COMP316

Project Documentation

Name and Student Number	
1. Shrivaad Gopal	218022956

Contents

Area of application

Introduction:

Sentiment analysis refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. We will explore deeper into the aspects of the classifiers that are used.

Problem:

Movie reviews are one of the most important ways to judge the performance of a movie. While providing a star rating to a movie tells us about the overall success or failure of that movie, a collection of reviews will be able to give us a deeper insight on different aspects of the movie. Depending on which classifier you use for the sentiment retrieval, the accuracy will vary.

Goal:

The goal of my project was to perform sentiment analysis on this dataset using 3 different classifiers to determine which has the highest accuracy. The 3 classifiers that I had used were, naïve Bayes (Multinomial Bayes classifier), Support Vector Machines and logistic regression.

Dataset:

For the dataset, I have chosen to use the IMBD 50k movie review dataset. This is because it has 50,000 reviews (the more data to train, the better), and out of these 50,000 reviews, 50% is positive and 50% is negative. This is good in a way that there is no imbalance problem and hence accuracy would be a very good metric for the performance as no sentiment would be favoured over the other.

Each sentiment is either marked as ‘negative’ or ‘positive’

(binary classification dataset)

Choice of NLP techniques

Sentiment Analysis

This is what the project was based on. Sentiment analysis can be performed in many different ways, as there exists many classifiers and algorithms. However, the classifiers I based my project on included Naïve Bayes (Specifically Multinomial Naïve Bayes), Logistic Regression, Support vector Machines, and Recurrent Neural Networks. By using these classifiers to obtain the sentiment of a review, I was able to measure the accuracy of them

Data cleaning with Regular Expressions

We know that REGEX can be used as FSA's/FST's for defining the rules of a language. Hence, I thought of using it in a way as a filter for cleaning raw data.

As you can see below it was used to remove non-alphabetic values and all other symbols: (very powerful, yet simple)

```
import re

def remove_non_alpha(text):
    text = re.sub('[^a-zA-Z]', ' ', text)
    text = re.sub('\[[^]]*\]', ' ', text)
    return text
```

Other important techniques

NLTK module

The above module was very important. With the help of stopwords, we could easily iterate through reviews and remove words that don't contain any value(stopwords).

Another important function was WordNetLemmatizer, it helped convert words back to their root words and make machine learning easier.

BOW via Vectorization

This is another very crucial technique needed. It helped convert the reviews into vectors using ‘CountVectorization’ (BOW approach), Hence, we could fit, train and test all of our classifiers.

Other

There was no one library that could provide me with everything needed for sentiment analysis, hence I used some of the libraries for these functions:

Matplotlib – assists with drawing graphs and is very useful for help visualizing data.

Sklearn – provides us with a very powerful function that can split data into test and training data randomly. It provides us with the classifiers needed for this project as well as reports on the results of the classifiers.

seaborn – Helps us with a lot of graphically representations, however the reason I use it, is for its heatmap representation of a confusion matrix

Design and modelling

The Design Process:

When we do sentiment analysis on any set of reviews, the first major thing to do is data cleaning. This is important as the machine needs to learn the data in the most simple and easiest form. When we do data cleaning, we basically making our own formal language that is accustomed to the scenario.

I implement many techniques to clean data and turn it into clean data (formal language)

The first step was making the reviews free of html tags, this was done easily and fast by the use of a library called bs4 (As seen below)

```
from bs4 import BeautifulSoup as bs

def remove_html_tags(text):
    soup = bs(text,'html.parser')
    cleaned = soup.get_text()
    return cleaned
```

The next part was of making our own formal language was to filter out any non-alphabetic values as well as any symbols. To do this, we used REGEX, as seen under the NLP techniques.

Another technique was the use of stopwords. StopWords are words that hold no important meaning as they are common words e.g. a, the etc. By the use of the nltk model that we learned during practical 4, I was successfully able to filter out all stopwords (as seen below)

```
import nltk
from nltk.corpus import stopwords

def remove_stopWords(text):
    sentence_without_stopWords = []

    # We basically break down the the entire review into words and then check if those individual words are stopwords or not
    text = text.split()
    for i in text:
        if i not in stopwords.words('english'):
            sentence_without_stopWords.append(i)
    return sentence_without_stopWords
```

After this most of the data was clean, however to prepare it before using vectorization on it, I had converted all the reviews to lower case. This would ensure that the vector counts won't interpret words like 'movie' and 'Movie' as different.

