter we discuss points and aspects of this report. Initially, we focus on which aspects of this report could be used to build an automatic sentiment analyzer on social media data. Afterwards we discuss the metric constructed in this report, how it relates to accuracy, and how useful it is. Then we move on to a more concrete discussion of the results obtained from our experiments, and how they compare to other work on the subjects found in Section 1.4. Our methods of resampling are then discussed, and we look into why the different sampling methods perform as they do. Lastly we take a look at the machine learning algorithms we used, and how the results we observed relates to other work done on the topic.

NORRIQ's interest

The scope of the project was, in accordance with Chapter 1, to examine the possibilities of automated sentiment analysis of text. This was done in collaboration with NORRIQ, and their initial idea was to do it on social media data. However, we found no suitable data source from social networks, and thus turned to sentiment analysis to social media data, namely on Trustpilot and Elgiganten reviews. While the Trustpilot data was well suited for learning sentiment, due to being labeled, the reviews are about companies, and as such they contain little information that can be directly linked to products. As a result of this, constructing an automatic system to analyze public opinion on products needs other sources of data. While the Trustpilot dataset was not perfect for the purpose, we found it suitable to illustrate that machine learning-based sentiment analysis is possible in Danish social media, something not previously explored in any sources we found.

If a way of labeling the social network data is not found only unsupervised machine learning algorithms can be used. Depending on the amounts of data to be labeled, an approach is to out-source this task, which might turn out costly. Looking at the articles in Section 1.4, the number of labeled data instances range from 2,000 - 40,000

to build state-of-the-art classifiers. Note that these articles focus on binary classification problems, and more data samples might be required to learn a model for a multiclass problem. This number might also vary depending on the domain in proportion to its vocabulary size.

There are other differences than simply the lack of labels in social network data. Firstly, Trustpilot and Elgiganten reviews are confined to a specific product or company, while social network data might contain a lot of noise that is not directly related to a product being written about. Secondly, social network data is not initially categorized with respect to products or companies. These issues has not been the main focus of the report, and therefore have not been explored sufficiently to make a fully automated system. Given ways of handling the previouly stated differences, we believe that the overall machine learning techniques is applicable to the social network data, along with results giving the same overall performance.

We have developed a simple fine-grained sentiment analyzer (explained in Section 3.2), that allows the user of the system to defined language specific rules to automatically detect multiple entities within text and calculate their sentiment. The analysis method also allows for user-defined entities, such as product names, to meet the interests of NORRIQ. While the method should in theory be usable for several languages, given enough data, we have only been able to try it out for Danish. However, we were not able to formally evaluate the approach due to lack of labeled test data.

Metrics

BRMSE is a metric introduced in this project, as described in Section 3.1. One of the main challenges during the development of our system, was the heavily imbalanced data set. Another core characteristics in the data is that labels are ratings, with 1 star being the most negative, and 5 star being the most positive. With respect to finding the sentiment, a misclassification of a 5 star review as a 1 star review, is worse than misclassifying it as a 4 star review, as 5 is semantically closer to 4 than 1. BRMSE is based on the RMSE, which handles the characteristics of the labels being ordered, and to additionally handle the imbalanced data by balancing across all labels.

The closely related literature that we have found used accuracy as the metric for evaluation. Table 5.1 show the results for the different configurations that we tested. From this table it is clear that there is no direct correlation between BRMSE and accuracy. This indicates that while the BRMSE is decreased, the accuracy will not necessarily increase. This is because accuracy completely ignore which label a review was misclassified as, whereas BRMSE will give less error if the misclassified label is close to the correct one. Based on this, we argue that BRMSE is better suited for the purpose of the system intended by NORRIQ.

From Table 5.1, the best BRMSE result was 1.32. While BRMSE is good to compare different configurations, it is not a very intuitive measure. Looking at the definition one can understand what the number means, but it is difficult to predefine a required

BRMSE goal for a newly developed system. While accuracy is more intuitive, it is not well suited for the ordered classes, as it is unclear "how much" the misclassified instances are misclassified.

