Capstone Project – 006

Multi-Class Text Sentiment Analysis

(using Python)

By

Abhiteja Achanta, M13023084

MS Business Analytics, University of Cincinnati

First Reader: Dr. Yichen Qin

Second Reader: Dr. Liwei Chen

Abstract

The goal of sentiment analysis is to extract human emotions from text. This project applies various machine learning algorithms to predict sentiment of reviewer from his textual review on Amazon food products. Metrics such as accuracy of prediction and precision/recall are presented to gauge the success of these different algorithms. Main purpose of this project is to introduce and apply different feature engineering techniques to convert text to numeric data and see how different Machine learning and Deep Learning algorithms perform with this data.

We have built following models:

1. Multinomial Naïve Bayes
2. Logistic Regression
3. Random Forest
4. Xgboost Classifier
5. Dense Neural Network
6. Recurrent Neural Network (LSTM)
7. Convolutional Neural Network (CNN)

Table of Contents

1. Introduction:

Text sentiment analysis is an important research topic for its wide applicability in real-world applications, and recent breakthroughs in text embedding and classification models led to state-of-the-art results. This project aims to apply recent innovations in machine learning to Amazon reviews. Different types of ML/DL models are applied to text data like Naïve Bayes, Logistic Regression, Random Forest, Xgboost Classifier, Dense Neural Network, Recurrent Neural Network (LSTM), Convolutional Neural Network (CNN).

2. Objective:

The objective of this project is to predict one of the three (positive, neutral, negative) sentiment classes given an Amazon food review.

3. Exploratory Data Analysis:

This is the first step in any data analysis project. We need to completely understand the data even before we implement machine learning models on the data.

3.1 Data Sources:

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Data includes:

- Reviews from Oct 1999 - Oct 2012
- 568,454 reviews
- 256,059 users
- 74,258 products
- 260 users with > 50 reviews

We are going to use the subset of this dataset with close to 100,000 reviews due to the memory and capacity constraint of the machine.

3.2 Data Definition:

- **Id:** Row Id.
- **ProductId:** Unique identifier for the product.
- **UserId:** Unique identifier for the user.
- **ProfileName:** Profile name of the user.
- **HelpfulnessNumerator:** Number of users who found the review helpful.
- **HelpfulnessDenominator:** Number of users who indicated whether they found the review helpful or not.
- **Score:** Rating between 1 and 5.
- **Time:** Timestamp for the review.
- **Summary:** Brief summary of the review.
- **Text:** Text of the review.

3.3 Data Wrangling:

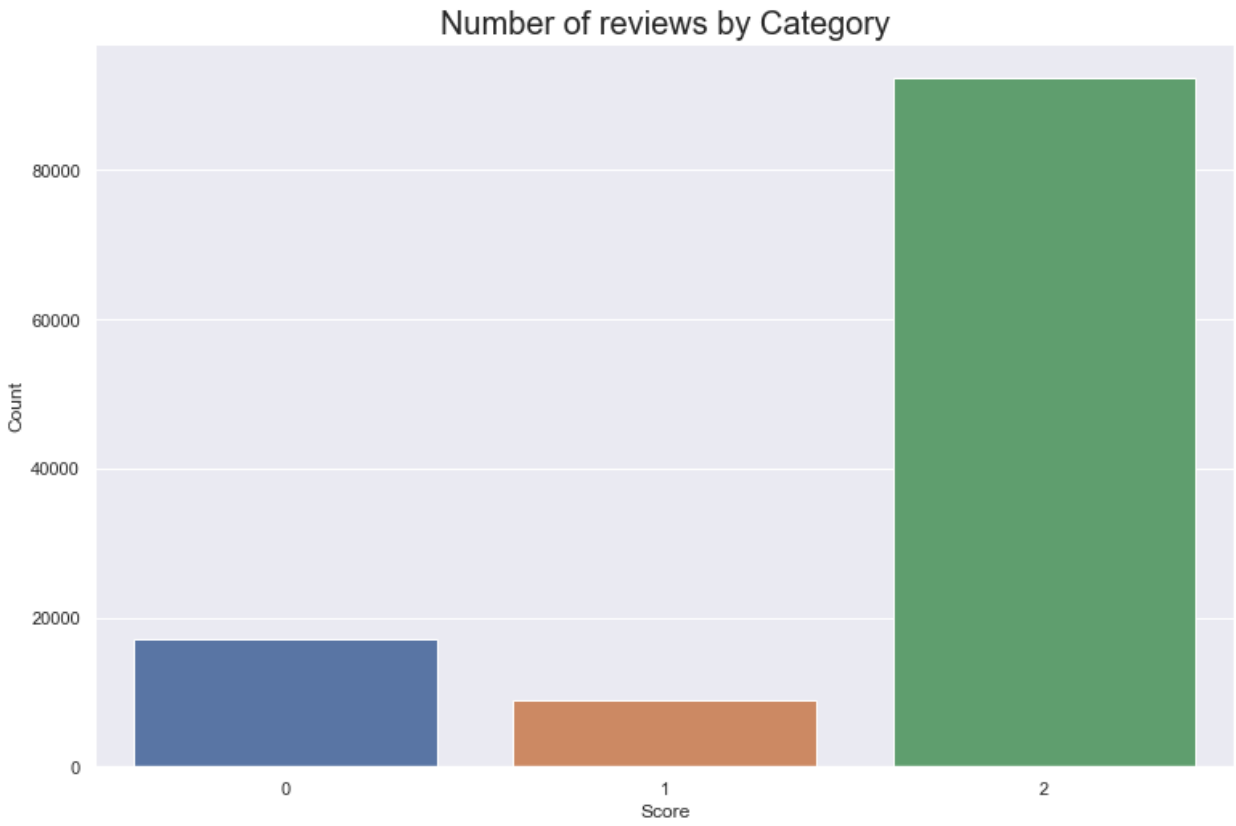
We perform couple of data wrangling tasks on the actual dataset to get the dataset into desired format for our analysis and modelling. Following manipulations are carried out on original dataset:

- Remove unnecessary columns from dataset. We only need Text and score columns.
- Remove the rows with missing values.
- Remove the duplicate rows in the dataset.
- Change the Score column such that ratings less than 3 are converted to 0, ratings with 3 are converted to 1 and ratings greater than 3 are converted to 2.
- Hence, we have 3 classes (positive-2, neutral-1, negative-0).
- After performing above tasks, take the subset of data with similar percentages of three classes to carry out further analysis and modelling because of memory and capacity constraint.

