Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

from sklearn.model_selection import train_test_split

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
```

```
..........
Mounted at /content/drive
```

In [3]:
```
# using SQLite Table to read data.
con = sqlite3.connect('drive/My Drive/Colab Notebooks/Assign - 3/databa
se.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 100000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [0]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text

	UserId	ProductId	ProfileName	Time	Score	Text
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...

In [7]:
```
display['COUNT(*)'].sum()
```

Out[7]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [10]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[10]:
```
(87775, 10)
```

In [11]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]:
```
87.775
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
print(final.shape)
```

```
#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(87773, 10)
```

Out[14]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [15]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)
```

```
sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [16]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
```

```
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how
-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
```

```
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [19]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [20]:
```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in
 the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
```

```
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', 
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

[5] Assignment 4: Apply Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)

2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

- Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB and print their corresponding feature names

4. **Feature engineering**

- To increase the performance of your model, you can also experiment with with feature engineering like :
  - Taking length of reviews as another feature.
  - Considering some features from review summary as well.

5. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

6. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this [link.]()

Applying Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

In [0]:
```
# Please write all the code with proper documentation
```

In [24]:
```
final.shape
```

Out[24]:
```
(87773, 10)
```

In [25]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:36<00:00, 2422.95it/s]
```

In [26]:
```python
# Combining all the above stundents and preprocessing the summary data
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_summary.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:23<00:00, 3729.00it/s]
```

In [27]:
```python
#added preprocessed reviews
#sample_preproc_revi
final['PreprocessedText'] = preprocessed_reviews
final['PreprocessedSummary'] = preprocessed_summary
final['Final_Text'] = final['PreprocessedText'] + final['PreprocessedSu
mmary']
final.head(3)
```

Out[27]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0
70677	76870	B00002N8SM	A19Q006CSFT011	Arlielle	0	0

In [0]:

```
#sorting based on time
final["Time"] = pd.to_datetime(final["Time"], unit = "s")
final = final.sort_values(by = "Time")
```

In [29]:

```
X_train , X_test ,y_train , y_test = train_test_split(final['Final_Tex
t'],final['Score'],test_size = 0.3,random_state=42)

X_train_cv, X_cv, y_train_cv, y_cv = train_test_split(X_train, y_train,
 test_size=0.33)

print(X_train.shape, y_train.shape)
print(X_train_cv.shape,y_train_cv.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(61441,) (61441,)
(41165,) (41165,)
(20276,) (20276,)
(26332,) (26332,)
```

In [30]:

```
vectorizer = CountVectorizer()
```

```
vectorizer.fit(X_train_cv)

X_train_cv_bow = vectorizer.transform(X_train_cv)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print(X_train_cv_bow.shape,X_cv_bow.shape,X_test_bow.shape)
```

```
(41165, 68417) (20276, 68417) (26332, 68417)
```

In [31]:

```
print(X_train_cv_bow.shape)
print(X_cv_bow.shape)
print(y_train_cv.shape)
print(y_cv.shape)
```

```
(41165, 68417)
(20276, 68417)
(41165,)
(20276,)
```

In [32]:

```
from sklearn.naive_bayes import MultinomialNB

# Creating alpha values in the range from 10^-4
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

from sklearn.model_selection import GridSearchCV

nb = MultinomialNB()
parameters = {'alpha':alpha}
clf = GridSearchCV(nb, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_cv_bow, y_train_cv)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(alpha, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
```

