Comp316 Natural Language Processing

Final Project Report

Topic: Using NLP to predict the rating of book reviews

Group members: Alkha L.S. Sewsunker (220011309) and Rionna P. Govender(220044250)

Introduction

Reviews have the power to influence any decision we make, whether it is which restaurant to dine at, which hotel to stay in, what book to read next, or whether the latest video game is worth playing. With that being said there are times we are too lazy and are unwilling to read lengthy reviews left under the products we are considering purchasing. This is where ratings come in. Ratings based on reviews give us a more concise idea of the overall sentiment of a review.

Predicting the rating of a review can be done by humans, however, this requires a lot of time, effort and understanding as each person can have their own perception of certain things. With the use of certain NLP techniques, we can limit the number of misconceptions that occur when a human attempts to perform such a task and should be able to more accurately determine the rating of a review left by a reviewer compared to what a potential customer may understand when they read the review. This gives us a more accurate overview of how a product is received by customers that have used the product.

Our system can be used by bookstores to classify ratings based on reviews from customers whose purchased books from the store. With a few tweaks, the system could potentially be used to determine which books are more popular and which are not.

Approach

We started off by researching how to go about creating a text classification model as well as what classification techniques were used for such a task.

We narrowed down our approach to the following 4 steps:

1. Retrieving the dataset
2. Preprocessing
3. Training the dataset

    3.1. Word embedding techniques

    3.2. Machine learning techniques

4. Evaluation of models

We chose to use Python as the default language for this project as it is a widely used NLP programming language. We decided to use Google Colab as we experienced a few issues when trying to make use of PyCharm and Visual Studios.

Retrieving the dataset

We found a Kindle Book Review dataset on Kaggle. It had 4 columns: Unnamed: 0, Summary, Ratings and Review Text. There are 12000 entries in the dataset. The 5 and 4-star labels have 3000 reviews linked to each of them. The 3, 2, and 1-star labels have 2000 reviews available for each of them. We retrieved our dataset and loaded it into our project using the panda's library.

```
Shape:  (12000, 2)

Value Counts :
5    3000
4    3000
1    2000
3    2000
2    2000
Name: rating, dtype: int64
```

Preprocessing

With regards to the preprocessing that was done on this dataset, we first removed any unnecessary columns that were not going to be used for this project. In this case, the columns that we decided to focus on are the "rating" and "reviewtext" columns in our dataset.

We created a function that takes in text and performs the following processing on it:

1. Using regular expressions, we removed any character that is not an alphabet
2. The text is normalized by converting each word to lowercase
3. Stop words are the common short function words in a language that don't add to the meaning of the sentence but just provide grammatical structure. In this step, stop words are then removed from the text as this allows the model to only be trained on the important words that serve a function in the sentences. (To do this we made use of the stop words from the NLTK library)
4. We then removed all of the punctuation from the text
5. Finally, using the NLTK WordNetLemmatizer, we lemmatize each word in the text. Lemmatization of the text means that each word in the text is converted to its root form from its inflected form. We decided to use lemmatization instead of stemming as lemmatization is better than stemming as it gives context to words. We had tried using stemming at first, however we quickly experienced two challenges with this approach, the over and under-stemming of words. Certain inflected words were cut off so much due to the stemming that the resulting stem was nonsensical. In certain other cases in which we had various forms of a certain root word, the resulting stemmed word of each of the different forms of the word were not the same.

The function was then applied to the review text and the results of this were stored in a new column called Processed Review.

```
21 def textprocessing(text):
22     text = re.sub(r'[^a-zA-Z\s]', '', text)
23     text = re.sub(r'<.*?>', '', text)
24     text = text.lower().strip()
25     text = text.translate(str.maketrans('', '', string.punctuation))
26     text = " ".join([lemmatizer.lemmatize(word) for word in str(text).split() if word not in stpwrds])
27
28     return text
29
```

