DM2583 Big Data in Media Technology

Project Report

Predicting Sentiment on Social Media Data Using Supervised Approaches

Fadhil Mochammad, Muhammad Irfan Handarbeni, Mehrdad Darraji
KTH Royal Institute of Technology

Abstract

In the last couple of decades, the use of social media websites has been increased tremendously. Millions of people use it to express their views and opinions on a wide array of topics. As a result, social media websites generate large volumes of data. In this paper, we will discuss several sentiment analysis algorithms and compare the performances of those algorithms using Twitter data. We have implemented the sentiment analysis algorithm using Multinomial Naive Bayes, Support Vector Classification, and Convolutional Neural Networks. Data preprocessing is a crucial step in sentiment analysis, since selecting the right preprocessing methods can increase the performance of our sentiment analysis. Based on this view, this paper will also discuss various preprocessing methods that usually use on sentiment analysis and compare the role of those different preprocessing methods to the performance of our sentiment analysis. The data preprocessing and sentiment analysis algorithm are done using NLTK, Scikit-learn, and Keras libraries. Experimental results obtained demonstrate that Linear Support Vector Machine classifier gives very high predictive accuracy and outperform others. Result also prove that the right feature selection and representation on data preprocessing can affect the performance of our sentiment analysis.

Keywords: Sentiment Analysis, Natural Language Processing, Neural Network, Support Vector Machine, Naive Bayes, Convolutional Neural Network

1 Introduction

Social media has been inseparable from modern human daily life. There has been a vast increase in social media users for the past several years. People use social media as a communication tool to communicate with others and share their experiences, views, and opinions on various topics. Twitter, a micro-blogging social media service is one of the most used social media nowadays. People share their thoughts and emotions through this platform within 140-characters length. Thus, the humongous amount of text data is widely available to use for various research.

Sentiment analysis is one of the most popular research topics on social media data. In sentiment analysis, people try to utilize social media data to get sentiment from the text. There are various applications of sentiment analysis, namely hate-speech detection, sentiment about a political figure, sentiment about particular issues and so on. Other earlier research suggested that social media data is an excellent data source for sentiment analysis task. However, since the nature of Twitter allows for free text, it can contain slangs, abbreviation, contraction, or even non-sense text. Hence, it is added as a challenge to preprocess the data before going forth to the sentiment analysis. Various approaches and algorithms have been used for this problem. Some of the widely used algorithms are Naive-Bayes, Support Vector Machine, and Deep Learning.

The main background of this research is to find out which algorithm produces the best result in sentiment analysis. We hypothesized that Deep Learning algorithm would give the best performance, compared to more traditional machine learning algorithms such as Naive Bayes and Support Vector Machine. Deep Learning algorithm learns by creating a more abstract representation of data as the network grows deeper and deeper. As a result, the model will automatically extract higher-level features. In this paper, we use the Convolutional Neural Network (CNN) architecture for our Deep Learning algorithm. Studies on CNN show that it has successfully established state-of-the-art results on various NLP sentence classification tasks also including sentiment analysis [7]. One thing to note, CNN will perform better when trained on a large amount of data.

Data preprocessing is a crucial step in sentiment analysis [2]. We believe that selecting the right preprocessing methods will generate a good data representation that can increase the performance of our sentiment analysis. Based on this background, we further want to investigate the correlation between data and the performance of our algorithms. We will compare the effects of various data preprocessing on the results of our sentiment analysis algorithm.

In order to test our hypotheses and beliefs, we implement end-to-end sentiment prediction process from data collection, data preprocessing, exploratory data analysis, feature extraction, and building a supervised model for the prediction. We use 1.6 million Twitter data and try to predict sentiment on each tweet. We use various data-preprocessing methods to clean the data and come up with several descriptive analysis of our dataset. For the feature extraction, we use three different approaches which are word count, TF-IDF, and tokenization. As for the modelling, we use three different supervised algorithms,

Naive Bayes, Support Vector Machine, and Convolutional Neural Network. Finally, we will compare the performances of our algorithms by measuring the accuracy score along with precision, recall, and area under curve.

2 Methodology

2.1 Data Collection

Crawling the data from Twitter using Twitter API will be taking too much time because Twitter limits the number of tweets on each API call. Also, crawling data using web-scrapping approach will also be taking a while since we have to learn the HTML structure first. Hence, due to the limited time constraint for this project assignment, we decided to utilize ready-to-use social media dataset available on the internet.

2.1.1 Sentiment140

Sentiment140 is a product created by Stanford University graduates. Their work allows one to discover the sentiment of brands, products, or topics on Twitter. They recorded about 1.6 million tweets in the format of polarity, id of the tweet, date of the tweet, the query, user of the tweet, and the text of the tweet. Specifically for this project, we decided to split the data into 600 thousand training data and 1 million test data[11].

