NLP Coursework Part A: Sentiment Analysis

Adam El Kholy

Bachelor of Computer Science with Artificial Intelligence
The University of Bath
March 21, 2024

Contents

Abstract

Sentiment analysis poses a useful and technically challenging task in the field of Natural Language Processing. In this report we investigate the leading methods of sentiment analysis using a dataset of 4000 IMDB movie reviews, Maas et al. (2011), sorted into positive and negative classes based on their numerical rating. We aim to evaluate different techniques of feature extraction as well as

differing model architectures in order to determine the highest performing approach. We develop our own classification models for comparison with the existing sklearn implementations and compare the metrics of each (accuracy, precision, recall and f1 score). Through our evaluations we hope to gain further insight into the available methods of sentiment analysis and the factors which may increase or decrease performance of a given model and feature set. We aim to identify the challenges and associated solutions which may arise in our experimentations in order to develop a clearer understanding of currently available methodologies.

Chapter 1

Introduction

1.1 Sentiment Analysis

We can define sentiment analysis as the task of determining the sentiment latent in a given text, where sentiment is a positive or negative evaluation expressed through language, Taboada (2016). It can therefore be treated, as in our case, as a binary classification task. The task is important in the field of Natural Language Processing (NLP) due to its many applications and uses. Surveys (e.g. questionnaires) offer valuable qualitative data for businesses, however analysing responses is often time consuming and labour intensive. Sentiment analysis can be used in such cases to automate the task of reading through customer feedback, saving time and money for the surveyors. Similarly in the context of research, interviews produce high quality data with the drawback being that a researcher is required to transcribe and conduct a content analysis after the fact. Once again sentiment analysis has a clear application here, allowing researchers to automate the process and direct their resources elsewhere. Sentiment analysis thus demonstrates its importance clearly in the context of research and business. Coupled with the challenges required to overcome in order to conduct sentiment analysis, the task proves to be valuable in the context of NLP research.

1.2 Challenges

Sentiment analysis is by no means a trivial task and it poses many challenges to overcome during development. There are common issues faced when dealing with any Natural Language Processing (NLP) task as well as problems specific to sentiment analysis. Zipf's law states that the most commonly used word in a language appears twice as often as the second most common word, Upton and Cook (2014). The usefulness of these high frequency words in sentiment analysis is very limited as they appear ubiquitously across all classes. In the English language for example a frequently used connective such as "and" is of little use to us in understanding whether a given text is positive or negative in expression as it appears frequently across both classes. These words are referred to as stop-words and are often removed from the data altogether in NLP tasks. Specific to sentiment analysis, the use of language in certain contexts pose a difficult challenge to overcome. Sarcasm, idioms and expressions are challenging to identify in writing and often times an uncommon use of a word will confuse a model's analysis. Take the toy example sentence of a negative movie review below.

"This film cheesed me off, I would love to give the director a piece of my mind"

The uncommon expression "cheesed me off" (i.e. "annoyed me") has an unintuitive meaning and is particularly difficult for a language model to decode due to the novel use of common words (the noun cheese is used here as a verb). The use of the word "love" is overwhelmingly positive in most use cases, however in our example it is being used in conjunction with the expression "to give [object] a piece of my mind" to denote a negative feeling. In this example sentiment must be heavily inferred from context and knowledge of English expressions. The inference, contextual and linguistic knowledge required to decode such a statement poses a severe challenge in sentiment analysis whilst it may come naturally to a native English speaker.

1.3 Approaches

For feature selection and normalisation the following approaches are available to us

- N-grams
- Tokenization
- Stemming
- Lemmatization
- Stopword removal
- Term Frequency Inverse Document Frequency (TF-IDF)
- Frequency Normalisation
- Positive Pointwise Mutual Information (PPMI)

Of these approaches we intend to evaluate all but Frequency Normalisation and PPMI which have been omitted due to their complexity to implement. The following are the model architectures available to us

- Multinomial Naïve Bayes (MNB)
- Gaussian Naïve Bayes (GNB)
- Logistic Regression with Stochastic Gradient Descent (SGD)
- Support Vector Machine (SVM)
- Bidirectional Encoder Representatons from Transformers (BERT)

We will test and evaluate at least one version of each of the above models. We also intend to implement our own MNB and GNB models and compare these to the sklearn models, Pedregosa et al. (2011). Note that all models will be trained upon the same corpus, split into the same test, train and development sets. We will also evaluate the sklearn SVM and SGD models and the pre-trained BERT model from HuggingFace, Wolf et al. (2019). It must also be noted that the Guassian Naïve Bayes approach is not listed in the specification of this project. This different approach was chosen due to the use of TF-IDF feature normalisation which generates continuous non binary features in the range of [0, 1], thus prompting experimentation with a probability distribution based model such as GNB.