Training the model

We decided to split our dataset using sklearn library, so that 70% would be used for training and 30% would be used for testing.

```
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

♦ Word Embedding

We first needed to convert our processed textual data into a numerical form This is because most of the algorithms in machine learning cannot process strings or plain text in their raw form. Instead, most algorithms in machine learning need numbers as inputs in order to work. We can process text data and make them suitable for machine learning algorithms by converting words into vectors.

We used the Count Vectorizer and TF-IDF Vectorizer from sklearn to convert text into vectors for this part of our project.

Count Vectorization:

CountVectorizer transforms a given text into a vector based on the number of times a word appears in the entire document. The CountVectorizer generates a matrix in which each unique word is represented by a column and each text sample from the document is represented by a row. The value of each cell is the count of the word in that particular text sample.

Some of the disadvantages of count vectorization are:

- Its inability in identifying more important and less important words for analysis.
- Words that have a low frequency may result in the word being ignored.
- It also doesn't identify the relationships between words such as linguistic similarity between words.

TF-IDF:

Term Frequency — Inverse Document Frequency provides a numerical representation of how important a word in a document is. It is based on the logic that words that appear too frequently in a corpus and words that are rarely found in the document are both not important for finding a pattern. The Logarithmic factor in TF-IDF mathematically penalizes these words by giving them low TF-IDF scores.A higher TF-IDF score signifies a higher importance of the words in the corpus while lower scores represent lower importance. Even though TF-IDF can provide a good understanding about the importance of words but just like Count Vectors, its disadvantage is that it fails to provide linguistic information about the words such as the real meaning of the words, similarity with other words etc.

- ## Machine Learning techniques

For our project, we implemented the following classifiers from sklearn using various classifiers in order to determine which classifier would be the best fit for our project:

1. MULTINOMIAL NAIVE BAYES

The Multinomial Naive Bayes classifier comes from a family of Naïve Bayes probabilistic classifiers. These classifiers are based on Bayes Theorem. We can say that they are "naïve" in the sense that with these classifiers, there is the assumption that features in the dataset are independent of each other. It is usually used to determine whether a document or text belongs to a particular category. The features used by the classifier are the frequency of the words present in the document. Bayes Theorem is used to calculate the conditional probability of each label given a piece of text, and the label with highest probability will be returned as the correct label for that piece of text.

2. LOGISTIC REGRESSION

Logistic regression models are statistical, supervised machine learning models and are discriminative models. This means that it tries to distinguish between classes or categories. In terms of our project, it is used to distinguish between the different star ratings. They are often used in classification problems and predictive analytics. It estimates the probability of an event occurring.