3.4 Data Visualization:

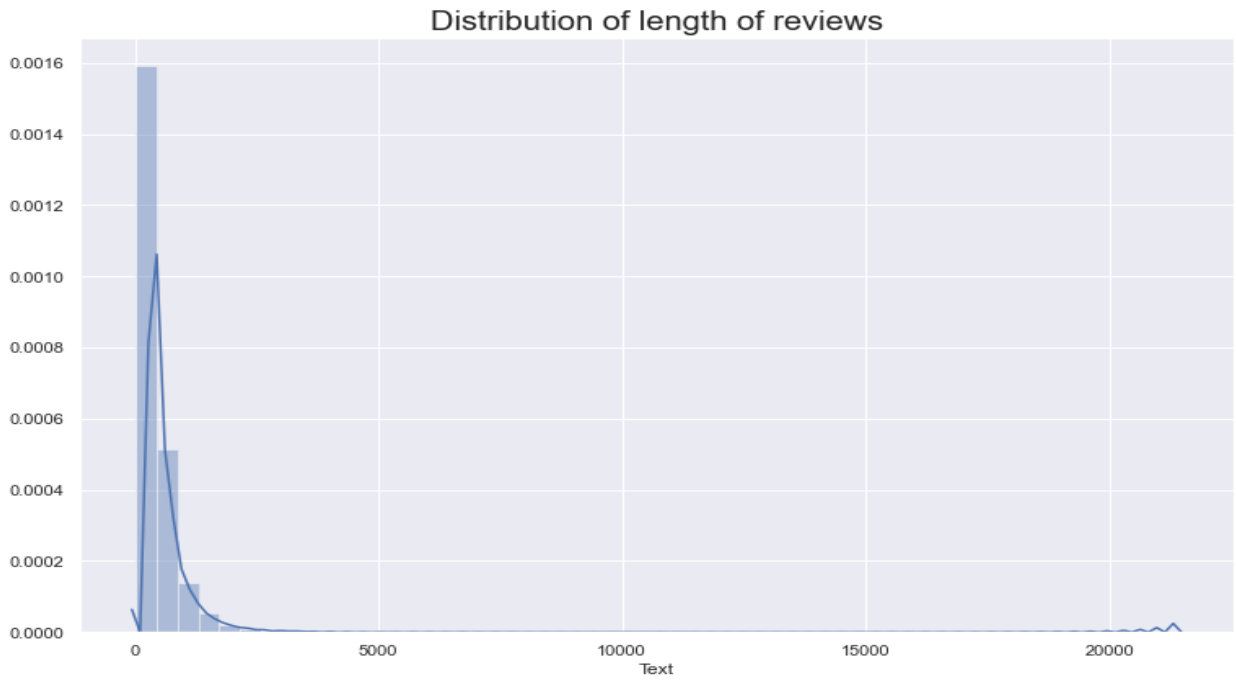
Using the dataset, we build couple of visualizations to get more understanding of the data. As we have text data, I made use of word clouds to highlight the words present each category of reviews.

Fig 3.1 Number of reviews by category



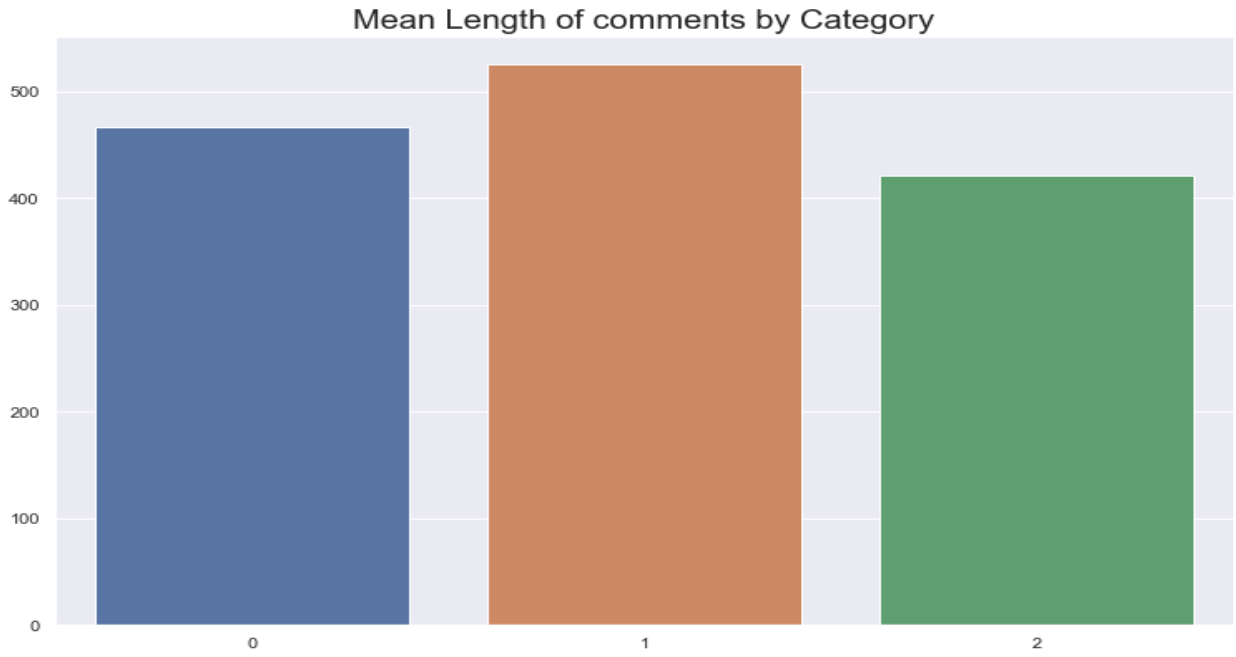
- Most of the reviews are positive.
- From the above chart, we can confirm this is highly imbalanced dataset.
- We can use techniques like under sampling or over sampling to deal with this issue.

Fig 3.2 Distribution of length of reviews



- We can observe that above distribution is right skewed.

Fig 3.3 Mean length of comments by category



- Length of neutral reviews is longer on average compared to positive, negative reviews.

3.5 Data Cleaning:

Before the feature engineering and modelling, we clean the text data after examining couple of reviews from each category. Following data cleaning steps are carried out:

- Converting entire text into Lowercase to avoid considering single word as two different words.
- Remove new line characters.
- Converting words like don't to do not using custom dictionary.
- Remove all the html tags like the one in the above neutral review.
- Remove all the punctuation marks.
- Remove all the numeric characters from the text.
- Remove the stop words like a, the etc. for the better performance of model.
- Carry out lemmatization to convert word like rocks to rock.

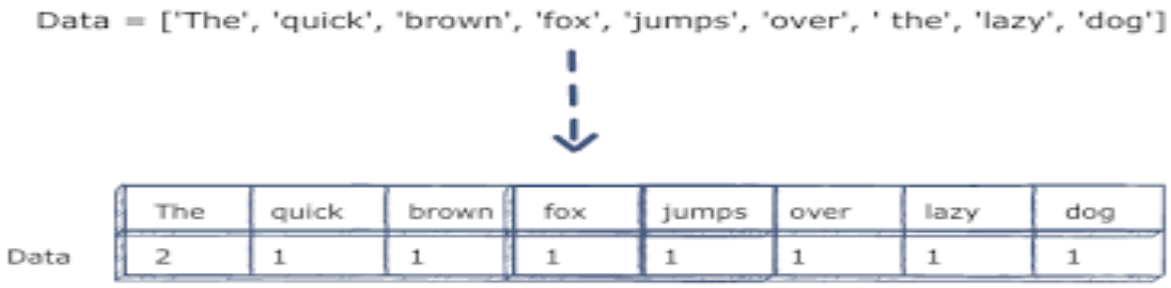
4. Feature Engineering:

Now that we have cleaned text, next step is to convert text data to numeric data which can be fed to Machine Learning/ Deep Learning algorithms. There are multiple techniques to accomplish this task. We will explore three major techniques in our project.

4.1 Count Vectorizer:

To use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called **tokenization**. These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms. This process is called **feature extraction (or vectorization)**.

Scikit-learn's Count Vectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.



4.2 TF-IDF Vectorizer:

One issue with simple counts is that some words like "*the*" will appear many times and their large counts will not be very meaningful in the encoded vectors.

An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF. This is an acronym than stands for "*Term Frequency – Inverse Document*" Frequency which are the components of the resulting scores assigned to each word.

- **Term Frequency**: This summarizes how often a given word appears within a document.
- **Inverse Document Frequency**: This downscales words that appear a lot across documents.

