2022

# A Report on Ratings Prediction

(A case Study.....)

Submitted to – Khusboo Garg
(SME of internship batch-28)


SUBMITTED BY- ABHISHEK BEHERA
DATA SCIENCE INTERN AT FLIP ROBO TECHNOLOGIES PVT LTD
INTERNSHIP BATCH-28

# Content

# INTRODUCTION

## Business Problem Framing

Websites and online stores increasingly rely on rating systems and interactive elements for visitors and customers. Users leave ratings on websites or give their opinions on products and companies using the comment boxes embedded on the page. The added value for users is clear. Customers and website visitors often gain important information through ratings and can read other user's experiences before investing in a product, service or a company. Since it's not possible to take a closer look at products online, these ratings and reviews fill information gaps. Online shopping is quite convenient, practical, time saving and fast. But nonetheless there's a distance between the provider and the customer. However, if a website contains ratings or a comment box, this can help to close the gap between the provider and the consumer. Customers can then use the feedback to help each other decide whether to go ahead with the purchase by providing information on the function, range and value of a product.


The rise in E-commerce has brought a significant rise in the importance of customer reviews. There are hundreds of review sites online and massive amounts of reviews for every product. Customers have changed their way of shopping and according to a recent survey 70 percent of customers say that they use rating filters to filter out low rated items in their searches. The ability to successfully decide whether a review will be helpful to other customers and thus give the product more exposure is vital to companies that support these reviews like Google, Amazon, Flipkart, Myntra, Reliance etc. There are two main methods to approach this problem. The first one is based on review text content analysis and uses the principles of natural language processing (the NLP method). This method lacks the insights that can be drawn from the relationship between costumers and items. The second one is based on recommender systems specifically on collaborative filtering and focuses on the reviewer's point of view.

## Conceptual Background of the Domain Problem

Rating prediction is a well-known recommendation task aiming to predict a user's rating for those items which were not rated yet by customers. Predictions are computed from users' explicit feedback i.e., their ratings provided on some items in the past. Another type of feedback are user reviews provided on

items which implicitly express users' opinions on items. Recent studies indicate that opinions inferred from users' reviews on items are strong predictors of user's implicit feedback or even ratings and thus, should be utilized in computation. As far as we know, all the recent works on recommendation techniques utilizing opinions inferred from users' reviews are either focused on the item recommendation task or use only the opinion information, completely leaving users' ratings out of consideration. The approach proposed in this project is filling this gap, providing a simple, personalized and scalable rating prediction framework utilizing both ratings provided by users and opinions inferred from their reviews. Experimental results provided on dataset containing user ratings and reviews from the real-world Amazon and Flipkart Product Review Data show the effectiveness of the proposed framework.

## Literature Review

**What is rating**?

Rating is a classification or ranking of someone or something based on a comparative assessment of their quality, standard or overall performance.

This project is more about exploration, feature engineering and classification that can be done on this data. Since we scrape huge amount of data that includes five stars rating, we can do better data exploration and derive some interesting features using the available columns.

We can categorize the ratings as: 1, 2, 3, 4 and 5 stars respectively.

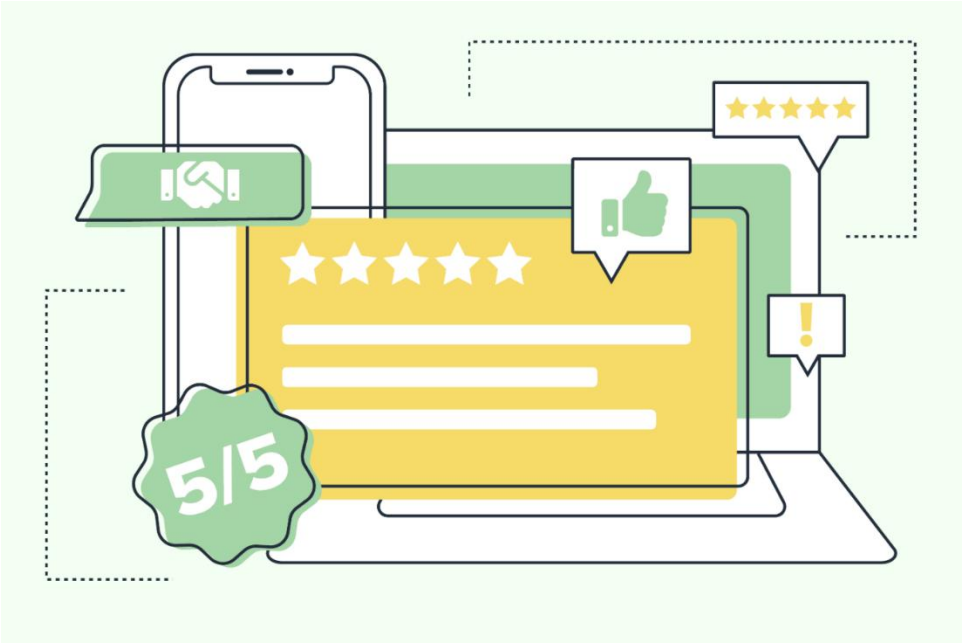
| Lower | Upper | Stars | Star Label |
|---|---|---|---|
| 1.0 | 1.2 | 1 | Bad |
| 1.3 | 1.7 | 1.5 | Bad |
| 1.8 | 2.2 | 2 | Poor |
| 2.3 | 2.7 | 2.5 | Poor |
| 2.8 | 3.2 | 3 | Average |
| 3.3 | 3.7 | 3.5 | Average |
| 3.8 | 4.2 | 4 | Great |
| 4.3 | 4.7 | 4.5 | Excellent |
| 4.8 | 5.0 | 5 | Excellent |

The goal of this project is to build an application which can predict the rating by seeing the review. In the long term, this would allow people to better explain and review their purchase with each other in this increasingly digital world.