3. RANDOM FOREST

Random forest classifiers are supervised learning algorithms that can be applied to regression and classification problems. It consists of multiple decision trees and uses randomness to enhance its accuracy and combat overfitting, which can be a huge issue for sophisticated algorithms. Based on a random selection of samples, these algorithms make decision trees and get predictions from each tree. Using votes, they then select the most viable solution.

4. LINEAR SVC

The Support Vector Classifier is a discriminative classifier that is formally designed by a separative hyperplane (a line in between different type objects). A Linear SVC is intended to fit to the data you provide, returning a hyperplane that categorizes your data based on its "best fit". Once the hyperplane is created, you should feed some features into the classifier to find out what the "predicted" class is.

Evaluation

In order to determine which classifier would be appropriate in the final model, we looked at the following metrics available from sklearn:

Accuracy is a measure of the correctly predicted tags over all the observations. It is easy to use however it is very weak and does not account for any slight errors. Accuracy looks at the prediction made as a whole and is best used when all the classes are equally important.

Precision is the measure of the correctly identified positive cases from all the predicted positive cases. This makes it useful when the costs of false positives are high.

Recall is the measure of the correctly identified positive cases from all the actual positive cases. It is important when the cost of false negatives is high. Precision and Recall go together as they measure each other's opposites. While precision refers to the percentage of your results which are relevant, recall refers to the percentage of total relevant results correctly classified by your algorithm

F1-score is the harmonic mean of Precision and Recall. If directly compared to accuracy, it gives a better measure of the incorrectly classified cases than accuracy does. Accuracy can be used when the class distribution is similar while F1-score is a better metric when there are imbalanced classes.

Confusion Matrix: A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

Results and Data Analysis

1. MULTINOMIAL NAIVE BAYES

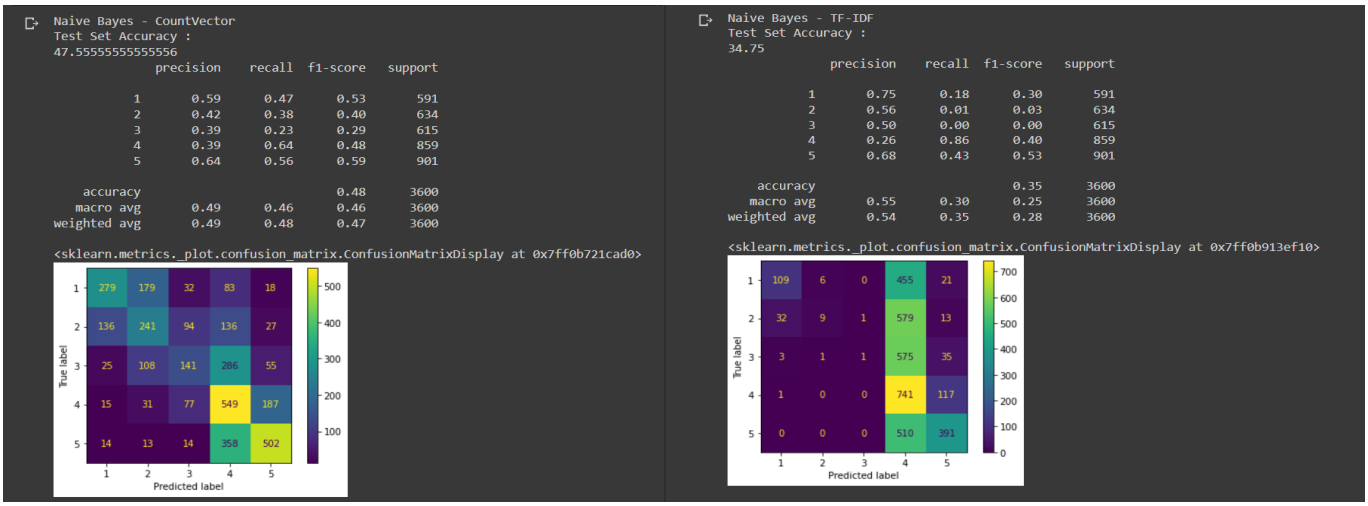
Using Count Vectorizer:

The overall accuracy is approximately 48%. For the precision of 1-star and 5-star ratings, the percentage of correctly identified positive cases when compared to the total number of predicted positive cases is much higher than the other star ratings at above 50%.

For the recall of the 3-star rating, the percentage of correctly identified 3-star reviews from all the actual 3-star rated reviews is much lower than the other star ratings at 23%. The recall of the other classes range between 38% and 64%.

Using TF-IDF:

The overall accuracy is approximately 35%. The precision percentage of the 4-star ratings, is much lower than the other star ratings at 26%. The recall percentage of the 1,2 and 3-star ratings, is much lower than the other star ratings at 18% ,1% and 0% respectively, while the recall of the 4-star class is 86% and the recall of the 5-star class is 43%.