Results

In Section 1.4 we investigate papers with projects similar to ours. The papers by Pang et al. [3] and Le and Mikolov [7] both uses machine learning techniques to determine sentiment in movie reviews from IMDB. The paper by Pang et al. explores an SVM approach much like the one used in this paper and Le and Mikolov explores the use of paragraph vectors in sentiment analysis. There are, however, some differences in the approaches taken in the papers and the one taken in this report. First, more data was available for our project as we have 2.5 million reviews compared to the 100,000 movie reviews used by Le and Mikolov and the 2,000 reviews used by Pang et al.. Second the paper by Pang et al. limits the sentiments to only positive and negative as opposed to the 1 to 5 star rating featured in the data of this project, the paper by Le and Mikolov presents results for both coarse- and fine-grained classification, where coarse-grained is only concerned with negative and positive sentiment, while fine-grained is much like our 1 to 5 star rating with five different levels of sentiment ranging from very negative to very positive.

There are many conditions that make it difficult to compare our results to the results of the mentioned papers. The data sets, and their domains are different. This means that inherently one of the data sets might be easier to classify. Another aspect is that the data sets used in the papers are approximately balanced, while the data set we use is very unbalanced. Additionally we test on data from a different source to the one we train on, and while the domain is the same, this could make a difference. The metric used to gauge the best configuration for our data set is BRMSE, as explained in Section 3.1, while the papers focus on accuracy.

To get an approximate idea of how the performance our system, we compare the accuracy values we obtain to those they report. The fine-grained classification presented by Le and Mikolov is the most comparable, as both projects classify sentiment into five ordered classes. The configuration that had the best result, on the BRMSE metric, in this report had an accuracy of 46.6 %. It should be noted that one configuration had an accuracy of 53.1 %, which shows the different goals of BRMSE and accuracy. Le and Mikolov obtained a best result with 48.7 % accuracy. These results are not directly comparable, but indicates that the results from the methods showed in this project are somewhat similar to the state of the art methods investigated in the paper by Le and Mikolov.

Resampling

The dataset used in the report is heavily imbalanced and we implemented resampling to counteract this. As described in Section 2.4.4, we did not implement oversampling

due to practicalities. The three different data sampling techniques we trained with were full training data, cost-based sampling, and undersampling. In Table 5.1, we see how the different types of sampling perform. Using BRMSE as a metric, the full training data sampling method gives worse results than cost-based and undersampling. This is probably because the learning algorithm will focus much more on the majority classes in the data set. Cost-based sampling and undersampling are much closer in terms of BRMSE result, but there is an indication that undersampling gives the best result. The cost-based sampling assigns heavier weights to the underrepresented classes and thereby simulates a balanced data set. Our results indicate that this approach is not as good as simply undersampling the over represented classes, as we have access to sufficient amounts of data. When undersampling we still have roughly 80,000 reviews of each class, or a total of 400,000 reviews. As discussed previously, the related articles used data sets of 2,000 - 40,000 entries, which explains why we still see good results with 400,000 reviews.

Machine learning

For machine learning we used several different classification algorithms: SVM, logistic regression, SVM with modified Huber loss, and perceptron. In Pang et al. [3] a comparison of three algorithms, with SVM and logistic regression among them, SVM was found to perform best. In Le and Mikolov [7] another result is obtained, that logistic regression with paragraph vectors perform better than SVM with BOW features. While it is not explicitly stated in the article, we assume that they tested several standard machine learning algorithms with the new feature space, and reported the one performing best, namely logistic regression. In this report we also found that logistic regression gives the best results.

7
Conclusion

We gathered and refined 2,535,938 Danish reviews from Trustpilot and 11,329 Danish reviews from Elgiganten. Each review is labeled with a rating from 1 to 5 stars. This rating is treated as the sentiment that we try to predict based on the review text. Due to the multiclass labels and highly imbalanced nature of the datasets we defined a balanced version of the well-known RMSE metric, which we call BRMSE. In addition, we also explored different resampling strategies, such as undersampling and cost-based training, to balance the data. Using the dataset from Trustpilot we built a paragraph vector representation of each review. These paragraph vectors were then used to train various linear classifiers using stochastic gradient descent and optimized with grid search on the BRMSE metric using different resampling techniques. The trained models were then evaluated on the Elgiganten dataset to demonstrate the generalizability of the models to other domains.