**Motivation for the Problem**

Every day we come across various products in our lives, on the 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. As we know ratings can be easily sorted and judged whether a product is good or bad. But when it comes to sentence reviews, we need to read through every line to make sure the review conveys a positive or negative sense. In the era of artificial intelligence, things like that have got easy with the Natural Language Processing (NLP) technology.

Therefore, it is important to minimize the number of false positives our model produces, to encourage all constructive conversation. Our model also provides beneficence for the platform hosts as it replaces the need to manually moderate discussions, saving time and resources. Employing a machine learning model to predict ratings promotes easier way to distinguish between products qualities, costs and many other features.




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

As per the client's requirement for this rating prediction project I have scraped reviews and ratings from well-known e-commerce sites.

This is then saved into CSV format file. Also, I have shared the script for web scraping into the GitHub repository.

Then loaded this data into a data frame and did some of the important natural language processing steps and gone through several EDA steps to analyse the data. After all the necessary steps I have built an NLP ML model to predict the ratings.

In our scrapped dataset our target variable column "Ratings" is a categorical variable i.e., it can be classified as 1, 2, 3, 4 and 5 stars. Therefore, we will be handling this modelling problem as a multi-class classification project.

## Data Sources and their formats

This project is done in two parts:

- Data Collection Phase
- Model Building Phase

## Data Collection Phase:

You have to scrape at least 20000 rows of data. You can scrape more data as well, it's up to you. More the data better the model. In this section you need to scrape the reviews of different laptops, Phones, Headphones, smart watches, Professional Cameras, Printers, monitors, home theatre, router from different e-commerce websites.

Basically, we need these columns:

1) reviews of the product.

2) rating of the product.

Fetch an equal number of reviews for each rating, for example if you are fetching 10000 reviews then all ratings 1,2,3,4,5 should be 2000. It will balance our data set. Convert all the ratings to their round number as there are only 5 options for rating i.e., 1,2,3,4,5. If a rating is 4.5 convert it 5.

## Model Building Phase:

After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps involving NLP. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science. Include all the steps mentioned below:

1. Data Cleaning

2. Exploratory Data Analysis and Visualization

3. Data Pre-processing

4. Model Building

5. Model Evaluation

6. Selecting the Best classification model

We have collected the data from difference e-commerce websites like Amazon and Flipkart. The data is scrapped using Web scraping technique and the framework used is Selenium. I scrapped nearly 77550 records of the reviews data and saved it in a CSV format file.

## Data Pre processing

Checked the ratings column and it had 10 values instead of 5 so had to clean it through and ensure that our target label was updated as a numeric data type instead of the object data type value. It was Make sure that the string entries were replaced properly.

```
df['Ratings'] = df['Ratings'].replace('1.0 out of 5 stars',1)
df['Ratings'] = df['Ratings'].replace('2.0 out of 5 stars',2)
df['Ratings'] = df['Ratings'].replace('3.0 out of 5 stars',3)
df['Ratings'] = df['Ratings'].replace('4.0 out of 5 stars',4)
df['Ratings'] = df['Ratings'].replace('5.0 out of 5 stars',5)
df['Ratings'] = df['Ratings'].astype('int')
df['Ratings'].unique()
```

```
array([2, 3, 1, 5, 4])
```

```
# Now combining the "Review_title" and "Review_text" columns into one single column called "Review"
df['Review'] = df['Review_title'].map(str)+' '+df['Review_text']
df
```

```
'''Here I am defining a function to replace some of the contracted words to their full form and removing urls and some
unwanted text'''

def decontracted(text):
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"don't", "do not", text)
    text = re.sub(r"can't", "can not", text)
    text = re.sub(r"im ", "i am", text)
    text = re.sub(r"yo ", "you ",text)
    text = re.sub(r"doesn't", "does not",text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    text = re.sub(r"<br>", " ", text)
    text = re.sub(r'http\S+', '', text) #removing urls
    return text

# Lowercasing the alphabets
df['Review'] = df['Review'].apply(lambda x : x.lower())
df['Review'] = df['Review'].apply(lambda x : decontracted(x))

# Removing punctuations from the review
df['Review'] = df['Review'].str.replace('[^\w\s]','')
df['Review'] = df['Review'].str.replace('\n',' ')
```

```
# Removing all the stopwords
stop = stopwords.words('english')
df['Review'] = df['Review'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly identifying the intended part of speech and meaning of a word in a sentence. As well as within the larger context

surrounding that sentence such as neighbouring sentences or even an entire document. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

```
# Defining functiom to convert nltk tag to wordnet tags
def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

# Defining function to lemmatize our text
def lemmatize_sentence(sentence):
    # tokenize the sentence and find the pos_tag
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    # tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x : (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatize_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            lemmatize_sentence.append(word)
        else:
            lemmatize_sentence.append(lemmatizer.lemmatize(word,tag))
    return " ".join(lemmatize_sentence)

df['Review'] = df['Review'].apply(lambda x : lemmatize_sentence(x))
```

```
# Noise removal function
def scrub_words(text):
    # remove HTML markup
    text = re.sub("(<.*?>)", "", text)
    # remove non-ascii and digits
    text = re.sub("(\\W)", " ", text)
    text = re.sub("(\\d)", "", text)
    # remove white space
    text = text.strip()
    return text

df['Review'] = df['Review'].apply(lambda x : scrub_words(x))
```

```
# Creating column for word counts in the review text
df['Review_WC'] = df['Review'].apply(lambda x: len(str(x).split(' ')))
df[['Review_WC', 'Review']].head(10)
```

