

# Sentiment Analysis on Women's E-Commerce Clothing Reviews

## Group Members:

Asma Abid Karim - 19685

Faryal Tarique - 17852

Huda Mumtaz - 19702

# Table of Contents

Introduction.....	6
Problem Description.....	7
Data Description.....	8
Rows and Columns.....	8
Missing Values.....	8
Imbalance Class.....	9
Data Preprocessing.....	11
Feature Selection.....	12
Missing Values.....	12
Count Vectorizer.....	13
Tokenization and Lowercasing.....	14
Removing Stop Words.....	14
Removing Punctuation.....	15
Lemmatization.....	15
Text Mining.....	16
Model Building.....	17
Multinomial Naive Bayes.....	18
Attempt 1.....	18
Attempt 2.....	19
Attempt 3.....	20
Attempt 4.....	21
Attempt 5.....	21

Attempt 6.....	23
Attempt 7.....	24
Attempt 8.....	25
Random Forest.....	27
Attempt 1.....	27
Attempt 2.....	28
Attempt 3.....	28
Attempt 4.....	29
Attempt 5.....	31
Attempt 6.....	31
Attempt 7.....	32
Attempt 8.....	32
Attempt 9.....	32
Attempt 10.....	33
Attempt 11.....	35
Decision Tree.....	36
Attempt 1.....	36
Attempt 2.....	37
Attempt 3.....	38
Attempt 4.....	39
Attempt 5.....	40
Attempt 6.....	42
Attempt 7.....	43

Gradient Boost.....	44
Attempt 1.....	44
Attempt 2.....	44
Attempt 3.....	45
K Nearest Neighbours.....	46
Attempt 1.....	46
Attempt 2.....	46
Attempt 3.....	47
Attempt 4.....	47
Attempt 5.....	48
Attempt 6.....	48
Attempt 7.....	49
Attempt 8.....	49
Attempt 9.....	50
Attempt 10.....	50
Attempt 11.....	51
Model Selection.....	53
Best Attempts.....	53
Multinomial Naive Bayes.....	53
Random Forest.....	53
Decision Tree.....	53
Gradient Boosting.....	53
K Nearest Neighbours.....	54

Best Working Model.....	54
Findings.....	56
Useful Insights.....	57
Limitations of the Study.....	56
Future Improvements.....	58
Deployment Plan.....	58
Model Maintenance.....	59
Conclusion.....	61

# Introduction

The Digital Age is a rapid epochal movement away from conventional industries to an economy based mostly on digitalization. Digital transformation isn't a buzzword, it's a necessary component of remaining in business. Every business owner wants a piece of the ecommerce pie, and those that don't keep up swiftly go out of business. Since every other firm is moving online, companies need powerful marketing strategies. This can only be completely achieved if the data gathered from all activities or transactions is transformed into something useful and valuable.

Data mining is a field that plays a significant part in this. For ecommerce, data from customer reviews and web browsing may be studied to reveal patterns in human behavior on that platform. Text classification and sentiment analysis is one of the most significant aspects of data mining that can be utilized to get useful insights from e-commerce platform reviews and ratings. This knowledge may then be used to make specific marketing decisions that have the best likelihood of increasing revenue.

# Problem Description

The Kaggle dataset used for this project contains a large number of customer reviews for products of an e-commerce store. Sentiment analysis of these product reviews can be very helpful for the e-commerce store to further analyze the customer engagements, improve their products in the future, and devise better marketing strategies. Further analysis can assist in determining which customer subsets are most likely to make repeated purchases, or which section of the customer base prefers which category of products.

Manually predicting the sentiments of such a large number of reviews can be significantly hectic and time consuming. Hence, the project uses machine learning and a classifier to accurately predict the sentiments of the reviews and provide useful insights about the data. Various classification models were used with different configurations to choose the best classifier with a reliable accuracy and reasonable computation time.

# Data Description

## Rows and Columns:

The dataset used contains Women's clothing e-commerce reviews and was retrieved from Kaggle.

<https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews>

This dataset includes 23486 rows and 10 feature variables. Each row corresponds to a customer review, and includes the variables:

Clothing ID: Integer variable that is treated as a categorical variable that relates to specific item review.

Age: An integer variable greater than zero that refers to the reviewer's age.

Title: Review title that is a string variable.

Review Text: Review body written by the customer as a string variable.

Rating: Positive Ordinal Integer variable for the product score granted by the customer from 1 Worst, to 5 Best.

Recommended IND: Binary type variable for whether the customer recommends (1) or does not recommend (0) the reviewed product.

Positive Feedback Count: An integer count referring to the number of people who agree with the customer review.

Division Name: Categorical name of the product high level division.

Department Name: Categorical name of the product department name.

Class Name: Categorical name of the product class name.

## Missing Values:

Missing values were checked in each column.



```
#Checking for null values
train.isnull().sum()
```

```
Unnamed: 0          0
Clothing ID         0
Age                 0
Title              3810
reviewText         845
overall rating      0
Recommended IND     0
Positive Feedback Count  0
Division Name       14
Department Name     14
Class Name          14
dtype: int64
```

The number of missing values amounted to 4697. The dataset had about 2.0% missing values.

### Imbalance Class:

To find out the total number of positive reviews (represented by 1) and the total number of negative reviews (represented by -1), we ran the following code.

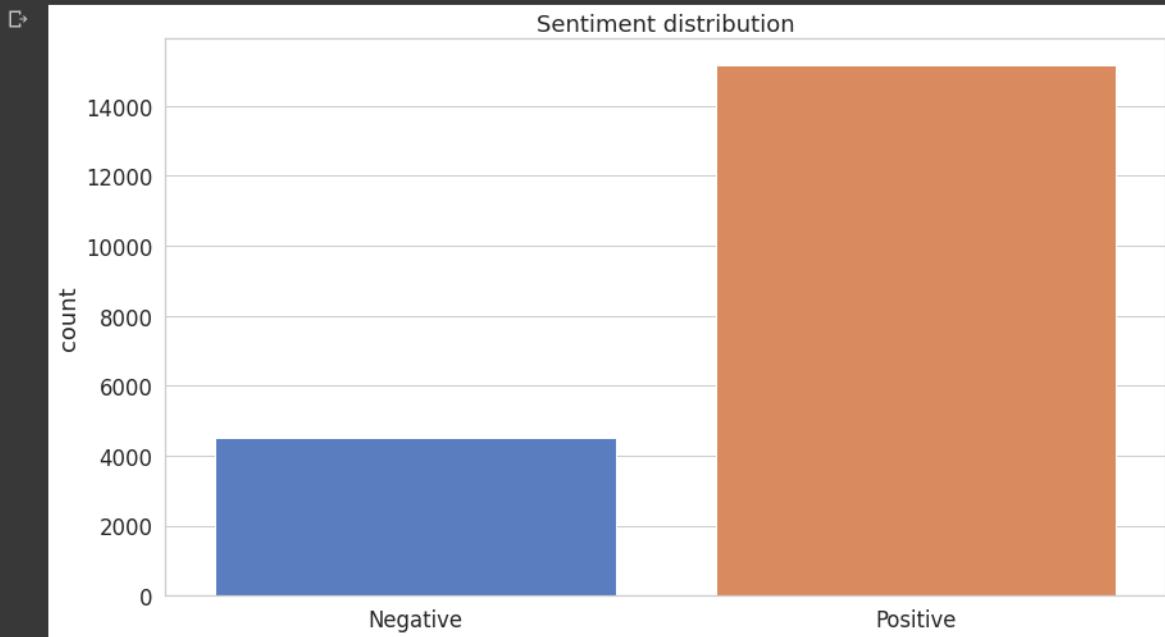
```
train['sentiment'].value_counts()

1      15160
-1      4515
Name: sentiment, dtype: int64
```

It turned out that there were 10,645 more positive reviews than negative reviews. The dataset had an imbalance class problem.

The imbalance is more visible in the graph below.

```
def SentimentDist(X):  
    Graph = sns.countplot(x='sentiment', data=X)  
    Graph.set_title("Sentiment distribution")  
  
    Graph.set_xticklabels(['Negative', 'Positive'])  
    plt.xlabel("");  
  
SentimentDist(train)
```



To deal with this problem, we decided to use F1 score instead of accuracy as the evaluation metric. Higher F1 score indicates better model, indicating good precision and recall.

# Data Preprocessing

## Feature Selection:

We made our own class variable in the dataset using the column 'overall rating'. If the overall rating given by the user was equal to 4 or 5, the review was positive, otherwise the review was negative (overall rating of value 1, 2 or 3).

```
[ ] def sentimentchecker(row):  
  
    '''This function returns sentiment value based on the overall ratings from the user'''  
  
    if row['overall rating'] == 1.0 or row['overall rating'] == 2.0 or row['overall rating'] == 3.0:  
        val = -1 # 'Negative'  
    elif row['overall rating'] == 4.0 or row['overall rating'] == 5.0:  
        val = 1 # 'Positive'  
    else:  
        val = -2  
    return val  
  
train['sentiment'] = train.apply(sentimentchecker, axis=1)  
train.head()
```

We saved the class variable ('negative' or 'positive') in a column called 'Sentiment'. The dataset had 12 columns now. Only two columns, reviewText and title from the original dataset seemed important for the training dataset. This is because the goal was to find whether a review was positive or negative based solely on the text of the review and the title, any other information was irrelevant for the data model. The column 'title' was equally important in the dataset as 'reviewText'. This is because a lot of users had written important information in the 'title', so 'title' could not be ignored.

```
[ ]  
train['reviews']=train['reviewText']+train['Title']  
train=train.drop(['reviewText', 'Title'], axis=1)  
  
train.head()
```

In the above piece of code, we merged 'title' and 'reviewText' in a column which we named as 'reviews', after which we removed 'reviewText' and 'title' from train, leaving only 'reviews', the new column, in the training data set.