2.1.2 Amazon Reviews for Sentiment Analysis

4 million Amazon reviews split into 3.6 million for training and 400 thousand for testing purposes formatted by the polarity and the review of products from the website. Although for this project, we only used the 1.5 millions of the actual training data and split it into 500 thousand training data and 1 million test data[3].

2.2 Preprocessing

Processing data from Twitter or other social media cannot be done directly because users can write anything as they want and use their own language. Users also prone to make mistakes in their writing (misspelling) and use slang words. Furthermore, for tweets especially, they sometimes contain a link/URL to another website or a mention ('@') to other users which we do not need for the sentiment analysis. These can be counted as errors and have the possibility to be a threat to sentiment analysis. Hence, we need to eliminate these errors to produce good quality data. A preprocessing part will play an important role on doing this[2]. There are various preprocessing methods that are used on solving Natural Language Processing problems, especially text classification. In this research, we use several preprocessing methods and investigate the effects on our sentiment analysis results.

2.2.1 Removal of Noisy Data and Word Contraction Mapping

Reviews or tweets can contain special characters, *mentions*, URL's, or HTML tag which have no value for sentiment analysis. Additionally, users often use contractions (*I'm, we're*) on their writing which will result in different feature representation on our model later. Hence, we need to remove those and map the contractions into formal writing. We create our custom function for noisy data removal and contraction mapping.
Input: "I don't realllly like the movie! :("
After removing the noisy data and word contraction mapping: "i do not realllly like the movie"

2.2.2 Spell Correction

As mentioned before, tweets and reviews are prone to have misspelled words. A misspelled word will end up in a different feature representation with the intended word later, and it may affect the performance of the algorithm. We implement Norvig's spell correction algorithm because it is one of the fastest and can process up to 10 words per second[1]. Instead of checking a whole corpus of all words in the dictionary, which will take much time to process since we have large number of words, the algorithm creates the 'most frequent misspelled word corpus' to reduce the computation time. This algorithm will calculate the probability of the correct words and use the Naive Bayes approach to replace the misspelled word with the correct word which has the highest probability.
After spell correction: "i do not really like the movie"

2.2.3 Removal of Stop-words

Stop word is a commonly used word that we can be ignored to reduce the number of features in our classifier. We implement 2 different stop-words removal techniques. The first one is the most common techniques where we use the stop-words corpus from NLTK which contains 127 stop-words. In the second technique, we use our custom stop-words corpus by picking the top 20 words that most frequently used

in the Twitter dataset.
After remove stop-words: "i like movie"

2.2.4 Lemmatization and Word-Stemming

The main purpose of lemmatization and word-stemming is to reduce the inflectional forms of a word into a common base or root of the word. However, these two methods are different. Word-stemming algorithms work by cutting off the common prefixes and suffixes. On the other hand, Lemmatization takes into consideration the morphological form of the words[5]. Lemmatization and word-stemming will ensure several words with different forms (plural, verb 2, etc) but have the same root word will have the same feature representation later.
Word-stemming: Studies → Studi
Lemmatization: Studies → Study

2.3 Exploratory Data Analysis

Before going deeper into the feature extraction and modelling phase, we decided to do an exploratory data analysis on the the Twitter dataset. The purpose of this exploratory data analysis is to understand and get a good grasp of the Twitter dataset. After pre-processing the data, we found that there are around 36,373 tweets that only contains links or mention users. Hence, we ditched those data and ended up with 1,563,627 tweets. Both negative and positive tweets are around the same number, so the dataset is balanced.

We explored the word frequency in each class, and the most frequent words in each class could be seen in figure 1. The bigger the word inside the plot, the more frequent it appears in that class. As expected, words like "bad," "miss," and "sad" frequently appear on negative class. On the other hand, words like "thank," "well," and "awesome" frequently appear in positive class. Interestingly, the word "love" appears frequently in both classes. That seems weird, but if we look deeper, we see that the actual negative tweets with the word "love" also contains the word "miss," or the word was just used sarcastically.