In terms of accuracy, the Multinomial Naïve Bayes classifier fitted with the Count Vectorizer performed better. For precision, the model with the fitted TF-IDF Vectorizer performed better however, the precision score for the 4-star label was better in the model fitted with the Count Vectorizer. With regards to recall, the model fitted with the Count Vectorizer performed better in all except in the 4-star class. Therefore, the model we determined that worked best for our project is the model fitted with the Count Vectorizer.

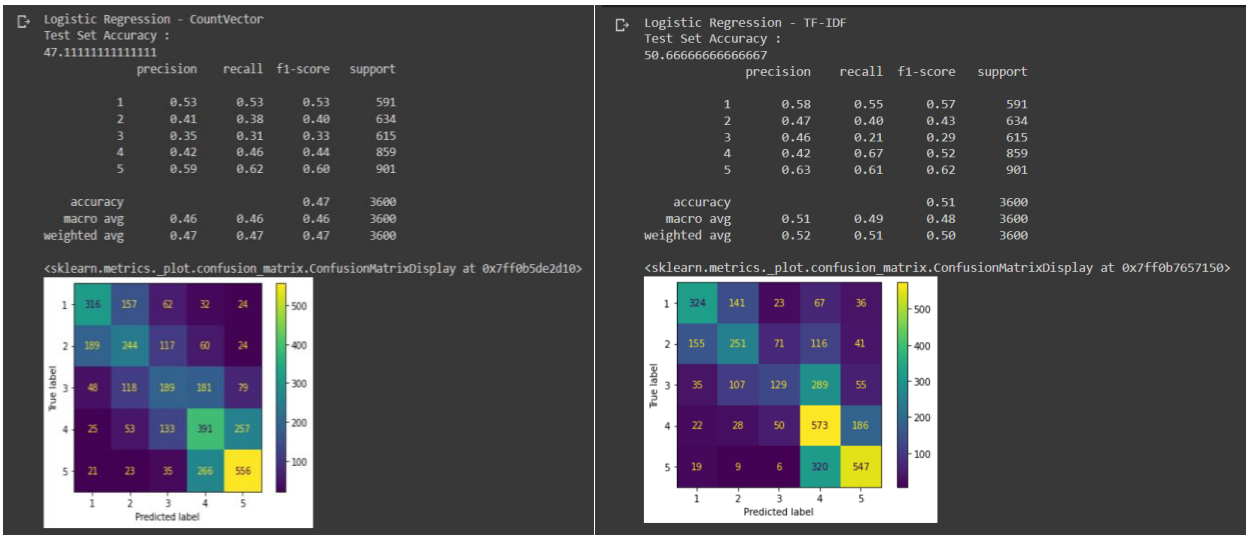
2. LOGISTIC REGRESSION

Using Count Vectorizer:

The overall accuracy is approximately 47%. The precision values for the labels range between 35% and 59%. The lowest value is for the 3-star label. It has a precision of 35%. The recall percentage of the 3-star rating is lower than the other star ratings at 31%. The recall of the other classes range between 38% and 62%.

Using TF-IDF:

The overall accuracy is approximately 51%. The precision of the 4-star ratings is the lowest amongst all the other star ratings at 42%. The other ratings have precision values range between 46% and 63%. The recall percentage of the 3-star rating, is lower than the other star ratings at 21%. The recall of the other classes range between 40% and 67%.



Logistic Regression - CountVector
Test Set Accuracy :
47.11111111111111

	precision	recall	f1-score	support
1	0.53	0.53	0.53	591
2	0.41	0.38	0.40	634
3	0.35	0.31	0.33	615
4	0.42	0.46	0.44	859
5	0.59	0.62	0.60	901
accuracy			0.47	3600
macro avg	0.46	0.46	0.46	3600
weighted avg	0.47	0.47	0.47	3600

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff0b5de2d10>

Logistic Regression - TF-IDF
Test Set Accuracy :
50.66666666666667

	precision	recall	f1-score	support
1	0.58	0.55	0.57	591
2	0.47	0.40	0.43	634
3	0.46	0.21	0.29	615
4	0.42	0.67	0.52	859
5	0.63	0.61	0.62	901
accuracy			0.51	3600
macro avg	0.51	0.49	0.48	3600
weighted avg	0.52	0.51	0.50	3600

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff0b7657150>

In terms of accuracy, the logistic regression classifier fitted with the TF-IDF Vectorizer performed better. For precision, once again the model with the fitted TF-IDF Vectorizer performed better however, the score for the 4-star label for both classifiers remained the same. With regards to recall, the TF-IDF model performed better in all except in the 5 and 3-star class. Therefore, the model we determined that worked best for our project is the model fitted with the TF-IDF vectorizer.

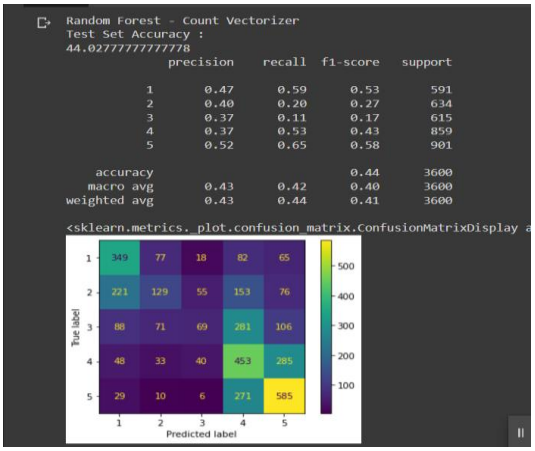
3. RANDOM FOREST

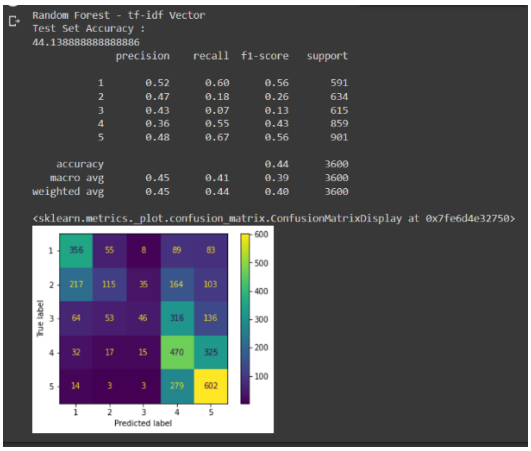
Using Count Vector:

The overall accuracy is approximately 44%. The precision percentage of 1-star and 5-star ratings is much higher than the other star ratings at 47% and 52% respectively, while the precision percentage of the other star ratings range between 37% and 40%. The recall percentage of the 3-star and 2-star classes is much lower than the other star ratings at 11% and 20% respectively, while the recall of the other star ratings range between 53% and 65%.

Using TF-IDF:

The overall accuracy is approximately 44%. The precision percentage of the 1-star rating is higher at 52%, while the precision percentage of the other star ratings ranges between 36% and 48%. The recall percentage of the 3-star and 2-star classes are much lower than the other star ratings at 7% and 18% respectively while the recall of the other star ratings range between 55% and 67%.