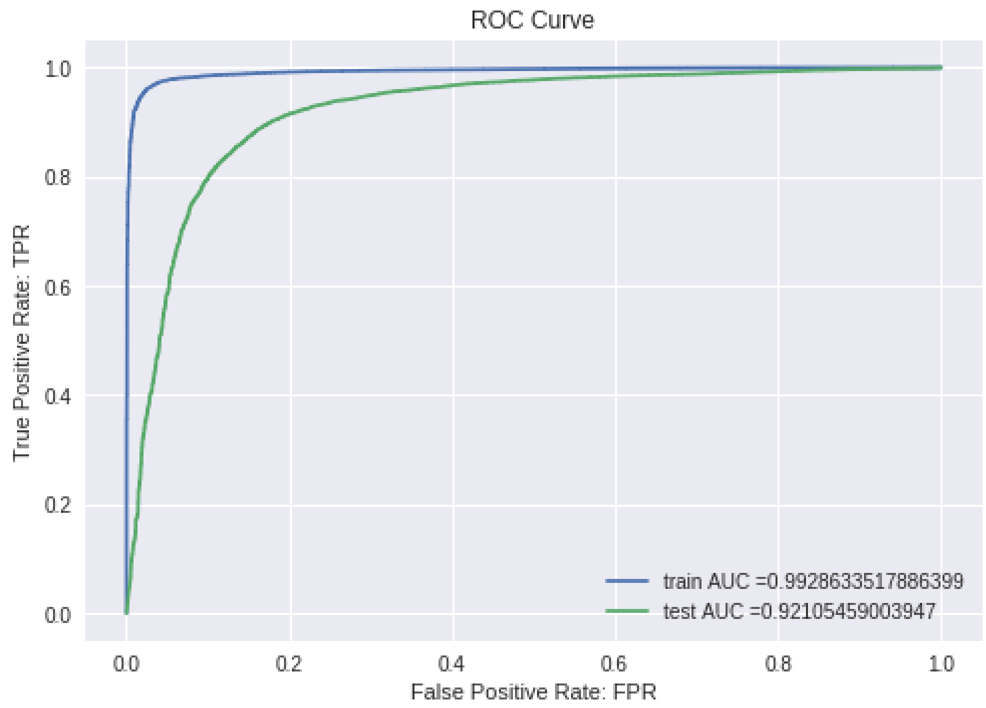
```
084039
plt.gca().fill_between(alpha,train_auc - train_auc_std,train_auc + trai
n_auc_std,alpha=0.2,color='darkblue')

plt.plot(alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
084039
plt.gca().fill_between(alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,al
pha=0.2,color='darkorange')
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.xscale('log')
plt.title("ERROR PLOTS")
plt.show()
```

In [33]:
```
print(clf.best_params_)
```

```
print(clf.best_estimator_)
print(clf.best_score_)
```

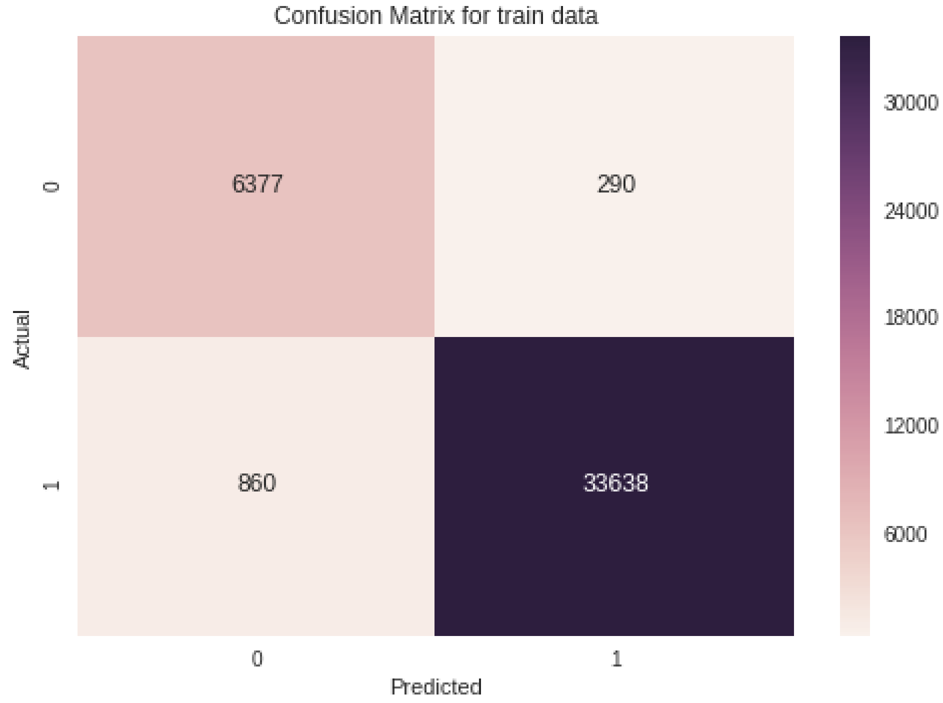
```
{'alpha': 0.1}
MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
0.9156317427616381
```

In [57]:
```
nb_optimal = MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
nb_optimal.fit(X_train_cv_bow, y_train_cv)

train_fpr, train_tpr, thresholds = roc_curve(y_train_cv, nb_optimal.pre
dict_proba(X_train_cv_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, nb_optimal.predict_p
roba(X_test_bow)[:,1])


train_acc_bow = auc(train_fpr, train_tpr)
test_acc_bow = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(train_acc_bow))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(test_acc_bow))

plt.legend()
plt.xlabel("False Positive Rate: FPR")
plt.ylabel("True Positive Rate: TPR")
plt.title("ROC Curve")
plt.show()
```