1.4 Related Work

In our experiments we use a subset of the IMDB movie review dataset, Maas et al. (2011), containing 2000 positive and 2000 negative movie reviews. As well as the methods we intend to test and evaluate there exist other approaches to sentiment analysis which are worthy of discussion. Deep learning techniques such as Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) networks can consider historical contextual data within a text with more nuance than any of the models we intend to test, Yu et al. (2019). The potentially higher performance of these models is offset by their computational complexity however as these approaches are by no means trivial to implement and can potentially consume a large amount of computational resources during training.

On the other end of the spectrum dictionary based approaches are perhaps the most simple methods of sentiment analysis, Van Atteveldt, Van der Velden and Boukes (2021). Scores are assigned to texts through use of predefined dictionaries which score words on their sentiment (positive, negative, or neutral). These simpler methods may lack the complexity to generalise well however and will be unable to consider context in their evaluation. The machine learning models we intend to test all fall under the umbrella of supervised learning approaches and were we to lack labelled data we would require unsupervised approaches instead. Latent Dirichlet Allocation (LDA) Blei, Ng and Jordan (2003), considers documents as mixtures of various topics and topics as distributions of words in order to generate a class label for a given text. Were we dealing with unsupervised approaches and more class labels than binary (positive, negative), LDA would be a sensible approach to take.

Chapter 2

Experiments and Results

2.1 N-Grams

N-Gram generation is an NLP technique which we can use to extract features from our dataset in order to conduct sentiment analysis. With N-Gram generation we group together consecutive *n* runs of words and use these as features. The intuition behind N-Grams is that, to borrow from linguist John Firth, "you shall know a word by the company it keeps". Contextual information is highly important in NLP tasks as the meaning of a given word can significantly vary depending on its context. Our model may thus achieve higher performance when extracting consecutive pairs (bigrams) or triplets (trigrams) of words from the corpus as opposed to single tokens (unigrams) as contextual information is better preserved in the former. N-Gram generation with higher values of *n* does increase the computational complexity substantially however as we generate a higher number of unique features. Below is an example of N-Gram generation (*for n = 1, 2, 3*) using a quote from semiotician Jacques Derrida.

"There is nothing outside the context", Derrida (2016)

```
Unigrams: (There) (is) (nothing) (outside) (the) (context)
Bigrams: (There is) (is nothing) (nothing outside) (outside the) (the
   context)
Trigrams: (There is nothing) (is nothing outside) (nothing outside the)
    (outside the context)
```

Despite the advantage of increased contextual information when using N-Grams, we use unigram feature generation for our feature sets. After preliminary experiments it becomes evident that the added computational complexity arising from a higher feature count renders bigrams and trigrams as unviable options for feature extraction. Table 2.1 shows the number of unique words in the shared vocabulary of the corpus, accuracy achieved with our own MNB model (*see section 2.6*) and time taken to train across varying N-Gram sizes.

N-Grams	Words in Shared Vocabulary	Training Time (minutes)	Accuracy
Unigrams	44377	1	0.852
Bigrams	392191	92	0.6375
Trigrams	509607	Untested	Untested

Table 2.1: Evaluation of N-Grams using our own MNB model (*see section 2.6*)

The higher number of unique words for bigrams and trigrams renders them unusable as the time complexity of training increases massively. As shown in Table 2.1 the time taken for training a feature set using bigrams is far higher than the same set using unigrams. In addition the accuracy of the bigram feature set is significantly lower than that of the unigram set. There could be multiple reasons for this, such as a lack of quality features in the over inflated bigram shared vocabulary. Consequently we use unigrams in our feature sets, as it is unviable to train every model for over an hour each time during development and because the unigram set appears to afford us a higher accuracy. Trigrams were left untested due to lack of computational resources. Potential further experimentation could be conducted using bigrams and trigrams given greater computational resources, however we will focus solely on unigrams for our investigation into sentiment analysis.