However, we did not completely remove the other columns from the original dataset, we selected only two columns, 'reviews' and 'sentiment' to go into our training model, rest of the information in the data set was necessary to help us analyze and understand the dataset, that further helped us in formulating marketing strategies for the e-commerce website.

```
X = train['reviews'].values  
y = train['sentiment'].values
```

## Missing Values:

It was important to handle missing values in the dataset. First, we wrote a code to find out whether we had any missing values in the dataset or not.

```
#Checking for null values  
train.isnull().sum()
```

Unnamed: 0	0
Clothing ID	0
Age	0
Title	3810
reviewText	845
overall rating	0
Recommended IND	0
Positive Feedback Count	0
Division Name	14
Department Name	14
Class Name	14
dtype: int64	

This piece of code displayed the total number of missing values in each feature. Because only reviewText, title and overall rating are selected to go into the training data set, we decided to handle the missing values in these columns only. There were several ways to handle the missing values, some of which were:

- Remove the row that has the missing value
- Fill the row with the majority in the bag of words
- Fill the row with N/A and treat it as a review



Overall rating did not have any missing values. It was decided that the rows missing values in reviewText or title be removed. The missing values were about 6.61% of our dataset. Removing them was the most convenient option in this case as there was an uncertainty that the other options mentioned above would have destroyed the context of the reviewText, defeating the purpose of our project. Removing them did not affect the size or the content of the dataset significantly.

```
#removing only those rows that have null in the reviews and title columns  
train=train.dropna(axis=0, subset=['reviewText', 'Title'])
```

```
train.isnull().sum()
```

Unnamed: 0	0
Clothing ID	0
Age	0
Title	0
reviewText	0
overall rating	0
Recommended IND	0
Positive Feedback Count	0
Division Name	13
Department Name	13
Class Name	13
dtype: int64	

## Count Vectorizer:

Countvectorizer was used to tokenize the dataset. It separated the documents into words and converted it into a vector on the basis of term frequency in the document.

Count vectorizer was imported from python sklearn library.

```
from sklearn.feature_extraction.text import CountVectorizer
```

```

input=train['reviews'].values
#input=[input]

cnt_vectorizer = CountVectorizer() #Try with diff
#cnt_vectorizer = CountVectorizer(ngram_range=(1,
features = cnt_vectorizer.fit_transform(input)
features_nd = features.toarray()
print(features_nd)

```

## Tokenization and Lowercasing:

Tokenization is the technique used to separate text documents into individual words and save them in a vector. A document can either be a sentence or a paragraph. In this dataset, the reviews column is tokenized into words.

```

def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

train['reviews'] = train['reviews'].apply(cleaningData)

```

To make up for our bag of words, the reviews column in the train data is passed to the function cleaningData where it is tokenized through word\_tokenize which is from the python library nltk and is converted to lowercase from .lower() function. Each word is classified into a token, which includes all alphabets between two spaces.

## Removing stop words:

Stop words refer to meaningless words, such as 'the', 'a', 'an', 'in'. We remove them to reduce our processing time and storage space. We used python nltk library to remove stop words.

```

from nltk.corpus import stopwords

stop_words = stopwords.words('english')

```

```
#Removing Stopwords
removeStopWords = [t for t in removePunct if t not in stop_words]
```

## Removing Punctuation:

To clean our dataset by removing punctuation marks, we had to download ‘punkt’ package from nltk and apply it as follows.

```
nltk.download("punkt")
```

```
#Punctuation Eraser
removePunct = [w for w in tokens if w.isalpha()]
```

## Lemmatization:

Both stemming and lemmatization are techniques used to bring a word to its root form. The only difference between stemming and lemmatization is that stemming can make a root form that might not have a meaning while lemmatization will always make a meaning root form that can be found in a dictionary.

For example:

For the words trouble, troubling and troubled

The root form from stemming will be ‘troubl’ while the root form from lemmatization will be ‘trouble’.

Because lemmatization brings in the actual root word preserving its context, we decided to use lemmatization in our project. This technique was also imported from python nltk library.

```
nltk.download('wordnet')
```

```
from nltk.stem import WordNetLemmatizer
```

```
#lemmatizing
text = [WordNetLemmatizer().lemmatize(t) for t in removeStopWords]
```

## Text Mining:

We combined the above mentioned text mining techniques to clean our data more efficiently.

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
nltk.download("punkt")

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    removeStopWords = [t for t in removePunct if t not in stop_words]

    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in removeStopWords]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()
```

First, the whole data lowercased and tokenized. Then punctuations and stop words were removed.



# Model Building

We used models from built in python libraries. In all the models, we partitioned 75% of the data into training dataset and 25% of the data into testing dataset.

```
▶ y = train['sentiment'].values  
  
X_train, X_test, y_train, y_test = train_test_split(features_nd, y, test_size=0.25, train_size=0.75, random_state=1234)
```

Following are the classification models we used:

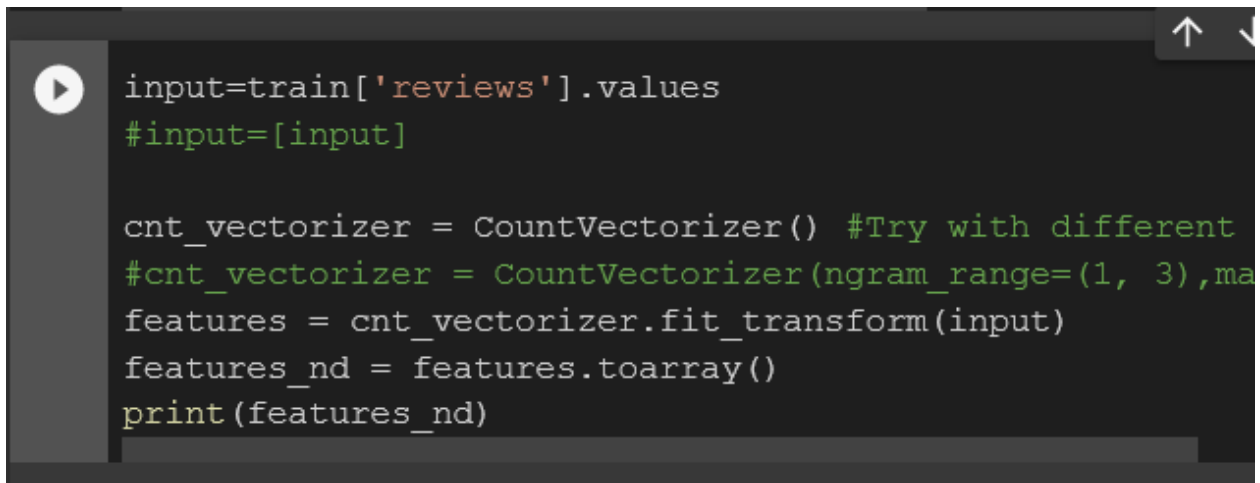
- Multinomial Naive Bayes
- Random Forest
- Decision Tree
- Gradient Boosting
- K Nearest Neighbours

## Multinomial Naive Bayes:

### Attempt 1:

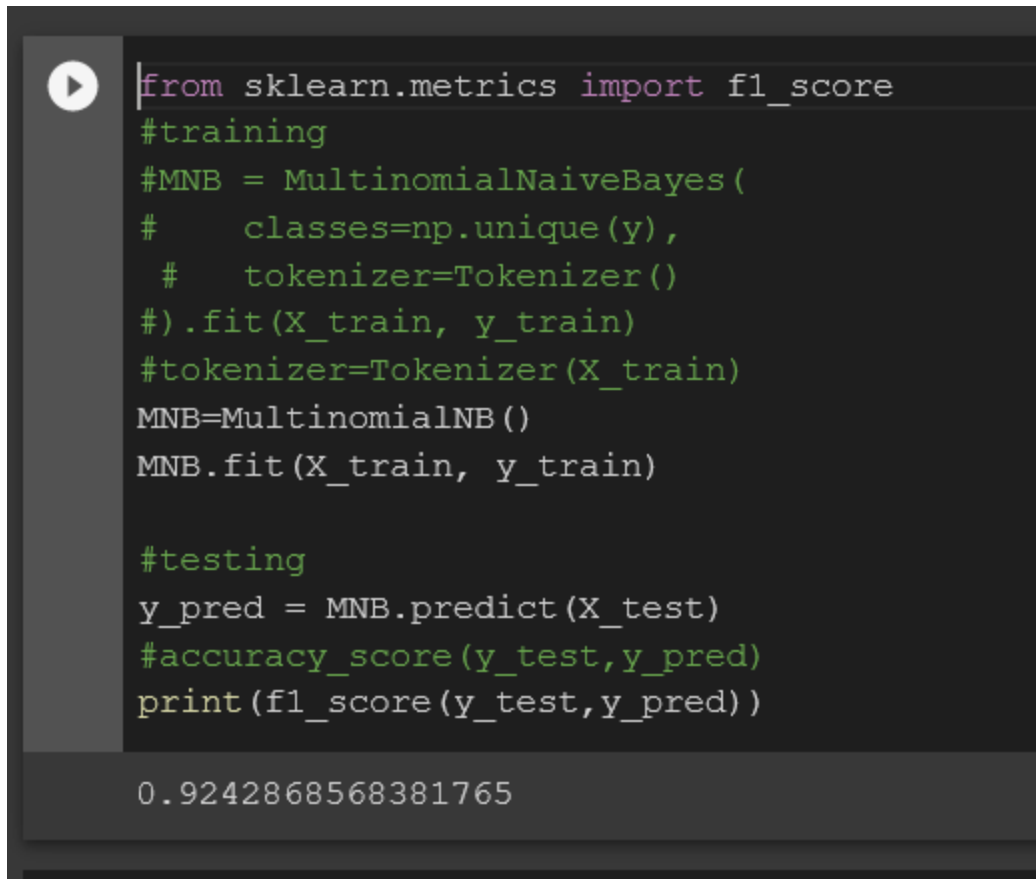
We preprocessed data only by removing missing values, count vectorization and feature selection as mentioned above in the document, under the heading of data preprocessing.

Multinomial naive bayes classifier was used by importing it from sklearn python library and default features for countvectorizer were used.



