Ratings Prediction

Submitted by-:
NITISH KUMAR SHARMA

ACKNOWLEDGEMENT

It is great pleasure for me to undertake this project. I feel overwhelmed doing this project entitled – “Ratings Prediction“.

Some of the resources that helped me to complete this project are as follows:

- Internet/web
- Stack overflow
- Analytics Vidhya
- Articles published in Medium.com
- Wikipedia

INTRODUCTION

BUSINESS PROBLEM FRAMING

We have a client who has a website where people write different reviews for technical products. Now they are adding a new feature to their website i.e. The reviewer will have to add stars(rating) as well with the review.

The rating is out 5 stars and it only has 5 options available 1 star, 2 stars, 3 stars, 4 stars, 5 stars. Now they want to predict ratings for the reviews which were wriiten in the past and they don't have rating.

So we, we have to build an application which can predict the rating by seeing the review.

CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM

Nowadays, a massive amount of reviews is available online. Besides offering a valuable source of information, these informational contents generated by users, also called User Generated Contents (UGC) strongly impact the purchase decision of customers. As a matter of fact, a recent survey (Hinckley, 2015) revealed that 67.7% of consumers are effectively influenced by online reviews when making their purchase decisions. More precisely, 54.7% recognized that these reviews were either fairly, very or absolutely important in their purchase decision making. Relying on online reviews has thus become a second nature for consumers.

People often rely on customer reviews of products before they buy online. These reviews are often rich in information describing the product. Customers often choose to compare between various products and brands based on whether an item has a positive or negative review. More often, these reviews act as a feedback mechanism for the seller. Through this medium, sellers strategize their future sales and product improvement.

REVIEW OF LITERATURE

The rapid development of Web 2.0 and e-commerce has led to a proliferation in the number of online user reviews. Online reviews contain a wealth of sentiment information that is important for many decision-making processes, such as personal consumption decisions, commodity quality monitoring, and social opinion mining. Mining the sentiment and opinions that are contained in online reviews has become an important topic in natural language processing, machine learning, and Web mining.

We have used different machine learning models to predict the above. Since we have categorical target data so classification model algorithms has been used.

We will start our project with the train dataset which contains categories of offensive words. We will observe all the features with following goals in mind:

- Relevance of the features
- Distribution of the features
- Data Cleaning of the features
- Visualization of the features

After having gone through all the features and cleaning the dataset, we will move on to machine learning classification modelling:

- Pre-processing of the dataset for models
- Testing multiple algorithms with multiple evaluation metrics
- Select evaluation metric as per our specific business application
- Doing hyper-parameter tuning using GridSearchCV for the best model
- Finally saving the best model
- Predicting with the test dataset

MOTIVATION FOR THE PROBLEM UNDERTAKEN

It’s a multi-label classification problem to solve which is a completely different project that I tried so far. So it will be exciting for me to get my hands on NLP Concepts which is required in this kind of projects.

Every day we come across various products through digital medium we swipe across hundreds of product choices under one category. It will be tedious for the customer to make selection. Here comes 'reviews' where customers who have already got that product leave a rating after using them and brief their experience by giving reviews.

Many product reviews are not accompanied by a scale rating system, consisting only of a textual evaluation. In this case, it becomes daunting and time-consuming to compare different products in order to eventually make a choice between them. Therefore, models able to predict the user rating from the text review are critically important.

Getting an overall sense of a textual review could in turn improve consumer experience.

ANALYTICAL PROBLEM FRAMING

MATHEMATICAL / ANALYTICAL MODELING OF THE PROBLEM

The dataset contains csv file: There are only two features that were scraped from e-commerce website that are ratings and reviews which were for different items like laptops, Phones, Headphones, smart watches, Professional Cameras, Printers, monitors, home theatre, router and printer.

In our scrapped dataset, our target variable "Rating " is a categorical variable i.e., it can be classified as ‘1.0’, ‘2.0’,’3.0’,’4.0’,’5.0’. Therefore, we will be handling this modelling problem as classification.

Fetched equal number of reviews for each rating. From an initial statistical overview of the dataset, we infer that our target variable is of multi label binary classification.

DATA SOURCES AND THEIR FORMATS

The data was in CSV format (Comma Separated Values). After reading the data by using function df=pd.read_csv (‘ ---file path --- ’) in jupyter notebook there were 24557 rows and 2 columns.

Dataset/Attributes Description :

1. Ratings: It is the Label column, which includes ratings in the form of integers from 1 to 5.
2. Full_review: It contains text data on the basis of which we have to build a model to predict ratings.

