# Sentiment Analysis on a Norwegian Review Corpus

Bernt Andreas Eide*
Department of Electrical Engineering and Computer Science (University of Stavanger)
Stavanger, Norway
ba.eide@stud.uis.no

Vegard Valberg*
Department of Electrical Engineering and Computer Science (University of Stavanger)
Stavanger, Norway
v.valberg@stud.uis.no

Stian Seglem Bjåland*
Department of Electrical Engineering and Computer Science (University of Stavanger)
Stavanger, Norway
ss.bjaland@stud.uis.no

## Abstract

There are many practical applications for sentiment analysis: consumer insight; discovering public opinion on controversial issues; categorising articles and reviews, and so on.

This paper is focused on determining the sentiments of movie-, game-, music- and book-reviews taken from a Norwegian review corpus. After testing multiple methods of classification, we pick the one that is overall the best, and present our recommendations as to how you can make an effective and accurate model for these sentiments.

## Keywords

preprocessing, deep learning, recurrent neural networks, long short term memory nets, naive bayes, support vector machine, sentiment analysis, natural language processing, pandas, numpy, keras, tensorflow

## 1 Introduction

Sentiment Analysis, or opinion mining, refers to the use of text analysis, and natural language processing, to identify and extract subjective information in text. Our focus was on constructing a set of models to accurately determine the sentiment of any provided review.

The dataset used in this project is the Norwegian Review Corpus (NoReC) [25]. This dataset contains more than 35 000 full text reviews which cover a range of categories, including: literature, movies, games and music. These reviews were scraped off of major Norwegian news sources [15]. For each full text review the dataset contains a manually assigned score of 1-6, which makes this a supervised learning problem. Since the dataset only contains Norwegian language reviews, we will not be using any prebuilt sentiment analysis models, as none of them have been trained on Norwegian text.

The project will detail the construction of several Python [24] models designed to identify positive and negative reviews. We will be using major Python libraries such as NumPy [18], Pandas [16] and NLTK [4] for processing. While scikit-learn [19], Keras and TensorFlow [1] will be used for model construction, and matplotlib [11] will be used for plotting results.

*All authors contributed equally to this research.

*Project in Data Science (DAT550), IDE, UiS*
2020.

### 1.1 Related Work

- BERT - Bidirectional Encoder Representations from Transformers, is a a popular NLP (Natural Language Processing) library developed by Google. [7]
- Mining opinion components from unstructured reviews: A review. [14]

## 2 Background

In the contemporary world there is immense access to information, enough to create one of the problems of the modern age: Chatrooms, forums, social-media, news sites, and so on generate far more information than any human being can process [26].

Ordinary search engines can tell if two documents both deal with the same topic, but they will not tell you what sentiments are expressed towards the subject [10, 22]. So if you look for material on the flat-earth theory, aliens, or nuclear energy, you will get everything related to these topics, pro and con.

By combining regular search methods with sentiment analysis it is possible to filter the search results, so as to only include material with a certain attitude towards the subject matter [22]. That is it is possible to specifically find articles debunking or disproving fringe theories. Or to find articles promoting or criticizing a certain political point of view. Or, of course, to find negative or positive reviews.

This ability is overall a good thing, even though it can result in a "media bubbles", or "echo chamber", where individuals can shelter themselves from dissenting views [2]. Such bubbles tend to be self-generating even without technology, and technology can also be used to pop the bubble. For instance political groups or corporations could use such methods to discover how popular a policy or product really is. As opposed to how popular it is with the members of an in-group.

Our report will be using the NoReC collection [25], which was gathered specifically to assist in developing document level sentiment analysis. Further these documents were all explicitly created to be reviews, leaving us with the relatively humble goal of deciding whether a review is positive or negative.

# 3 Dataset

In this chapter the dataset and its key features will be introduced.

## 3.1 Dataset

The dataset contain just shy of 43 000 reviews divided into 35 000 training reviews, about 4000 testing, and 4000 development reviews. The NoReC dataset [25] consists of three files, a zipped archive consisting of coNLL-U files [21], a zipped archive containing all the reviews in plain text files, and finally a JSON file containing metadata for each review. The size of the dataset is roughly 1.4 GB.

The metadata file contains 17 fields, some of which are irrelevant for our purpose. Given that the classification is to be done by rating we found that category, rating, id and tags were very relevant. On the other hand fields like year, author, day, month, and URL, were judged irrelevant. There were also two fields, title and excerpt which included the first few lines from the plain text files. These fields were also discarded, as everything in them was also in the plain text file section for content. The metadata also contained two fields containing pros, and cons, these fields were mostly empty, and we considered dropping them. However we decided against this when we discovered that all reviews from Dinside used these fields.

For this project the coNLL-U files [5, 21] will not be used. These files are concerned with teaching complex language models to a computer, i.e. teaching it the inherent meaning of a word. Since this is not the goal of this project these files were discarded.