```
# Creating column for character counts in the review text
df['Review_CC'] = df['Review'].str.len()
df[['Review_CC','Review']].head(10)
```

```
# Applying zscore to remove outliers
z_score = zscore(df[['Review_WC']])
abs_z_score = np.abs(z_score)
filtering_entry = (abs_z_score < 3).all(axis = 1)
df = df[filtering_entry]
print("We have {} Rows and {} Columns in our dataframe after removing outliers".format(df.shape[0], df.shape[1]))
```

```
We have 67260 Rows and 6 Columns in our dataframe after removing outliers
```

Dealt with very lengthy comments that may have been detected as an outlier for our classification model.

## Data Inputs- Logic- Output Relationships

The libraries/dependencies imported for this project are shown below:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from wordcloud import WordCloud

# Importing nltk libraries
import re
import string
import missingno
import pandas_profiling
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)
from nltk import FreqDist
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import FreqDist

from scipy import stats
from scipy.stats import zscore
from scipy.sparse import hstack
import scikitplot as skplt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score, precision_score, confusion_matrix, accuracy_score, classification_report

from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost.sklearn import XGBClassifier

import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib
```

I have analysed the input output logic with word cloud and I have word clouded the reviews as per their ratings classification. A tag/word cloud is a novelty visual representation of text data, typically used to depict keyword metadata on websites or to visualize free form text. It's an image composed of words used in a particular text or subject, in which the size of each word indicates its frequency or importance.

Code:

```
# Getting insight of loud words in each rating
cols = 2
ratings = np.sort(df.Ratings.unique())
rows = len(ratings)//2
if len(ratings) % cols != 0:
    rows += 1
fig = plt.figure(figsize=(15,20))
plt.subplots_adjust(hspace=0.3)
p = 1
for i in ratings:
    word_cloud = WordCloud(height=800, width=1000, background_color="white", max_words=50).generate(' '.join(df.Review[df.R
    axis = fig.add_subplot(rows,cols,p)
    axis.set_title(f"WordCloud for Rating: {i}\n")
    axis.imshow(word_cloud)
    for spine in axis.spines.values():
        spine.set_edgecolor('r')
    axis.set_xticks([])
    axis.set_yticks([])

    plt.tight_layout(pad=5)
    p += 1
plt.show()
```

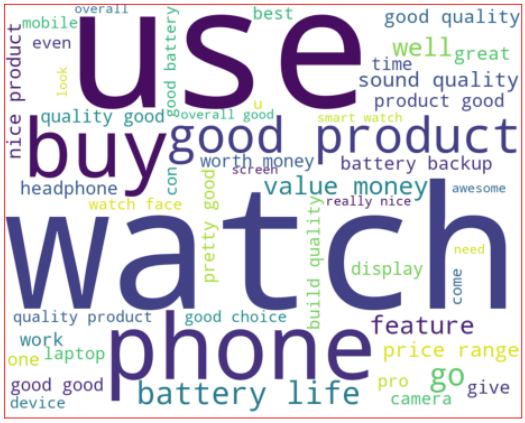
**Output:**

WordCloud for Rating: 1


WordCloud for Rating: 2


WordCloud for Rating: 3


WordCloud for Rating: 4




WordCloud for Rating: 5


These are the comments that belongs to different rating types so with the help of these word clouds we can see all the frequently used words in each and every ratings class. It is observed that 5-star rating comments have mostly positive words while the 1-star rating comments are loaded with negative descriptions.

**Set of assumptions (if any) related to the problem under consideration**

By looking into the target variable/label, we assumed that it was a multiclass classification type of problem. Also, we observed that our dataset was imbalance so we will have to balance the dataset for better prediction accuracy outcome.

The 5-star rating system allows respondents to rank their feedback on a 5-point scale from 1 to 5. The more stars that are selected, the more positively your customer is responding to the purchased products. People tend to overlook businesses with a less than four-star rating or lower. Usually, when people are researching a company, the goal is to find one with the highest overall score and best reviews. Having a five-star rating means a lot for your reputation score and acquiring new customers.

- Hardware and Software Requirements and Tools Used

  Hardware technology being used.

  RAM : 8 GB

  CPU : AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz

  GPU : AMD Radeon ™ Vega 8 Graphics and NVIDIA GeForce GTX 1650 Ti

  Software technology being used.

  Programming language : Python

  Distribution : Anaconda Navigator

  Browser based language shell : Jupyter Notebook

  Libraries/Packages specifically being used.

  Pandas, NumPy, matplotlib, seaborn, scikit-learn, pandas-profiling, missingno, NLTK