Dataset datatypes are as follow :

```
df.info() # to know datatype of each columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24557 entries, 0 to 24556
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Ratings      24557 non-null  int64 
 1   Full Review  24557 non-null  object
dtypes: int64(1), object(1)
memory usage: 383.8+ KB
```

Number of unique elements present in each column

```
# to count number of unique values in each columns
df.nunique()
```

```
Ratings            5
Full Review    15313
dtype: int64
```

Data Distribution also looks fine

```
# to get high understanding of dataset or to get overview/stats of the dataset
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Ratings	24557.0	3.991611	1.4039	1.0	3.0	5.0	5.0	5.0

DATA PREPROCESSING DONE

Imported RegEx (regular expression) to identify sequence of characters that specifies a search pattern, installed nltk library for words classification, tokenizing, stemming, parse tree visualization etc and wordcloud for visualizing loudwords used in the comments.

```
def text_cleaner(text):
    clean_text = re.sub(r'@[A-Za-z0-9]+','',text)
    clean_text = re.sub('#','',clean_text)
    clean_text = re.sub(r"'s\b",'',clean_text)
    clean_text = re.sub(r'[%$#@&}{]','',clean_text)
    clean_text = re.sub(r'[.,:;!]','',clean_text)
    letters_only = re.sub("[^a-zA-Z]",' ',clean_text)

    lower_case = letters_only.lower()
    tokens = [w for w in lower_case.split() if not w in stop_words]
    clean_text=''
    for i in tokens:
        clean_text = clean_text + lemmatizer.lemmatize(i)+ ' '
    return clean_text.strip()
```

- Converting all Full Reviews feature texts into lower case
- Removed punctuation, stopwords, digits from Full Reviews column
- Using Lemmatizing for converting a word to its base form

```
cleaned_text=[]
for i in df['Full Review']:
    cleaned_text.append(text_cleaner(i))
```

```
df['Cleaned_Full Reviews'] = cleaned_text
```

- After doing all data cleaning on Full Reviews column, then storing it to a empty list named cleaned text and then storing to a dataframe cleaned full reviews.

```
# dropping original column
df=df.drop(columns='Full Review')
```

- Dropping column Full Review as it is not useful for further prediction.

HARDWARE / SOFTWARE REQUIREMENTS AND TOOLS USED

- Anaconda Navigator 1.10.0
- Jupyter Notebook 6.1.4 , Python 3

Libraries

```
import pandas as pd # for handling dataset
import numpy as np  # for mathematical computation

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
                            plot_roc_curve, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier

from sklearn.decomposition import PCA
from scipy.stats import skew

# for visualization
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import style
import seaborn as sns
import pickle

import warnings
warnings.filterwarnings('ignore')
```

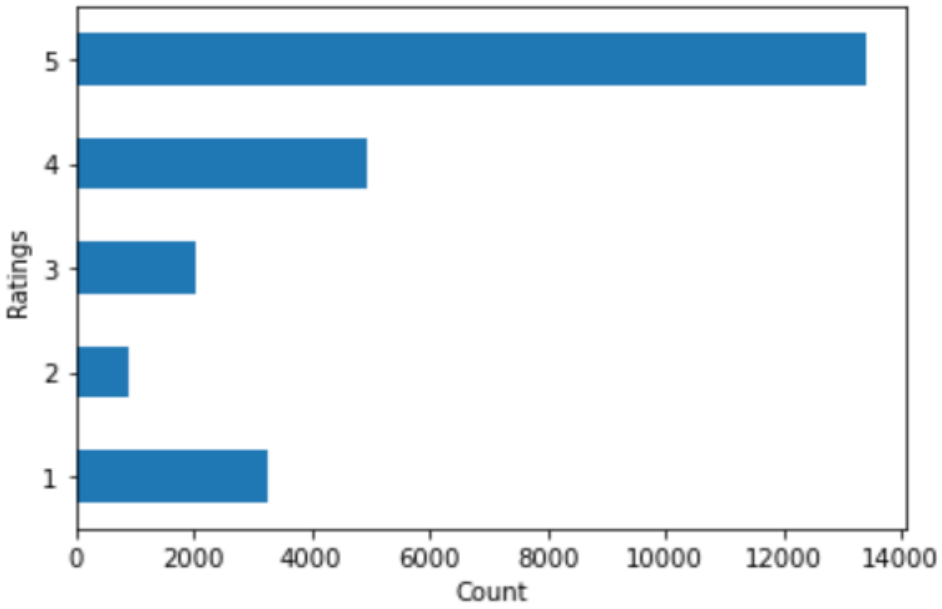
Libraries and Packages used:

- Pickle for saving & loading machine learning model.
- GridSearchCV for Hyper-parameter tuning.
- Cross validation score to cross check if the model is overfitting or not.
- Seaborn and matplotlib for visualization.
- Wordcloud to visualize sense of loud words used
- Lemmatizing for converting words to its base form

MODEL/s DEVELOPMENT AND EVALUATION

IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS)