(a) Negative class (b) Positive class

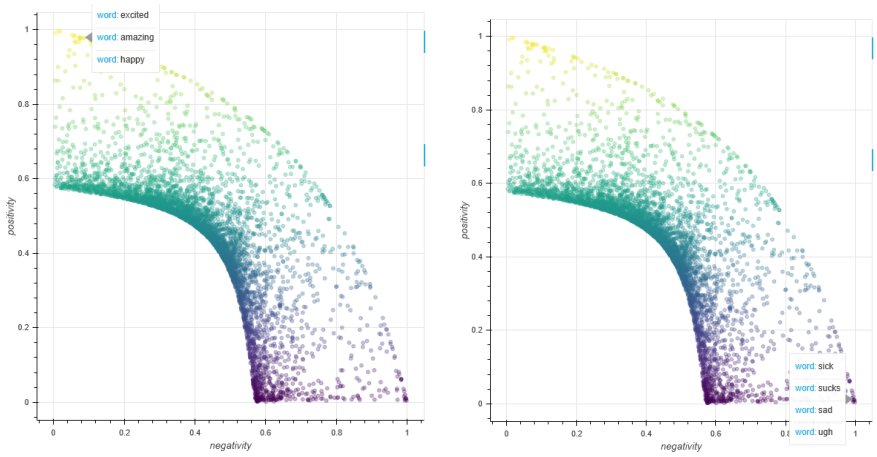
Figure 1: Frequent words on each class

Lastly, we try to evaluate negativity and positivity of 5000 most frequent words in the Twitter dataset. To do this, we first calculated the positive and negative rate along with the positive and negative percentage of each word. The rate is calculated by dividing its appearance in a particular class with the total appearance. The percentage is calculated by dividing its appearance in a particular class with the total of all words appearing in that particular class. Then we applied cumulative distribution function on both the rate and the percentage. At last, to get the polarity of each word we calculated the harmonic mean between those values from cumulative distribution function. The positivity and negativity of each word can be seen in figure 2. From the figure, we see that words like "excited," "amazing," and "happy" have a great value of positivity and almost zero negativity. On the other hand, words like "sick," "sucks," and "suck" have a great value of negativity and zero positivity. From this value, we could find which words are highly positive, highly negative or even neutral.

2.4 Feature Extraction

2.4.1 N-Gram

To generate the features for the dataset, we decided to try three different word representation. The first is the standard approach of the individual word as the features, and the other two use n-gram representation. N-gram is a method of text data representation which uses n consecutive sequence from the given text data. Using this representation as the features would allow capturing a little context on the text.



(a) Words with high positivity and low negativity

(b) Words with high negativity and low positivity

Figure 2: Positivity and Negativity of each word

2.4.2 Count Vectorizer

The Scikit-learn library's Count Vectorizer function converts a collection of text documents to a matrix of token counts. The count matrix has rows that represent the occurrence counts of different words while taking into account the high-dimensional sparsity.

2.4.3 Tf-idf

The TfidfVectorizer function of the Scikit-learn library transformed our tweets into feature vectors that will be used to as input estimators. It has a dictionary that converts each word to feature index in the matrix, giving them a unique index. The TfidfVectorizer is different from CountVectorizer in that it increases the value proportionally to the count and is offset by the frequency of the word in the corpus. This difference helps adjust the words that appear more frequently. The inverse document frequency helps even out the value to less valuable words such as stopwords.

2.4.4 Tokenization

Tokenization is the process of dividing a sequence of string into individual words, keywords, or phrases called tokens. We used this feature extraction method specifically for the convolutional neural network model. We used word tokenization where all tweets are tokenized into sequences of words, and then for each distinct word, it was given a unique number. For each tweet, the final representation of the data will then be represented as the sequence of the number of its words. To illustrate this, say we have two tweets: "They hate the movie" and "They love the performance". Therefore the tokenization results are as in table 1.

They	love	hate	the	movie	performance
1	2	3	4	5	6

Table 1: Example of tokenization result

Hence, the first tweet representation would be [1, 3, 4, 5] and the second tweet representation would be [1,2,4,6].

2.5 Classifier

After the data being pre-processed and extracted, we use it to train our classifier model and test the performance of our sentiment analysis. As mentioned earlier, in this research we will focus on implementing three different classifier models, which are Naive Bayes Classifier, Support Vector Machine Classifier, and Convolutional Neural Network Classifier. Each classifier has its own parameters which we need to determine and the value of the parameters will affect the performance of the classifier itself.

2.5.1 Naive Bayes

Naive Bayes is a probabilistic classifier that uses the Maximum A Posteriori decision rule in a Bayesian setting to make classifications. It uses Bayes rule: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ to predict the likelihood of

our binary sentiment classification. This classifier assumes that features in a class are independent and unrelated. They are known to be fast at predicting a test dataset, easy to build, and useful for large datasets. There are several Naive Bayes models that can use to perform on text classification, such as Multinomial, Binarized, and Bernoulli. We choose Multinomial Naive Bayes because it outperforms the other [9]. For the implementation, we use the Multinomial Naive Bayes function from Scikit-learn library.