2.2 Feature Selection

Through the use of a combination of Tokenization, Stemming/Lemmatization, Stop-word Removal, and TF-IDF scoring we generate three feature sets to train and evaluate our models upon. Below is a short description of each technique and its application to sentiment analysis.

Tokenization: Tokenization is simply the process of converting a block of text into individual tokens, which are often single words. In doing so we complete the first step in the feature selection process as individual tokens can be used as features whereas entire paragraphs (in most cases) cannot. For our purposes we will split our texts into tokens whereby each word is a token and whitespace, punctuation and stop-words (*see below*) are removed.

Stop-word Removal: In accordance with Zipf's law, Upton and Cook (2014), the most commonly used words in a language are seen so often across the corpus that their usefulnessin sentiment analysis becomes limited. These words are present in both positive and negative examples with similar frequencies, meaning they cannot be used to discern between classes. Conversely words which appear rarely may be of great use to us in categorising a review as positive or negative. For example if we know that the word "dreadful" appears in ten reviews, nine of which are negative, the word then becomes a useful feature for discerning negative reviews as opposed to the word "and" which appears equally as frequently in both classes. Therefore we simply remove these frequently used useless words, which are referred to as stop-words, in order to preserve only those features which are useful for our model in distinguishing between classes.

Stemming/Lemmatization: One challenge faced in NLP tasks is the notion of morphology within language. Morphology refers to conjugations, prefixes and suffixes applied to words to change their meaning. For example the word run can become runner, running, ran etc. For sentiment analysis we would like to simplify these morphed words in order to glean the common meaning for our model (in this case that would be "run"). Stemming and lemmatization allow us to do this by reducing words to either their stem or lemma. A word stem is the root of a word, Merriam-Webster (1990), and a lemma is "a word without any morphological changes made to it", Merriam-Webster (1990). The meaning of the words in our corpus is thus preserved whilst different forms of the same word are removed. By reducing the words in our dataset to their lemmas or stems we reduce the complexity of the task and allow for simpler feature selection in sentiment analysis.

TF-IDF: Similar to some of the aforementioned methods the motivation behind Term Frequency - Inverse Document Frequency (TF-IDF) scoring is to utilise word frequency data to extract

features from our dataset. Term Frequency refers to the count of terms within a single document as a proportion of the total words in said document. Inverse Document Frequency is the reciprocal of the number of documents containing a given word as a proportion of the total number of documents. We then multiply the two values together to generate a TF-IDF score which is then used as a feature in our classification models. A high TF-IDF score for a given feature indicates that it is of use as a distinguishing feature between classes due to its rare occurence in the corpus. For example a word such as "and" will have a low TF-IDF score due to its abundance across documents of all classes. TF-IDF scores also allow us a method of feature normalisation as the original text consisting of words has now been encoded as numerical values.

2.3 Feature Sets

Having discussed the feature selection methods available we now list the combinations of approaches used in generating our three feature sets. Note that all sets have been subject to tokenisation.

1. **Stemming**: Stemming with Unigrams and TF-IDF. As explained in section 2.1 any N-Gram higher than unigrams is unviable due to lack of computational resources. In this set we choose not to remove stop-words in order to assess the performance of this feature set against those which use stop-word removal, in order to discern its usefulness in sentiment analysis.
2. **Stemming + Stop-words**: Stemming with Unigrams, stop-word removal and TF-IDF. In this set we remove stop-words to compare with **Set 1** as both sets are otherwise identical.
3. **Lemmatization + Stop-words**: Lemmatization with Unigrams, stop-word removal and TF-IDF. This set is identical to **Set 2** aside from the use of lemmatization over stemming, allowing us to compare the performance of the two methods.