**Model/s Development and Evaluation**

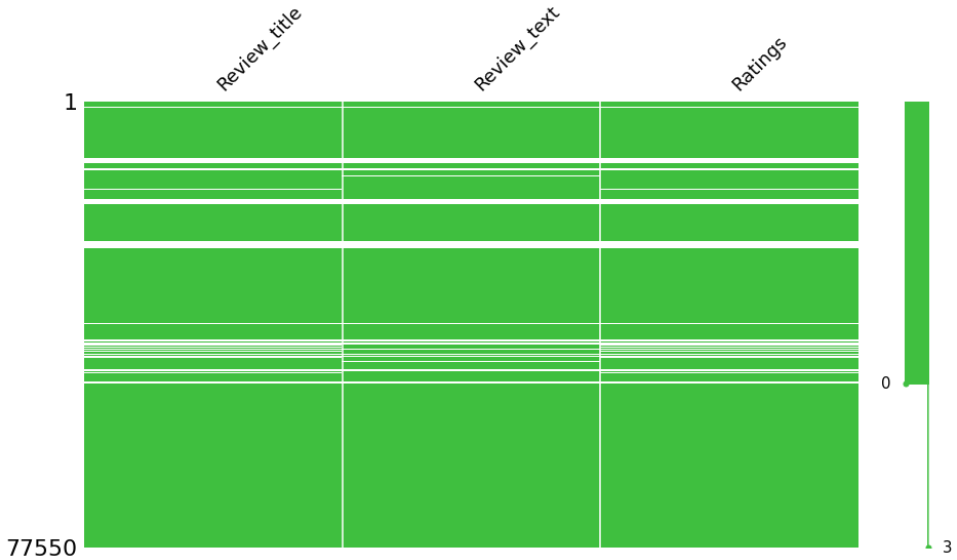
- Identification of possible problem-solving approaches (methods)

  Checked for missing values in our originally imported dataset and were able to notice NaN values in them.

```
df.isna().sum() # checking for missing values
```

```
Review_title    9029
Review_text     8085
Ratings         9027
dtype: int64
```

  Visual representation of the missing data in matrix format.




We dropped all the NaN values from our dataframe since we could afford to lose that much data.

```
print("We have {} Rows and {} Columns in our dataframe before removing NaN".format(df.shape[0], df.shape[1]))
df.dropna(inplace=True)
print("We have {} Rows and {} Columns in our dataframe after removing NaN".format(df.shape[0], df.shape[1]))
```

```
We have 77550 Rows and 3 Columns in our dataframe before removing NaN
We have 68294 Rows and 3 Columns in our dataframe after removing NaN
```

We then checked for the datatype details present in our dataframe. Using the info method, we are able to confirm the non-null count details as well as the datatype information. We noticed all the 3 columns showing as object datatype along with our target label.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68294 entries, 0 to 77549
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Review_title  68294 non-null  object
 1   Review_text   68294 non-null  object
 2   Ratings       68294 non-null  object
dtypes: object(3)
memory usage: 2.1+ MB
```

## Testing of Identified Approaches (Algorithms)

The complete list of algorithms that were used in training and testing the classification model are listed below:

1. Logistic Regression
2. Linear Support Vector Classifier
3. Random Forest Classifier
4. Bernoulli Naïve Bayes
5. Multinomial Naïve Bayes

6. Stochastic Gradient Descent Classifier
7. LGBM Classifier
8. XGB Classifier

From all of these above models Random Forest Classifier gave me good performance and I performed parameter tuning process to further improve the model confidence.

**Run and Evaluate selected models**

I created a classification function that included the evaluation metrics details for the generation of our Classification Machine Learning models. We defined the various classification models mentioned above and assigned them to user created variables. Then we defined the function that would train and predict the multiclass labels for us along with the evaluation metrics. I did not include the cross-validation part in the function and rather created a separate function for that metrics to evaluate only the best scored classification models amongst the original list. After calling the classification function we were able to obtain the accuracy score, classification report and confusion matrix details.

```
# Defining the Classification Machine Learning Algorithms
rf = RandomForestClassifier()
lr = LogisticRegression(solver='lbfgs')
svc = LinearSVC()
bnb = BernoulliNB()
mnb = MultinomialNB()
sgd = SGDClassifier()
lgb = LGBMClassifier()
xgb = XGBClassifier(verbosity=0)

# Creating a function to train and test the model with evaluation metrics
def BuiltModel(model):
    print('*'*30+model.__class__.__name__+'*'*30)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_train)
    pred = model.predict(x_test)
    accuracy = accuracy_score(y_test, pred)*100
    print(f"ACCURACY SCORE PERCENTAGE:", accuracy)
    # Confusion matrix and Classification report
    print(f"CLASSIFICATION REPORT: \n {classification_report(y_test, pred)}")
    print(f"CONFUSION MATRIX: \n {confusion_matrix(y_test, pred)}\n")
    print("-"*120)
    print("\n")
```

Output:

```
*****************************LogisticRegression*****************************
ACCURACY SCORE PERCENTAGE: 70.984230560087
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.74      0.78      0.76      1834
           2       0.63      0.62      0.62      1862
           3       0.62      0.64      0.63      1822
           4       0.70      0.70      0.70      1819
           5       0.86      0.81      0.84      1858

    accuracy                           0.71      9195
   macro avg       0.71      0.71      0.71      9195
weighted avg       0.71      0.71      0.71      9195

CONFUSION MATRIX:
 [[1428  282   90   26    8]
 [ 328 1158  281   76   19]
 [ 126  294 1158  203   41]
 [  27   82  264 1272  174]
 [  21   28   61  237 1511]]
```

- **Key Metrics for success in solving problem under consideration**

The key metrics used here were accuracy_score, cross_val_score, classification report, and confusion matrix. We tried to find out the best parameters and also to increase our scores by using Hyperparameter Tuning and we will be using GridSearchCV method.

**1. Cross Validation:**

Cross-validation helps to find out the over fitting and under fitting of the model. In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces where each part being 20% of full dataset. While running the Cross-validation the 1st part (20%) of the 5 parts will be kept out as a holdout set for validation and everything else is used for training data. This way we will get the first estimate of the model quality of the dataset.

In the similar way further iterations are made for the second 20% of the dataset is held as a holdout set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross-validation process to get the remaining estimate of the model quality.