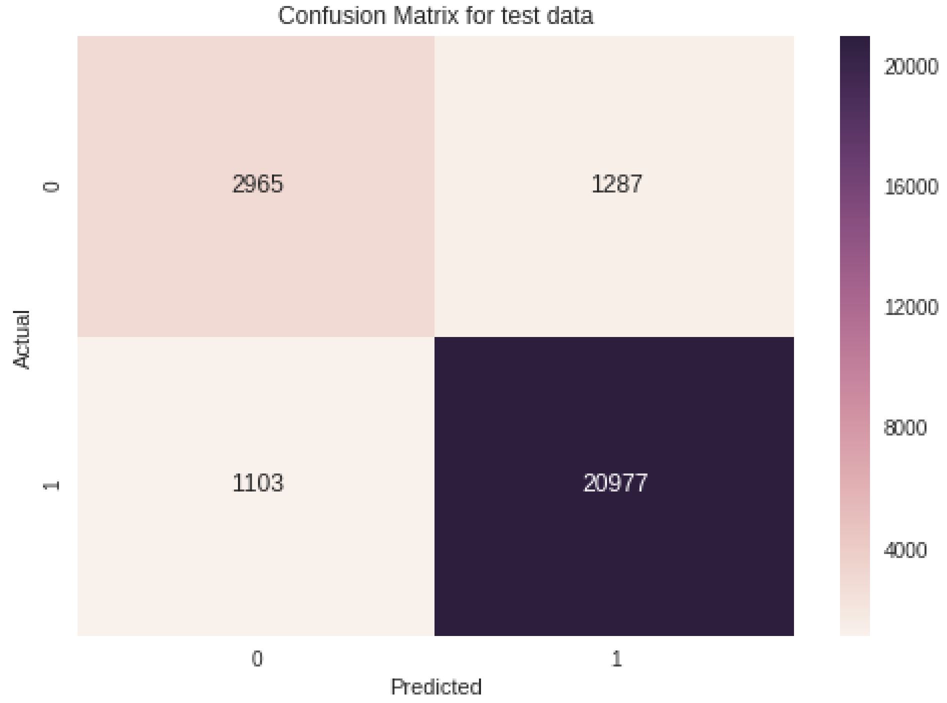


In [35]:

```
conf_matrix = confusion_matrix(y_train_cv, nb_optimal.predict(X_train_c
v_bow))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix for train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [36]:
```
conf_matrix = confusion_matrix(y_test, nb_optimal.predict(X_test_bow))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix for test data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



[5.1.1] Top 10 important features of positive class from SET 1

In [0]:
```
# Please write all the code with proper documentation
```

In [38]:
```
# Now we can find log probabilities of different features for both the
 classes
class_features = nb_optimal.feature_log_prob_

#  row_0 is for 'negative' class and row_1 is for 'positive' class
negative_features = class_features[0]
positive_features = class_features[1]

# Getting all feature names
feature_names = vectorizer.get_feature_names()
```

```
# Sorting 'positive_features' in descending order using argsort() funct
ion
sorted_positive_features = np.argsort(positive_features)[::-1]


print("\n\nTop 10 Important Features and their log probabilities For Po
sitive Class :\n\n")
for i in list(sorted_positive_features[0:10]):
    print("%s\t -->\t%f  "%(feature_names[i],positive_features[i]))
```

```
Top 10 Important Features and their log probabilities For Positive Clas
s :


not      -->    -3.747137
like     -->    -4.553464
good     -->    -4.648461
great    -->    -4.737225
coffee   -->    -4.920770
one      -->    -4.931519
taste    -->    -4.952459
tea      -->    -5.023463
flavor   -->    -5.046554
love     -->    -5.084413
```

[5.1.2] Top 10 important features of negative class from SET 1

In [0]:
```
# Please write all the code with proper documentation
```

In [40]:
```
# Please write all the code with proper documentation
#Here doing for negative features.
sorted_negative_features = np.argsort(negative_features)[::-1]
print("Top 10 Important Features and their log probabilities For Negati
ve Class :\n\n")
for i in list(sorted_negative_features[0:10]):
    print("%s\t -->\t%f  "%(feature_names[i],negative_features[i]))
```

```
Top 10 Important Features and their log probabilities For Negative Clas
s :


not        -->    -3.300935
like       -->    -4.400482
taste      -->    -4.671580
would      -->    -4.698164
product    -->    -4.743983
one        -->    -4.932028
good       -->    -5.075326
coffee     -->    -5.129305
flavor     -->    -5.172776
no         -->    -5.180681
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
#TFIDF

tfidf_vect = TfidfVectorizer(min_df=10 , max_features=500)
tfidf_vect.fit(X_train_cv)

X_train_cv_tfidf = tfidf_vect.transform(X_train_cv)
X_cv_tfidf = tfidf_vect.transform(X_cv)
X_test_tfidf = tfidf_vect.transform(X_test)
```

In [43]:
```
from sklearn.naive_bayes import MultinomialNB

#taking alpha values
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

from sklearn.model_selection import GridSearchCV

nb = MultinomialNB()
```