To put it in more formal mathematical terms, the TF-IDF score for the word t in the document d from the document set D is calculated as follows:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

Where:

$$tf(t, d) = log(1 + freq(t, d))$$

$$idf(t, D) = log(\frac{N}{count(d \in D : t \in d)})$$

We can use TfidfVectorizer function from sklearn to perform this task.

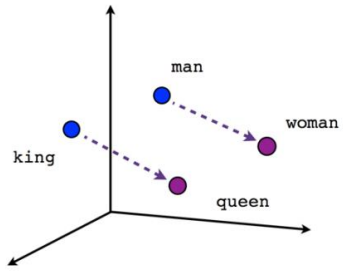
4.3 Word2Vec

Word2Vec is a type of word embedding. Word embedding is a form of representing words and documents using a dense vector representation. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. Word embeddings can be trained using the input corpus itself or can be generated using pre-trained word embeddings such as **Glove, FastText,** and **Word2Vec**.

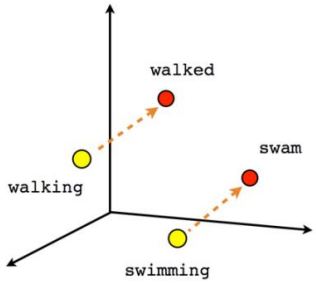
Word vectors are positioned in the vector space such that words that share common contexts in the corpus are in the proximity to one another in space. Word2Vec is particularly computationally effective predictive model for learning word embeddings from raw text.

It comes in two flavors, the Continuous Bag-of-Words (CBOW) model and the Skip-Gram model. Algorithmically, these models are similar.

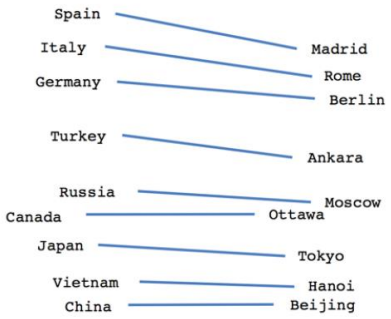
Word2Vec can capture multiple different degrees of similarity between words, such that semantic and syntactic patterns can be reproduced using vector arithmetic. Patterns such as “Man is to Woman as Brother is to Sister” can be generated through algebraic operations on the vector representations of these words such that the vector representation of “Brother” - ”Man” + ”Woman” produces a result which is closest to the vector representation of “Sister” in the model. Such relationships can be generated for a range of semantic relations (such as Country—Capital) as well as syntactic relations (e.g. present tense—past tense).



Male-Female



Verb tense



Country-Capital

We use genism package in Python to create word embeddings using Word2Vec. There are two ways we can use embeddings. We can create word embeddings from our corpus, or we can use pre trained word embeddings by downloading them online like word2vec-google-news-300 where each is represented in 300-dimensional vector space. In this project we explored both the techniques for at least one model.

5. Model Building and Evaluation:

We perform feature engineering on the dataset using above three techniques mentioned. These three different datasets are used to build machine learning models and their evaluation metrics like accuracy/ f1 score etc. are compared. For each algorithm, we report training accuracy, test accuracy, classification report with metrics like precision/recall and confusion matrix. For naïve Bayes algorithm we do not use dataset from Word2Vec because it cannot handle negative values in the dataset.

5.1 Machine Learning:

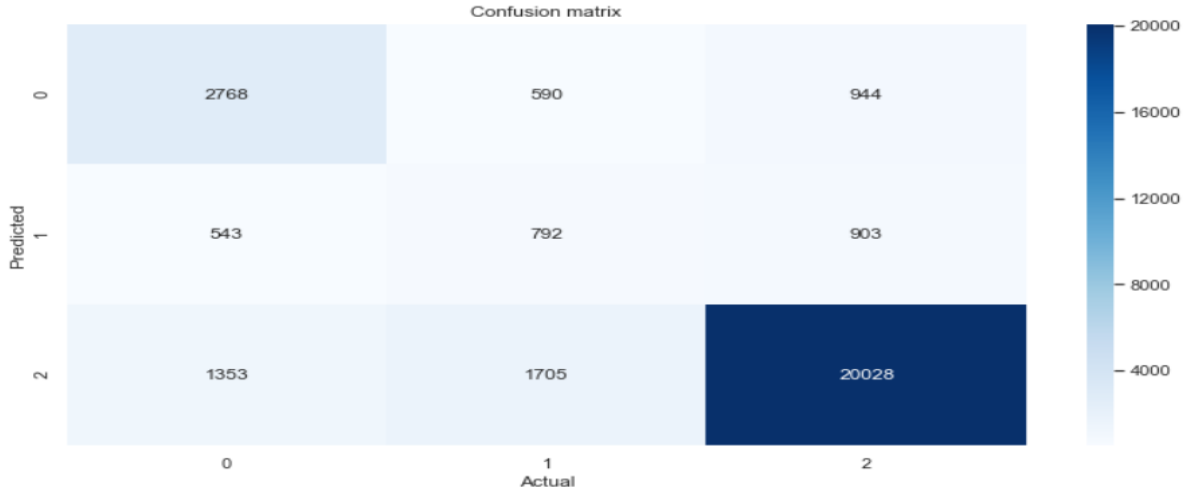
5.1.1 Multinomial Naïve Bayes

```
Train accuracy of Multinomial Naive BayesClassifier with Count Vectorizer is 0.815606188466948
Test  accuracy of Multinomial Naive BayesClassifier with Count Vectorizer is 0.7961925335853642

Classification report is :
              precision    recall  f1-score   support

           0       0.59      0.64      0.62      4302
           1       0.26      0.35      0.30      2238
           2       0.92      0.87      0.89     23086

    accuracy                           0.80     29626
   macro avg       0.59      0.62      0.60     29626
weighted avg       0.82      0.80      0.81     29626
```

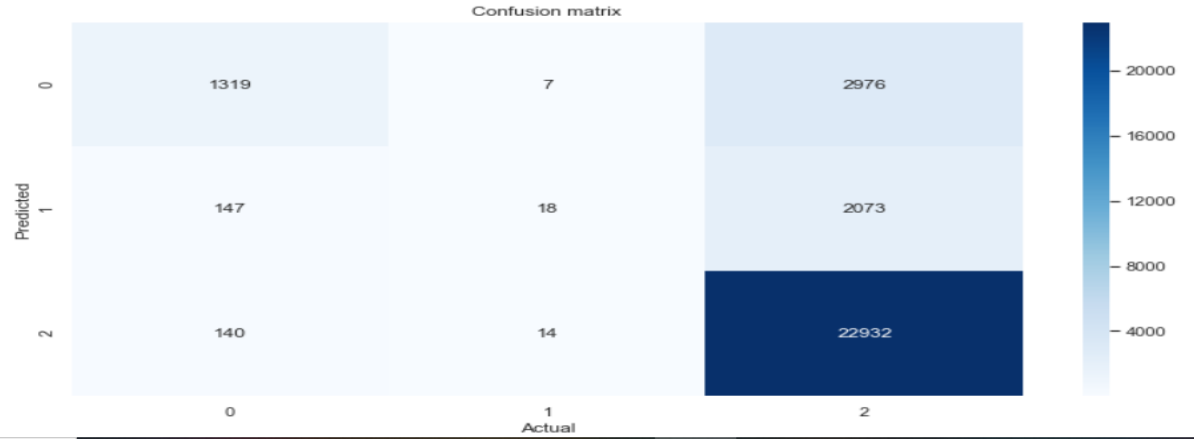



```
Train accuracy of Multinomial Naive BayesClassifier with TF-IDF Vectorizer is 0.8267116736990154
Test  accuracy of Multinomial Naive BayesClassifier with TF-IDF Vectorizer is 0.8191790994396814

Classification report is :
              precision    recall  f1-score   support

           0       0.82      0.31      0.45      4302
           1       0.46      0.01      0.02      2238
           2       0.82      0.99      0.90     23086

    accuracy                           0.82     29626
   macro avg       0.70      0.44      0.45     29626
weighted avg       0.79      0.82      0.77     29626
```