2.4 Data Splits

For our sentiment analysis models we split the dataset into three subsets; training (70%), testing (20%) and development (10%). The training set is used to calculate the relevant likelihoods, priors and weights of our models. The test set is used to evaluate the performance of our models after hyperparameter tuning. Finally the development set is used after training and before testing in order to tune the relevant hyperparameters of our models and make any necessary adjustments in order to increase performance. We use the sklearn *test_train_split* function with a constant value for the *random_state* parameter in order to ensure the same splits across all models. Defining a sensible split of the data is crucial in implementing a model that generalises well. Separate sets must be reserved for testing and development because the model will achieve 100% accuracy if evaluated on the training data. Similarly we must separate the development and testing data in order to fairly assess the performance of our models, as test data is by definition unseen. By defining a split of data in this way we aim to maximise the performance of our models and increase their generalisability.

2.5 Bayesian Models

Bayesian models are a popular choice for classification tasks and in this case they will be used for sentiment analysis. The intuition behind Bayesian models stems from Bayes' rule which states that the probability of an event A occurring given event B can be calculated when using the Naïve independence assumption.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

In training we calculate the priors and likelihoods for our classes (positive and negative) which are then used to classify new inputs. Priors refer to the proportion of documents belonging to a given class.

$$P(C) = \frac{\text{Number of documents belonging to class } C}{\text{Total number of documents}}$$

Likelihoods are calculated per feature and indicate the probability that a document with said feature belongs to a given class.

$$P(\text{feature } f | \text{ class } C) = \frac{\text{Number of documents in class } C \text{ with feature } f}{\text{Total number of documents in class } C}$$

Putting the above equations together we can estimate the probability that a document x belongs to a given class C. Because the denominator *p(x)* remains constant across all calculations it is dropped, as it has no effect on the final comparison of probabilities between classes. Thus we have a formula for determining the probability that a given document belongs to a given class which will be the central mechanism of our Bayesian models.

$$P(C|x) = \prod_{i=1}^{n} P(x_i|C) \cdot P(C)$$

We often take the logarithm of all calculations in order to avoid dealing with very small numbers, giving us our final mathematical model.

$$P(C|x) = \sum_{i=1}^{n} \log P(x_i|C) + \log P(C)$$

2.6 Multinomial Naïve Bayes

For evaluating Multinomial Naïve Bayes (MNB) for sentiment analysis on our dataset we implement our own model and compare it against the sklearn implementation. We make the Naïve assumption that our features are independent of one another, i.e. that the presence of one feature does not impact the likelihood of another feature's presence. This assumption is made purely for modelling purposes, in actuality it does not stand. This particular approach to Bayesian learning is known as Multinomial as we assume a multinomial distribution of features among our documents, from which we can infer class likelihoods and probabilities. The class likelihoods and priors are calculated in training and used to classify new documents as per the formulae in section 2.5. A vector of TF-IDF scores is passed into the model as input when classifying a new document. We then sum the logarithms of the feature likelihoods for each present feature for each class (positive and negative). After then adding the logarithm of the

given class prior we have a value for each class representing the probability of the input document belonging to said class. In order to avoid taking the logarithm of 0 at any point we apply Laplace Smoothing to the likelihoods, whereby we add a constant *alpha* (= 1) to avoid any 0 values. Whichever value is higher determines the class label. Note that the values being compared are not strictly probabilities in the range of [0, 1] due to the application of the logarithm in order to avoid very small probabilities. Table 2.2 shows an evaluation on dev set of both models.

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Stemming	Ours	0.802	0.927	0.623	0.745
	Sklearn	0.835	0.779	0.901	0.836
Stemming + Stop-Words	Ours	0.875	0.927	0.794	0.855
	Sklearn	0.856	0.816	0.892	0.852
Lemmatization + Stop-Words	Ours	0.867	0.939	0.762	0.841
	Sklearn	0.865	0.846	0.865	0.856

Table 2.2: Comparison of performance on feature sets for our own Multinomial Naïve Bayes model and the pre-built sklearn MNB model.

It is clear that removing stopwords increases performance across both models. Lemmatization and stemming with stop-word removal both perform highly on the development set with little difference in evaluation metrics. We will evaluate our models on the test set using stemming and stop-word removal because of the slightly higher accuracy in comparison to the lemmatization set. Our own implementation of MNB generally matches or outperforms the sklearn model on the feature sets with stop-word removal. Table 3.1 (*see section 3.1*) shows the performance on the test set of both models on the stemming with stop-word removal feature set.