```
input=train['reviews'].values
#input=[input]

cnt_vectorizer = CountVectorizer() #Try with different
#cnt_vectorizer = CountVectorizer(ngram_range=(1, 3),ma
features = cnt_vectorizer.fit_transform(input)
features_nd = features.toarray()
print(features_nd)
```



```
from sklearn.metrics import f1_score
#training
#MNB = MultinomialNaiveBayes(
#    classes=np.unique(y),
#    tokenizer=Tokenizer()
#).fit(X_train, y_train)
#tokenizer=Tokenizer(X_train)
MNB=MultinomialNB()
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))
```

0.9242868568381765

Encoder was used because sklearn multinomial naive bayes takes in float values while our data was in the form of text, so it was necessary to convert it to float using encoder.

We ended up with an F1 score of about 92.43% with a computation time of 20s on Google Colab.

Attempt 2:

With the same data preprocessing settings as attempt 1, the only difference in attempt 2 was that we set the smoothing parameter in multinomial naive bayes to 0.

```
MNB=MultinomialNB(alpha = 0)
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/naiv
% _ALPHA_MIN
0.8925029750099166
```

F1 score of 89.3 % was achieved with a computation time of 14s on Google Colab.

Attempt 3:

With the same data preprocessing as in previous attempts, we changed the value of alpha in the multinomial naive bayes classifier to 0.5 now.

```
#naive bayes attempt 3
MNB=MultinomialNB(alpha = 0.5)
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))
```

```
0.9202603742880392
```

F1 score was greater than attempt 2 but lesser than attempt 1. Computation time was 17 s on Google Colab.

Attempt 4:

With the same data preprocessing as in previous attempts, we changed the value of alpha in the multinomial naive bayes classifier to 0.5 now.

```
#naive bayes attempt 3
MNB=MultinomialNB(alpha = 0.5)
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))

0.9202603742880392
```

F1 score of 92.03 % was observed, which was lesser than attempt 1 but greater than attempts 2 and 3, which led to the observation that our dataset classifies better with larger values of alpha in multinomial naive bayes classifier.

Computation time was 15s on Google Colab.

Attempt 5:

In this attempt, we included text mining in the data preprocessing techniques. The dataset cleaned after text mining was then passed to encoder and then passed to the multinomial naive bayes classifier with default settings.

```

import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('wordnet')

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    removeStopWords = [t for t in removePunct if t not in stop_words]

    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in removeStopWords]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```

from sklearn.metrics import f1_score
#training
#MNB = MultinomialNaiveBayes(
#    classes=np.unique(y),
#    tokenizer=Tokenizer()
#).fit(X_train, y_train)
#tokenizer=Tokenizer(X_train)
#naive bayes attempt 1
MNB=MultinomialNB()
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))

0.922539489671932

```

F1 score observed was about 92.25%. This was lesser than attempt 1 but greater than attempts 2, 3 and 4. Computation time was 24s on Google Colab.

Attempt 6:

With the same data preprocessing techniques as in attempt 5, we changed the fit prior in naive bayes classifier to false from default true.

```

MNB=MultinomialNB(alpha = 1, fit_prior=False)
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))

0.9168875721191331

```

Decrease in F1 score from attempt 5 was observed with a computation time of 24s on Google colab.

Attempt 7:

Experimented with removing removePunction (keeping the punctuation in the dataset), but the F1 score did not change. Then, we removed the removing stopwords filter (F1 score did not change this time too). We then removed the lemmatizer which followed a slight decrease in the F1 score.

```
stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    text = [t for t in removePunct if t not in stop_words]

    #lemmatizing
    #text = [WordNetLemmatizer().lemmatize(t) for t in removePunct]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()
```



```
from sklearn.metrics import f1_score
#training
#MNB = MultinomialNaiveBayes(
#    classes=np.unique(y),
#    tokenizer=Tokenizer()
#).fit(X_train, y_train)
#tokenizer=Tokenizer(X_train)
#naive bayes attempt 1
MNB=MultinomialNB()
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))

0.9223758164970378
```

Attempt 8:

We removed the remove punctuation filter and the remove stop words filter from attempt 5.

```

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    #removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    #text = [t for t in removePunct if t not in stop_words]

    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in tokens]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```

from sklearn.metrics import f1_score
MNB=MultinomialNB()
MNB.fit(X_train, y_train)

#testing
y_pred = MNB.predict(X_test)
#accuracy_score(y_test,y_pred)
print(f1_score(y_test,y_pred))

0.9223758164970378

```

There was a slight decrease in F1 score from attempt 5, with around the same computation time on Google Colab.

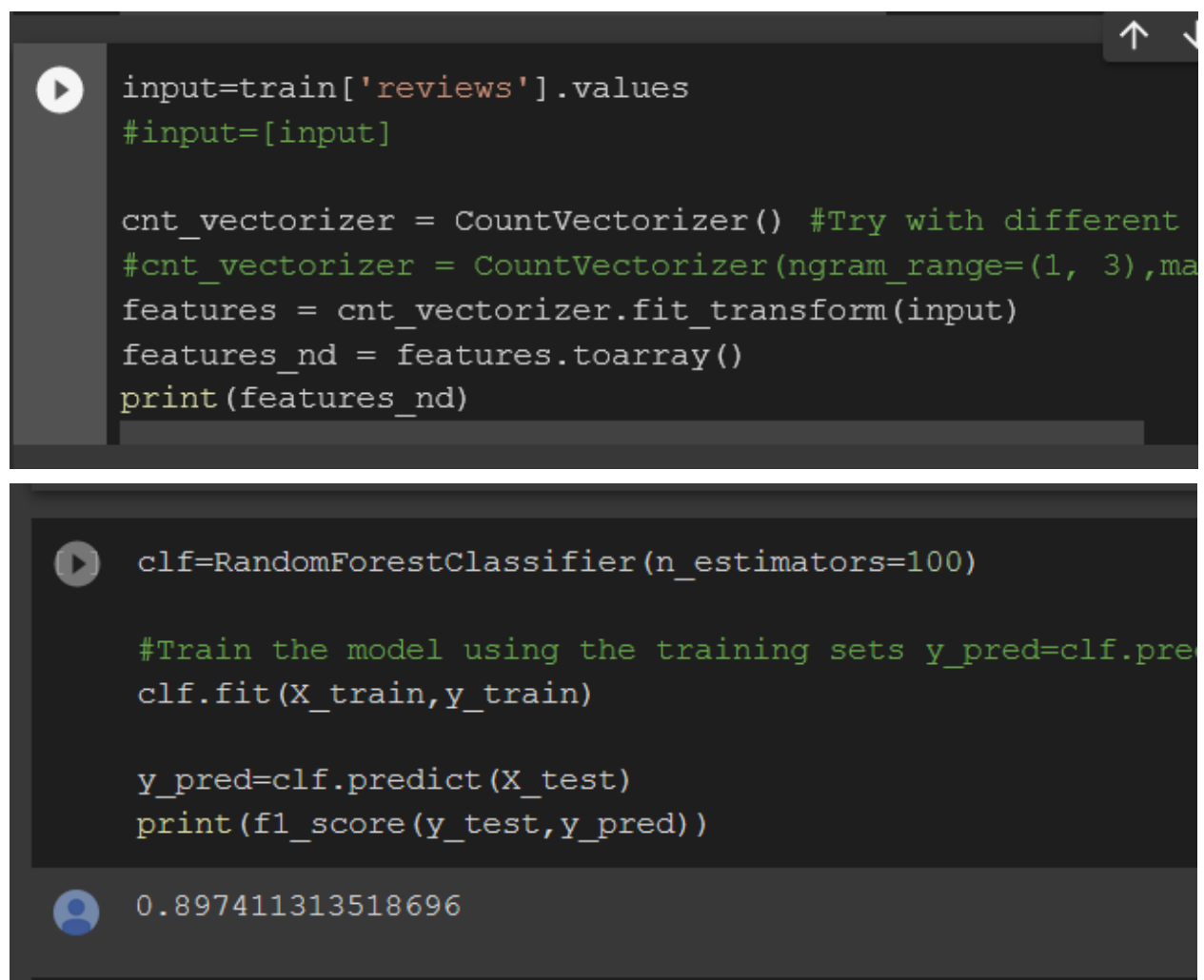
## Random Forest:

### Attempt 1:

We preprocessed data only by removing missing values, count vectorization and feature selection as mentioned above in the document, under the heading of data preprocessing.

Random forest classifier was used by importing it from sklearn python library and default features for countvectorizer were used.

The number of trees was set to 100.



```
input=train['reviews'].values
#input=[input]

cnt_vectorizer = CountVectorizer() #Try with different
#cnt_vectorizer = CountVectorizer(ngram_range=(1, 3),ma
features = cnt_vectorizer.fit_transform(input)
features_nd = features.toarray()
print(features_nd)

clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.pre
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.897411313518696
```

F1 score of about 89.7% was achieved with a computation time of 1m 9s on Google Colab.

Attempt 2:

Using the same data preprocessing techniques as in attempt 1, we increased the number of trees to 150 by increasing the value of `n_estimators` to 150 in the random forest classifier.