```
# I am checking cross-validation score only for those algorithms which are giving us better accuracies

def cross_val(model):
    print('*'*30+model.__class__.__name__+'*'*30)
    scores = cross_val_score(model,train_features,y, cv = 3).mean()*100
    print("Cross validation score:", scores)
    print("\n")

for model in [lr,svc,sgd,rf,lgb,xgb]:
    cross_val(model)
```

```
******************************LogisticRegression******************************
Cross validation score: 70.163132137031


******************************LinearSVC******************************
Cross validation score: 69.88308863512779


******************************SGDClassifier******************************
Cross validation score: 69.7389885807504


******************************RandomForestClassifier******************************
Cross validation score: 70.29363784665578


******************************LGBMClassifier******************************
Cross validation score: 69.76345840130506


******************************XGBClassifier******************************
Cross validation score: 69.67917346383904
```

## 2. Confusion Matrix:

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e., commonly mislabelling one as another).

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

## 3. Classification Report:

The classification report visualizer displays the precision, recall, F1, and support scores for the model. There are four ways to check if the predictions are right or wrong:

a. TN / True Negative: the case was negative and predicted negative

b. TP / True Positive: the case was positive and predicted positive

c. FN / False Negative: the case was positive but predicted negative

d. FP / False Positive: the case was negative but predicted positive

**Precision:**

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class, it is defined as the ratio of true positives to the sum of a true positive and false positive. It is the accuracy of positive predictions. The formula of precision is given below:

Precision = TP/ (TP + FP)

**Recall:**

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. It is also the fraction of positives that were correctly identified. The formula of recall is given below:

Recall = TP/(TP+FN)

**F1 score:**

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy. The formula is:

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

**Support:**

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

**4. Hyperparameter Tuning:**

There is a list of different machine learning models. They all are different in some way or the other, but what makes them different is nothing but input parameters for the model. These input parameters are named as Hyperparameters. These hyperparameters will define the architecture of the model, and the best part about these is that you get a choice to select these for your model. You must select from a specific list of hyperparameters for a given model as it varies from model to model.

We are not aware of optimal values for hyperparameters which would generate the best model output. So, what we tell the model is to explore and select the optimal model architecture automatically. This selection procedure for hyperparameter is known as Hyperparameter Tuning. We can do tuning by using GridSearchCV.

GridSearchCV is a function that comes in Scikit-learn (or SK-learn) model selection package. An important point here to note is that we need to have Scikit-learn library installed on the computer. This function helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end we can select the best parameters from the listed hyperparameters.

**Visualizations**

I used the pandas profiling feature to generate an initial detailed report on my dataframe values. It gives us various information on the rendered dataset like the correlations, missing values, duplicate rows, variable types, memory size etc. This assists us in further etailed visualization separating each part one by one comparing and research for the impacts on the prediction of our target label from all the available feature columns.