## 3.2 Structure

Within the dataset the overall distribution of features (Table 1) are heavily biased toward a majority value. For instance the `movie`-tag effectively define $\approx$ 27% of all reviews based on the tags-feature. Additionally, and more problematic for supervised learning is the distribution between good and bad ratings. We will mostly work under the presumption that a review is either positive or negative, thus we define a review as negative if the original rating is $1-3$, and positive if it's between $4-6$, this results in the dataset containing $\approx$ 76% positive reviews, and $\approx$ 24% negative reviews.

The data is collected from seven different sources. Data is collected from the respective web pages of these sources. Two of these sources are pure websites, five are newspapers (sa, dagbladet, fvn, bt, ap), and one (p3) is a radio channel. The dataset consist of 8 main categories, where music is the majority class encompassing 30% of all reviews.

# 4 Methodology

## 4.1 Preprocessing

Preprocessing was done in two steps:

(1) Loading the metadata file into a Pandas dataframe, and removing unnecessary columns from it. The full text reviews (content) would be loaded separately and eventually added as a new column to the same dataframe. This would ensure that the most important data was kept together with the content itself.
(2) Removing common stopwords, and performing stemming on all the words in the content, pros and cons column. The pros and cons column was however only used for reviews published by Dinside.

Columns like when a review was written, who wrote the review and any other column which would have had no impact on the training was dropped from the metadata. There were two columns, title and excerpt which essentially was a brief introduction to the review. These columns were also dropped because they were repeated in the raw text files, which were loaded directly for each review in a new column named content. The `content`-column contained the full review of the articles and as such this field had to be processed further. The processing consisted of using the NLTK (Natural Language ToolKit) Python package [4], this package contain essential modules and wordlists that were used. Specifically the nltk.stopwords list and the snowball stemmer provided in the nltk.stem module. These were used to efficiently remove Norwegian stopwords and stem the words from the text-fields. Stemming is helpful because it tries to return a word to its root. Regular expressions were also used to filter away symbols and numbers.

## 4.2 Initial Testing

Testing the different algorithms used in this project required different embeddings for optimal results, the content, pros and cons fields had to be embedded in a numerical way so that they could be processed by the models. Embeddings like hashed ngrams, bucket of words, etc. were primarily used in these initial steps. Whenever ngram was used, the ngram size was either 2 or 4.

Multiple variations were tested:

(1) Mixed (Content, Pros and Cons) with ngrams and bucket of words.
(2) Mixed (Content, Pros and Cons) with only ngrams.
(3) Mixed (Content, Pros and Cons) and bucket of words.
(4) Content only ngrams.
(5) Content only word bucket.

Where ngrams were used the 10000 most common ngrams were chosen and fed into a count matrix (i.e. the occurrences of each ngram in each article was counted). In variation 1 and 2, the 6000 most common words for content, and the 500 most common words for both pros and cons were used for the construction of the count matrix. Similarly for the pure word bucket, the 6000 most common words for content, and 500 most common words for pros and cons were chosen.

| Category | Count | Percent |
|---|---|---|
| music | 13204 | 30.41 |
| misc | 4619 | 10.64 |
| literature | 4313 | 9.93 |
| products | 3470 | 7.99 |
| games | 1799 | 4.14 |
| restaurants | 915 | 2.11 |
| stage | 764 | 1.76 |
| sports | 233 | 0.54 |

| Source | Count | Percent |
|---|---|---|
| sa | 6996 | 16.11 |
| dagbladet | 6693 | 15.42 |
| p3 | 5708 | 13.15 |
| fvn | 3348 | 7.71 |
| dinside | 3280 | 7.55 |
| bt | 2589 | 5.96 |
| ap | 2139 | 4.93 |

| Rating | Count | Percent |
|---|---|---|
| 6 | 2722 | 6.27 |
| 5 | 16046 | 36.96 |
| 4 | 14142 | 32.57 |
| 3 | 7477 | 17.22 |
| 2 | 2768 | 6.38 |
| 1 | 459 | 1.06 |

| Tags* | Count | Percent |
|---|---|---|
| ['movie'] | 11595 | 26.71 |
| ['tv'] | 1785 | 4.11 |
| ['culture'] | 1141 | 2.63 |
| ['car'] | 744 | 1.71 |
| ['concert'] | 712 | 1.64 |
| ['theatre'] | 350 | 0.81 |
| ['technology'] | 70 | 0.16 |

* only reviews with a single tag included

**Table 1: Sorted distribution frequency of non-content key-features found within the NoReC dataset.**

Once the count matrix was generated it was tested with normalized columns (N(0,1)) and simple count columns. Whichever count matrix provided the best result was kept. Generating the count matrix and normalizing it (where applicable) was done by running a script developed for this project.