Finally, the last step was just to lemmatize the words. This was another thing that I learnt from practical 4 and I thought it would be perfect to implement here. This was because, the simpler the data, the better the machine would learn (Code implementation below):

```
def lemmatizing_words(text):
    text = [lemmatizer.lemmatize(word) for word  in text]
    text = join(text)
    return text
```

After all of the above modelling, we finally achieve 'clean data':

	review	sentiment	Cleaned_review
0	One of the other reviewers has mentioned that ...	positive	one reviewer mentioned watching oz episode hoo...
1	A wonderful little production. The...	positive	a wonderful little production the filming tech...
2	I thought this was a wonderful way to spend ti...	positive	i thought wonderful way spend time hot summer ...
3	Basically there's a family where a little boy ...	negative	basically family little boy jake think zombie ...
4	Petter Mattei's "Love in the Time of Money" is...	positive	petter mattei love time money visually stunnin...
5	Probably my all-time favorite movie, a story o...	positive	probably time favorite movie story selflessnes...
6	I sure would like to see a resurrection of a u...	positive	i sure would like see resurrection dated seahu...
7	This show was an amazing, fresh & innovative i...	negative	this show amazing fresh innovative idea first ...
8	Encouraged by the positive comments about this...	negative	encouraged positive comment film i looking for...
9	If you like original gut wrenching laughter yo...	positive	if like original gut wrenching laughter like m...

Review is the 'raw data' and Cleaned_review is the 'clean data'

Techniques for modelling the data

After splitting the data, we finally got out training and testing datasets along with their corresponding labels

(Please note the labels were converted from positive and negative to 0 and 1 using label encoder from sklearn)

Now that we had our reviews and their sentiments, the last thing needed was to convert the reviews into vectors. To do this I used CountVectorizer as it implements a Bag of words approach (BOW). Basically, this function, provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. This is needed when fitting the classifiers with training data.

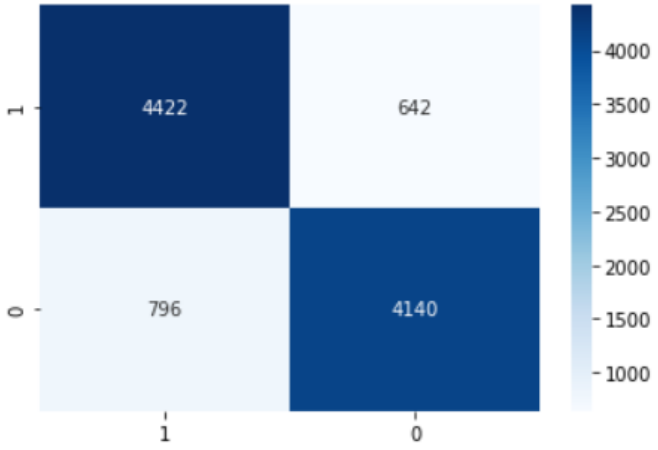
All that was left, was to fit the 3 classifiers with the vectors and their labels and the models were trained and ready to be tested.

Implementation

After I trained the models with data (dataset(vectors) + labels) and the model was trained, I simply used the 'model.predict' function on the test sets. Since the goal is to find the classifier that performs the best, the only thing required was the accuracy score, a report and a confusion matrix.

Hence, there is not much implementation required in this type of project. We only require to see the accuracy and report of each of the 3 classifiers and from the results determine the classifier with the best accuracy. However, I did all the report generation and confusion matrix designs by re-using sklearn and seaborn libraries for each of the 3 classifiers.