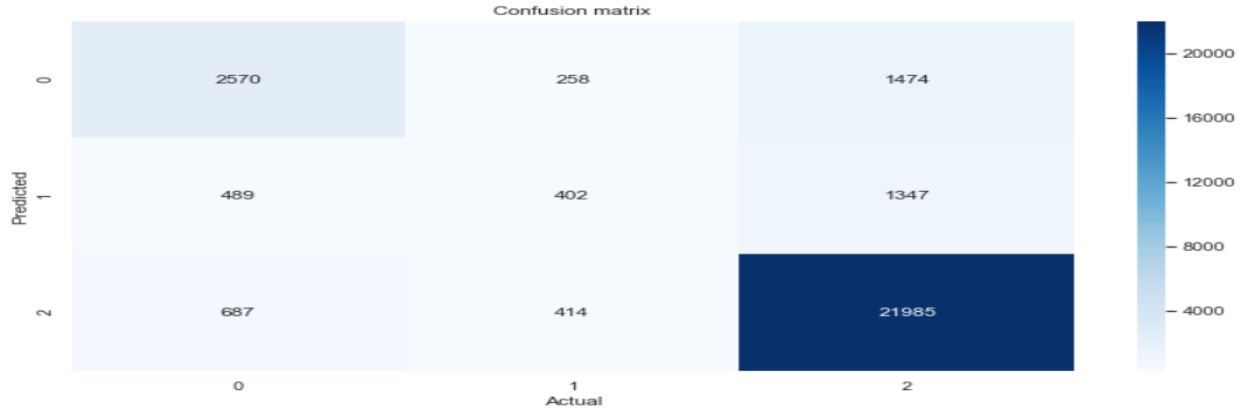
5.1.2 Logistic Regression

```
Train accuracy of Logistic RegressionClassifier with Count Vectorizer is 0.9045850914205344
Test  accuracy of Logistic RegressionClassifier with Count Vectorizer is 0.8424019442381692

Classification report is :
              precision    recall  f1-score   support

           0       0.69      0.60      0.64      4302
           1       0.37      0.18      0.24      2238
           2       0.89      0.95      0.92     23086

    accuracy                           0.84     29626
   macro avg       0.65      0.58      0.60     29626
weighted avg       0.82      0.84      0.83     29626
```

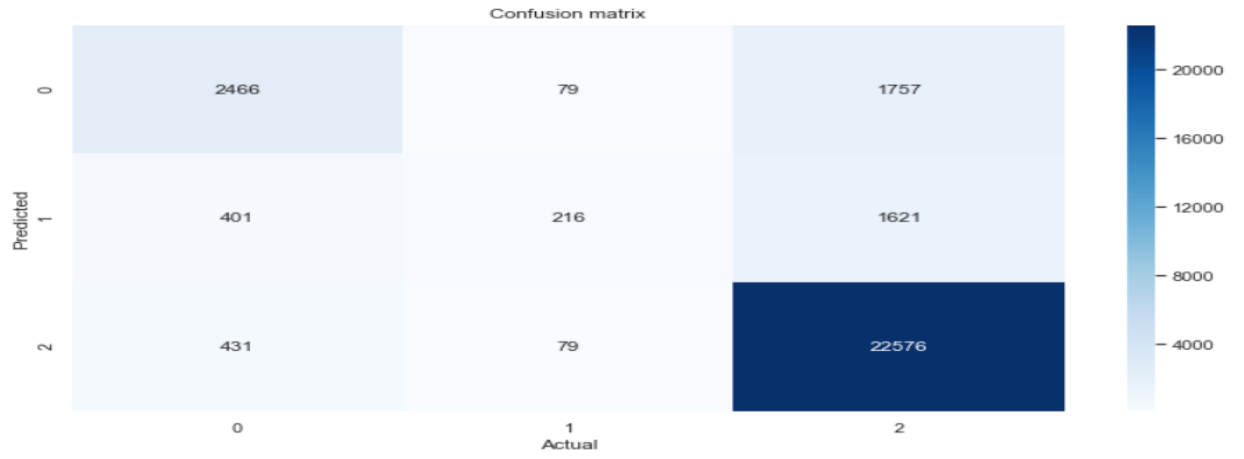



```
Train accuracy of Logistic RegressionClassifier with TF-IDF Vectorizer is 0.8688270042194093
Test  accuracy of Logistic RegressionClassifier with TF-IDF Vectorizer is 0.8525619388375076

Classification report is :
              precision    recall  f1-score   support

           0       0.75      0.57      0.65      4302
           1       0.58      0.10      0.17      2238
           2       0.87      0.98      0.92     23086

    accuracy                           0.85     29626
   macro avg       0.73      0.55      0.58     29626
weighted avg       0.83      0.85      0.82     29626
```

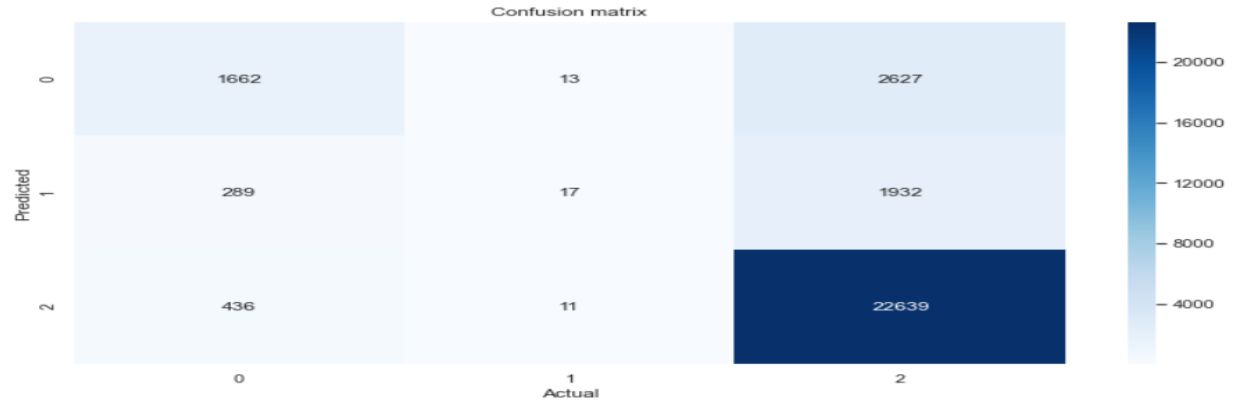


```
Train accuracy of Logistic RegressionClassifier with Word2Vec is 0.8208045007032349
Test  accuracy of Logistic RegressionClassifier with Word2Vec is 0.820833052048876

Classification report is :
              precision    recall  f1-score   support

           0       0.70      0.39      0.50      4302
           1       0.41      0.01      0.01      2238
           2       0.83      0.98      0.90     23086

    accuracy                           0.82     29626
   macro avg       0.65      0.46      0.47     29626
weighted avg       0.78      0.82      0.77     29626
```