2.5.2 Support Vector Machine

Support Vector Machine(SVM) is a supervised learning method for classification problems which creates a line that separates data into classes. It will find the best line to separate the data by finding the points closest to the line from both of the classes. Then, it computes the distance between the line and the points in order to maximize the margin. The line with the max-margin is the most ideal.

On implementing SVM Classifier, we need to determine which kernel to use on our SVM model, such as linear, RBF, polynomial, etc. A kernel is a similarity function that will calculate the similarity between two data points in a feature space. The kernel that we choose will affect the model's performance. Ideally, the best kernel depends on the data distribution in a feature space of extracted data. Since we do not know the data distribution of our data, in general, the first reasonable choice is the RBF kernel. The reason is that it can handle the case when the relation between class labels and features is nonlinear.[4] However, most of the text categorization are linearly separable.[6] Hence, we implement the SVM with both kernels, linear kernel and RBF kernel.

An important thing to note, choosing the right values kernel's parameters is essential since it will affect the classification result. We implement Grid Search with Cross Validation to find the best kernel parameters for both to be able to generate the best result without overfitting the model to the training data.[4] We plot the score from the grid search. As we can see from the figure 3, the best kernel parameters are `gamma = 3.72759` and `C = 0.001`. We use the Scikit-learn library to create the Support Vector Classifier(SVC).

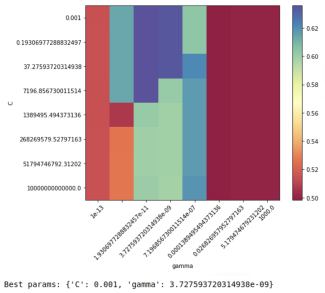


Figure 3: Grid search score plot for Tuning SVM Kernel Parameters

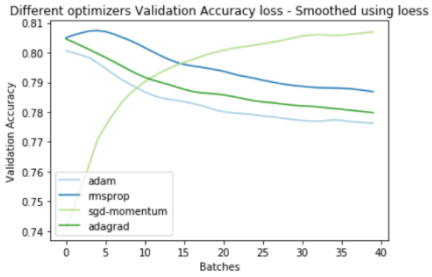


Figure 4: Loss Rate Curve on training CNN Classiefier

2.5.3 Convolutional Neural Network

The Convolutional Neural Network(CNN) is one of the most popular deep learning architecture. The Deep Learning approach is widely known for its superior performance on the machine learning problem. CNN is widely used for computer vision such as object detection and image recognition. However, CNN is also applicable on text classification problems. Studies show that it performs well in solving Natural Language Processing problems [10].

For this project, we implemented the CNN using Keras machine learning library with Tensorflow backend. The feature extraction process for this model is different from the other two models. We use tokenization so that each tweet is represented as a sequence of numbers. We limited the maximum length of the sequence to 20 because the average number of word on the Twitter dataset is just around 12 words. Hence, tweets with more than 20 words in it will be cut into its first 20 words. As for the tweets that are less than 20 words, we apply zero padding to it so that every tweet representation will have the same length.

Our model consists of 4 hidden layers. During the training process of our models, we use *categorical cross entropy* for the Loss function and 4 different optimization algorithm, which are *Adam*, *Stochastic Gradient Descent - Momentum*, *Adagrad*, and *RMSProp*. An optimization algorithm is an algorithm that we use to search for *learnable parameters* that will minimize our Loss function ($J(\theta)$) result. For example, we have weights (`w`) and bias (`b`) in each node of our neural network that are the *learning parameters* of our model. During the training process, the values of `w` and `b` will be updated in every epoch. The optimization function will be responsible to calculate these values in the direction to minimize the loss.

Hence, optimization function plays a major role in the training process of the Neural Network model in order to generate a model that can perform best.

To prevent the model from overfitting problem; we use Lasso Regression and Dropout. The Lasso Regression will shrink the less important features' coefficient to zero while Dropout will randomly ignore some nodes on training. We tune the parameters manually for both regularization to reduce the overfitting without reducing much of the model performance.

In the first experiment, we run the training using the setup mentioned above. As a result, we have 4 CNN models that have been trained using different opitimization function. The validation accuracy during the training process is plotted in figure 4. Based on this result, we can see that Adam, RMSProp, and Adagrad converge faster. However, if we compare to SGD-Momentum, it converges slower while keeping a positive trend for the validation accuracy. On the second experiment, we do the training process again and we determine the number of training epochs based on the best validation accuracy achieved on the first experiment. The experiment result shows that the best models are the models that have trained using 1) RMSProp for 5 number of epochs and 2) SGD-Momentum for 70 number of epochs. We choose the RMSProp model for our final CNN since it is the most efficient one.

2.6 Evaluation

There are several ways to measure the performance of our machine learning model. On this project, there 4 performance metrics that we use, which are Classification Accuracy, Confusion Matrix, Area Under Curve, and Precision Recall.