2.7 Gaussian Naïve Bayes

As opposed to MNB, Guassian Naïve Bayes (GNB) assumes a Guassian distribution of values for each feature. In practice this means that we calculate a probability distribution for each possible feature during training and fit the value for a feature in the given document to the distribution during classification. As before we implement our own GNB model and compare its performance to the pre-built sklearn model. GNB has the added potential advantage over MNB of being more robust and less sensitive to small datasets, as well as better suited to continuous data. In our own implementation of GNB we pre-process the mean and variance of each feature's TF-IDF scores across all documents. This is then used to calculate the likelihood of a particular document's feature belonging to a given class using the following formula.

$$P(x_i|C) = \frac{1}{\sqrt{2\pi\sigma_{C,i}^2}} \cdot \exp\left(-\frac{(x_i - \mu_{C,i})^2}{2\sigma_{C,i}^2}\right)$$

We take the logarithm of this class likelihood and sum it for each feature in the input document and finally we add the logarithm of the class prior, giving us a value indicating the probability of the document belonging to each class as before with our MNB model. Table 2.4 shows the results on the development set of our own GNB model and the prebuilt sklearn model across our three feature sets.

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Stemming	Ours	0.860	0.831	0.879	0.854
	Sklearn	0.650	0.612	0.672	0.641
Stemming + Stop-Words	Ours	0.865	0.829	0.892	0.860
	Sklearn	0.650	0.611	0.677	0.643
Lemmatization + Stop-Words	Ours	0.867	0.833	0.892	0.861
	Sklearn	0.635	0.593	0.686	0.636

Table 2.3: Comparison of performance on feature sets for our own Gaussian Naïve Bayes model and the pre-built sklearn GNB model.

Once again the feature sets with stop-word removal outperform the set without and stemming and lemmatization with stop-word removal perform very similarly. As opposed to our MNB results our model decisively outperforms the sklearn implementation, achieving much higher scores for every metric across all sets. For evaluation on our test set we will once again use stemming with stop-word removal, as the performance of sklearn's model is higher in this set than with lemmatization. Table 3.2 (*see section 3.2*) shows the performance of both models on the test set.

2.8 Logistic Regression

With Logistic Regression we train the weight vector w of a model such that when multiplied by a feature vector as input and passed through a sigmoid function a class probability is output, Menard (2002). The sigmoid function is necessary to constrain the value between [0, 1] and thus output a probability. For Logistic Regression (and Support Vector Machines) we are required to encode our features as a one-hot vector meaning that our feature vector consists only of binary values. We choose to lose the information gained by TF-IDF scoring in order to utilise these one-hot vectors instead. Stochastic Gradient Descent (SGD) is an algorithm often used to update the weight vector w during training. A loss function is defined to measure the disparity between the results of the classifier and the true class labels. SGD iteratively minimises the loss function through a gradient descent technique which updates weights such that the loss function moves towards a local minimum Menard (2002). We use the sklearn *LogisticRegression* model in our evaluation and test the SGD algorithm in comparison to others such as the L-BFGS solver.

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Stemming	Sklearn	0.838	0.811	0.848	0.823
Stemming + Stop-Words	Sklearn	0.833	0.812	0.834	0.823
Lemmatization + Stop-Words	Sklearn	0.858	0.841	0.857	0.849

Table 2.4: Comparison of performance on feature sets for the sklearn Logistic Regression model.

The best feature set is evidently lemmatization with stop-word removal. Continuing to work with the development set we will now optimise the hyperparameters of the sklearn LogisticRegression model. Following hyperparameter optimisation we evaluate the final tuned model on the test set. Below are the hyperparameters we experimented with during the fine tuning of the model, all of which were tested using the development set and lemmatization with stop-word removal and compared to the baseline sklearn model with no hyperparameter tuning using SGD as the update algorithm.