Using grid search on the Trustpilot dataset we found the best method to be a combination of DBOW paragraph vectors, random undersampling, and logistic regression with a BRMSE of 0.95. This also proved to be the most efficient method on the Elgiganten dataset, achieving a BRMSE of 1.32. It is problematic to compare our results directly to state-of-the-art methods, as these often report results using the accuracy metric, which is unsuitable for the highly imbalanced data used in this project. Nonetheless, we have compared the results of the state-of-the-art fine-grained classifiers presented by Le and Mikolov [7] to that of our method and found that it achieves similar accuracy.

Overall, we believe that BRMSE is a suitable metric for evaluating classifiers on imbalanced datasets, while equally weighting errors in the prediction of each class. To our knowledge, this is the first attempt at applying machine learning-based sentiment analysis to Danish text. Our results show that sentiment analysis of Danish is possible and achieve results comparable to that of English.

8
Future work

All Nearest Neighbors

As mentioned in Section 4.4.1, we found a number of problems with resampling due to the computationally hard task of calculating nearest neighbors for each point. The all nearest neighbor problem is an extension of the nearest neighbor problem into n points. A problem with nearest neighbor queries (and spatial queries in general) is the *curse of dimensionality* where algorithms rapidly increase in processing time the more dimensions are introduced [45].

Vaidya [46] describes an algorithm for computing all points nearest neighbors in a d-dimensional space in $O((cd)^d\, n \log n)$ time where c is a constant independent of d. If we consider d to be fixed, this leads to optimal $O(n \log n)$ asymptotic runtime [46]. While d is fixed at 100 in our case this still leads to a intractably large constant factor of $(cd)^d$.

Another option is to trade precision for better performance. The ϵ-approximate nearest neighbor is the problem of finding a point $p \in P$ for the query q such that for all $p' \in P$ it holds that $d(p, q) \leq (1 + \epsilon)\, d(p', q)$, where P is the set of all points and d is the distance function [45]. Indyk and Motwani [45] proposes an algorithm for this problem that uses *locality sensitive hashing* to construct a data structure in $\tilde{O}(n) \cdot O(1/\epsilon)^d$ time that can query nearest neighbors in $\tilde{O}(d)$ where $\tilde{O}(g(n)) = O(g(n) \log^k g(n))$, i.e. $\tilde{O}(d)$ is loglinear in d. Solving all nearest neighbors using this would yield a total runtime of $\tilde{O}(n) \cdot O(1/\epsilon)^d + O(n) \cdot \tilde{O}(d) = \tilde{O}(n) \cdot O(1/\epsilon)^d$. It is worth noting that this algorithm is impractical for small ϵ as the preprocessing time is dependent on $1/\epsilon$. We have not been able to find any prior research into how these nearest neighbor errors might affect resampling correctness.

Either of the algorithms above could be used for improving resampling, depending on whether approximate nearest neighbors is sufficient. Implementing these algorithms and integrating them into our current framework could improve the performance of our sentiment analyzer, and could therefore be interesting future work.

Entity sentiment detection

Our shallow approach to detailed entity sentiment detection, described in Section 3.2, yielded interesting results. When analyzing simple subjective Danish texts the method could easily identify different entities, determine their scope, and calculate their sentiment. But as the length and complexity of the reviews increased, fewer of the intended sentiments were identified through the analysis process. For complex reviews the method also struggles to correctly identify all words, or combinations of words, that carry sentiment, and has a hard time correctly identifying all contexts of entities, e.g. when a context is not coherent. These problems and some short notes on possible solutions are described in Section 3.2.5. More interestingly the complexity of the reviews seems to be less important when aggregating the sentiments of a lot of related reviews, e.g. thousands of reviews for one particular business. This is likely the result of the majority of reviews being simple and/or short. This seems to indicate that the approach could be used to identify sentiments of entities when given a sufficiently large and reliable language-specific dataset.