```
df.groupby('Ratings')['Ratings'].count().plot(kind='barh')
plt.xlabel('Count')
plt.show()
```



- From the above we can observe that the dataset was highly imbalanced so we have used SMOTETomek to balance the dataset.

TESTING OF IDENTIFIED APPROACHES (ALGORITHMS)

Algorithms used:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- KNN Classifier
- Support Vector Machines
- Gradient Boosting Classifier
- Stochastic Gradient Descent

RUN AND EVALUATE SELECTED MODELS

1.Logistic Regression

```
log_reg = LogisticRegression()
log_reg.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
y_pred = log_reg.predict(x_test)
```

```
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

```
0.9289551155944359
```

```
# Confusion Matrix
conf_mat =confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[32881,  3013],
       [ 2079, 33700]], dtype=int64)
```

```
print('\n----------------Classification Report--------------------')
print (classification_report(y_test,y_pred,digits=2))
```

```
----------------Classification Report--------------------
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     35894
           1       0.92      0.94      0.93     35779

    accuracy                           0.93     71673
   macro avg       0.93      0.93      0.93     71673
weighted avg       0.93      0.93      0.93     71673
```

Accuracy score : 68.23%

2. Decision Tree Classifier

```
dt_clf = DecisionTreeClassifier()
dt_clf.fit(x_train,y_train)
pred=dt_clf.predict(x_train)
dt_clf_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{dt_clf_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#********************** Test Score ************************

pred = dt_clf.predict(x_test)
dt_clf_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{dt_clf_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 74.90% Test Result : 73.28%

3. Random Forest Classifier

```
rand_clf = RandomForestClassifier(random_state=101)
rand_clf.fit(x_train,y_train)
pred=dt_clf.predict(x_train)
rand_clf_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{rand_clf_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#********************** Test Score ************************

pred = rand_clf.predict(x_test)
rand_clf_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{rand_clf_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 82.38% Test Result : 79.62%

4. KNN Classifier

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
pred=knn.predict(x_train)
knn_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{knn_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#********************** Test Score ************************

pred = knn.predict(x_test)
knn_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{knn_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 71.79% Test Result : 69.20%

5. Support-Vector Machines

```
svc = SVC(kernel = 'rbf',C=1)
svc.fit(x_train,y_train)
pred=svc.predict(x_train)
svc_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{svc_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#********************** Test Score ************************

pred = svc.predict(x_test)
svc_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{svc_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 78.04% Test Result : 73.58%

6. Gradient Boosting Classifier

```
gbdt_clf = GradientBoostingClassifier()
gbdt_clf.fit(x_train,y_train)
pred=gbdt_clf.predict(x_train)
gbdt_clf_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{gbdt_clf_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#********************* Test Score ************************

pred = gbdt_clf.predict(x_test)
gbdt_clf_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_____________________________________________")
print(f"CLASSIFICATION REPORT:\n{gbdt_clf_report}")
print("_____________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 61.06% Test Result : 58.70%