In terms of accuracy, both Random Forest models had a value of 44%. For precision, the model with the fitted TF-IDF Vectorizer performed better however, the score for the 4-star and 5-star labels for the models fitted with the count vectorizer performed slightly better. With regards to recall, the TF-IDF model performed better in all except in the 2 and 3-star classes. The model we determined that worked best for our project is the model fitted with the TF-IDF vectorizer.

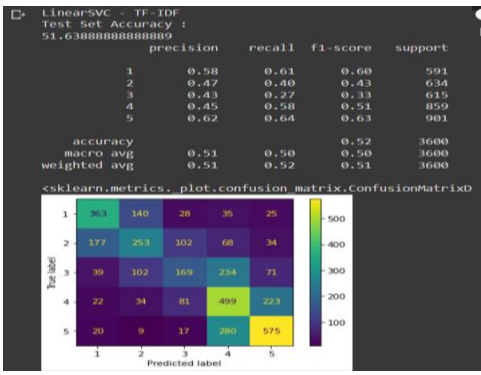
4. LINEAR SVC

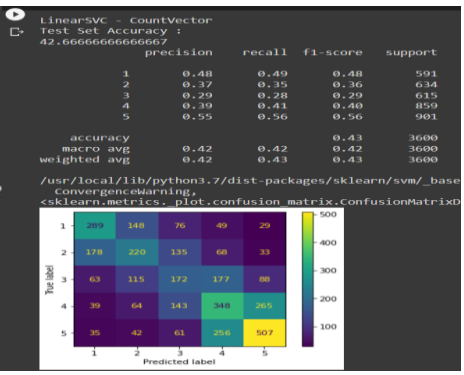
Using Count Vector:

The overall accuracy is approximately 43%. The precision percentage of the 1-star and 5-star ratings is much higher than the other star ratings at 48% and 55% respectively, while the precision percentage of the other star ratings ranges between 29% and 39%. The recall score of the 3-star class, is lower than the other star ratings at 28%, while the recall of the other star ratings range between 35% and 56%.

Using TF-IDF:

The overall accuracy is approximately 52%. The precision percentage of the 1- and 5-star ratings, is much higher at 58% and 62% respectively, while the precision percentage of the other star ratings ranges between 43% and 47%. The recall score of the 3-star class, is much lower than the other star ratings at 27% while the recall of the other star ratings range between 40% and 64%.





In terms of accuracy, the LinearSVC classifier fitted with the TF-IDF Vectorizer performed better. For precision, once again the model with the fitted TF-IDF Vectorizer performed better. With regards to recall, the TF-IDF model performed better in all except in the 3-star class. The model we determined that worked best for our project is the model fitted with the TF-IDF.

Conclusion

After analyzing our models, we came up with the following conclusions:

We found that the TF-IDF vectorizer was a better technique to use for all the models with the exception of the multinomial naïve bayes model.

Since the count vectorizer counts the frequency of the words in our corpus, sorts them and grabs the most recurring features, the results produced have a high possibility of being biased causing our models to lose out on some of the more important but less frequently appearing features or words. TF-IDF on the other hand corrects this since it not only focuses on the frequency of words present in the corpus but also provides the importance of the words, which results in words that appear less frequently being given a higher score.

Recall Values Using TF-IDF

	Multinomial Naïve Bayes	Logistic Regression	LinearSVC	Random Forest	Best Value
1 star	18	55	61	60	61
2 stars	1	40	40	18	40
3 stars	0	21	27	7	27
4 stars	86	67	58	55	86
5 stars	43	61	64	67	67

We decided to value the recall scores over the other metrics as we preferred to have more false negatives than false positives and since TF-IDF was found to be the better word embedding technique, our next step was to compare the recall score of each star class in each of the models we used that used TF-IDF. This is depicted in the table below.

We then made note of the highest recall value for each star class. Finally, we compare the models to find which model has the greatest number of classes with the highest recall score.

Based on our findings, we chose the LinearSVC model for predicting the ratings of book reviews because it had the highest number of classes in which the best recall score was found.