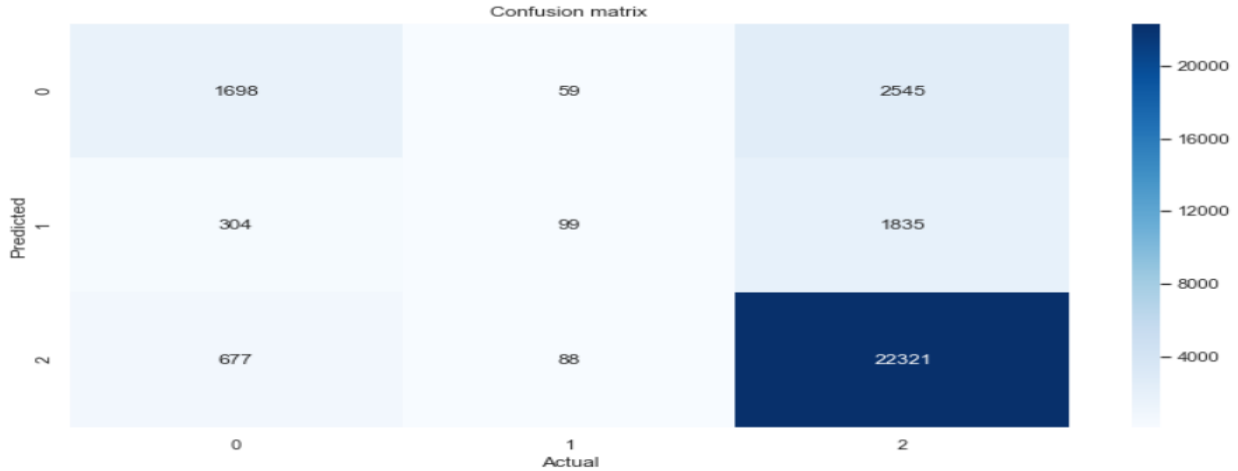
5.1.3 Random Forest

```
Train accuracy of Random ForestClassifier with Count Vectorizer is 0.99143741209564
Test  accuracy of Random ForestClassifier with Count Vectorizer is 0.8140822250725714

Classification report is :
              precision    recall  f1-score   support

           0       0.63      0.39      0.49      4302
           1       0.40      0.04      0.08      2238
           2       0.84      0.97      0.90     23086

    accuracy                           0.81     29626
   macro avg       0.62      0.47      0.49     29626
weighted avg       0.77      0.81      0.78     29626
```

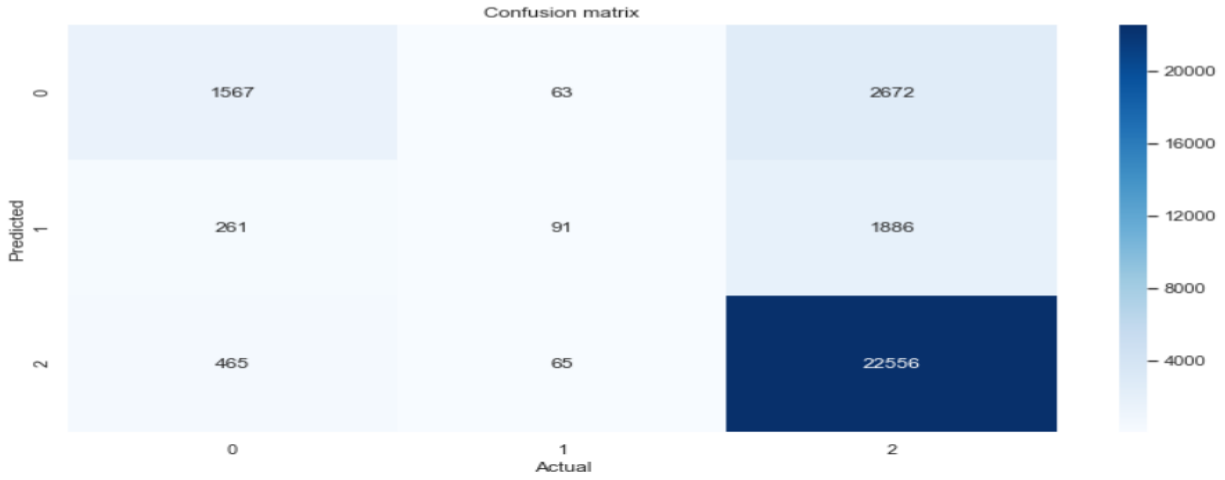


```
Train accuracy of Random ForestClassifier with TF-IDF Vectorizer is 0.9923825597749648
Test  accuracy of Random ForestClassifier with TF-IDF Vectorizer is 0.8173226220211975

Classification report is :
              precision    recall  f1-score   support

           0       0.68      0.36      0.48      4302
           1       0.42      0.04      0.07      2238
           2       0.83      0.98      0.90     23086

    accuracy                           0.82     29626
   macro avg       0.64      0.46      0.48     29626
weighted avg       0.78      0.82      0.77     29626
```

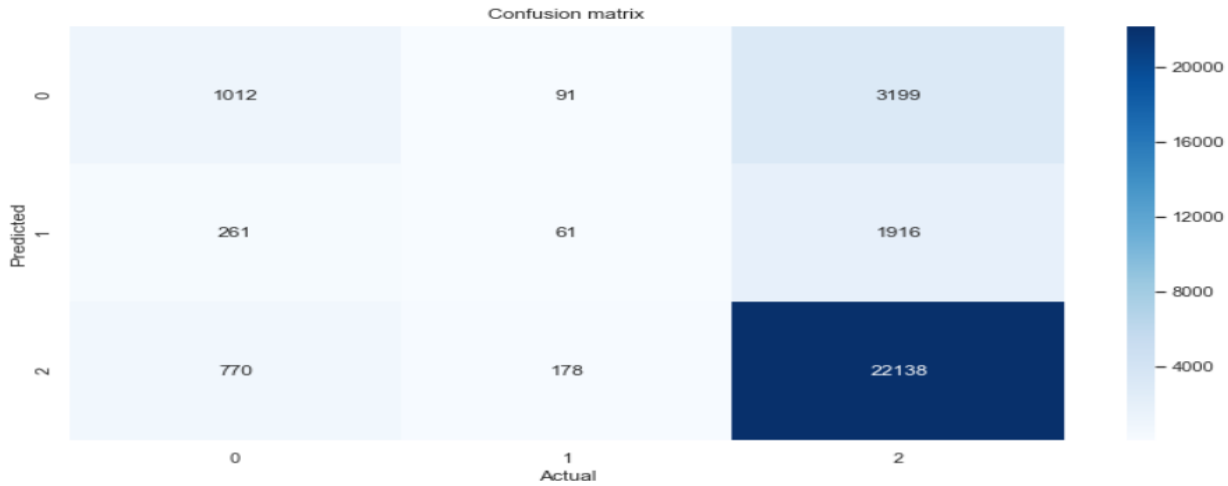



```
Train accuracy of Random ForestClassifier with Word2Vec is 0.992056258790436
Test  accuracy of Random ForestClassifier with Word2Vec is 0.78346722473503

Classification report is :
              precision    recall  f1-score   support

           0       0.50      0.24      0.32      4302
           1       0.18      0.03      0.05      2238
           2       0.81      0.96      0.88     23086

    accuracy                           0.78     29626
   macro avg       0.50      0.41      0.42     29626
weighted avg       0.72      0.78      0.74     29626
```