2.6.1 Classification Accuracy

We use classification accuracy to measure the number of correct predictions made by the model over all of the predictions made. Measuring accuracy is a good way to evaluate the performance if the data are nearly balance.

2.6.2 Confusion Matrix

Confusion Matrix is one of the most intuitive metrics used for finding correctness and accuracy of the model. On the confusion matrix result, we can see the number of true positive, true negative, false positive, and false negative.

2.6.3 Area Under Curve

The Area Under Curve(AUC) metric of evaluation is one of the most used binary classification problems. It is the probability that our classifiers will choose a randomly chosen positive tweet higher than a randomly chosen negative tweet. It is the area under the curve of a plot of False Positive Rate vs True Positive Rate. False Positive Rate is the proportion of negative data that are wrongly considered positive in respect to all negative data - calculated by False Positive divided by the sum of True Positive and False Positive. True Positive Rate is the proportion of positive data that are wrongly considered positive in respect to all positive data - calculated by True Positive divided by the sum of True Positive and False Positive.

2.6.4 Precision Recall

As mentioned before that confusion matrix will give us true positive, true negative, false positive, and false negative value - through these numbers we can compute a various amount of rates, such as precision and recall, for binary classifiers. Precision is the number of tweets retrieved that are relevant, and is calculated by the number of true positives over the sum of true positives and false positives. Recall is the number of relevant tweets that are retrieved, and is calculated by the number of true positives over the sum of true positives and false negatives.

3 Result

Before going to the actual model building, we tried to investigate the effects of various data preprocessing and feature extraction techniques on the performance of the model. For this purpose, we use a Naive Bayes model which is implemented using the Multinomial Naive Bayes package in Scikit-Learn library. After getting the optimal preprocessing method, we proceed to build the actual model with the three different algorithms.

3.1 Pre-processing & Feature Extraction

3.1.1 Removal of stopwords Effect

From figure 5 it can be seen that removing the stopwords, both using NLTK English stopwords and our custom stopwords, negatively affects the model performance. Thus we ended up choosing to keep the stopwords in the tweets. The decrease in the performance is due to removing the stopwords because

we are at a risk of changing the context of the tweet. For instance, the word "not" would be erased and changes the context of the tweet from negative to positive.

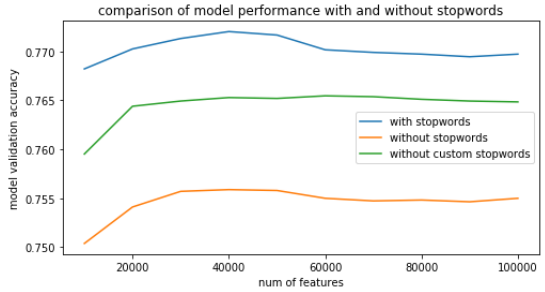


Figure 5: Effect of removing stopwords in the model performance

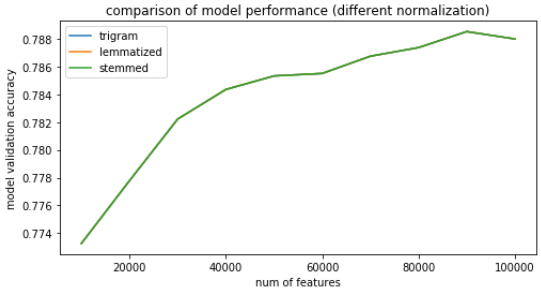


Figure 6: Model performance on lemmatization and word stemming

3.1.2 Lemmatization and Word Stemming

We try to explore the effects of word lemmatization and stemming to the performance of the model. As we see in figure 6, performing lemmatization and word-stemming does not have any effects on the performance. Hence we proceed without lemmatizing and stemming the words.

3.1.3 Spell Correction

We investigate that by adding spell correction on our preprocessing, it was able to improve the accuracy by 2% on SVM and Naive-Bayes classifier. We implemented it using 30,000 training data and 5,000 test data. However, implementing spell correction has a huge trade-off with the computation time. Regarding this trade-off, we did not implement the spell correction for our final classifiers.

3.1.4 N-Gram Effect

We three different word representations in this experiment, which are uni-gram, bi-gram, and tri-gram. The result of the experiment in figure 7 shows that using sequences of words in bi-gram and tri-gram give an improvement on the model performance compared to using the individual word in unigram. That is because by using sequences of words, we can capture a little context of the tweet. It is also better to capture the similarity between the tweets.

3.1.5 Word count & Tf-Idf Vectorizer

For the feature extraction, we try two different approaches - word count and tf-idf. We run an experiment to compare the model performance using features from those two approaches. From the result in figure 8, we find that tf-idf performs better than word count. Hence, we can conclude that tf-idf gives better feature representation than word count.