During this initial phase; convention was followed and the part of the dataset labelled training was used for training purposes, and the part labelled dev was used for testing which algorithm was most suitable. Given that SVM is extremely resource intensive it was decided to exclude it from the initial testing, instead attempting to use: Ada boost, Naïve Bayes Gaussian, K-nearest Neighbours, Decision Tree, Random Forest, and MLP.

The built in models from scikit-learn was used when constructing these models. These specific models were chosen due to their easy implementation, and due to pre-existing familiarity with the scikit-learn framework [19].

As results from section 5.1.1 had been consistent when using the generated column matrices, it was decided to use existing libraries for the SVM and LSTM models (see section 6.2 for details). In addition it was decided to put Ada Boost aside temporarily while testing SVM and LSTM.

## 4.3 Models

During the construction of models scikit-learn [19] and TensorFlow [1] was mostly used.

*4.3.1 K-nearest neighbours(KNN)* The model was implemented with scikit-learns KNeighborsClassifier. Number of neighbours were set to the square root of the number of entries in the learning set.

K-nearest neighbours proved not only be computational intensive, but also became very unstable when using binary split.

*4.3.2 Decision Tree and Random Forest* Implemented with sklearn DecisionTreeClassifier and RandomForestClassifier respectively. No limits were placed on the number of layers. Both of these methods provided consistently good results with a low computing overhead.

*4.3.3 Multi Layer Perceptron (MLP)* Implemented with sklearns MLPClassifier, and a maximum iteration of 2000, learning rate being set to adaptive.

Although it was overall good at making predictions, it was very resource demanding and became less stable when trained on two classes in comparison to three classes.

*4.3.4 Naive Gaussian Bayes* Implemented with sklearns KNeighborsClassifier, it was one of the simplest methods available but provided remarkably good results. However, it seems to have suffered from the biased reviews.

*4.3.5 Adaptive Boost* Implemented with AdaBoostClassifier, n_estimators set to 200 and using the SAMME.R algorithm [27].

Adaptive Boost proved quite consistent, which is reasonable since it is explicitly made to be adaptive when used with weak learners [8].

*4.3.6 Support Vector Machine/Classifier* All models were built using scikit-learn's support vector classifier, but several parameters were tried, and we went forward with the values that gave the best result. Initially the content datafield was collected and a count vectorizer applied to it, returning an array of how frequently a word occurs. This count vectorizer was replaced with a Term Frequency Inverse Document Frequency(TF-IDF) vectorizer, to lower the high dimensionality we got when using the countvectorizer. By using the TF-IDF vectorizer we could more easily filter out terms which occured too frequently, thus lowering the dimensionality of the data. A sweetspot was discovered in the results when filtering out terms with a TF-IDF score above 5.0, and below 0.4, this lowered the number of terms from ~380 000 using Count Vectorizer down to a total of ~65 000 terms.

Initially only two variables were changed in the models: the gamma function and the kernel. Once the optimal kernel had been picked, specifically the linear kernel(Table 3), we constructed two models using a threeway split in the ratings. The first model used a 'one-versus-one' approach, while the second model used a 'K one-versus-rest' approach.

Finally 5-fold cross-validation was performed on the optimal kernel, and validation and learning curves were constructed(Figure 3). This was done to ensure that the reported results were real, and not a result of excessive under-/overfitting.

*4.3.7 Recurrent Neural Network with LSTM* RNNs with LSTM are widely used for natural language processing, voice recognition and sentiment analysis [20]. Significant computational power is required, the dataset used has a huge vocabulary size (368281 unique words!) and therefore was limited to the 25 000 most common words. And the average length of a review in terms of the amount of words were roughly 256, meaning reviews would be padded to a length of 256 words. Padded in this sense means that any review less than 256 words will have the requisite number of zeros placed in front of the initial embedded vector. Similarly a vector that was too long would be cropped. This ensures that all reviews have the same length when being fed to the neural network.

To be able to feed numerical data to the neural net a matrix with the embedded content was constructed. To be specific each word was represented by its frequency. That is to say the most common word would be represented as 1, while the fifth most common word would be represented as 5, etc.

There is a weakness in the way traditional RNN memory works, since their back-propagation necessarily use floating point numbers. This means that a gradient may vanish or explode into infinity. The two most common methods for countering this is LSTM (Long Short Term Memory Net) and GRU (Gated Recurrent Unit). Both ensure that the neurons will be able to update their weights based on previous knowledge. The reason for LSTM being preferred over GRU is that LSTM has more parameters and can therefore be tuned more, and GRUs will in most cases perform poorly when trying to learn simple languages. [6]

Tensorflow was used to train, test and evaluate the RNN model. Tensorflow can run on the CPU or GPU, which is useful since running deep neural networks on a CPU alone can be extremely time consuming. For this project a GPU was utilized (specifically an NVIDIA graphics card) for training, testing and evaluating the model. Supported NVIDIA GPUs can utilize CUDA to improve parallelization, allowing for GPU acceleration. In our case the use of GPU acceleration sped up the train and test time considerably. [23]