1. **Solver**: Specifies the algorithm used for optimisation. Experimenting with different solvers reveals that *lbfgs* achieves a higher performance on the development set over *sag* (SGD), despite the potential advantages of a stochastic gradient descent algorithm. Due to this higher performance we choose to use the lbfgs solver instead of the SGD algorithm in our final tuned model.
2. **Regularization Parameter (C)**: Defines the strength of regularization as inversely proportional to the value given. Hence a lower value causes higher regularization and vice versa. During development numeruous values for C were evaluated and ultimately a value of 1.5 was found to be optimal.
3. **Penalty**: Determines the norm of the penalty. For the lbfgs solver either *l2* or *None* can be specified in the penalty parameter. An L2 penalty norm gives a higher accuracy than no norm whatsoever, thus we use an L2 norm in our model.
4. **N_jobs**: Controls the number of CPU cores used when parallelizing. Setting this parameter to -1 indicates an unlimited number of cores. Surprisingly the task was performed faster when n_jobs = None, i.e. when the task was not parallelized. For this reason we use n_jobs = None in our model, constraining the task to a single CPU core.

2.9 Support Vector Machines

Support Vector Machines (SVMs) operate by aiming to find the optimal hyperplane separating different classes whilst maximising the margin between said classes, Yu et al. (2019). The Support Vectors are those data points lying closest to the decision boundary that the hyperplane aims to divide. As with Logistic Regression, SVMs take one-hot vectors as input. SVMs are an efficient method of classification due to their generalisation capabilities and ability to deal with high dimensionality data. Once again we utilise the sklearn library in order to evaluate the performance of an SVM model for sentiment analysis. Training an SVM model can be potentially time consuming as SVMs do not scale well with large datasets. In addition complexity increases as the number of features increases, meaning that SVMs may not be the optimal model choice were we to test with any N-Gram selection higher than unigrams.

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Stemming	Sklearn	0.852	0.825	0.865	0.844
Stemming + Stop-Words	Sklearn	0.852	0.825	0.865	0.844
Lemmatization + Stop-Words	Sklearn	0.850	0.821	0.865	0.843

Table 2.5: Comparison of performance on feature sets for the sklearn Support Vector Machine model.

Interestingly our results show that the removal of stop-words has no effect on the results of the SVM classifier as we see identical results for stemming with and without stop-word removal. We will choose stemming with stop-word removal as the best feature set for use with SVMs due to its higher performance over stemming without stop-word removal in our other models. We now proceed to optimise the hyperparameters of the sklearn model and subsequently evaluate upon the test set (*see table Table 3.4*)

1. **Regularization Parameter**: Defines the strength of regularization as inversely proportional to the value given. Hence a lower value causes higher regularization and vice versa.

During development a value of 0.9 was found to outperform the default value of 1.0, hence we use this in our final model.

2. **Kernel**: Specifies the kernel to be used in the algorithm. Experiments showed that the *rbf* kernel achieved a much higher accuracy over the *linear* kernel, making it a suitable choice for our fine tuned model.

3. **Gamma**: Determines the kernel coefficient. Having tested the two options, *scale* and *auto*, it is clear that auto is the vastly superior choice, as scale achieves the lowest possible accuracy of around 0.50. The value of the coefficient using auto is defined as *1 / n features*.

2.10 BERT

Bidirectional Encoder Representations from Transformers (or BERT) is an NLP model that utilises a transformer architecture with deep learning techniques. The Bidirectional Representation element of the model refers to the fact that BERT uses a self-attention mechanism to consider both the left and right context of a masked word, where the model is attempting to predict the masked word as in a masked language model. This affords the model a deeper contextual understanding of a given text making BERT a potentially high performing solution for the task of sentiment analysis.

In our testing we evaluate both the cased and uncased versions of the pre-trained BERT model available from HuggingFace Wolf et al. (2019). Case here refers to the capitalisation of letters, as uncased BERT does not consider capitalisation whereas cased BERT does. Identifying proper nouns may therefore be less accurate in the uncased BERT model, potentially leading to lower performance in sentiment analysis. We test both versions of the model on the same test, train and development split as with all previous models. The BERT models have a much higher time complexity and require a much larger amount of computational resources than any other models thus far. This is due to the complexity of the model architecture itself as this is the first deep learning transformer and self-attention model to be evaluated. Due to the complexity requirements we have trained the BERT model using Google's Colab environment, Bisong et al. (2019), through which we can access a powerful GPU that can complete training in approximately 5 minutes. The performance of the BERT model could likely be increased given more training data, as a dataset of 4000 samples is insufficient for BERT to achieve its maximal performance. Table 2.10 shows the results of both the cased and uncased BERT models on the development set. Note that the usual feature sets do not apply in this case as BERT requires its own encoding and tokenization methods.