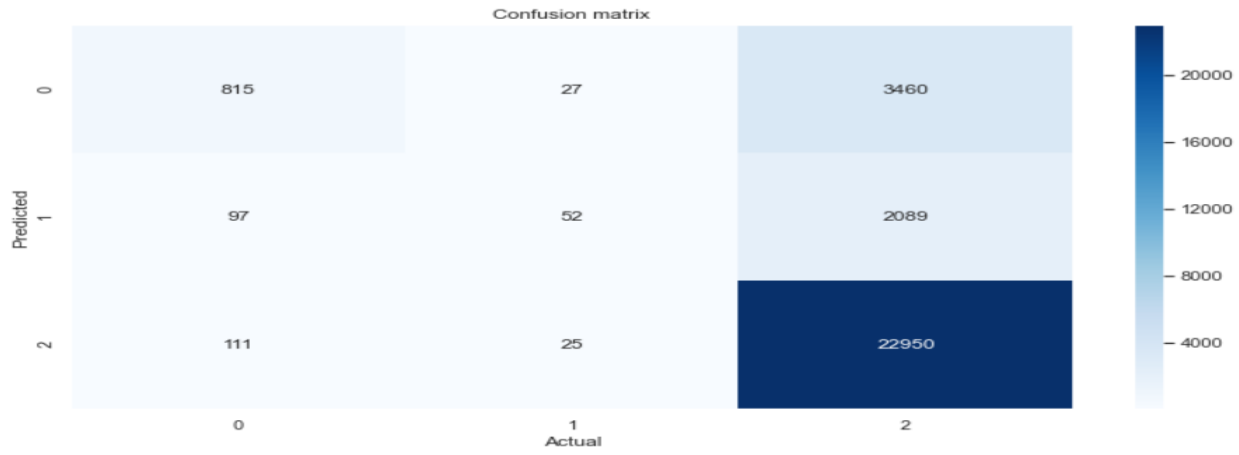
5.1.4 Xgboost Classifier

```
Train accuracy of XGBoost ClassifierClassifier with Count Vectorizer is 0.8079662447257384
Test  accuracy of XGBoost ClassifierClassifier with Count Vectorizer is 0.803922230473233

Classification report is :
              precision    recall  f1-score   support

           0       0.80      0.19      0.31      4302
           1       0.50      0.02      0.04      2238
           2       0.81      0.99      0.89     23086

    accuracy                           0.80     29626
   macro avg       0.70      0.40      0.41     29626
weighted avg       0.78      0.80      0.74     29626
```

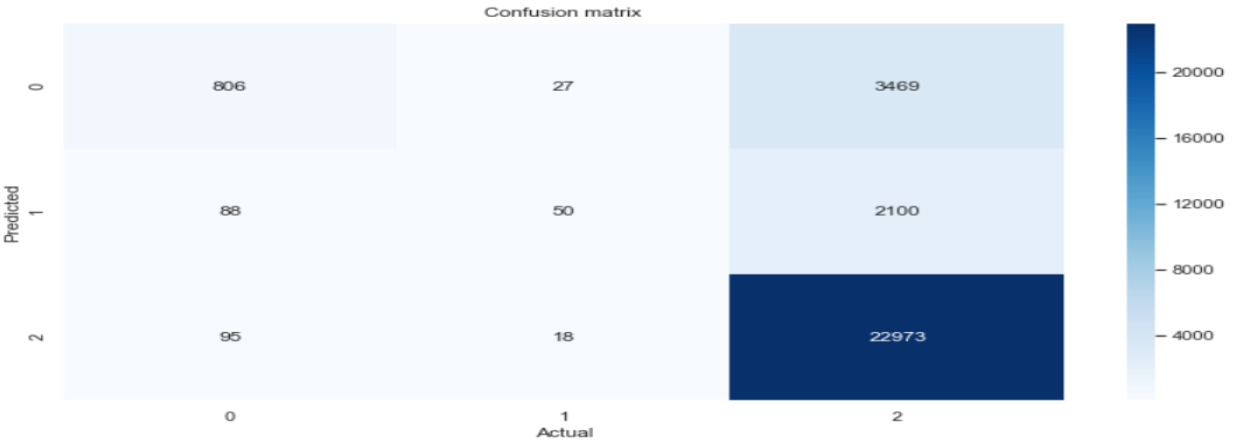



```
Train accuracy of XGBoost ClassifierClassifier with TF-IDF Vectorizer is 0.8099915611814346
Test  accuracy of XGBoost ClassifierClassifier with TF-IDF Vectorizer is 0.8043272800918112

Classification report is :
              precision    recall  f1-score   support

           0       0.81      0.19      0.30      4302
           1       0.53      0.02      0.04      2238
           2       0.80      1.00      0.89     23086

    accuracy                           0.80     29626
   macro avg       0.72      0.40      0.41     29626
weighted avg       0.79      0.80      0.74     29626
```



```
Train accuracy of XGBoost ClassifierClassifier with Word2Vec is 0.8107341772151899
Test  accuracy of XGBoost ClassifierClassifier with Word2Vec is 0.8056436913521906

Classification report is :
              precision    recall  f1-score   support

           0       0.71      0.25      0.37      4302
           1       0.36      0.00      0.00      2238
           2       0.81      0.99      0.89     23086

    accuracy                           0.81     29626
   macro avg       0.63      0.41      0.42     29626
weighted avg       0.76      0.81      0.75     29626
```

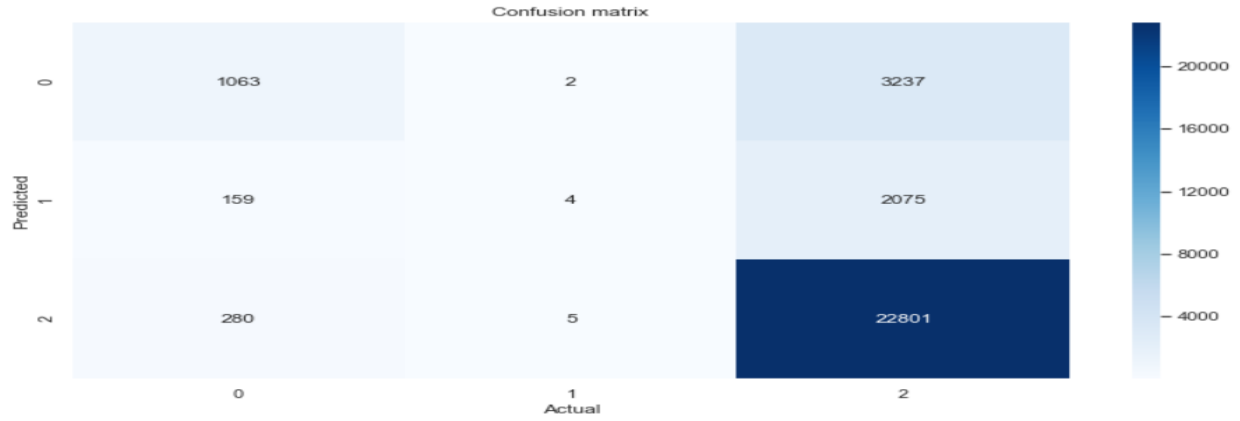


Table 4.1 Comparison of different classifiers with different embeddings

	Classifier	Word Embeddings	Train Accuracy	Test Accuracy	f1 score
0	Multinomial Naive Bayes	Count Vectorizer	0.815606	0.796193	[0.62, 0.3, 0.89]
1	Multinomial Naive Bayes	TF-IDF Vectorizer	0.826712	0.819179	[0.45, 0.02, 0.9]
2	Logistic Regression	Count Vectorizer	0.904585	0.842402	[0.64, 0.24, 0.92]
3	Logistic Regression	TF-IDF Vectorizer	0.868805	0.852562	[0.65, 0.17, 0.92]
4	Logistic Regression	Word2Vec	0.820805	0.820833	[0.5, 0.01, 0.9]
5	Random Forest	Count Vectorizer	0.991831	0.814352	[0.49, 0.06, 0.9]
6	Random Forest	TF-IDF Vectorizer	0.991977	0.817188	[0.48, 0.07, 0.9]
7	Random Forest	Word2Vec	0.991876	0.781206	[0.31, 0.04, 0.88]
8	XGBoost Classifier	Count Vectorizer	0.807966	0.803922	[0.31, 0.04, 0.89]
9	XGBoost Classifier	TF-IDF Vectorizer	0.809992	0.804327	[0.3, 0.04, 0.89]
10	XGBoost Classifier	Word2Vec	0.810734	0.805644	[0.37, 0.0, 0.89]