```
clf=RandomForestClassifier(n_estimators=150)

#Train the model using the training sets y_pre
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8959827833572452
```

F1 score of about 89.60% was achieved with a computation time of 1m 40s on Google Colab. Computation time increased with no significant increase in F1 score.

Attempt 3:

With the same data preprocessing as in previous attempts, we tried to see the results with decreasing the number of trees.

```
# random forest attempt 2
clf=RandomForestClassifier(n_estimators=50)

#Train the model using the training sets y_p
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8943771730008392
```

F1 score of 89.44% was observed with a computation time of 41s on Google Colab. No significant difference from attempts 1 or 2 was observed in terms of F1 scores.

Attempt 4:

With the same data preprocessing techniques as in previous attempts, we decided to experiment with 100 trees and changed the criterion time from 'gini' to 'entropy'.

```
# random forest attempt 4
clf=RandomForestClassifier(n_estimators=100, criterion='entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8993498675656152
```

F1 score of 89.93 was achieved which was the highest F1 score of all the previous attempts. Computation time was 1m 6s on Google Colab.

Attempt 5:

In this attempt, we included text mining in the data preprocessing techniques. The dataset cleaned after text mining was sent to the count vectorizer was then passed to the random forest classifier with 100 trees.

```

import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('wordnet')

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    removeStopWords = [t for t in removePunct if t not in stop_words]

    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in removeStopWords]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```

# random forest attempt 1
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8990110043181502

```

With a computation time of 1m 12s on Google Colab, F1 score is more or less similar to previous attempts.

#### Attempt 6:

With the same data preprocessing techniques as in attempt 5, we changed the criterion from default 'gini' to 'entropy'.

```
# random forest attempt 1
clf=RandomForestClassifier(n_estimators=100, criterion = 'entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.9005864283719631
```

F1 score of 90.06% was observed which was the highest F1 score of all the attempts of random forest classifier until now. Computation time was 1m 26s on Google Colab.

#### Attempt 7:

With the same data preprocessing techniques as in attempt 5, we changed the criterion from default 'gini' to 'entropy' and number of trees to 150.

```
# random forest attempt 1
clf=RandomForestClassifier(n_estimators=150, criterion = 'entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8991234172812022
```

A slight decrease in F1 score was observed with a computation time of 1m 43s.

#### Attempt 8:

After removing the stopwords filter and lemmatizer from text preprocessing, we ran the random forest classifier with 100 trees and with entropy as the criterion.

```
#Tokenize and lowercasing
tokens = word_tokenize(data.replace("'", "").lower())

#Punctuation Eraser
text = [w for w in tokens if w.isalpha()]

#Removing Stopwords
removeStopWords = [t for t in removePunct if t not in stop_words]

#lemmatizing
text = [WordNetLemmatizer().lemmatize(t) for t in removeStopWords]

#joining
return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()
```

```
# random forest attempt 4
clf=RandomForestClassifier(n_estimators=100, criterion='entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))
```

```
0.8993498675656152
```

No significant change in F1 from previous attempts. Computation time was 1m 5s on Google Colab.

#### Attempt 9:

After removing the punctuation filter and lemmatizer from text preprocessing, we ran the random forest classifier with 100 trees and with entropy as the criterion.



```

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    #removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    text = [t for t in tokens if t not in stop_words]

    #lemmatizing
    #text = [WordNetLemmatizer().lemmatize(t) for t in tokens]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```

# random forest attempt 4
clf=RandomForestClassifier(n_estimators=100, criterion='entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8992053534086156

```

No significant change in F1 from previous attempts. Computation time was 1m 12s on Google Colab.

Attempt 10:

After removing the lemmatizer from text preprocessing, we ran the random forest classifier with 100 trees and with entropy as the criterion.

```

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    text = [t for t in tokens if t not in stop_words]

    #lemmatizing
    #text = [WordNetLemmatizer().lemmatize(t) for t in removeS

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```

# random forest attempt 4
clf=RandomForestClassifier(n_estimators=100, criterion='entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8987978753145093

```

No significant change in F1 score from previous attempts. Computation time was 1m 18s on Google Colab.

Attempt 11:

After removing the stop words filter and punctuations filter from text preprocessing and using lemmatization only, we ran the random forest classifier with 100 trees and with entropy as the criterion.

```
stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    #removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    #text = [t for t in tokens if t not in stop_words]

    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in tokens]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()
```

```
# random forest attempt 4
clf=RandomForestClassifier(n_estimators=100, criterion='entropy')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
print(f1_score(y_test,y_pred))

0.8999161308358959
```

No significant change in F1 score from previous attempts. Computation time was 1m 16s on Google Colab.

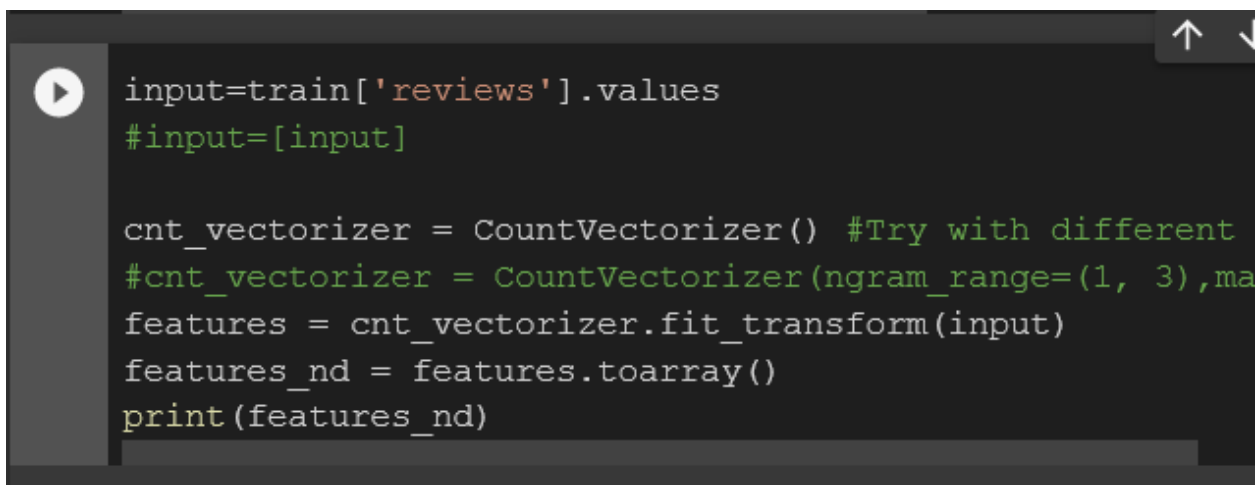
## Decision Tree:

### Attempt 1:

We preprocessed data only by removing missing values, count vectorization and feature selection as mentioned above in the document, under the heading of data preprocessing.

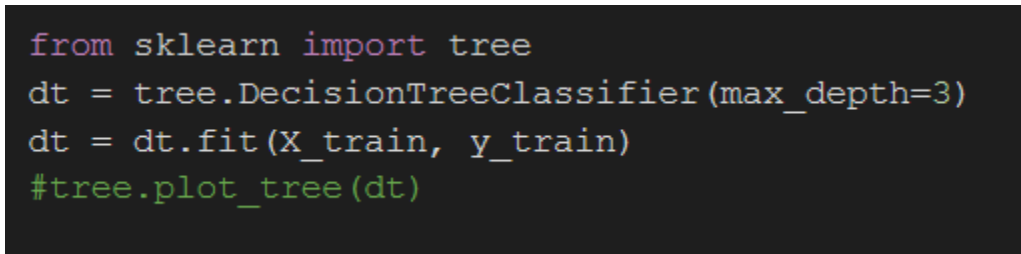
Decision tree classifier was used by importing it from sklearn python library and default features for countvectorizer were used.

Maximum depth of the tree was set to 3 with the result of the classifier settings as default.

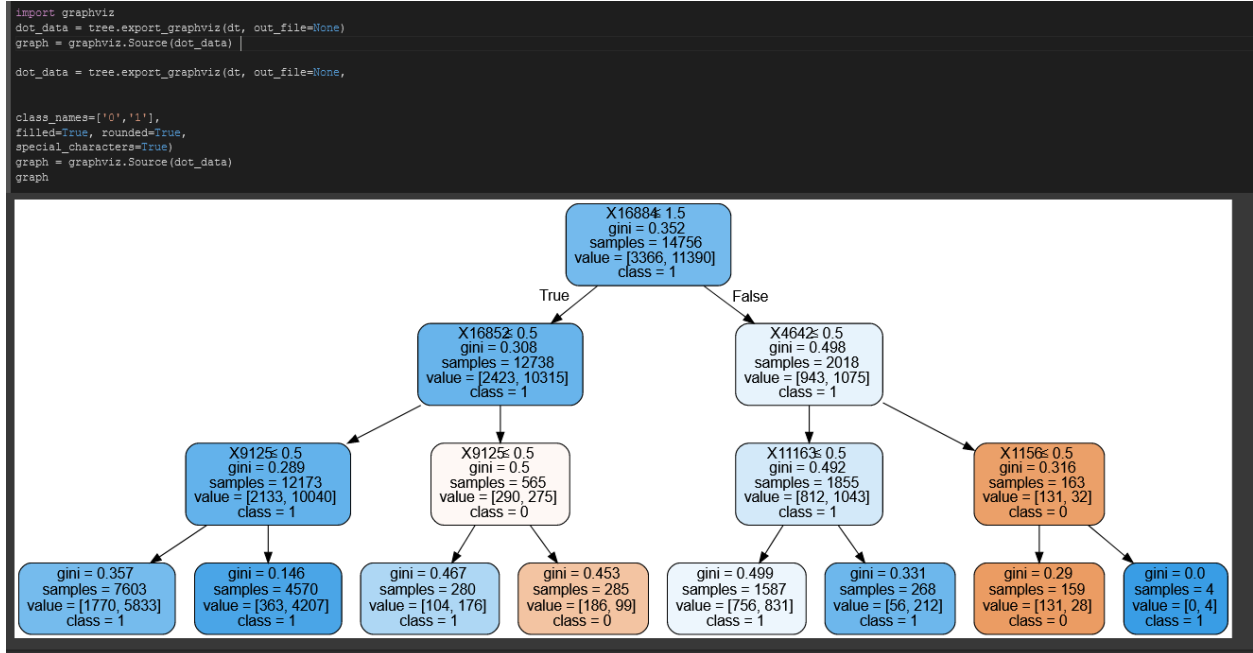
A screenshot of a Jupyter Notebook code cell. It features a dark background with a play button icon on the left and a scroll bar on the right. The code is written in a syntax-highlighted font.