```
model = LinearSVC(max_iter=200,C=0.1).fit(X_train_tV, y_train)
y_pred = model.predict(X_test_tV)
cf_matrix = confusion_matrix(y_test, y_pred)

print('LinearSVC - TF-IDF')
print("Test Set Accuracy : ")
print(metrics.accuracy_score(y_test, y_pred) * 100)
print(metrics.classification_report(y_test, y_pred))
disp = ConfusionMatrixDisplay(confusion_matrix=cf_matrix, display
disp.plot()
```

```
LinearSVC - TF-IDF
Test Set Accuracy :
50.638888888888886
              precision    recall  f1-score   support

           1       0.57      0.59      0.58       591
           2       0.46      0.36      0.40       634
           3       0.48      0.17      0.26       615
           4       0.43      0.66      0.52       859
           5       0.61      0.64      0.62       901

    accuracy                           0.51      3600
   macro avg       0.51      0.48      0.48      3600
weighted avg       0.51      0.51      0.49      3600
```

We played around with the parameters of the model to increase recall, however the default values of the models from sklearn were found to be optimal. The value of parameter C changed the values of the recall scores the most, however, whilst some recall values increased, majority decreased. This statement proved true when we both increased and decreased the value of C. Some parameters, such as max_iterations, had no effect at all on our results.

Shortcomings

While trying to improve our model, one of the issues we noted was that when it came to predicting a rating, any negative review that contained the word "not" in it was usually given a higher predicted star rating than what it should have been given. This is because "not" is included in the stopwords list and was therefore removed from the reviews and so it was not contributing to the overall sentiment of the review. To counter this, we removed "not" from the stopwords list. This enabled us to more accurately predict the ratings of the reviews that contained a negative sentiment and classified them in their correct star rating class.

One shortcoming that we noticed is that the model was not able to detect sarcasm. This could have been solved if we were able to get our hands on a set of reviews in which sarcasm was prominent so that the model could be trained to identify sarcasm. Unfortunately, we were not able to do so. We also found that contracted words that are included in the stopwords list, despite attempting to remove them from the list, were still not being considered and we have not been able to rectify this.