As you can see below, is an example for Naïve Bayes Classifier:



As you can see from above that a heatmap is returned in the form of a confusion matrix showing us the 'True Positives', 'False Positives', 'False negatives' and 'True Negatives'. The values shown in the blocks are the counts of how the model performed judging from the Predicted and Actual Labels. The darker the shade of blue, the more values within that respective category.

	precision	recall	f1-score	support
0	0.85	0.87	0.86	5064
1	0.87	0.84	0.85	4936
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

FINAL ACCURACY SCORE WITH MNB IS: 0.8562

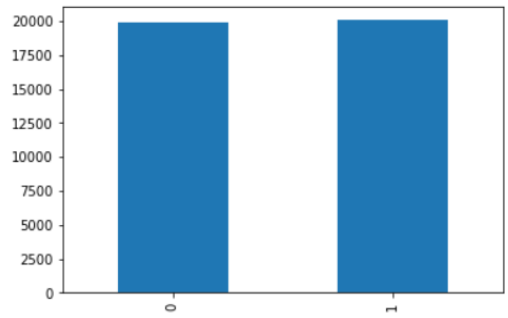
As you can see from above that, a report is also returned (using sklearn function) displaying to us the various performance metrics, as well as the overall score for that particular classifier.

Empirical evaluation of the NLP system

The dataset was split into 80% for training and 20% for testing. In terms of review counts this would be: 40000 and 10000 respectively. When the data was shuffled and split, this is what the training data looked like:

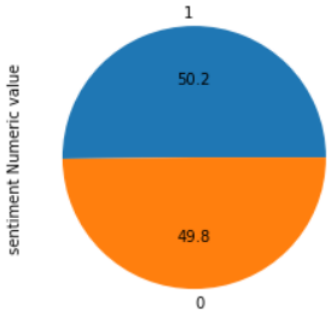
```
Training set information

1    20064
0    19936
Name: sentiment Numeric value, dtype: int64
```




```
In Pie chart Display:

<AxesSubplot:ylabel='sentiment Numeric value'>
```



As you can see even after splitting up the data and shuffling it, there were still about 50% of positive reviews and 50% of negative reviews. This was a really good split, as this means the data won't favour any sentiment when classifying a review as it was trained with almost equal amounts of positive and negative reviews. Hence, from this data I decided to base the performance of classifiers on 'Accuracy'.

Each of the classifier's was then fitted using the same training and testing data as this ensured fairness.

As you can see from below these were the classifiers used:

Naïve Bayes Classifier:

```
model = MultinomialNB().fit(cv_train,y_train)
```

Support Vector Classifier:

```
lin_svc = LinearSVC(C=0.5,random_state=42,max_iter=10000)
lin_svc.fit(cv_train1,y_train)
```

Logistic Regression Classifier:

```
lr = LogisticRegression(solver = 'liblinear', random_state = 42, max_iter=1000)
lr.fit(cv_train1,y_train)
```

Now to the results:

For each matrix,

The top left indicates the True Positive's(that is how many sentiments with class '1' was predicted correctly against actual data).

The top right indicates the False Positives(that is how many sentiments with class '1' was Incorrectly predicted as '0' against actual data)

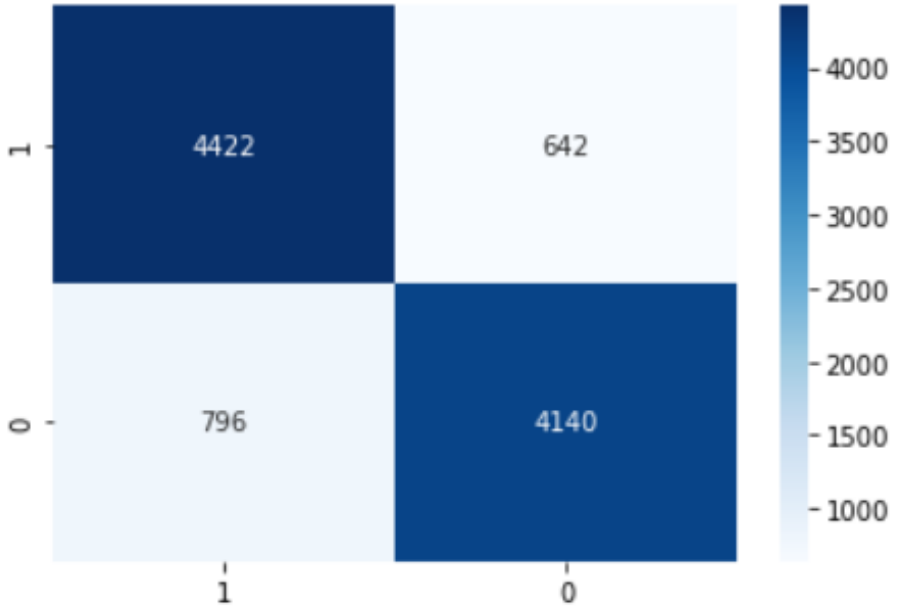
The bottom left indicates the False Negative's(that is how many sentiments with class '1' was incorrectly predicted as '1' against actual data).