As Section 3.2.5 describes, the method cannot be tested formally without access to a specialized dataset of manually analyzed Danish texts. Any further development of this or any similar approach should make retrieval of such a dataset a high priority. One way of obtaining such a dataset would be with the use of crowd-sourcing by paying willing Danish speaking people to manually identify entities and sentiments in text.

As described above and in Section 3.2.5 the weakest point in the process is likely the detection of context-sensitive or word-combination sentiments. The current method relies solely on lookup where all words that carry sentiment needs to be defined. It might be interesting to examine whether or not the calculation of sentiments could be improved by using the classification models that we have generated. When treating each detected context as an isolated document it can be classified in the same way as a whole review and should yield a similarly meaningful result.

In our current implementation negation is handled on a per context basis. If a negation word is found in a context all sentiment words in the context has their score inverted. This is quite naive and does not make sense in most situations and marks a key area of potential improvement. It might be interesting to look into a negation-context recognizer built upon the same principle as our entity-context recognizer, where context is determined by word-class rules. Better detection of the context of negation should lead to an overall greater accuracy in the detection of sentiment. It might also be possible that negation in part could be handled by a well-trained classification model if that route is pursued.

In other areas of computer science this method might be usable in recommender systems if reviews are user-tagged. If a user has strong sentiment towards certain entities like “price”, “quality”, “style”, etc. they could be recommended products based on public opinion of said entities. For example, if the public opinion of a product is that it is stylish but low quality it could be targeted towards users who usually complain about lack of style or praises style but rarely complains about product quality.

Other data

As is described in Section 1.2 and Section 3.2.5 the need for more data and specialized datasets is crucial in further exploring whether our sentiment analysis models can generalize to Danish text in other domains, and whether it is possible to determine the sentiment of individual entities in text.

First is the problems described in Section 1.2 concerning the availability of text from social media sites such as Facebook and Twitter. If such data was obtained and filtered, leaving us with spam-free Danish text data, it would be interesting to investigate the generalizability of our sentiment classifier on the data. For such an experiment to yield interesting results, the data would have to be mapped beforehand in order to make a sensible evaluation. This mapping could be made through the means of crowd-sourcing services, that could map the large amounts of Danish text data to corresponding sentiment, e.g. positive/negative, or some rating.

Second is the problem described in Section 3.2.5 concerning the need of specialized data to evaluate our method for detecting sentiments of single entities in text. The most obvious way of creating such a dataset would be to manually mark entities and evaluate their sentiment polarities. This would of course require a lot of work, and is too big a task for our team, but through the means of crow-sourcing services it might be feasible. If such a dataset was available, our Entity Sentiment Detection method could be formally evaluated and opened up for further improvements and optimization. A verification of this method would also be in the interest of NORRIQ, as it closely relates to their problem with detecting sentiment based on predefined product catalogues.

9
Acknowledgments

We would like to thank NORRIQ for proposing and cooperating with us on the problem. By working with companies and actual problems the end result tend to be more intriguing than working on a broad topic of study. We would also like to thank Peter Dolog for excellent supervision and feedback throughout the project period.

Bibliography

[1] NORRIQ website. URL `http://www.norriq.dk/`. Accessed: 2016-05-24.

[2] Socialhub. Use of social media in Denmark. URL `http://socialhub.dk/indsigt/`. Accessed: 2016-05-24.

[3] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, volume 10, pages 79–86. Association for Computational Linguistics, 2002.

[4] Cataldo Musto, Giovanni Semeraro, and Marco Polignano. A comparison of lexicon-based approaches for sentiment analysis of microblog posts. *Information Filtering and Retrieval*, page 59, 2014.

[5] Preslav Nakov, Zornitsa Kozareva, Alan Ritter, Sara Rosenthal, Veselin Stoyanov, and Theresa Wilson. SemEval-2013 task 2: Sentiment analysis in Twitter. 2013.

[6] Nicola Burns, Yaxin Bi, Hui Wang, and Terry Anderson. Sentiment analysis of customer reviews: balanced versus unbalanced datasets. In *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 161–170. Springer, 2011.

[7] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[8] Finn Årup Nielsen. A new anew: Evaluation of a word list for sentiment analysis in microblogs. *arXiv preprint arXiv:1103.2903*, 2011.