Training neural networks is normally not straight forward, due to all the work that is happening in the hidden layers. Hyper parameter tuning is necessary in order to identify the most suitable parameters for the model, the different parameters used for this model are listed in table 2.

| | |
|---|---|
| Optimizer | adam, rmsprop |
| LSTM Units | 25, 50, 100 |
| Dropout | 0, 0.1, 0.2 |
| Learning Rate | 0.001, 0.003, 0.01, 0.1 |
| Batch Size | 64, 128, 256 |
| Epochs | 3 |
| Vocabulary | 25000 |
| Embedding Size | 100 |

**Table 2: RNN LSTM Hyper Parameters**

Training, testing and evaluating took roughly 4 hours and 30 minutes, this was done for both multiclass and binary sentiments. Multiclass sentiments would be separated into negative, neutral and positive (0,1,2). Binary sentiments were either negative or positive (0,1).

*4.3.8 Multinomial Naive Bayes* This is a classifier based on Bayes probability theory, but with the assumption of full independence. This assumption makes the classifier naive, and relatively fast to train. The reviews are represented as a bag of words, each review is converted to a vector. [12] Each review will be a row with a length equal to the total amount of unique words in the content itself(e.g. as in figure 1). For each column in this row there will be a frequency of words that are present for that review, words that are not present will be set to zero. This way the classifier can detect patterns such as negative reviews having a large frequency of words like *bad, terrible, slow, crap, etc*.

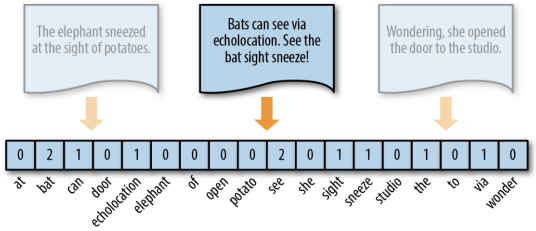



**Figure 1: Bag of Words embedding example. [9]**

SKLearns CountVectorizer was used to embed the reviews in a bag of words fashion, the transformation could be fitted directly on the classifier. To account for the high bias related to the division between positive and negative reviews, a *class_prior* keyword was used on the classifier to give extra weight to negative reviews.

## 5 Results

The models had varying results, mostly dependent on whether or not we carried out a binary or ternary split. In general the models with two outcomes had a higher accuracy, compared to the models with three outcomes, this difference was in general roughly 20%. In the following subsection we will present results based on both binary and ternary splits.

### 5.1 Models

*5.1.1 DT, RF, MLP, ADABoost, Gaussian Naive Bayes* In figure 2 and figure 3 it is clear that a three way split is unsatisfactory for all the classifiers. A binary split performs better, however the K-nearest neighbour algorithm becomes very unstable at this level.

For KNN, depending on how the data was chosen, accuracy was either in the 0.7 area or in the 0.25 area. There is no clear pattern, but it seems likely there is a "perverse" overlap from the classes being fairly close in N-dimensional space. This is likely to be a problem for other algorithms as well, even

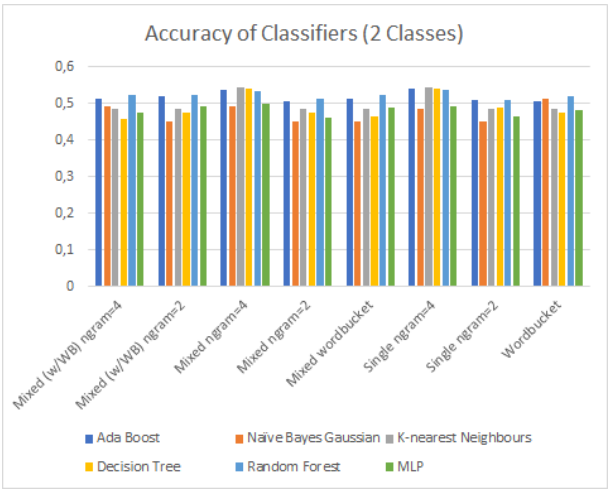



**Figure 2: Accuracy of Classifiers (2 Classes)**

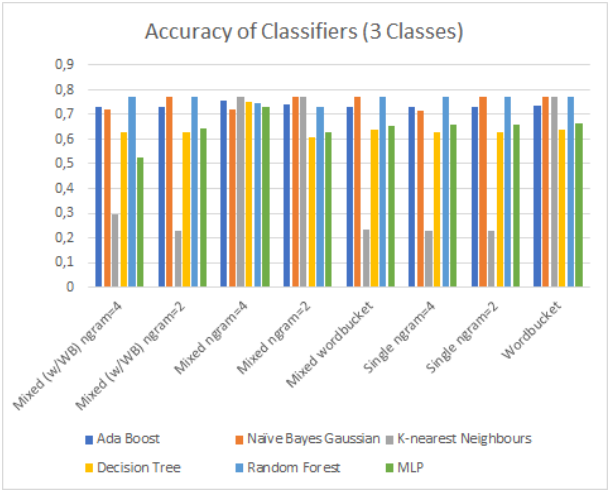



**Figure 3: Accuracy of Classifiers (3 Classes)**

though they do not express it to the same degree.