```
pandas_profiling.ProfileReport(df)
```

Summarize dataset: 100% 20/20 [00:15<00:00, 1.77it/s, Completed]

Generate report structure: 100% 1/1 [00:03<00:00, 3.45s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.09s/it]

Pandas Profiling Report | Overview | Variables | Interactions | Correlations | Missing values | Sample

# Overview

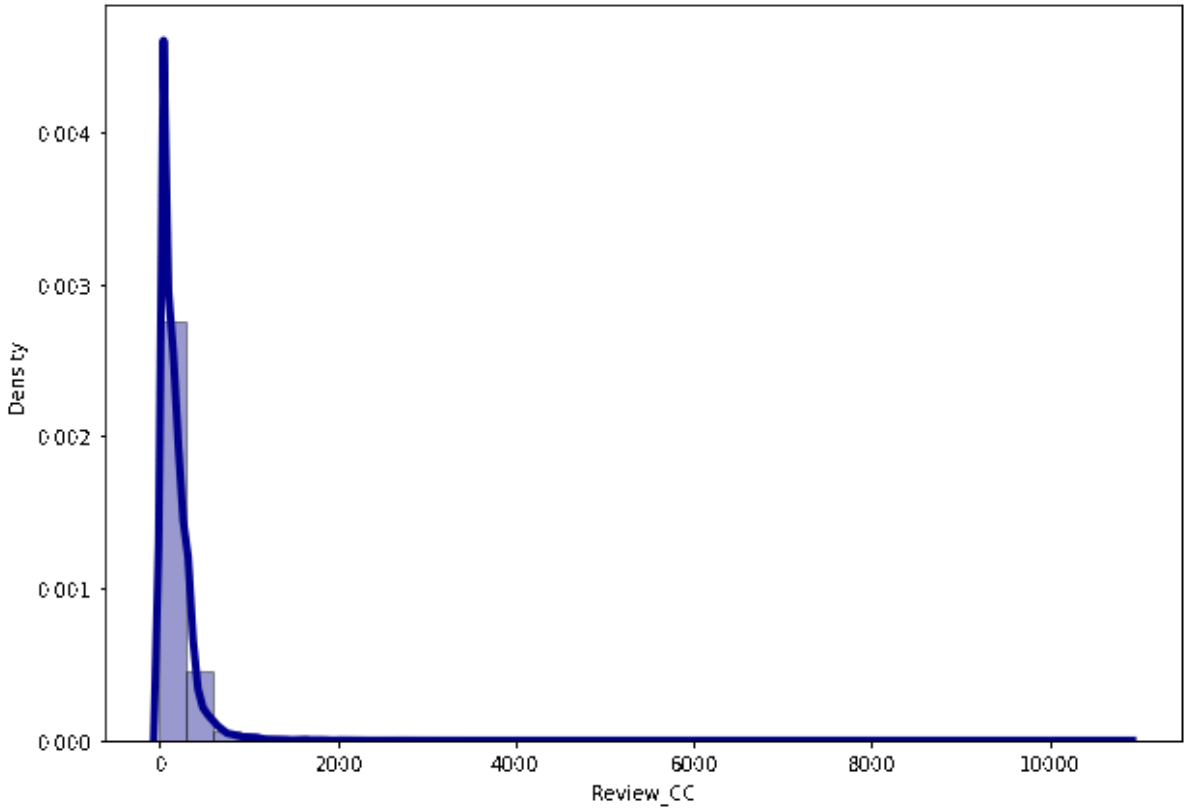
Overview | Warnings 12 | Reproduction

## Dataset statistics

| | |
|---|---|
| **Number of variables** | 7 |
| **Number of observations** | 67260 |
| **Missing cells** | 0 |
| **Missing cells (%)** | 0.0% |
| **Duplicate rows** | 0 |
| **Duplicate rows (%)** | 0.0% |
| **Total size in memory** | 3.3 MiB |
| **Average record size in memory** | 52.0 B |

## Variable types

| | |
|---|---|
| **Numeric** | 3 |