```
input=train['reviews'].values
#input=[input]

cnt_vectorizer = CountVectorizer() #Try with different
#cnt_vectorizer = CountVectorizer(ngram_range=(1, 3),ma
features = cnt_vectorizer.fit_transform(input)
features_nd = features.toarray()
print(features_nd)
```

A screenshot of a Python code block with a dark background and syntax-highlighted text.

```
from sklearn import tree
dt = tree.DecisionTreeClassifier(max_depth=3)
dt = dt.fit(X_train, y_train)
#tree.plot_tree(dt)
```



```

y_pred=dt.predict(X_test)
#print(y_pred)
print(f1_score(y_test,y_pred))
#accuracy_score(y_test,y_pred)

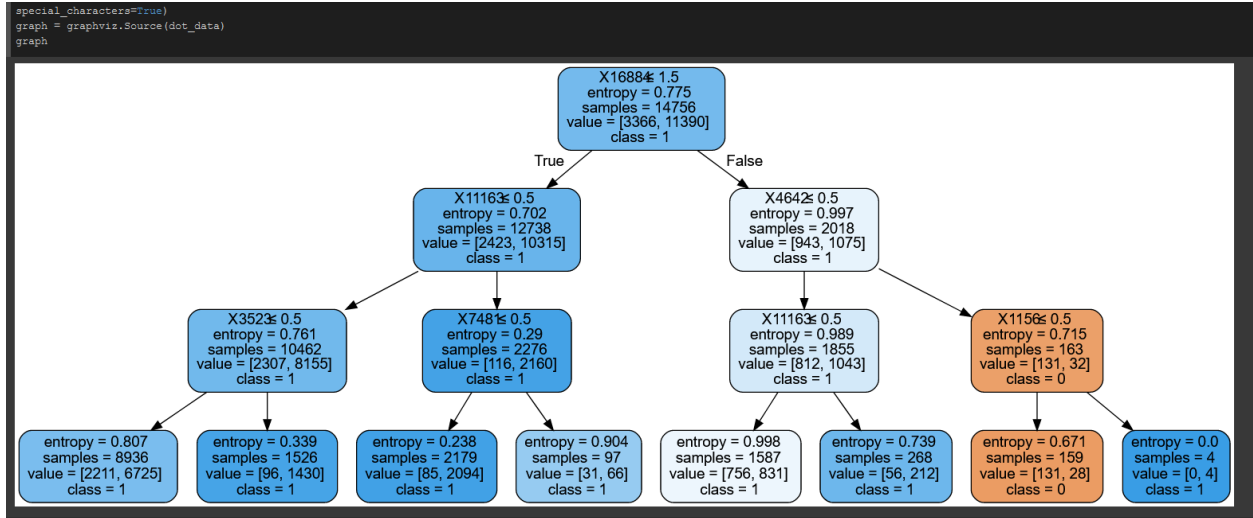
0.8712431294585429

```

F1 score of 87.12% was achieved with a computation time of 19s on Google Colab.

Attempt 2:

With the same data preprocessing used in attempt 1, we changed the criterion in the decision tree classifier from default 'gini' to 'entropy' using the same tree depth.



```

[25] y_pred=dt.predict(X_test)
      #print(y_pred)
      print(f1_score(y_test,y_pred))
      #accuracy_score(y_test,y_pred)

0.8713490959666204

```

F1 score of about 87.13% was observed. This took a computation time of 20s with no significant increase in the F1 score from attempt 1.

Attempt 3:

With the same data preprocessing techniques as in previous attempts, we changed the max depth to 10 and set criteria as 'entropy'.

```

# decision tree attempt 2
dt = tree.DecisionTreeClassifier(max_depth=10, criterion='entropy')
dt = dt.fit(X_train, y_train)

```

```
[38] y_pred=dt.predict(X_test)
      #print(y_pred)
      print(f1_score(y_test,y_pred))
      #accuracy_score(y_test,y_pred)

0.8749999999999999
```

F1 score of about 87.50 was observed which was not significantly different than in previous attempts. Computation time was 21 s on Google Colab.

Attempt 4:

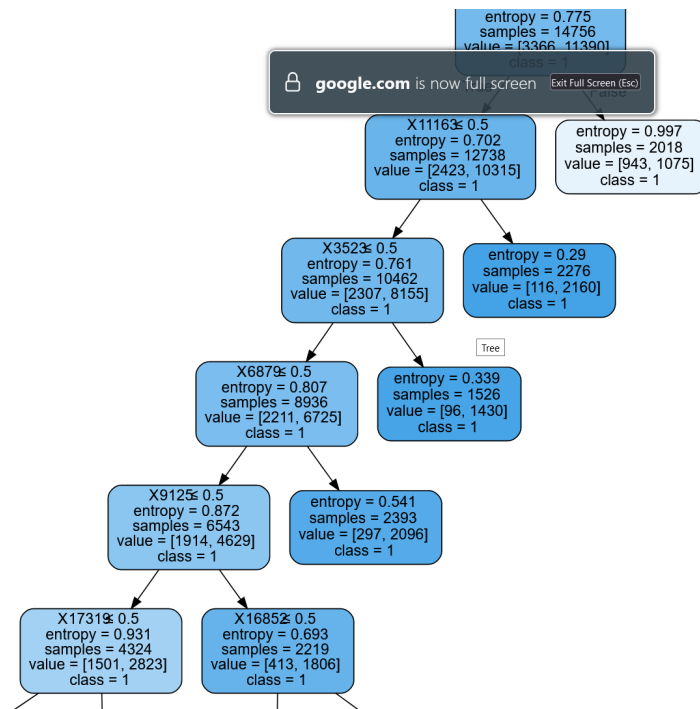
Same data preprocessing as in previous attempts, settings changed were that max leaf nodes were fixed to be no more than 8.

```
dt = tree.DecisionTreeClassifier(max_depth=10, max_leaf_nodes = 8, criterion='entropy')
dt = dt.fit(X_train, y_train)

import graphviz
dot_data = tree.export_graphviz(dt, out_file=None)
graph = graphviz.Source(dot_data)

dot_data = tree.export_graphviz(dt, out_file=None,

class_names=['0','1'],
filled=True, rounded=True,
special_characters=True)
graph = graphviz.Source(dot_data)
graph
```



```

y_pred=dt.predict(X_test)
#print(y_pred)
print(f1_score(y_test,y_pred))
#accuracy_score(y_test,y_pred)

0.8698067353315588

```

F1 score of 86.98 was achieved, a decrease from previous attempts, with a computation time of 18 s on Google Colab.

Attempt 5:

In this attempt, we included text mining in the data preprocessing techniques. The dataset cleaned after text mining was then passed to the decision tree classifier with tree depth of 3.



```

import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('wordnet')

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    removeStopWords = [t for t in removePunct if t not in stop_words]

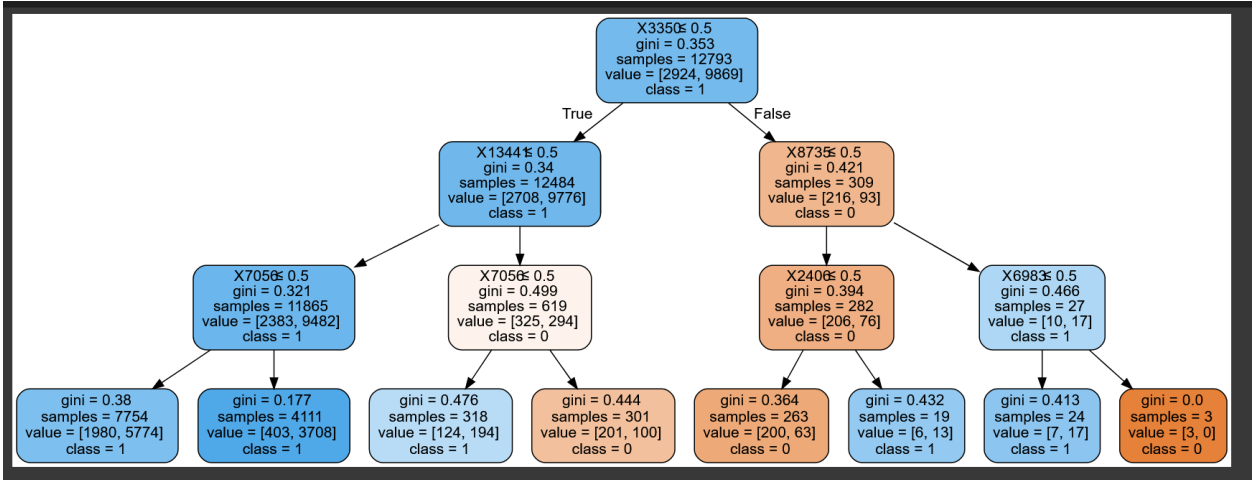
    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in removeStopWords]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```
# decision tree attempt 1
from sklearn import tree
dt = tree.DecisionTreeClassifier(max_depth=3)
dt = dt.fit(X_train, y_train)
#tree.plot_tree(dt)
```



```
y_pred=dt.predict(X_test)
#print(y_pred)
print(f1_score(y_test,y_pred))
#accuracy_score(y_test,y_pred)

0.8712358631966208
```

F1 score is more or less similar to previous attempts. Computation time was 12s on Google Colab.

Attempt 6:

With the same data preprocessing techniques as in attempt 5, we decided to increase the tree depth to 6.