7. Stochastic Gradient Descent

```
sgd=SGDClassifier(loss='modified_huber',shuffle=True,random_state=101)
sgd.fit(x_train,y_train)
y_pred = sgd.predict(x_test)
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

Training Result : 69.86% Test Result : 64.95%

KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION

Precision: It is the ratio between the True Positives and all the Positives. It can be seen as a measure of quality, higher precision means that an algorithm returns more relevant results than irrelevant ones

Recall : It is the measure of our model correctly identifying True Positives. It is used as a measure of quantity and high recall means that an algorithm returns most of the relevant results.

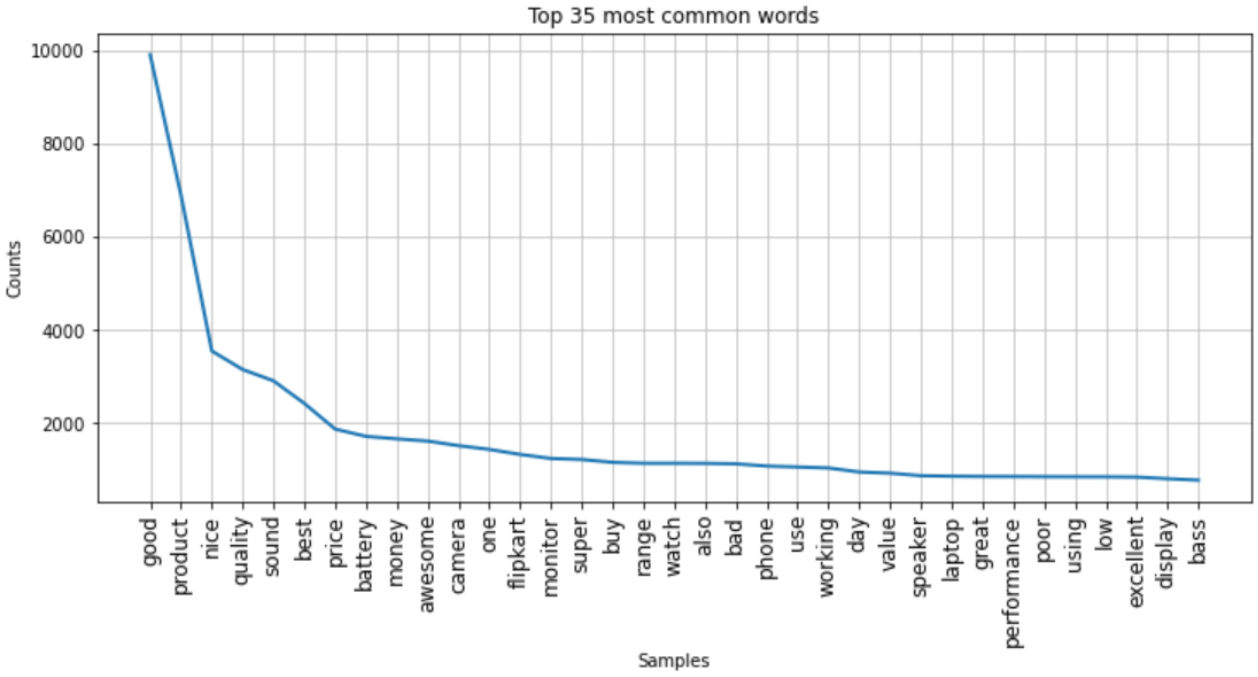
Accuracy score : It is the ratio of the total number of correct predictions and the total number of predictions. It is used when the True Positives and True negatives are more important. Accuracy can be used when the class distribution is similar

F1-score : It is used when the False Negatives and False Positives are crucial. Hence F1-score is a better metric when there are imbalanced classes.

Cross_val_score : To run cross-validation on multiple metrics and also to return train scores, fit times and score times. Get predictions from each split of cross- validation for diagnostic purposes. Make a scorer from a performance metric or loss function.

roc _auc _score : ROC curve. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0

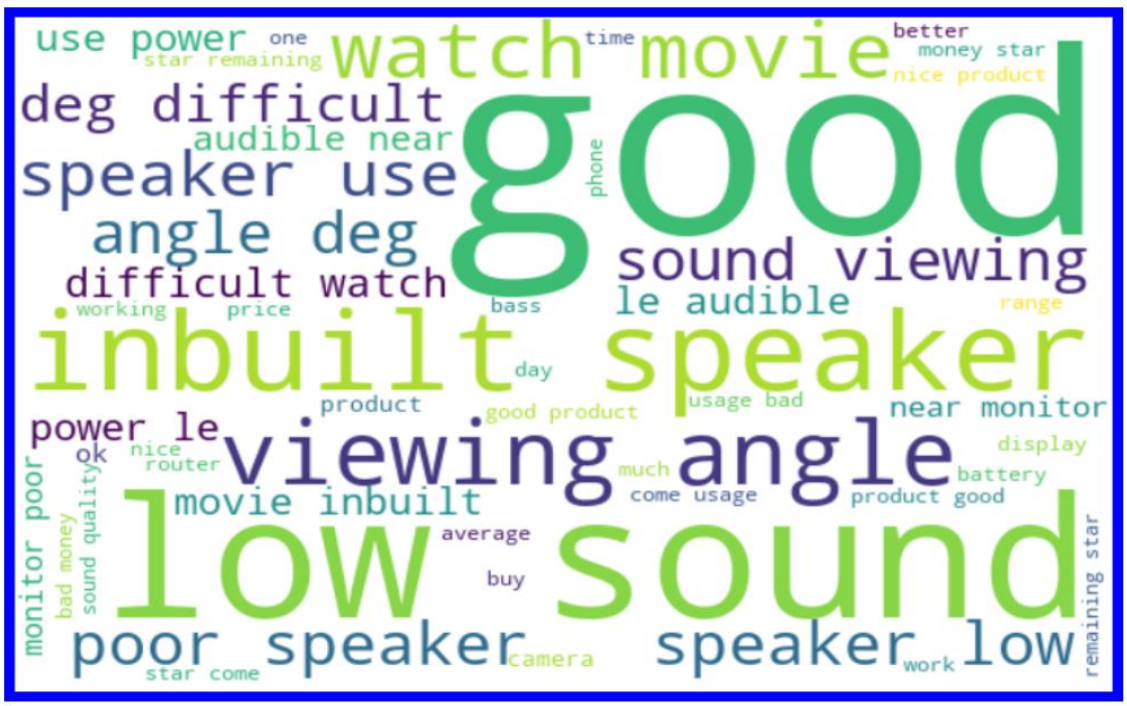
VISUALIZATION



- These are 35 most frequently used common words from the Cleaned_Full Reviews column according to its number of occurences in a descending order.

- Words appearing more frequently in Cleaned_Full Reviews column that is given rating five by consumers.

- Words appearing more frequently in Cleaned_Full Reviews column that is given rating four by consumers.



- Words appearing more frequently in Cleaned_Full Reviews column that is given rating three by consumers.

- Words appearing more frequently in Cleaned_Full Reviews column that is given rating two by consumers.

- Words appearing more frequently in Cleaned_Full Reviews column that is given rating one by consumers.

INTERPRETATION OF RESULTS

- Our model performs reasonably well on 1,4 and 5-star ratings and relatively bad on 2 and 3-star ratings.
- Ratings 5: Words like perfect, nice, best, excellent, good has been mostly used.
  Ratings 4: Words like good, nice, best, better has been mostly used
  Ratings 3: Words like average, low, poor, difficult has been mostly used
  Ratings 2: Words like slow, poor, barely, problem has been mostly used
  Ratings 1: Words like bad, disappointed, poor, one star has been mostly used
- The analysis also reveals that the user reviews are inconsistent with user numeric ratings, and that numeric ratings are higher than user reviews might suggest.
- Some of the reviews were bad and the texts had more wrong information about the product.
- Results demonstrate that tree-based bagging ensemble classifiers perform much better than boosting-based classifiers on account of their support for nonlinearity, colinearity, and tolerance to data noise.