| **Categorical** | 4 |

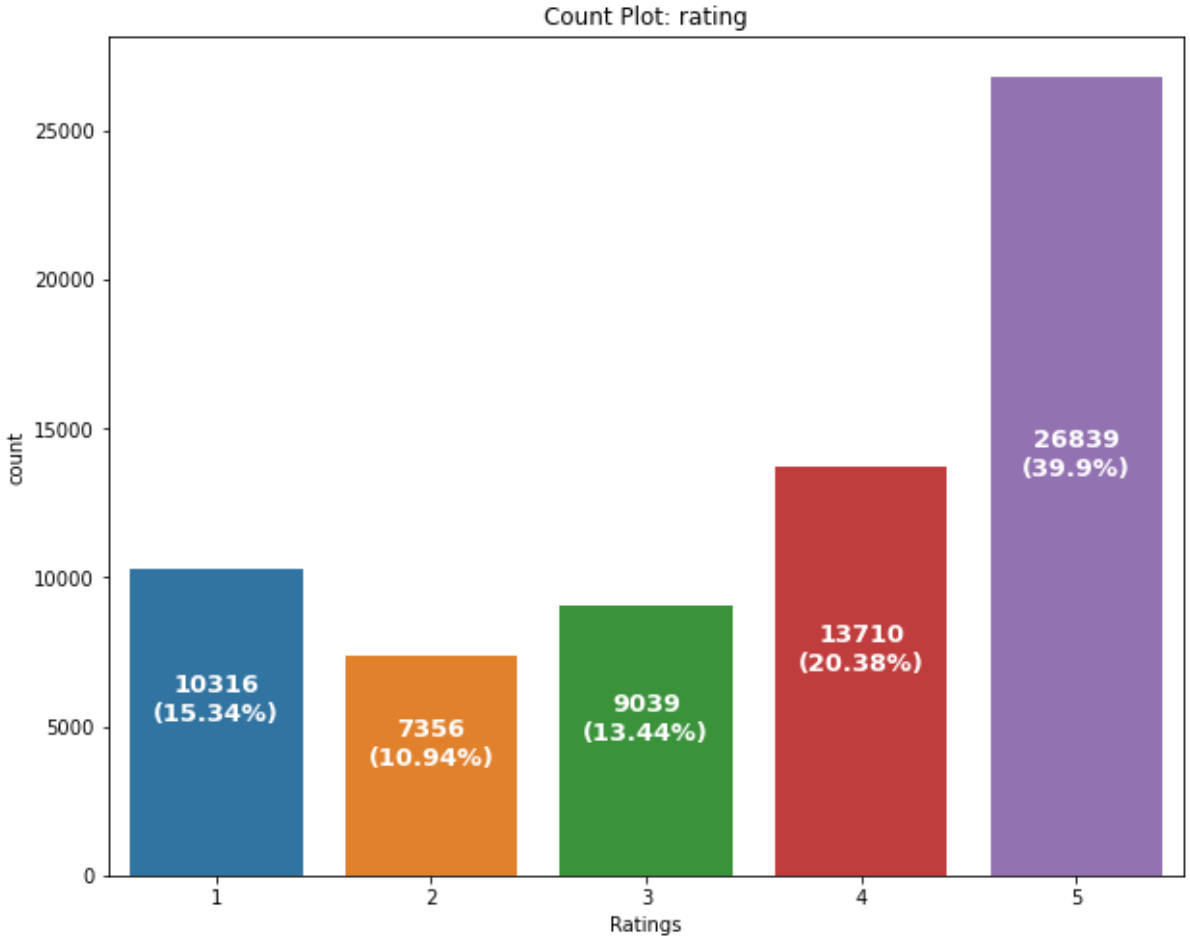



In the above histogram + distribution plots we can see the word count and character count visuals provide with density details and indicate towards the lengthy comments that may act as an outlier when used as an input for our classification model. Therefore, we will be removing the outliers accordingly.

**Code:**

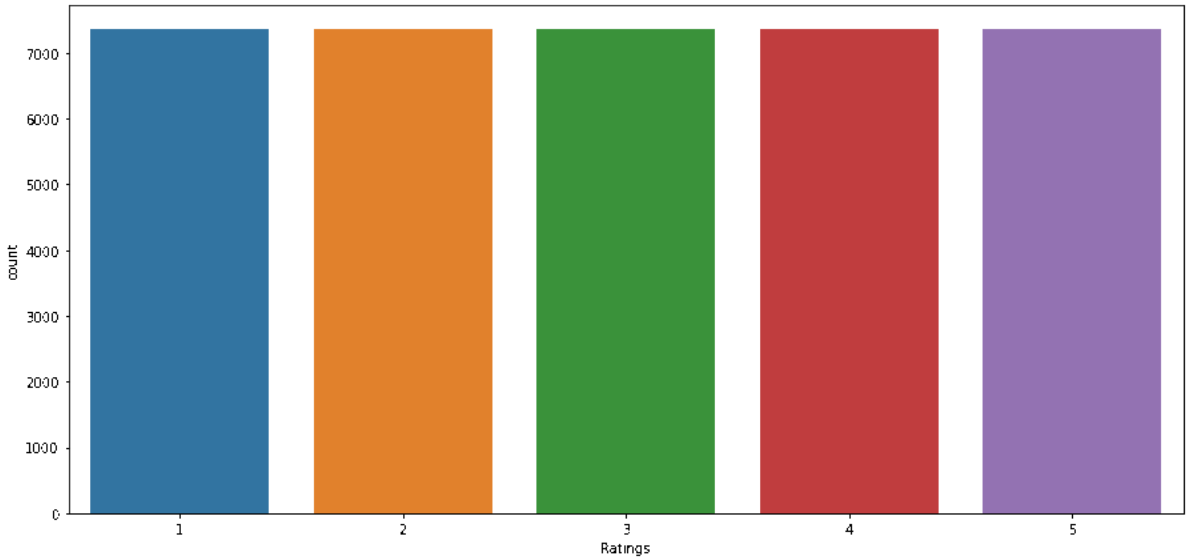
```
# Checking the ratings column details using count plot
x = 'Ratings'
fig, ax = plt.subplots(1,1,figsize=(10,8))
sns.countplot(x=x,data=df,ax=ax)
p=0
for i in ax.patches:
    q = i.get_height()/2
    val = i.get_height()
    ratio = round(val*100/len(df),2)
    prn = f"{val}\n({ratio}%)"
    ax.text(p,q,prn,ha="center",color="white",rotation=0,fontweight="bold",fontsize="13")
    p += 1