```
# decision tree attempt 1
from sklearn import tree
dt = tree.DecisionTreeClassifier(max_depth=6)
dt = dt.fit(X_train, y_train)
#tree.plot_tree(dt)
```

```
y_pred=dt.predict(X_test)
#print(y_pred)
print(f1_score(y_test,y_pred))
#accuracy_score(y_test,y_pred)

0.8748269177513153
```

We achieved the highest F1 score until now for decision tree classifier with a computation time of 10s on Google Colab.

Attempt 7:

With the same data preprocessing techniques as in attempt 5, we set the tree depth to 5 and changed the criterion to entropy.

```
# decision tree attempt 1
from sklearn import tree
dt = tree.DecisionTreeClassifier(max_depth=6, criterion = 'entropy')
dt = dt.fit(X_train, y_train)
#tree.plot_tree(dt)
```

```
y_pred=dt.predict(X_test)
#print(y_pred)
print(f1_score(y_test,y_pred))
#accuracy_score(y_test,y_pred)

0.8706979295214591
```

F1 score decreased from attempt 6 but was similar to other attempts with a computation time for 22s on Google Colab.

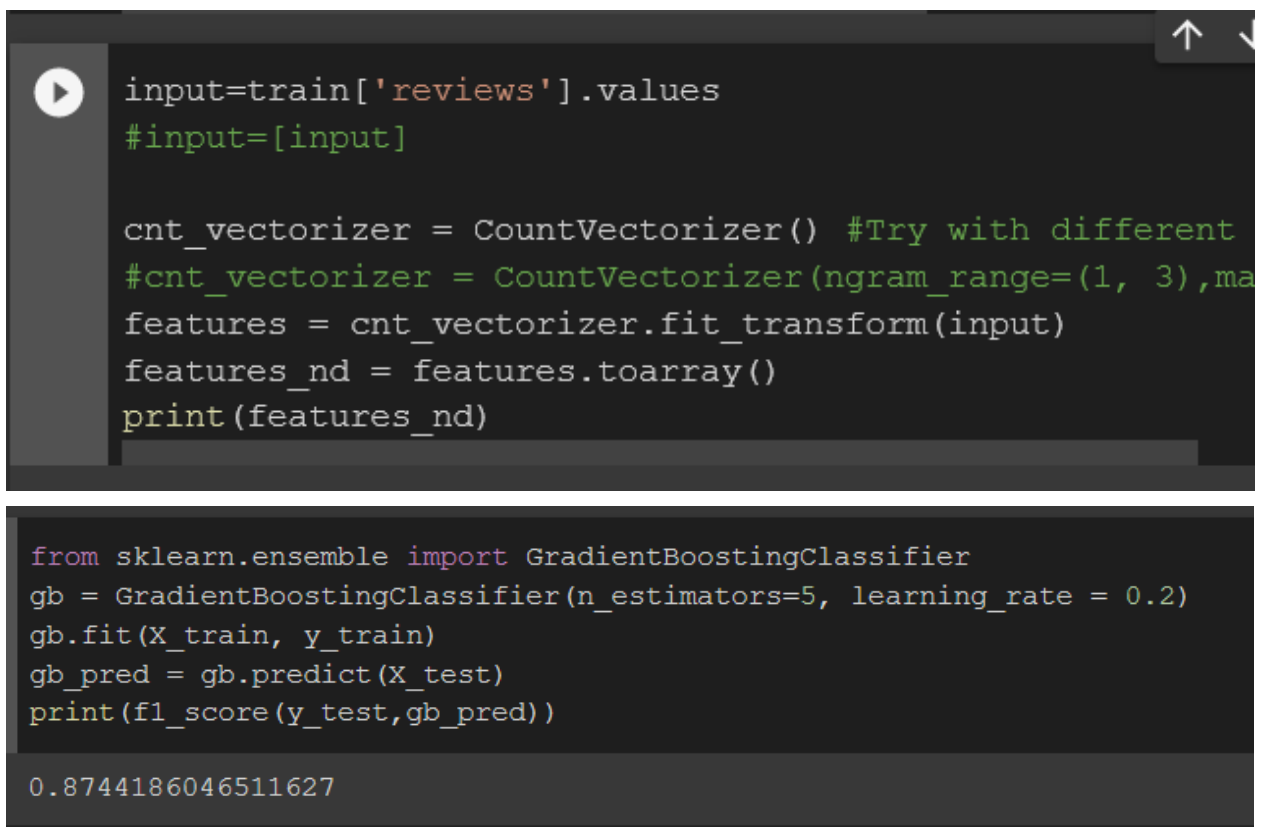
## Gradient Boosting:

### Attempt 1:

We preprocessed data only by removing missing values, count vectorization and feature selection as mentioned above in the document, under the heading of data preprocessing.

Gradient Boosting classifier was used by importing it from sklearn python library and default features for countvectorizer were used.

5 models with a learning rate of 0.2 were used in the classifier, rest of the settings were set to default.



```
input=train['reviews'].values
#input=[input]

cnt_vectorizer = CountVectorizer() #Try with different
#cnt_vectorizer = CountVectorizer(ngram_range=(1, 3),ma
features = cnt_vectorizer.fit_transform(input)
features_nd = features.toarray()
print(features_nd)

from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=5, learning_rate = 0.2)
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
print(f1_score(y_test,gb_pred))

0.8744186046511627
```

F1 score of 87.44% was achieved with a computation time of 52s on Google Colab.

### Attempt 2:

Using the same data preprocessing as in attempt 1, we increased the number of estimators to 100.

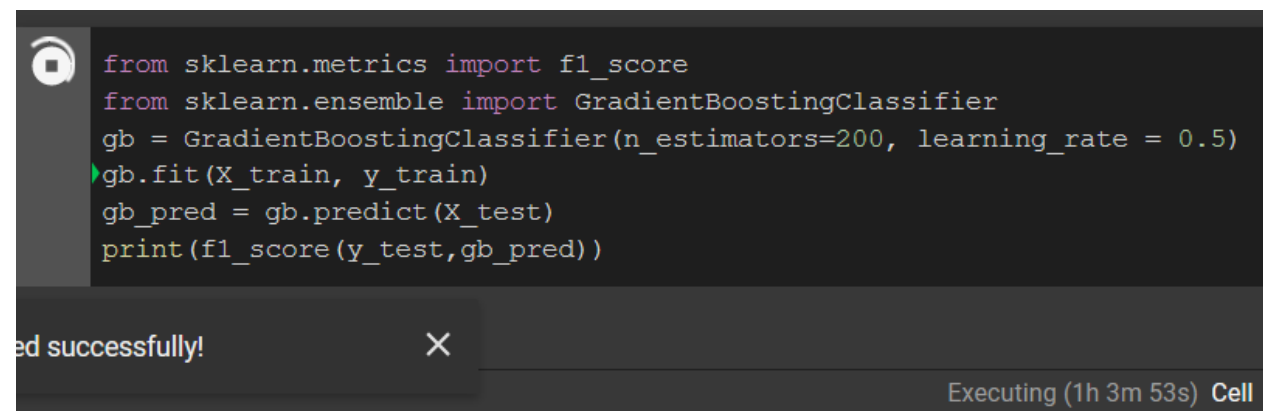
```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate = 0.2)
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
print(f1_score(y_test,gb_pred))

0.9136020151133502
```

F1 score of about 91.36% was achieved, which was an increase of 3.92% from attempt 1. However, computation time increased from 52s to 14m 6s.

Attempt 3:

Changing anything to gradient boost resulted in a lot of computation time. (1 hr 3m in the screenshot below and still running), so we decided to stop experimenting with gradient boosting.