CONCLUSION

KEY FINDINGS AND CONCLUSIONS OF THE STUDY

From the whole project evaluation these are the inferences that I could draw from the visualization of data.

- The numeric ratings given for products may be biased and overrated because higher ratings given by users potentially attract several new users disproportionately.
- However, the large increase in review-based data implies a need to focus also on performing predictions. This process is challenging yet fruitful, as user reviews are qualitative while ratings are essentially quantitative.
- WordCloud was not showing proper text which had more positive and negative weightage.
- User reviews are limited to identifying polarity and subjectivity.
- It is very difficult to model a customer's rating behaviour properly. The correctness of a model highly depends on what kind of information the dataset provides and the structure of the model.

LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE

Through Visualization it was clear that the target variable was a imbalanced dataset. Also the features that depicted a lot of story telling when visualization was done for all of them. Visualization gives meaning to a data which helps drawing inference from it.

This project allowed me to work with two different deep learning models and additionally, I was able to implement them on a Natural Language Processing use-case. The various data pre-processing and feature engineering steps in the project made me cognizant of the efficient methods that can be used to clean textual data.

Decision Tree was the best model to be deployed in production because its accuracy and CV value was least among all models.

Through this project we were able to learn various Natural language processing techniques like lemmatization, stemming, removal of stopwords.
This project has demonstrated the importance of sampling effectively, modelling and predicting data.

The dataset was highly imbalanced so we balanced the dataset using smote technique. We converted text data into vectors with the help of tfidf vectorizer.

LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK

However, our method will not be able to judge common affection of user in capturing the sentiment of review texts.

For further improvement of results, Recurrent neural networks offer extremely high performance on natural language processing problems, and if the architecture for the inference step were implemented efficiently the computational overhead would be minimal.

While we couldn’t reach out goal of maximum accuracy in Ratings prediction project, we did end up creating a system that can with some improvement and deep learning algorithms get very close to that goal.

As with any project there is room for improvement here. The very nature of this project allows for multiple algorithms to be integrated together as modules and their results can be combined to increase the accuracy of the final result. This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others. Once that condition is satisfied, the modules are easy to add as done in the code. This provides a great degree of modularity and versatility to the project.