5.1.5 Logistic Regression with Oversampling

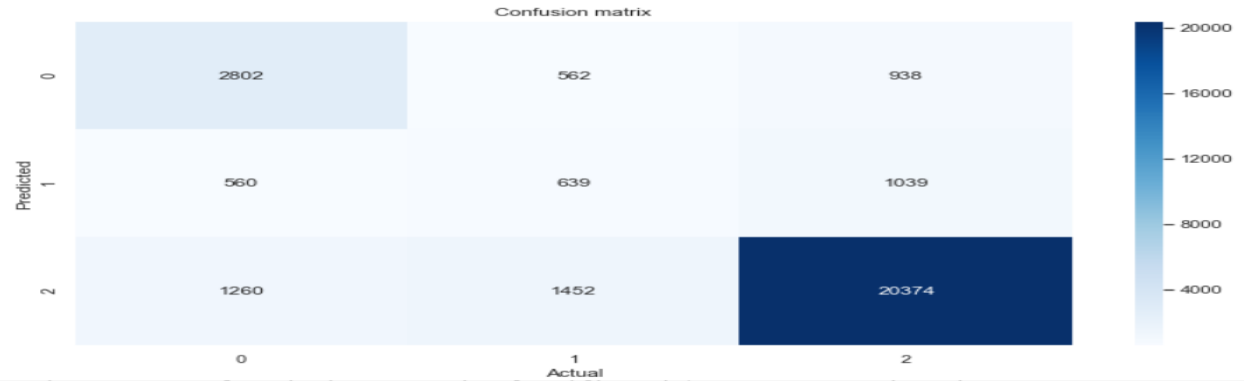
From the above table, we can see how different algorithms are performing with different word embeddings. As this dataset is highly imbalanced with very few neutral reviews, f1 score for that category is very less even though accuracy overall is decent. Generally, there are couple of techniques like Under sampling and Over sampling that can be used to overcome this issue. I am performing Over sampling with Logistic Regression which is best performed among all algorithms. I will be using Synthetic Minority Oversampling Technique (SMOTE) to perform over sampling.

```
Train accuracy of Logistic RegressionClassifier with Count Vectorizer is 0.7953660520476871
Test  accuracy of Logistic RegressionClassifier with Count Vectorizer is 0.8038547222034699

Classification report is :
              precision    recall  f1-score   support

           0       0.61      0.65      0.63      4302
           1       0.24      0.29      0.26      2238
           2       0.91      0.88      0.90     23086

    accuracy                           0.80     29626
   macro avg       0.59      0.61      0.60     29626
weighted avg       0.82      0.80      0.81     29626
```

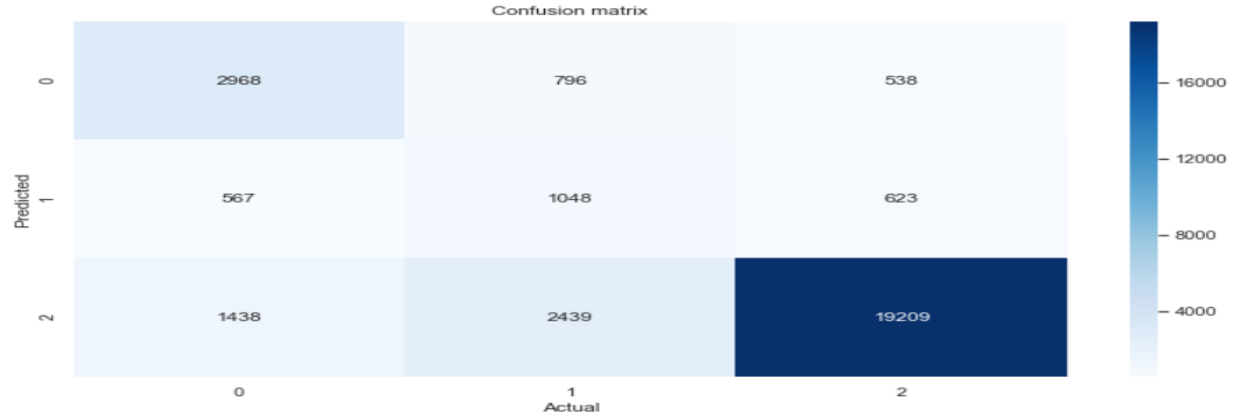



```
Train accuracy of Logistic RegressionClassifier with TF-IDF Vectorizer is 0.862170370263415
Test  accuracy of Logistic RegressionClassifier with TF-IDF Vectorizer is 0.7839397826233714

Classification report is :
              precision    recall  f1-score   support

           0       0.60      0.69      0.64      4302
           1       0.24      0.47      0.32      2238
           2       0.94      0.83      0.88     23086

    accuracy                           0.78     29626
   macro avg       0.59      0.66      0.62     29626
weighted avg       0.84      0.78      0.81     29626
```



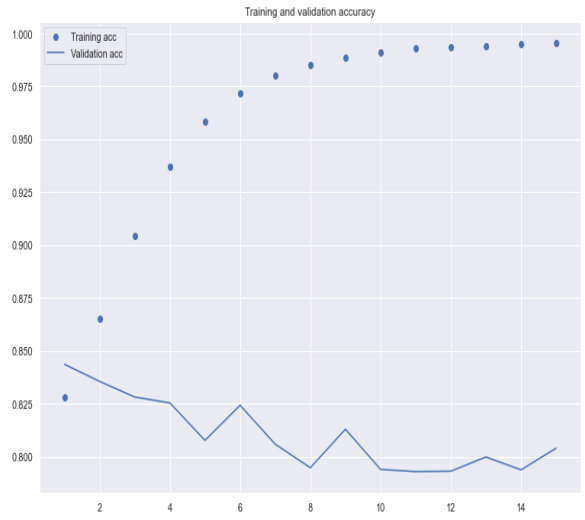
5.2 Deep Learning:

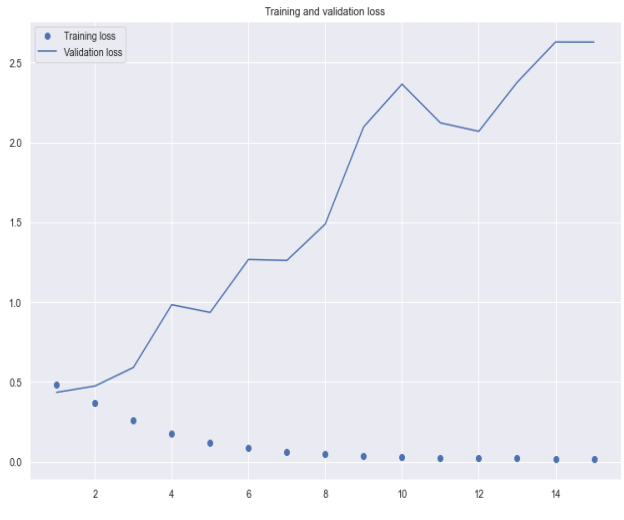
After using the machine learning algorithms, we will now explore more complicated deep learning algorithms. Word embeddings for this model can be built in couple of ways. We can learn embeddings while fitting the model or we can pre train word embeddings and directly feed them to embedding layer. We will apply prior method with Linear NN, LSTM and later with Text CNN.