```
from sklearn.metrics import f1_score
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=200, learning_rate = 0.5)
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
print(f1_score(y_test,gb_pred))
```

ed successfully! X

Executing (1h 3m 53s) Cell

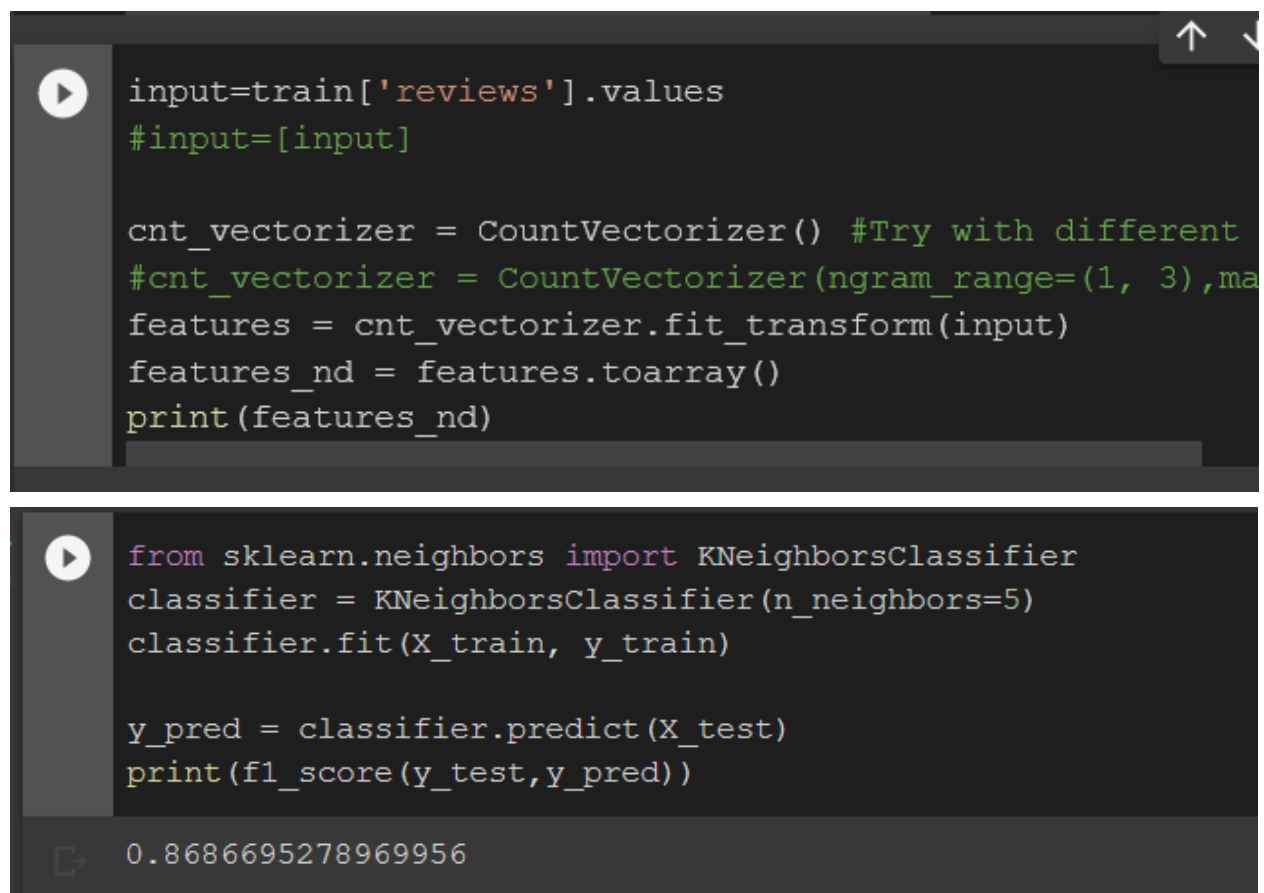
## K Nearest Neighbours:

### Attempt 1:

We preprocessed data only by removing missing values, count vectorization and feature selection as mentioned above in the document, under the heading of data preprocessing.

K nearest neighbours classifier was used by importing it from sklearn python library and default features for countvectorizer were used.

K nearest neighbours classifier was run with 5 nearest neighbours and rest of the classifier configuration as default.



```
input=train['reviews'].values
#input=[input]

cnt_vectorizer = CountVectorizer() #Try with different
#cnt_vectorizer = CountVectorizer(ngram_range=(1, 3),ma
features = cnt_vectorizer.fit_transform(input)
features_nd = features.toarray()
print(features_nd)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test,y_pred))

0.8686695278969956
```

F1 score of about 86.87% was achieved with a computation time of 1m 34s on Google Colab.

### Attempt 2:

With the same preprocessed data as in attempt 1, we decreased the number of nearest neighbours from 5 to 3.

```
# k nearest neighbours attempt 2
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8655493482309123
```

F1 score was 85.55%, more or less similar to attempt 1, with a computation time of 1m 18 s on Google Colab.

Attempt 3:

With the same data preprocessing techniques as in previous attempts, k was set to 5 and neighbours were weighted by distance.

```
# k nearest neighbours attempt 3
classifier = KNeighborsClassifier(n_neighbors=5, weights='distance')
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.868776060829041
```

No significant change in F1 score, which was now about 86.88%. Computation time was 1m 16s on Google Colab.

Attempt 4:

With the same data preprocessing techniques, k was increased to 11 and it was specified to use euclidean distance ( $p = 2$ ) and weighted by distance.

```
# k nearest neighbours attempt 3
classifier = KNeighborsClassifier(n_neighbors=11, weights='distance', p = 2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8760806916426512
```

F1 score, about 87.61% increased slightly from previous attempts. Computation time was 1m 12s on Google Colab.

Attempt 5:

Using same data preprocessing techniques and configurations, manhattan distance was used in the classifier instead of euclidean ( $p = 1$ ).

```
# k nearest neighbours attempt 3
classifier = KNeighborsClassifier(n_neighbors=11, weights='distance', p = 1)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8723554301833568
```

F1 score was similar to attempt 2 but computation time increased significantly to 20m 17s on Google Colab.

Attempt 6:

With the same data preprocessing techniques, we tried to see the effect of increasing k.



```
# k nearest neighbours attempt 3
classifier = KNeighborsClassifier(n_neighbors=15, weights='distance', p = 2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test,y_pred))

0.8741058655221744
```

There was no significant change in F1 score. Computation time was 1m 15s on Google Colab.

Attempt 7:

In this attempt, we included text mining in the data preprocessing techniques. The dataset cleaned after text mining was sent to the count vectorizer which was passed to K nearest neighbours classifiers with 11 nearest neighbours weighted by distance and distance calculated set as euclidean distance.

```
# k nearest neighbours attempt 3
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=11, weights='distance', p = 2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test,y_pred))

0.8676291998380785
```

F1 score was similar to attempt 1, attempt 2 and attempt 3 and was lower than attempt 4, attempt 5 and attempt 6. Total computation time was 59s on Google Colab.

Attempt 8:

With the same data preprocessing techniques as in attempt 7, we changed the number of nearest neighbours to 7.

```
# k nearest neighbours attempt 3
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=7)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8670678336980305
```

F1 score was similar to attempt 7. Computation time was 57s on Google Colab.

Attempt 9:

With the same data preprocessing techniques as in attempt 7, we changed the number of nearest neighbours to 13.

```
# k nearest neighbours attempt 3
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=13)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8672590199246095
```

F1 score decreased from previous attempt. Computation time was 55s on Google Colab.

Attempt 10:

We removed the remove punctuation filter and the remove stop words filter from our text mining techniques.

```

stop_words = stopwords.words('english')
def cleaningData(data):

    #Tokenize and lowercasing
    tokens = word_tokenize(data.replace("'", "").lower())

    #Punctuation Eraser
    #removePunct = [w for w in tokens if w.isalpha()]

    #Removing Stopwords
    #text = [t for t in tokens if t not in stop_words]

    #lemmatizing
    text = [WordNetLemmatizer().lemmatize(t) for t in tokens]

    #joining
    return " ".join(text)

train['reviews'] = train['reviews'].apply(cleaningData)
train['reviews'].head()

```

```

# k nearest neighbours attempt 3
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=11)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8676291998380785

```

F1 score was similar to other attempts. Computation time was 55s on Google Colab.

Attempt 11:

With the same data preprocessing techniques as in attempt 7, we changed the number of nearest neighbours to 17.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=17, weights='distance', p=2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(f1_score(y_test, y_pred))

0.8667380251538668
```

F1 score was similar to other attempts. Computation time was 1m on Google Colab.

# Model Selection

## Best Attempts:

### Multinomial Naive Bayes:

All attempts from multinomial naive bayes resulted in almost the same F1 score with almost the same computation time.

The highest F1 score from multinomial naive bayes was from attempt 1, which was 93.43%, in which no text mining techniques were used.

However, we decided to declare the model from attempt 5 as our best multinomial naive bayes, as it was more explainable than the other models, because it used all the combined text mining techniques. F1 score from attempt 5 was 92.25% with a computation time of 24s on Google Colab.

### Random Forest:

The highest F1 score for Random Forest was 90.06% from attempt 6, computation time 1m 26s on Google Colab, in which we used all the text mining techniques described under the heading of Data Preprocessing on Page 16. Random Forest Classifier was used with 100 tree estimators and 'entropy' as the criterion in the classifier.

### Decision Tree:

All attempts from decision trees resulted in almost the same F1 score with almost the same computation time.

The highest F1 score from decision trees was from attempt 6, which was 87.48%, computation time of 10s on Google Colab in which all text mining techniques mentioned on Page 16 were used. Attempt 6 also resulted in the shortest computation time for decision tree model. Decision tree classifier with max tree depth of 6 and 'gini' as the criterion was used.

### Gradient Boosting:

Gradient Boosting was taking a lot of computation time, so, after three attempts, we decided to stop experimenting with gradient boosting to find out the best model.

Out of the three attempts we did, we forced stopped the third model as it took more than one hour to run and was still running.

Attempt 1 resulted in a F1 score of 87.44%, taking a computation time of 52s.  
Attempt 2 resulted in a F1 score of 91.36% taking a computation time of 14m 6s.

### K Nearest Neighbours:

The highest F1 score for K nearest neighbours was 87.61% from attempt 4, in which text mining techniques were not used. Computation time on Google Colab for this model was 1m 12s.

The second highest F1 score for K nearest neighbours was 87.41% from attempt 6. This model too, did not use any text mining techniques and was similar to the model used in attempt 4. The only difference was that this model used 15 nearest neighbours while the model with the highest accuracy used 13 nearest neighbours. Computation time was 1 m 16 s on Google Colab.