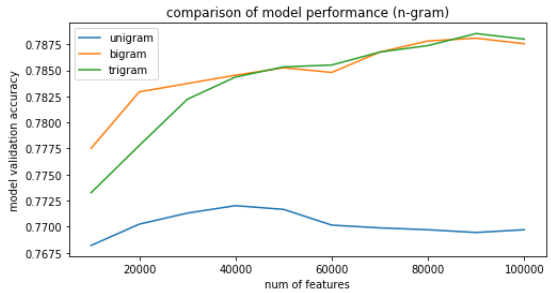


Figure 7: Effect of using n-gram in the model performance

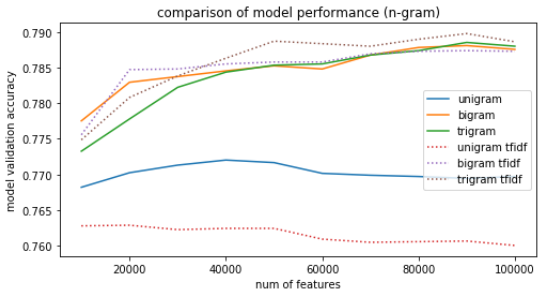


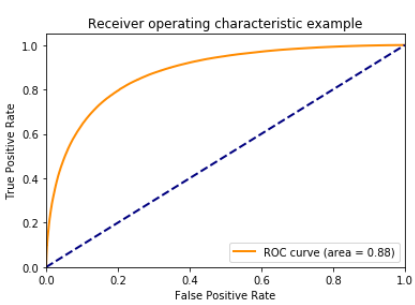
Figure 8: Model performance on different feature extraction approach

3.2 Naive Bayes Classifier

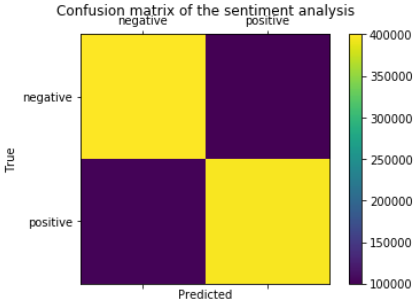
We computed the Area Under the Receiver Operating Characteristics Curve and the Confusion Matrix for the Multinomial Naive Bayes classifier. The AUC-ROC curve gave an area of 0.88. The Confusion Matrix shows that there is about four hundred thousand true negative data, about a hundred thousand false negative data, a hundred thousand true positive data, and about four hundred thousand false positive data. The overall accuracy of this classifier was 79.73% which is higher than the result on Korovkinas et al[8]. They have accuracy score of 75.77% for Naive Bayes Classifier with the same dataset.

Accuracy	79.73%	
Class	**Precision**	**Recall**
Negative	0.7958	0.8001
Positive	0.7990	0.7947

Table 2: Naive Bayes Classification Result



(a) AUC-ROC Curve



(b) Confusion Matrix

Figure 9: Naive-Bayes Classification Result

3.3 Support Vector Machine

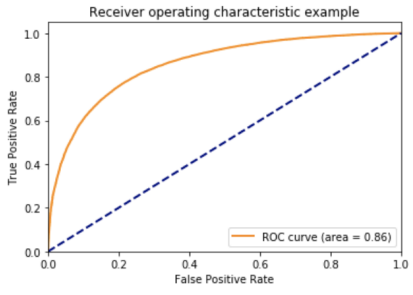
We make two different SVM classifier model on this project, the SVM classifier model using a linear kernel and SVM classifier model using the RBF kernel. The results show that the SVM classifier with a linear kernel generates a better classification than the one with the RBF kernel; linear kernel has a better score on both classification accuracy and AUC score. As shown in figure 11, both classifier models have greenish colour on the true positive container, which means that both have errors in predicting the positive sentiment. This result also supported by the recall score on table 3 and table 4. The RBF model has lower on all precision and recall score, but the difference in recall score for the positive class is relatively bigger than the difference of other scores. Our accuracy score for the SVM Classifier also higher compare to the result by Korovkinas et al.[8] where they get 78.08% for the accuracy. It is also true that Linear SVM performs better than RBF on text categorization[6].