It was also observed in figure 2 and 3 that using the ngrams; from cons, pros, and content did not provide a significant boost in the model accuracy; relative to accuracy scores achieved by just using the content column.

Overall the results were remarkably stable regardless of what type of data was fed into the model. However as it was desirable to obtain an accuracy of at least 80 % we decided to proceed with tests on SVM and RNN.

*5.1.2 SVM* The overall execution times and accuracy scores of the SVC models have been summarized in table 3, once these results had been generated the best case models were rerun through scikit-learns inbuilt learning curve and cross-validation curves, to see the effect varying testsizes had on the results and whether or not the constructed models were excessively overfitted, see figure 4.

| **Binary Classification [positive and negative]** | | | | |
|---|---|---|---|---|
| **Kernel ǀ Gamma** | **Vector** | **Train[s]** | **Test[s]** | **Score[%]** |
| RBF ǀ auto | Count | 1185.54 | 358.37 | 75.29 |
| RBF ǀ scale | Count | 1089.76 | 371.47 | 76.22 |
| Linear ǀ auto | Count | 6771.56 | 240.39 | 81.74 |
| **Linear ǀ auto** | **TF-IDF** | **1668.44** | **283.11** | **85.82** |
| RBF ǀ scale | TF-IDF | 1217.50 | 339.58 | 75.23 |
| Polynom ǀ auto | TF-IDF | 695.38 | 289.29 | 75.44 |
| Polynom ǀ scale | TF-IDF | 699.33 | 290.59 | 75.83 |
| Sigmoid ǀ auto | TF-IDF | 714.67 | 296.17 | 75.53 |
| Sigmoid ǀ scale | TF-IDF | 1152.59 | 314.54 | 75.43 |
| **Three-split Classification [positive, neutral and negative]** | | | | |
| **Kernel ǀ Type** | **Vector** | **Train[s]** | **Test[s]** | **Score[%]** |
| Linear ǀ OvO | TF-IDF | 2645.79 | 472.93 | 71.26 |
| Linear ǀ OvR | TF-IDF | 2647.32 | 478.00 | 71.72 |

**Table 3: Execution time and accuracy score comparison between SVM-models**

| Accuracy | Loss | Optimizer | Batches | Units | Dropout | Learning Rate |
|---|---|---|---|---|---|---|
| 0.7981 | 0.522 | rmsprop | 64 | 100 | 0.0 | 0.003 |
| 0.7979 | 0.612 | rmsprop | 64 | 50 | 0.1 | 0.003 |
| 0.7957 | 0.617 | rmsprop | 64 | 25 | 0.2 | 0.01 |
| 0.795 | 0.5395 | rmsprop | 64 | 50 | 0.2 | 0.001 |
| 0.7943 | 0.544 | rmsprop | 64 | 100 | 0.2 | 0.01 |

**Table 4: RNN, parameters for the top 5 accuracies obtained when using multiclass sentiments.**

| Accuracy | Loss | Optimizer | Batches | Units | Dropout | Learning Rate |
|---|---|---|---|---|---|---|
| 0.7908 | 0.5127 | rmsprop | 64 | 25 | 0.2 | 0.001 |
| 0.7911 | 0.518 | rmsprop | 64 | 100 | 0.1 | 0.001 |
| 0.7981 | 0.522 | rmsprop | 64 | 100 | 0.0 | 0.003 |
| 0.789 | 0.5228 | rmsprop | 64 | 25 | 0.2 | 0.003 |
| 0.7728 | 0.5235 | rmsprop | 64 | 50 | 0.2 | 0.003 |

**Table 5: RNN, parameters for the 5 lowest loss obtained when using multiclass sentiments.**

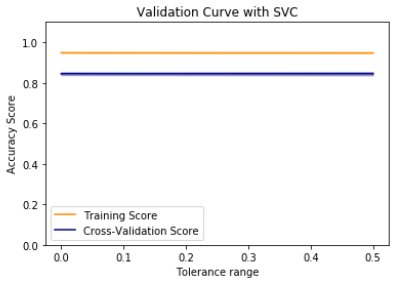
*5.1.3 RNN* Table 4, 5, 6 and 7 summarizes the most suitable parameters for the neural network when using either two or three classes. Two of them are based on the best accuracy and the other two are based on the lowest loss. And in figure 5b it is possible to extract the probability that the two class model will separate negative and positive reviews.