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
BERT Features	Uncased	0.846	0.831	0.839	0.835
BERT Features	Cased	0.844	0.803	0.879	0.839

Table 2.6: Comparison of performance on the development set by Google's cased and uncased BERT model.

Chapter 3

Discussion

3.1 Mutinomial Naïve Bayes Evaluation

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Stemming + Stop-word Removal	Ours	0.809	0.876	0.729	0.796
	Sklearn	0.820	0.809	0.849	0.829

Table 3.1: Comparison on the test set of performance of stemming with stop-word removal feature set for our own Multinomial Naïve Bayes model and the pre-built sklearn MNB model.

Both our own implementation and the sklearn model perform well on the test set with both achieving accuracies higher than 80%. Sklearn's model slightly outperforms our own across all metrics aside from precision. This may explain the disparity in performance, as it is possible that our model's higher precision leads to a lower recall and thus a lower accuracy. We aimed to achieve the same or higher performance in comparison to the sklearn implementation however the disparity in evaluation metrics is minimal to the point that our model remains viable. Potential improvements to precision and recall could be made by examining the values assigned for each class in cases of ambiguous samples, i.e. in cases where our model has a low certainty about the given class label.

3.2 Gaussian Naïve Bayes Evaluation

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Stemming and Stop-word Removal	Ours	0.828	0.811	0.866	0.837
	Sklearn	0.653	0.658	0.671	0.664

Table 3.2: Comparison on the test set of performance of stemming with stop-word removal feature set for our own Gaussian Naïve Bayes model and the pre-built sklearn GNB model.

For the GNB models our own implementation vastly outperforms the sklearn model on every available metric. Such a large disparity in accuracies can be explained through a number of differences between the two implementations of the method. Firstly, the sklearn GNB model has a much lower time complexity than our own model, with the former training in a matter of seconds as opposed to minutes for the latter. This could suggest an optimisation that sacrifices

accuracy for speed in the case of the sklearn model. In our implementation we use multiple slightly unconventional techniques to improve performance. The first is Laplace Smoothing, which is not regularly used in the context of a GNB model and is more commonly found in MNB implementations. We found that our model performed worse without Laplace Smoothing as it encountered 0 and infinity values during its calculations. Secondly we choose to sum the logarithms of our likelihoods and priors which is once again a technique seen more often in MNB models. We did this in order to avoid errors arising from calculations involving very small numbers. These differences in approach may help to explain the large performance gap between our own and the sklearn GNB model.

3.3 Logistic Regression Evaluation

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Lemmatization + Stop-words	Sklearn	0.836	0.833	0.851	0.842

Table 3.3: Performance on the test set of the sklearn Logistic Regression model with hyperparameter optiisation.

For our final logistic regression model we utilise one hot vectors as features over TF-IDF scores as we found an increase in performance when using the former. During development we found that the sklearn lbfgs solver outperformed the SGD based sag solver for our purposes of sentiment analysis. We used the lbfgs solver in conjunction with other hyperparameter optimisations to fine tune the model and achieve the results as displayed in Table 3.3. Once again our results for this model are very close to the average performance of models thus far. There is room for further hyperparameter tuning and more extensive testing of the sklearn SVM model, however these experiments remain outside the scope of this project. The logistic regression model achieves the highest f1 score of all models which suggests a robust performance when dealing with unseen data.

3.4 Support Vector Machines Evaluation

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
Lemmatization + Stop-words	Sklearn	0.826	0.809	0.866	0.836

Table 3.4: Performance on the test set of the sklearn Support Vector Machine model with hyperparameter optimisation.

The sklearn SVM model achieves a similar accuracy to the highest performing MNB and GNB models tested thus far. Similar results across multiple models is expected when using the same features and split of the dataset and indicates to us that our own implementations are functioning properly. The SVM model achieves a high recall, from which we can infer that the model is effectively discerning the true positive instances of the dataset, suggesting that the hyperplane has been properly adjusted during training. It appears that an SVM approach is well suited to this task, as the sklearn SVM model achieves the highest overall performance when compared with the other models (aside from BERT). Once again further work could be conducted in tuning the hyperparameters given more time and resources. Given that all of our models achieve similar accuracies it would be of interest to examine the differences between models given a much larger dataset and more computational resources.