Accuracy	82.00%	
Class	**Precision**	**Recall**
Negative	0.8072	0.8411
Positive	0.8341	0.7991

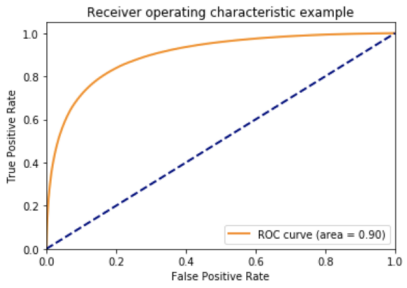
Table 3: SVM Classification using Linear Kernel Result

Accuracy	77.77%	
Class	**Precision**	**Recall**
Negative	0.7615	0.8120
Positive	0.7964	0.7430

Table 4: SVM Classification using RBF Kernel Result

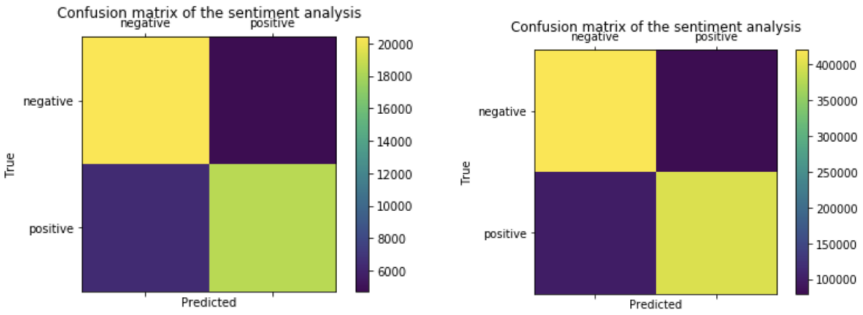


(a) SVM Classification with RBF kernel



(b) SVM Classification with linear kernel

Figure 10: AUC-ROC Curve from SVM Classification Results



(a) SVM Classification with RBF kernel (b) SVM Classification with linear kernel

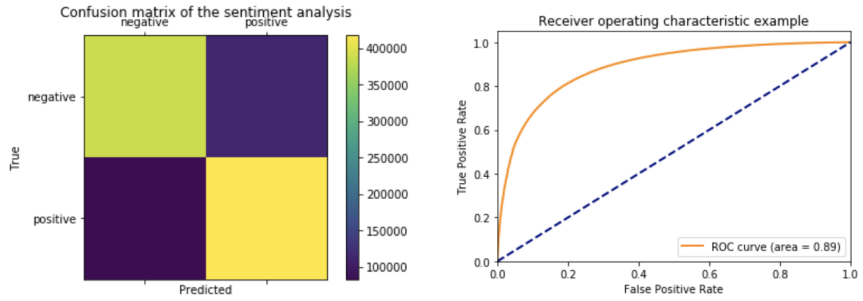
Figure 11: Confusion Matrix from SVM Classification Results

3.4 Convolutional Neural Network

For our CNN model, we implement four hidden convolution layers. 1D convolution layer was used because the data representation is in a 1-dimensional sequence. From figure 12 & table 5 we can see that the CNN model gives good classification result. Comparing the performance metrics, the CNN model performs slightly below than our SVM model, this is different from our initial assumption that the CNN model will perform better. However, our CNN result also shows that it outperforms others in Precision and Recall score. For this experiment, we cannot compare our implementation result due to a different dataset that is used on training and evaluating the model. They show that their implementation can achieve the accuracy score 70% - 88% using CNN algorithm.

Accuracy	81.26%	
Class	**Precision**	**Recall**
Negative	0.8047	0.8155
Positive	0.8130	0.8021

Table 5: CNN Classification Result



(a) Confusion Matrix (b) AUC-ROC Curve

Figure 12: CNN Classifier Performance Result

3.5 Model Performance on Different Dataset

In order to investigate the effects of different datasets on our models performance, we also use the Amazon review dataset to train and evaluate our algorithm. As we expected, due to the nature of Amazon dataset which have richer words and longer texts, our models succeeded to perform really well to predict the sentiment. The Naive Bayes classifier managed to achieve accuracy of 89.97%, SVM gave 95.71% and CNN gave 94.75%. Again, linear SVM classifier have the best performance, beating the CNN classifier. We assume that this is because the data is linearly separable thus linear model beat even the Neural Network model. We also tried to train the model on Twitter dataset, and test on Amazon dataset, vice versa. We found that using dataset with different domain on tranining and test, would make the model suffer on the test data. The test accuracy for these experiments is only around 60%. From these experiments, we found that the quality of the dataset effects the performance of the model. The Amazon

dataset has higher quality because it contains longer text and more words, also fewer slang. Hence, our models could perform better on that dataset. We also found that when we train and test the model with different datasets, it would not give a good result - this is mainly because the difference in the context between the two datasets.

4 Analysis and Discussion

In this project, our primary purpose is to learn and get our hands dirty on how to work with machine learning related problem and apply the machine learning algorithm to big data. We don't want to learn only how to design and implement efficient and optimal machine learning algorithm to a vast amount of data, but to also on how to pre-process the data. We choose to work on classification problem on sentiment analysis as it is a good starting point for us to work with big data since there are many researches on it for us to use as learning resources. We also want to investigate how various algorithms perform in sentiment analysis.