*5.1.4 Multinomial Naive Bayes* In figure 5a it is possible to extract the probability that the model will separate negative and positive reviews, and in table 8 it is possible to extract the score for this model. F1 score depicts the harmonic mean of the precision and recall, where precision reflects the rate of correct positive classifications and recall reflects a similar measure but for the opposite class.

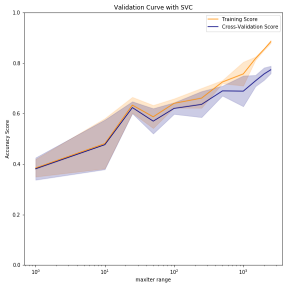
$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

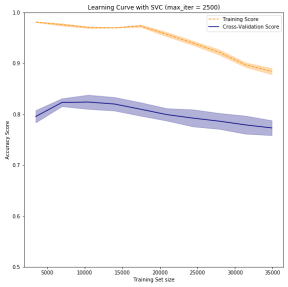
$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$




**(a) SVM validation curve**
**Varying tolerance**




**(b) SVM validation curve**
**Varying number of max iterations**




**(c) SVM learning curve**
**Varying ratio between training and testing data**
**Model is limited to 2500 iterations**

**Figure 4: Overview of some generated validation and learning curves based on builtin scikit-learn methods. Orange lines refer to testing directly on the training set, while blue lines refer to crossvalidation scores based on unseen data.**

| Accuracy | Loss | Optimizer | Batches | Units | Dropout | Learning Rate |
|---|---|---|---|---|---|---|
| 0.8513 | 0.4058 | rmsprop | 64 | 100 | 0.1 | 0.003 |
| 0.8488 | 0.3615 | rmsprop | 64 | 100 | 0.2 | 0.001 |
| 0.8484 | 0.3901 | rmsprop | 64 | 50 | 0.2 | 0.001 |
| 0.8471 | 0.3911 | rmsprop | 64 | 50 | 0.2 | 0.003 |
| 0.8459 | 0.3625 | rmsprop | 128 | 100 | 0.2 | 0.01 |

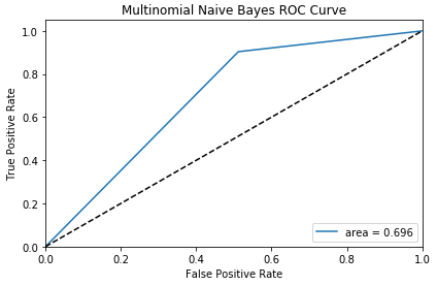
**Table 6: RNN, parameters for the top 5 accuracies obtained when using binary sentiments.**

| Accuracy | Loss | Optimizer | Batches | Units | Dropout | Learning Rate |
|---|---|---|---|---|---|---|
| 0.8488 | 0.3615 | rmsprop | 64 | 100 | 0.2 | 0.001 |
| 0.8459 | 0.3625 | rmsprop | 128 | 100 | 0.2 | 0.01 |
| 0.8421 | 0.3676 | rmsprop | 128 | 100 | 0.1 | 0.003 |
| 0.8337 | 0.3758 | rmsprop | 128 | 25 | 0.2 | 0.003 |
| 0.8407 | 0.3771 | rmsprop | 128 | 25 | 0.2 | 0.001 |

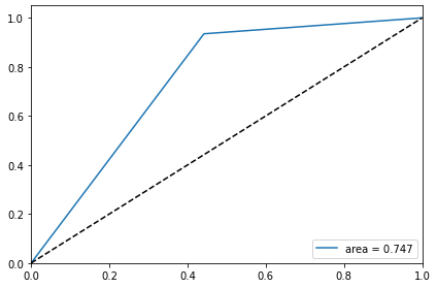
**Table 7: RNN, parameters for the 5 lowest loss obtained when using binary sentiments.**

| F1 | Recall | Precision |
|---|---|---|
| 0.881 | 0.904 | 0.859 |

**Table 8: Multionomial Naive Bayes score.**




**(a) ROC (Receiver Operating Characteristics) & AUC (Area Under Curve) curve for Multinomial Naive Bayes.**




**(b) ROC (Receiver Operating Characteristics) & AUC (Area Under Curve) curve for RNN with LSTM.**

## 6 Discussion

All the initial models detailed in section 5.1.1 and the initial SVM models had average accuracy scores of around 75%. As mentioned previously the dataset consists of about 75% positive reviews. A confusion matrix for one of the SVM models was constructed to see if these results were due to under-/overfitting, and if the models were able to classify negative reviews. The results were that the model could correctly classify 95% of all positive reviews, while it only manages to correctly classify 54% of all negative reviews. This indicates that the model performs poorly when predicting negative reviews, but it also show that the model does predict negative reviews; thus not always predicting the majority class.

### 6.1 Initial Testing

Though the results of the initial models (see section 5.1.1) had returned consistent accuracy scores based on all the various test inputs, it was decided that existing Python libraries were prioritized for generating the model inputs for the rest of the constructed models. Part of the reason for this decision was related to how poorly the ngram-based matrix performed when used as an input in the SVM model; the model building process took over 12 hours (when it was stopped) and possible workarounds were considered (section 5.1.1).