3.5 BERT Evaluation

Feature Set	Model	Accuracy	Precision	Recall	F1 Score
BERT Feature Set	Uncased	0.891	0.889	0.900	0.895
BERT Feature Set	Cased	0.883	0.866	0.912	0.888

Table 3.5: Performance on the test set of the Google's cased and uncased BERT models

Due to the similar performance from both BERT models on the development set we evaluated both on the test set in order to determine the best model. Surprisingly the uncased model outperforms the cased model on the test set, which could be due to the fact that the data has less variation in the uncased encodings. By taking into account the case of words, the cased model may encode more unique features, leading to higher dimensionality and thus lower performance in the case of a small dataset. The disparity in performance between the two models remains minimal however, and both BERT models appear to outperform all other tested models. This is expected due to their complex architectures and is reflected in time complexity and computational resources required.

Chapter 4

Conclusion

4.1 Conclusion

We have experimented with different approaches to feature analysis and modelling in the context of sentiment analysis using a dataset of 4000 IMDB movie reviews. Our results found that stop-word removal produces a marked increase in performance across all models tested. We found that any N-Gram size higher than unigrams was unviable in our circumstances due to time complexity and computational demand. We used TF-IDF scoring to normalise our features which appeared to positively affect performance.

Of the approaches tested we found that all models performed very similarly, with all bar one achieving accuracies of around 80%. We determine that the best performing model (not including BERT) is sklearn's SVM model and the worst is sklearn's GNB model. Our own implementation of MNB achieves a nearly equal performance to the sklearn implementation. For the GNB models our own implementation vastly outperforms the sklearn model, scoring significantly higher on every evaluation metric. We found that hyperparameter optimisation led to increased performance with our SVM and Logistic Regression models and would be interested in further optimisation given more time. Through our experimentation we found that our dataset is insufficiently large for optimised performance with BERT, however we still achieve high accuracies using the model. We note that the time complexity of the BERT model is significantly higher than any other.

4.2 Future Work

Given the size of our dataset it would be interesting to investigate the differences between sentiment analysis models on a very large dataset, for example the original IMDB dataset of size 50,00 Maas et al. (2011). For our logistic regression and SVM models the effects of hyperparameter tuning on performance could be examined further, with more extensive testing and more combinations experimented with. Finally further research could be conducted into feature extraction and normalisation techniques, as we experimented with only one tokenizer and chose not to implement PPMI for feature normalisation. Examining the effects of these approaches on model performance could form the basis of interesting future work in the field of sentiment analysis.

Bibliography

Bisong, E. et al., 2019. *Building machine learning and deep learning models on google cloud platform*. Springer.

Blei, D.M., Ng, A.Y. and Jordan, M.I., 2003. Latent dirichlet allocation. *Journal of machine learning research*, 3(Jan), pp.993–1022.

Derrida, J., 2016. *Of grammatology*. Jhu Press.

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

Menard, S., 2002. *Applied logistic regression analysis*, 106. Sage.

Merriam-Webster, I., 1990. *Webster's ninth new collegiate dictionary*, vol. 10. Merriam-Webster.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al., 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct), pp.2825–2830.

Taboada, M., 2016. Sentiment analysis: An overview from linguistics. *Annual review of linguistics*, 2, pp.325–347.

Upton, G. and Cook, I., 2014. *A dictionary of statistics 3e*. Oxford quick reference.

Van Atteveldt, W., Velden, M.A. Van der and Boukes, M., 2021. The validity of sentiment analysis: Comparing manual annotation, crowd-coding, dictionary approaches, and machine learning algorithms. *Communication methods and measures*, 15(2), pp.121–140.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M. et al., 2019. Huggingface's transformers: State-of-the-art natural language processing. *arxiv preprint arxiv:1910.03771*.

Yu, Y., Si, X., Hu, C. and Zhang, J., 2019. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7), pp.1235–1270.