[9] Ernesto William De Luca Grothe, Lena and Andreas Nümberger. A comparative study on language identification methods. *LREC*, 2008.

[10] William B Cavnar and John M. Trenkle. N-gram-based text categorization. *Ann Arbor MI 48113.2*, pages 161–175, 1994.

[11] Fela Winkelmolen and Viviana Mascardi. Statistical language identification of short texts. In *ICAART (1)*, pages 498–503. Citeseer, 2011.

[12] Martin F Porter. Snowball: A language for stemming algorithms, 2001.

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

[14] John Rupert Firth. *A Synopsis of Linguistic Theory, 1930-1955*. Blackwell, 1957.

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

[16] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*, volume 1. Pearson Addison Wesley Boston, 2005.

[17] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[18] Peter McCullagh. Regression models for ordinal data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 42(2):109–142, 1980. ISSN 00359246.

[19] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

[20] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[21] Society for Danish Language and Literature. KorpusDK. URL `http://korpus.dsl.dk/`. Accessed: 2016-05-24.

[22] Thierry Poibeau and Leila Kosseim. Proper name extraction from non-journalistic texts. *Language and computers*, 37(1):144–157, 2001.

[23] Andreas Eiselt and Alejandro Figueroa. A two-step named entity recognizer for open-domain search queries. In *IJCNLP*, pages 829–833, 2013.

[24] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SigKDD Explorations Newsletter*, 6(1):1–6, 2004.

[25] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, pages 321–357, 2002.

[26] Thien M Ha and Horst Bunke. Off-line, handwritten numeral recognition by perturbation method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(5):535–539, 1997.

[27] Gustavo EAPA Batista, Ana LC Bazzan, and Maria Carolina Monard. Balancing training data for automated annotation of keywords: a case study. In *WOB*, pages 10–18, 2003.

[28] Thomas Landgrebe and R Duin. A simplified extension of the area under the ROC to the multiclass domain. In *Seventeenth annual symposium of the pattern recognition association of South Africa*, pages 241–245, 2006.

[29] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[30] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on Knowledge and Data Engineering*, 21(9): 1263–1284, 2009.

[31] Ivan Kozik and Grab-Site contributors. Grab-Site, 5 2016. URL `https://github.com/ludios/grab-site`. Commit: `842fab4b23ef0c6a8302449affc63592c36ca05c`.

[32] Information and documentation – WARC file format. ISO 28500:2009, International Organization for Standardization, Geneva, Switzerland, 2009.

[33] Trustpilot A/S. Trustpilot. URL `https://dk.trustpilot.com/`. Accessed: 2016-05-24.

[34] Elgiganten website. URL `http://www.elgiganten.dk/`. Accessed: 2016-05-24.

[35] Nakatani Shuyo. Language detection library for Java (language-detection), 3 2014. URL `https://github.com/shuyo/language-detection`. Commit: `a1b65d981fc40aad0763dd782acbc99ab40a6228`.

[36] Nakatani Shuyo, Fabian Kessler, Francois Roland, and Robert Theis. Language detection library for Java (language-detector), 2 2016. URL `https://github.com/optimaize/language-detector`. Commit: `57cdf47d4c1d7145658773aa3afce0534f4e39f0`.

[37] Cédric Champeau. A language detection library for the JVM (JLangDetect). URL `https://github.com/melix/jlangdetect`. Commit: `aa8b0647c1fad469ef5e37e54a4f47166ca87a3a`.

[38] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[39] Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

[40] Martin Porter and Richard Boulton. Danish stemming algorithm. URL `http://snowballstem.org/algorithms/danish/stemmer.html`. Accessed: 2016-05-24.

[41] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[43] Shoushan Li, Zhongqing Wang, Guodong Zhou, and Sophia Yat Mei Lee. Semi-supervised learning for imbalanced sentiment classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1826, 2011.

[44] Kay H Brodersen, Cheng Soon Ong, Klaas E Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 3121–3124. IEEE, 2010.

[45] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[46] Pravin M Vaidya. An O(n log n) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4(2):101–115, 1989.