5.2.1 Linear Neural Network:

We are training Linear Neural Network with an embedding layer, two hidden dense layers along with drop out layer which takes care of Overfitting. Model summary can be seen below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 100, 200)          10485800
_________________________________________________________________
flatten_2 (Flatten)          (None, 20000)             0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               2560128
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 195
=================================================================
Total params: 13,054,379
Trainable params: 13,054,379
Non-trainable params: 0
_________________________________________________________________
None
```





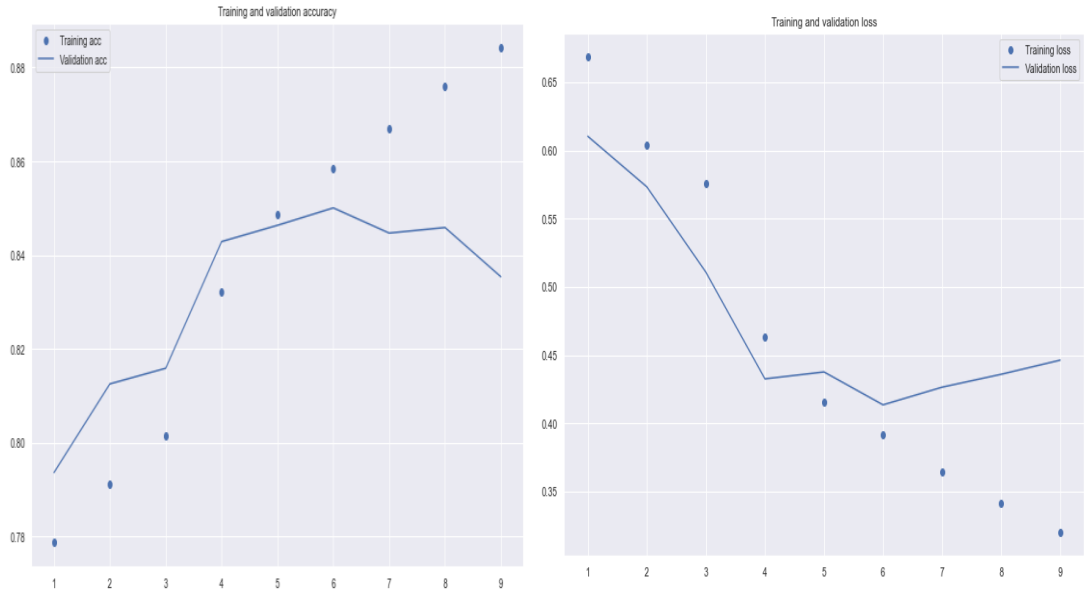
Training accuracy for Linear Neural Network is : 0.99
Test accuracy for Linear Neural Network is : 0.80

We can clearly observe that even after using dropout technique it is Overfitting.

5.2.2 Recurrent Neural Network (LSTM)

We are training LSTM model with an embedding layer, LSTM layer and 2 dense layers along with dropout layer. Model summary and accuracy/loss plots can be seen below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 100, 200)          10485800
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               168448
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 195
=================================================================
Total params: 10,662,699
Trainable params: 10,662,699
Non-trainable params: 0
_________________________________________________________________
None
```

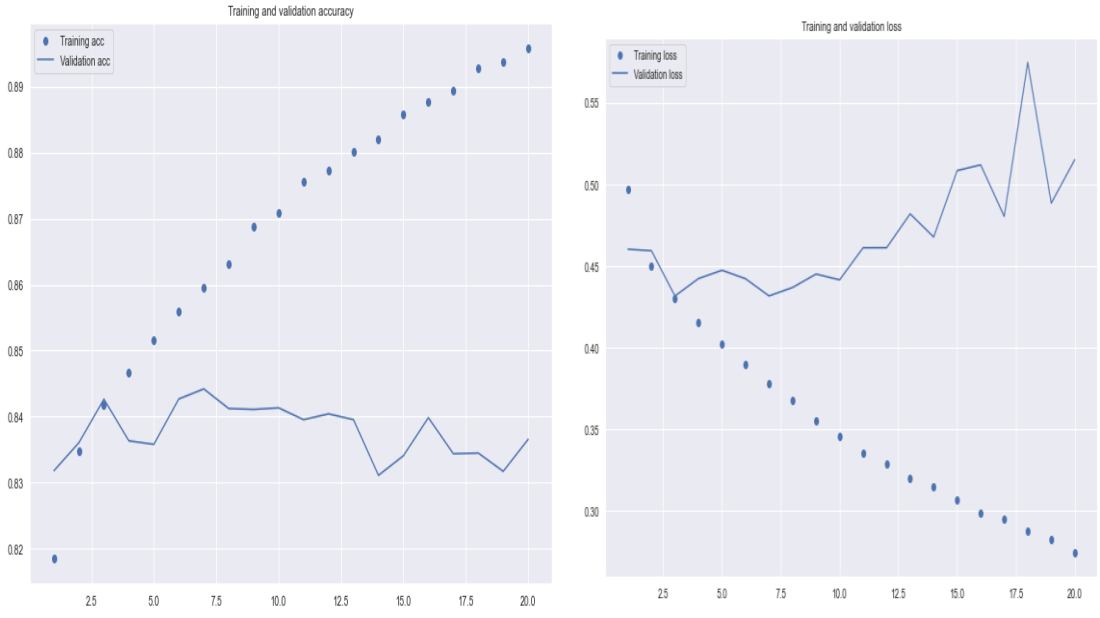


Training accuracy for Linear Neural Network is : 0.88
Test accuracy for Linear Neural Network is : 0.84

5.2.3 Convolutional Neural Network (Text CNN)

In this model, we train the word embeddings prior using Word2Vec function in genism. We add the embedding layer directly after converting the input to embedding matrix using above trained embeddings. To this, we add 2 CNN layers and finally one dense layer. Model summary and accuracy/loss plots can be seen below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 200)          10613200
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 94, 64)            89664
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 18, 64)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 18, 64)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 12, 32)            14368
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 32)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 99
=================================================================
Total params: 10,718,387
Trainable params: 105,187
Non-trainable params: 10,613,200
_________________________________________________________________
None
```



Training accuracy for Linear Neural Network is : 0.89
Test accuracy for Linear Neural Network is : 0.84

6 Conclusion:

In this project, we implemented multiple ML/DL algorithms for amazon reviews to carry out text sentiment classification and evaluated them against themselves and each other. We experimented with different sampling, embedding techniques, and utilized various visualizations to get insight of model performance.

We can observe that more complicated LSTM and Text CNN models are not giving any significant improvement over Logistic Regression with TF-IDF Vectorizer. This is because sentiment of the model is based on the usage of certain words rather the semantic or order in which words are used. These sophisticated deep learning algorithms with word embeddings will be extremely beneficial when we are performing tasks like Language Translation, Question answering etc.

From here, we can extend our project to doing hyperparameter tuning which cannot be done on local machine because of memory and capacity constraints. We can use the entire dataset and extend this project to predict numerical rating from 1 to 5 instead of 3 classes.

7 References:

- https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/
- https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras/
- https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456
- https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
- Deep Learning with Python – Book by François Chollet