Due to the excessive execution time, the Curse of Dimensionality [3] was considered. And for this reason PCA was applied to the dataset in order to reduce the overall dimensionality. However after PCA had been performed, the overall dimensionality of the dataset had been reduced by 45%, at the cost of introducing a variance of 0.1. A second PCA-attempt was performed, where the goal was to get the variance below 0.01. The second attempt resulted in a dimensionality reduction of 9.7%. Both PCA attempts would however reduce the accuracy

by 50%, as compared to accuracies obtained before applying PCA.

Due to the mentioned issues with PCA, it was decided that library vectorizers were preferred based on their faster response, and lower impact on achieved model accuracy. In addition other dimensionality reduction methods were chosen. Such as placing an upper and lower bound on the TF-IDF vectorizer, or encoding each word in a dataset with a number (keras [13]).

## 6.2 Models

After rejecting the earlier naive approach to vectorizing the data (resulting in value-vectors of length 12000-25000) we decided to test a default scikit-learn CountVectorizer. It was however quickly decided that this was not optimal either, due to the excessive dimensions of the resulting vectors; - 380 000 value-vector. Though we could construct models using the CountVectorizer, this took several hours (see table 3 - rows where Vector is equal to Count). Therefore we began to consider further alternatives, of which three seemed most promising:

(1) SVM models created using a TF-IDF vectorizer with an upper and lower bound.
(2) Variations of the above where initial models were constructed using a limited subset of the most common ngrams.
(3) RNN models utilizing another frequency embedding.

Since classifying a review as either positive or negative is a linear problem, we made the initial assumption that a linear kernel would be optimal for constructing the model. Comparing the accuracy scores of the SVM models listed in table 3, it became clear that the non-linear kernels (RBF, polynomial and sigmoid) had an accuracy loss of about 10% relative to the linear kernel, supporting our earlier assumption.

## 6.3 Results

The results from the SVC models, and the result curves in figure 4, needs to be discussed a bit. The first subgraph a) is a validation curve where the tolerance variable is increased from 0.0001 to 0.5, but the maximum number of iteration the SVC-solver is allowed to do is not limited, and the solver will thus iterate until convergence is reached.

The second subgraph b) was then constructed to see the effect limiting the number of maximum iterations the solver is allowed to do based on the accuracy. This graph clearly show that the variance in the score is initially high (shaded area) with an overall poor accuracy, and as the number of maximum iterations increase, the overall accuracy of the model increase, and the uncertainty in the resulting score decrease. An interesting detail with this graph is that there is a sudden discrepancy between the cross-validation accuracy and the training accuracy indicating that the model has begun overfitting.

The final graph c) was implemented to see the effect of varying the size of the test-set, and the results were as expected. A peak was reached at about 20% test/train split, and as the relative size of the test-set increased the uncertainty and accuracy of the results decreased. This would indicate that the model had too little data to train on.

## 6.4 Limitations

We discovered early that the dataset was heavily biased towards positive reviews. This is undesirable for supervised learning, since it can "learn" that certain classes are inherently unlikely thus giving us false negatives.

Another problem is that NoReCs dataset is a selection made by someone, by which we mean that the selection of reviews is not a simple random selection from all available reviews of the period. This may, or may not, skew the samples, but it is hard to decide without a careful review of the selection process. Such a review is however beyond the scope of this project.

There are of course ways to get around this, for instance by giving increased weight to certain classes. Another method could be to randomly select an equal number of samples from each class, and then do a classification based on this. However, in terms of reviews this runs into the problem that this would not increase variety. In other words our samples still would not cover the full span of the class of negative reviews.

As all the reviews were in Norwegian we were not able to utilize any of the pre-existing sentiment analysis models. This is the case as the already existing models would not have been trained using Norwegian text, thus the models would not have been able to interpret for instance contextual clues. Furthermore, the natural language tool kits are not as extensive in Norwegian as they are in English, however the NLTK tools used in this project functioned adequately.

Additionally if there had been pre-existing Norwegian sentiment analysis models, these could have been used as a baseline, to further quantify how well the models constructed throughout this project actually performed.

Another challenge was the lack of computational power, most of the constructed models could not take advantage of GPU-acceleration. In the case of the scikit-learn models most of these couldn't even be multi-threaded.

# 7 Conclusion

Several models have been constructed to perform sentiment analysis on the NoReC dataset. We found that models utilizing linear decision boundaries provided better results in correctly classifying positive and negative reviews based on a manually assigned truth-value.

Hyper-parameter tuning were performed on all of the best

performing models to attempt to identify the most optimal parameters in this use case. We did however find that a lack of computational power proved to be the biggest bottleneck in this process, as properly computing the optimum parameters is extremely computationally expensive. It is thus our belief that we have found a set of good hyper-parameters, but further tuning and training using more epochs might lead to more optimal values for real world purposes.