The bottom right indicates the True Negatives(that is how many sentiments with class '0' was predicted correctly against the actual data)

The values shown in the blocks are the counts of how the model performed judging from the Predicted and Actual Labels. The darker the shade of blue, the more values are within that respective category.

Naïve Bayes:

Confusion Matrix:



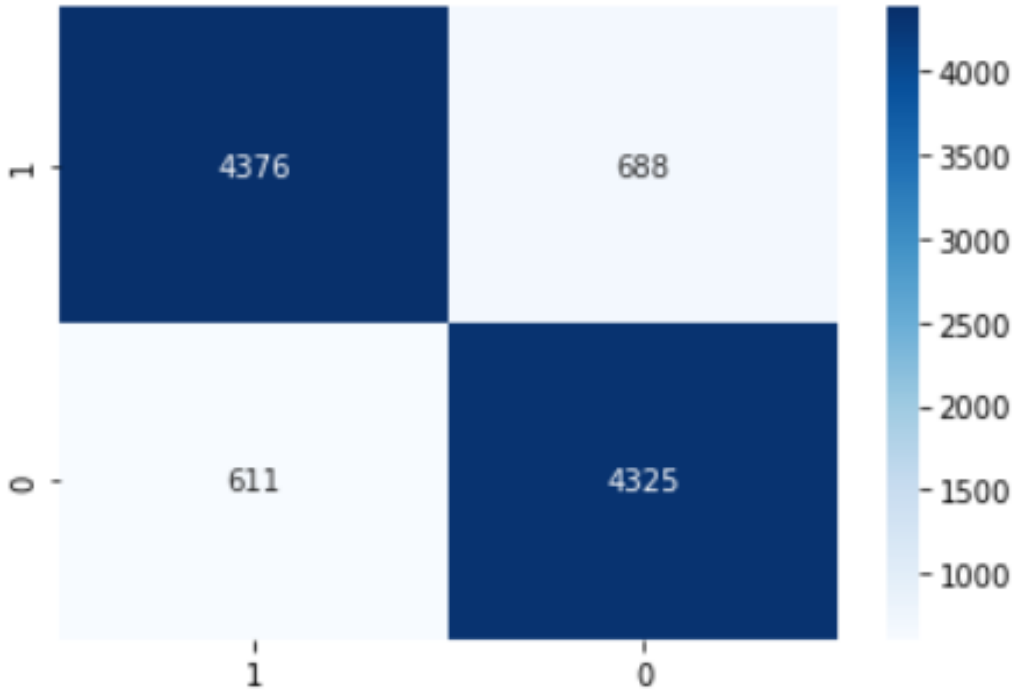
Report on the different performance metrics:

	precision	recall	f1-score	support
0	0.85	0.87	0.86	5064
1	0.87	0.84	0.85	4936
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

FINAL ACCURACY SCORE WITH MNB IS: 0.8562

Support Vector Classifier:

Confusion Matrix:



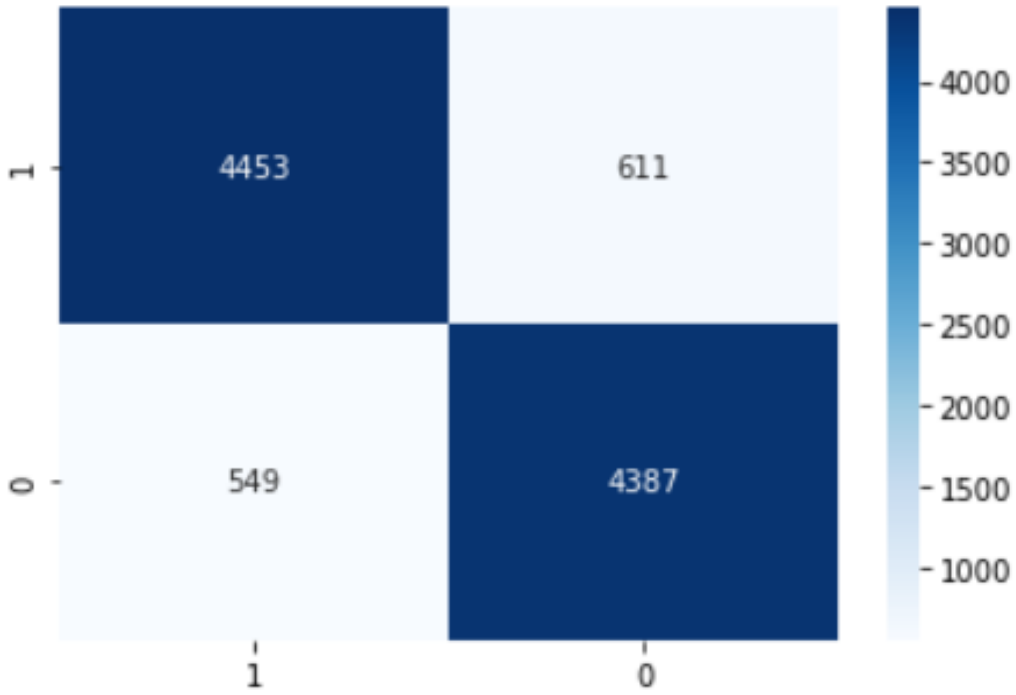
Report on the different performance metrics:

	precision	recall	f1-score	support
0	0.88	0.86	0.87	5064
1	0.86	0.88	0.87	4936
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

FINAL ACCURACY SCORE WITH LINEAR SVC IS: 0.8701

Logistic Regression Classifier:

Confusion Matrix:



Report on the different performance metrics:

	precision	recall	f1-score	support
0	0.89	0.88	0.88	5064
1	0.88	0.89	0.88	4936
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

FINAL ACCURACY SCORE WITH Logistic Regression IS: 0.884

As we can see from the results from the 3 above classifiers, the one that produced the best accuracy was the logistic Regression classifier, followed in second place by the Support Vector Classifier followed in third place by the Naïve Bayes Classifier.

Although the Logistic Regression classifier was first, we could see that the accuracy between all 3 classifiers differed from each other by +/- 1.

The results of the Naïve Bayes were expected to be the lowest as its is commonly used as a baseline accuracy when doing sentiment analysis.

By looking at each of the confusion matrices we can also see that the logistic Regression classifier had the highest number of True Positives and True Negatives, due to its high precision. Hence, was able to give us the best accuracy.

Shortcomings of the project

The only shortcomings experienced during the project was that of the dataset. I think if there were more datasets out there that not only contained 'positive' and 'negative' sentiments, it would really show a change of our results and on the accuracy of the classifiers.

I was going to use the twitter airline dataset, as it is one of the few datasets that contain positive, negative and neutral sentiments, however there was a big class imbalance, which would have caused the classifiers to become biased.

Overall, I think the current dataset used for this project was good.

Conclusion

After carrying out this project, I learned that from the way I designed and tested the 3 different classifiers, logistic regression was the best classifier in terms of accuracy with a score of 88% and Naïve Bayes had lowest accuracy with a score of 86.5%.

This test proved that Naïve Bayes can provide a good baseline when doing sentiment analysis and that the logistic regression classifier and the Support Vector Classifier are good to use for training as they yield higher accuracy scores.

To bring this project to an end, I found this project was a good test of my skill attained this year and I'm satisfied with the results. Even though I may have not conducted things in the incorrect way, I am still happy I was able to see the classifiers put to a good test and still give very good accuracy scores.

References:

'https://www.kaggle.com/prashant111/naive-bayes-classifier-in-python'