For the models that used text mining techniques, the average F1 score was 86.73%, but computation time was lower than the models that did not use text mining techniques. None of the models in this category took more than 1 minute to run.

So we decided to choose a model which used text mining, because of its feasible computation time.

We declared the model from attempt 7, which used all the text mining techniques, had an F1 score of 86.76% and a computation time of 59s on Google Colab. The drop in F1 score from the highest F1 score (87.41%) was only 0.65%.

### Best Working Model:

We got our best F1 scores in the order that follows (from highest to lowest):

- Multinomial naive bayes
- Random forest
- Decision tree
- K nearest neighbours

We got our best computation times in the order that follows (from lowest to highest):

- Decision tree
- Multinomial naive bayes
- K nearest neighbours
- Random forest

We declared multinomial naive bayes as the best working model for sentiment analysis on this dataset as it gave the highest F1 scores with reasonable computation times. The best configuration and data preprocessing for naive bayes were the ones we used in the attempt 6 of

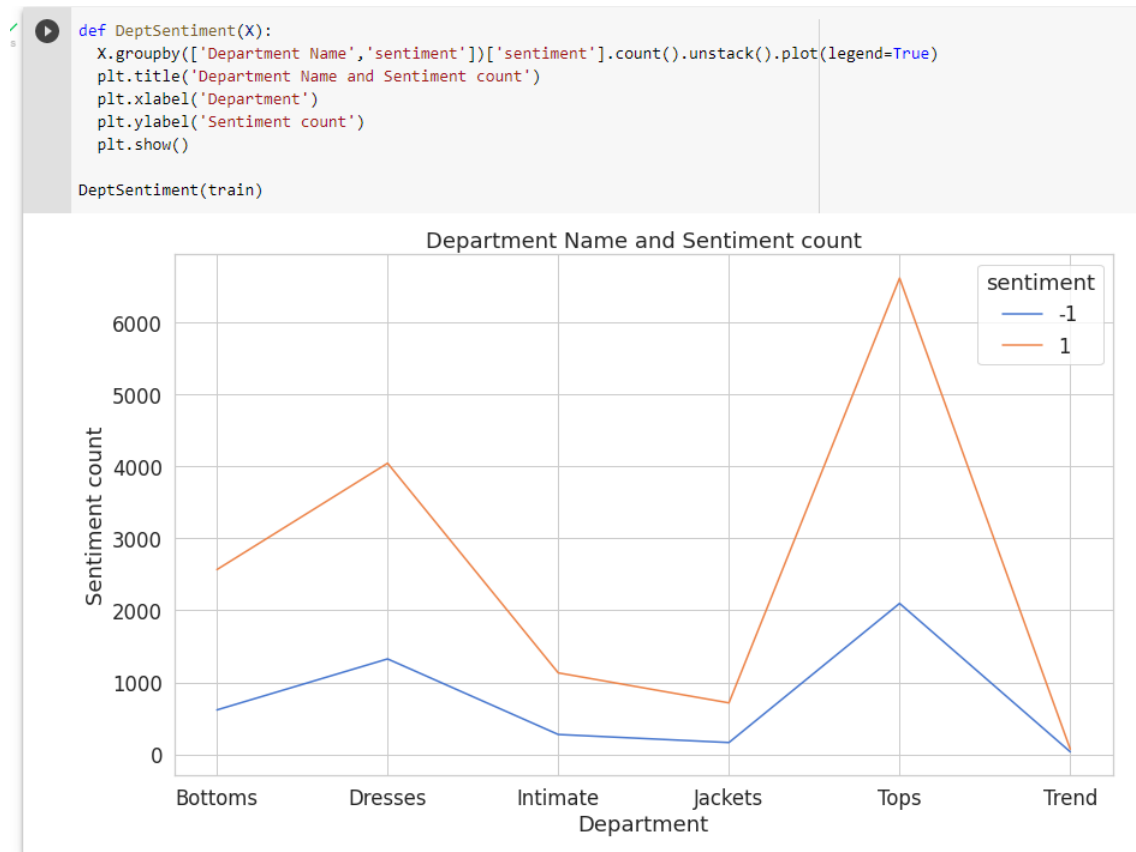
building our multinomial naive bayes classifier model. The model can be referred to on Page 23 of this document.

# Findings

## Useful Insights:

The detailed sentiment analysis on the dataset provided some useful insights about customer preference, sentiments, product quality and popularity, and the relationship between customer age and product review sentiment.

The 'Department Name and Sentiment count' graph below implies customer preference and popularity of each department category in the ecommerce store. As per the readings, most customers' purchases were more inclined towards the categories 'Dresses' and 'Tops'. These two categories also had the two highest positive sentiment counts with negative sentiment count being less than half of the positive sentiment count. This suggests that the products belonging to these two categories exhibit good quality and more customer satisfaction.



Similarly, the graph below shows the relationship between customer age and their review sentiment counts derived from the dataset. As can be seen in the figure below, women belonging to the age group of 30 to 50 years old were the most active in providing product reviews. Further,



the women aged 35 to 40 years provided the highest number of positive reviews which implies greater customer satisfaction in this age group.

```
[15] def AgeSentiment(X):  
      X.groupby(['Age', 'sentiment'])['sentiment'].count().unstack().plot(legend=True)  
      plt.title('Age and Sentiment count')  
      plt.xlabel('Age')  
      plt.ylabel('Sentiment count')  
      plt.show()  
  
AgeSentiment(train)
```



Such insights from the dataset can significantly contribute in predicting potential repeated purchases. These customers can also be specifically targeted in the marketing campaigns for popular products to increase sales.

## Limitations of the Study:

While the built model provides a preferable F1 score with reasonable computation cost, there are certain limitations of the study and the model. The dataset provided included more positive reviews than negative reviews which limits the understanding of the available product quality.

It is possible that there was a higher number of dissatisfied customers who did not write text reviews. Such records with null values in the review text column were eliminated by the model which creates a possibility for the model overlooking some more negative reviews. In this way, the model is limited to explaining the quality and preference of the existing products only on the basis of the positive review counts.

## Future Improvements:

To retrieve more useful information in terms of devising more effective and profitable marketing strategies, it is advisable to collect more data related to the customers writing the reviews. Such as, the dataset can include information about the demographics of the customer leaving the review. This would enable the model to generate more useful insights about which products are popular among which customer group. In turn, this would help create a specific customer base that can be targeted in marketing campaigns to increase website traffic and potential purchases.

Moreover, in order to improve product quality assessment, the dataset can be expanded to include names of the specific products the customer is leaving a review for. By doing so, the organization will increase its chances of identifying which product is not liked by the customers and needs improvement.

Along with this, to tackle the issue of the current limitation of the model, the organization can also provide a feature for a list of quick, automatic review texts for the customers to select from. This way, customers who prefer to save time in typing the reviews can simply select a text review from the provided list. This would prevent most customers from sending null review text which leads the current model to totally remove the entire record from the dataset.

## Deployment Plan:

The created classification is quite simple to implement in a business setting. For the model to predict the reviews sentiments, the user only needs to upload the relevant dataset with all the required attributes.

Once the model is run, the predictions of the review sentiment are displayed in an easy-to-understand confusion matrix like the one below.



For further analysis, the graphs generated by the model will be helpful. However, this would require the user to be learned enough to understand and analyse the displayed graphs and visuals.

To make this model more interactive and hassle-free, an online application can be developed. The software application would take as input a CSV file containing the dataset and return the predicted sentiments of the individual reviews. Moreover, the user can be given the flexibility to retrieve whatever useful information or graphs they require in order to understand the input dataset. This would in turn speed up the process and make it more efficient for the personnels belonging to the marketing department to derive useful insights as discussed in the earlier section

## Model Maintenance:

As discussed above, if the model is implemented via a software application, it would provide ease of usability. This will further reduce the maintenance requirement of the model as the user will have limited access. Such restricted access will prevent the user from making uninformed changes to the model.

If the model is protected and maintained in such a manner, then this would also ensure longevity of the proposed model. However, if there are changes in the form of the collected structured data, then the created model would not work. One primary requirement of this model is that the input will be structured data. If at any point the organization switched to collecting unstructured review data, then the model would not be applicable anymore.

## Conclusion

To reliably forecast the sentiments of the reviews and give relevant insight into the data, the project employs machine learning and a classifier. Women's clothes e-commerce reviews are included in the dataset, which has 23486 rows and 10 feature variables. A customer review is represented by each row. Using the column 'overall rating,' we created our own class variable in the dataset. In a column labeled 'Sentiment,' we saved the class variable ('negative' or 'positive'). First, the missing values were eliminated. Then, instead of using accuracy as an assessment statistic, we chose to employ F1 score. To tokenize the dataset, sometimes a count vectorizer was utilized from python sklearn library and sometimes tokenizer from python nlkt library was used. To remove stop words and punctuation from train data, we utilized the Python nltk package.

To train and evaluate computer models, we employed models from built-in Python packages. Multiple attempts were conducted with these models by adjusting needed parameters and evaluating the f1 score. Multinomial naive bayes, Random forest, Decision tree, and K nearest neighbors were the methods that yielded the best F1 scores (in order of highest to lowest). On this dataset, we declared multinomial naive Bayes to be the best working model for sentiment analysis.

The most popular categories for client purchases were 'Dresses' and 'Tops.' The number of negative sentiments was less than half of the number of favorable sentiments. Women between the ages of 30 and 50 were the most active reviewers on Amazon, according to the research. It is recommended that more data connected to the consumers writing the reviews be collected in order to obtain more meaningful information. The dataset may include information about the customer's demographics when they leave a review. A CSV file containing the dataset may be fed into an online software tool, which will provide the projected sentiments of the individual reviews.