Sentiment analysis comprises collection and processing of data which is then used to construct models that should identify the underlying sentiment of any provided text. The use cases for such models are enormous; and modern usages of this include e.g. delivering ads to users based on previous user traffic. Sentiment Analysis can be used for good by expanding the horizon of a person, an example being GoodReads which can recommend similar books by unknown authors based on your reading history and book ratings. It can however also be use maliciously by only providing a user with articles that the user agrees with, this might result in the Echo Chamber effect [2], the Cambridge Analytica case is another well known example of sentiment augmented advertisement delivery being abused.

## 8 Further Work

With more computational power it would have been possible to try more parameters, and thus made more in depth hyper-parameter tuning of the models.

If we had had more time we would also have looked into the possibility of expanding the possible model outcomes, from 2/3 outcomes to 6 outcomes. (classes) This was not done during this project due to the dataset being heavily biased toward positive reviews, a noticeable decrease in model accuracy was recorded when using three outcomes. The accuracy would have degraded even further with more outcomes, which evidently would require us to correct the bias in the dataset by adding more negative reviews.

Trying different word embeddings could have also been beneficial, such as word2vec, skipgram, continuous bag of words, etc. These embeddings have properties that help the model learn the context of words faster, which can improve the overall accuracy of the model. [17]

## References


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Pablo Barberá, John T. Jost, Jonathan Nagler, Joshua A. Tucker, and Richard Bonneau. 2015. Tweeting From Left to Right: Is Online Political Communication More Than an Echo Chamber? *Psychological Science* 26, 10 (2015), 1531–1542. https://doi.org/10.1177/0956797615594620 arXiv:https://doi.org/10.1177/0956797615594620 PMID: 26297377.

[3] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. 1957. *Dynamic Programming*. Princeton University Press. https://books.google.it/books?id=wdtoPwAACAAJ

[4] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

[5] Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *In Proceedings of the 10 th CoNLL*. 149–164.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Y. Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. (12 2014).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Yoav Freund and Robert E. Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, Paul Vitányi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 23–37.

[9] Shreya Ghelani. 2019. From Word Embeddings to Pretrained Language Models — A New Age in NLP. https://towardsdatascience.com/from-word-embeddings-to-pretrained-language-models-a-new-age-in-nlp-part-1-7ed0c7f3dfc5 [Online; accessed April 21, 2020].

[10] Louis A Gottschalk and Goldine C Gleser. 1979. *The measurement of psychological states through the content analysis of verbal behavior*. Univ of California Press.

[11] John D Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering* 9, 3 (2007), 90–95.

[12] Daniel Jurafsky and James Martin. 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Vol. 2.

[13] Keras. 2020. IMDB Movie reviews sentiment classification. https://keras.io/datasets/ [Online; accessed April 24, 2020].

[14] Aurnagzeb Khan Ashraf Ullah Khairullah Khan, Baharum Baharudin. 2014. Mining opinion components from unstructured reviews: A review. *Journal of King Saud University-Computer and Information Sciences, 26(3), 258-275* (2014).

[15] ltgoslo/norec. 2019. NoReC: The Norwegian Review Corpus. https://github.com/ltgoslo/norec.

[16] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems* 26 (10 2013).

[18] Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[20] Suman V. Ravuri and Andreas Stolcke. 2015. Recurrent neural network and LSTM models for lexical utterance classification.. In *INTERSPEECH*. ISCA, 135–139. http://dblp.uni-trier.de/db/conf/interspeech/interspeech2015.html#RavuriS15

[21] Emil Stenstrom. 2020. CoNLL-U Parser. https://github.com/EmilStenstrom/conllu.

[22] Philip J Stone, Dexter C Dunphy, and Marshall S Smith. 1966. The general inquirer: A computer approach to content analysis. (1966).

[23] Heronimus Tresy Renata Adie, Ignatius Aldi Pradana, and Pranowo Pranowo. 2018. Parallel Computing Accelerated Image Inpainting using GPU CUDA, Theano, and Tensorflow. 621–625. https://doi.org/10.1109/ICITEED.2018.8534858

[24] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[25] Erik Velldal, Lilja Øvrelid, Eivind Alexander Bergem, Cathrine Stadsnes, Samia Touileb, and Fredrik Jørgensen. 2018. NoReC: The Norwegian Review Corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Nicoletta Calzolari (Conference chair), Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga (Eds.). European Language Resources Association (ELRA), Miyazaki, Japan.

[26] Tomasz Wiktorski. 2019. *Introduction*. Springer International Publishing. https://doi.org/10.1007/978-3-030-04603-3_2

[27] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. 2006. Multi-class AdaBoost. *Statistics and its interface* 2 (02 2006). https://doi.org/10.4310/SII.2009.v2.n3.a8