From the results of all the experiments we have done for our project, we found that the best algorithm for the sentiment analysis is the SVM with linear kernel. Thus, our prior hypothesis was rejected. That is most likely due to the data, as it is actually linearly separable. So that linear classifier produces the best performance. we also found that proper data preprocessing and feature extractions are needed to produce optimal and efficient machine learning model. The data collection, data preprocessing and feature extraction process are highly crucial for the model to give it excellent performance since the data is the most important thing on all machine learning-related problems. At first, we expected higher accuracy - around 90%. However, judging from the nature of the Twitter data which has the maximum of 140 characters, is a free text and may contain subjectivity, different tone, sarcasm, and irony, we can say that our models perform quite well in performing sentiment classification from the Twitter dataset with accuracy around 80%. We also happened to try our model on Amazon review dataset which contains richer words and longer text, and it performs significantly well in the sentiment classification, even reaching an accuracy of 93% on our SVM model. Therefore, we can say that the model would give different performance depending on the data that is applied to it. Those results verified our hypothesis about the data affects the algorithm performance.

As for designing and implementing the model, we find that after the preprocessing and feature extraction it is easier to tune the Naive Bayes and SVM model. Since they only need a small number of parameters to be tuned. On the other hand, tuning the Neural Network model is a non-trivial model; this is due to so many parameters that need to be tuned on the Neural Network model: namely learning rate, regularization rate, and the number of hidden layers. Moreover, for CNN, we also need to tune the filter size, padding size and others.

If we are given more time on the project, we want to explore several things for further research. We are specifically interested in the Neural Network model for sentiment analysis. Since the performance of our CNN is not better than the SVM, we want to modify it, in the hope of increasing the performance. We want to try to build tokenization of n-gram as the feature extraction instead of just individual word tokenization. Another interesting thing to explore is to implement a recurrent neural network with long-short-term memory as the model for sentiment classification.

The workload on this project is distributed evenly between all the group members. The reason is that so all of us can have a complete picture of this sentiment analysis project.

5 Conclusion

The goal of this project is to find the answers to our research questions, which are: Which algorithm produces the best sentiment analysis performance, and does the data affect the algorithm performance in sentiment analysis. From the experiments in this project, we found that proper data preprocessing is crucial in doing sentiment analysis, for it lets the algorithm to perform better. Also, the data quality also plays a role in the algorithm performance, since richer and better text data would give a better result. Our final models managed to achieve a quite good result both in Twitter and Amazon dataset. The models produced an accuracy of around 80% on the Twitter dataset and around 90% on Amazon dataset. Surprisingly, contrary to our prior hypothesis, the SVM model performed slightly better than the Deep Neural Network model on both datasets. Hence, from the experiments, we found the answers to our research questions that SVM is the best algorithm for this sentiment analysis and the data does indeed affect the algorithm performance.

Nonetheless, by doing this sentiment analysis project, we have learned a lot about real big data/machine learning project. We got our hand on end-to-end machine learning project from data collection, data preprocessing, feature extraction, model building and model evaluation.

References

[1] August, 2016. URL: http://norvig.com/spell-correct.html.

[2] Giulio Angiani et al. "A Comparison between Preprocessing Techniques for Sentiment Analysis in Twitter". In: *KDWeb* (2016).

[3] Adam Mathias Bittlingmayer. *Amazon Reviews for Sentiment Analysis*. May 2017. URL: https://www.kaggle.com/bittlingmayer/amazonreviews.

[4] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. "A practical guide to support vector classification". In: (2003).

[5] A.G. Jivani. "A Comparative Study of Stemming Algorithms". In: *IJCTA* (2011).

[6] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *European conference on machine learning*. Springer. 1998, pp. 137–142.

[7] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: *CoRR* abs/1404.2188 (2014). arXiv: 1404.2188. URL: http://arxiv.org/abs/1404.2188.

[8] Konstantinas Korovkinas, Paulius Danėnas, and Gintautas Garšva. "SVM and Naive Bayes Classification Ensemble Method for Sentiment Analysis". In: *Baltic Journal of Modern Computing* 5.4 (2017), pp. 398–409.

[9] Andrew McCallum, Kamal Nigam, et al. "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1. Citeseer. 1998, pp. 41–48.

[10] Andreea Salinca. "Convolutional Neural Networks for Sentiment Classification on Business Reviews". In: *arXiv* (2017).

[11] *Sentiment140 - a twitter sentiment analysis tool*. URL: http://www.sentiment140.com/.