plt.title("Count Plot: rating")
plt.show()
```

Output:

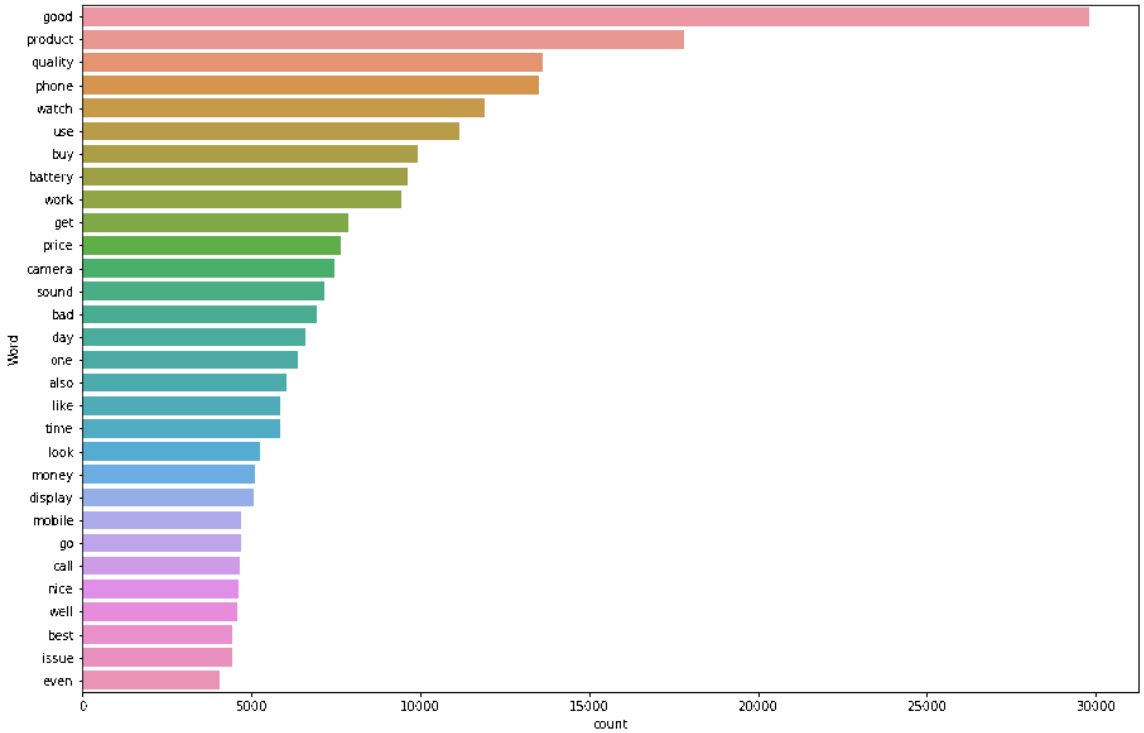



**Observation:**

We can see that the highest number of customer rating received are for 5 stars Then we have 4-star rating reviews present in our dataset However, we see a high 1-star rating as well compared to 2- and 3-star rating reviews

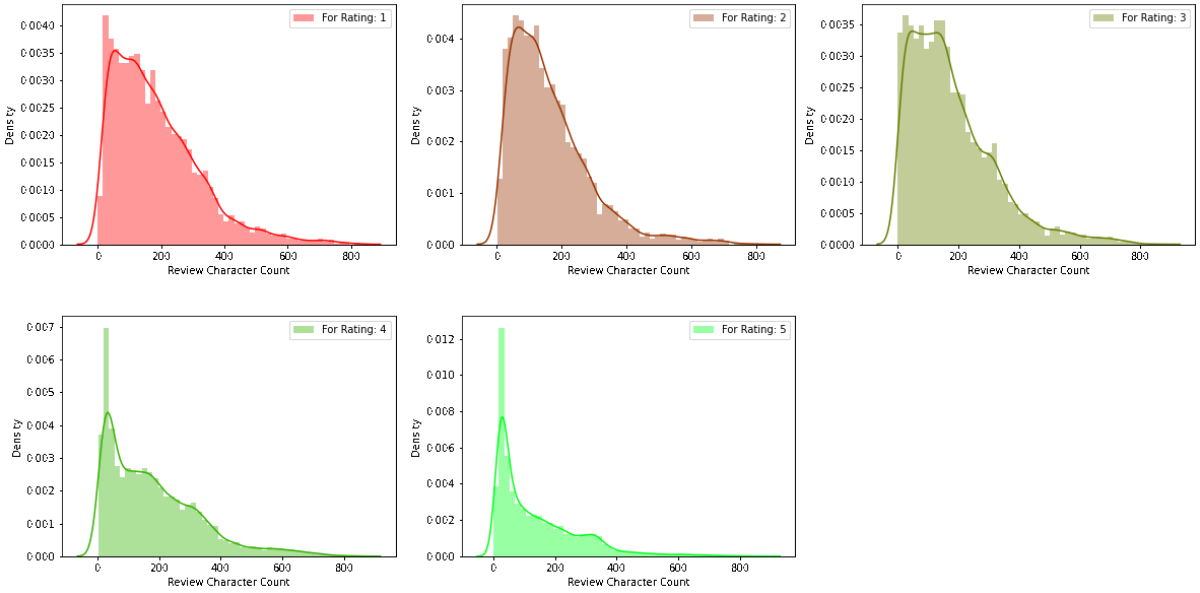



In the above plot we see how initially we had an imbalanced class concern that we later rectified by manually choosing the same number of records for each and every class and ensuring that the dataset get balanced multiclass label variable.




In this bar plot we have the most frequently and rarely used words.

Histogram-Distribution plots for Ratings column:




## Interpretation of the Results

Starting with univariate analysis, with the help of count plot it was found that the data consists of in equal amount for each rating (i.e., from 1 to 5). Moving further with the removal and replacement of certain terms (like punctuations, extra spaces, numbers, money symbols, smiley etc) as well as removal of stop words. It was evident that the length of review text decreases by a large amount. This was also depicted by using distribution plots and histograms. With the help of word cloud, it was found that rating 1 consists of words like waste, money, slow, worst, issue, horrible etc, rating 2 consists of words like problem, issue, bad, poor, slow etc, rating 3 consists of word like problem, bad, issue, slow, life, average, nice etc, rating 4 consists of word like good, value, money, nice, performance, great, better, wonderful etc and rating 5 consists of words like excellent, must buy, great, perfect, super, awesome, mind blowing etc.

# CONCLUSION

## Key Findings and Conclusions of the Study

This research evaluated the rating of a product classification using machine learning and deep learning techniques. Using real data, we compared the various machine learning algorithms' accuracy by performing detailed experimental analysis while classifying the text into 5 categories.

Generally, Random Forest Classification machine learning algorithms have shown a better performance with our real-life data than others, and the most performing models are all ensemble classifiers.

## Learning Outcomes of the Study in respect of Data Science

In this project we were able to learn various Natural Language Processing techniques like lemmatization, stemming, removal of Stop Words, etc. This project has demonstrated the importance of sampling effectively, modelling and predicting data. Through different powerful tools of visualization, we were able to analyse and interpret different hidden insights about the data. The few challenges while working on this project are:

1. Imbalanced Dataset.
2. Lots of Text data.

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

## Limitations of this work and Scope for Future Work

The future of sentiment analysis is going to continue to dig deeper, far past the surface of the number of likes, comments and shares, and aim to reach, and truly understand, the significance of social media interactions and what they tell us about the consumers behind the screens. This forecast also predicts broader applications for sentiment analysis – brands will continue to leverage this tool, and so will individuals in the public eye, governments, non-profits, education centres and many other domain organizations.

Sentiment analysis is getting better because social media is increasingly more emotive and expressive. A short while ago, Facebook introduced "Reactions," which allows its users to not just 'Like' content, but attach an emoticon, whether it be a heart, a shocked face, angry face, etc. To the average social media user, this is a fun, seemingly silly feature that gives him or her a little more freedom with their responses. But, to anyone looking to leverage social media data for sentiment analysis, this provides an entirely new layer of data that wasn't available before. Every time the major social media platforms update themselves and add more features, the data behind those interactions gets broader and deeper.

Negative reviews can carry as much weight as positive ones. One study found that 82% of those who read online reviews specifically seek out negative reviews. Research indicates that users spend five times as long on sites when interacting with negative reviews, with an 85% increase in conversion rate.

Customers like to see lots of reviews. A single review with a few positive words makes up an opinion, but a few dozen that say the same thing make a consensus. The more reviews, the better, and one study found that consumers want to see at least 40 reviews to justify trusting an average star rating. However, a few reviews are still better than no reviews.

## References:


1)https://www.google.com/

2) https://www.youtube.com/

3) https://scikit-learn.org/stable/user_guide.html

4) https://github.com/

5) https://www.kaggle.com/

6) https://medium.com/

7) https://towardsdatascience.com/

8) https://www.analyticsvidhya.com/

9) https://www.nltk.org/api/nltk.html

10) https://www.nltk.org/data.html


*********