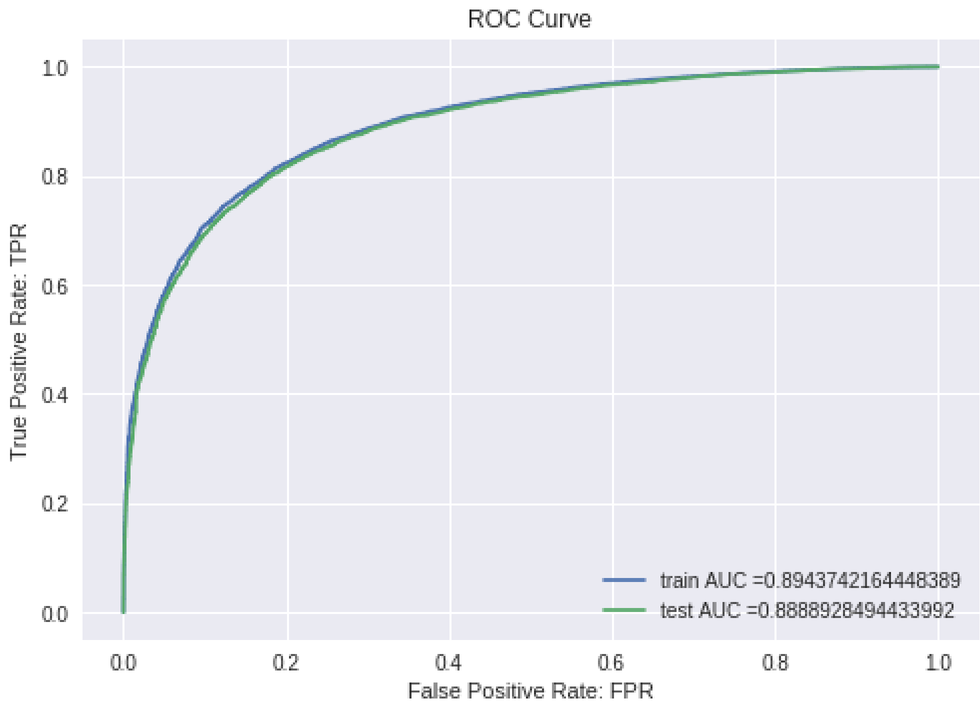
```
parameters = {'alpha':alpha}
tfidf = GridSearchCV(nb,parameters,cv=3 , scoring='roc_auc')
tfidf.fit(X_train_cv_tfidf,y_train_cv)

train_auc_tfidf = tfidf.cv_results_['mean_train_score']
train_auc_std_tfidf = tfidf.cv_results_['std_train_score']
cv_auc_tfidf = tfidf.cv_results_['mean_test_score']
cv_auc_std_tfidf = tfidf.cv_results_['std_test_score']


plt.plot(alpha, train_auc, label='Train AUC')
plt.gca().fill_between(alpha,train_auc - train_auc_std,train_auc + trai
n_auc_std,alpha=0.2,color='darkblue')


plt.plot(alpha , cv_auc , label='CV AUC')
plt.gca().fill_between(alpha , cv_auc - cv_auc_std , cv_auc + cv_auc_st
d , alpha = 0.3 , color = 'darkorange')
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.xscale('log')
plt.title("ERROR PLOTS")
plt.show()
```

In [44]:
```
print(tfidf.best_params_)
print(tfidf.best_estimator_)
print(tfidf.best_score_)
```

```
{'alpha': 1}
MultinomialNB(alpha=1, class_prior=None, fit_prior=True)
0.8869871780048908
```

In [58]:
```
nb_optimal_tfidf = MultinomialNB(alpha=0.1, class_prior=None, fit_prior
=True)
nb_optimal_tfidf.fit(X_train_cv_tfidf, y_train_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train_cv, nb_optimal_tfi
df.predict_proba(X_train_cv_tfidf)[:,1])
```

```
test_fpr, test_tpr, thresholds = roc_curve(y_test, nb_optimal_tfidf.pre
dict_proba(X_test_tfidf)[:,1])


train_acc_tfidf = auc(train_fpr, train_tpr)
test_acc_tfidf = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("False Positive Rate: FPR")
plt.ylabel("True Positive Rate: TPR")
plt.title("ROC Curve")
plt.show()
```



[5.2.1] Top 10 important features of positive class from SET 2

In [0]:
```
# Please write all the code with proper documentation
```

In [47]:
```
# Please write all the code with proper documentation
# Now we can find log probabilities of different features for both the
 classes
class_features = nb_optimal_tfidf.feature_log_prob_

#  row_0 is for 'negative' class and row_1 is for 'positive' class
negative_features = class_features[0]
positive_features = class_features[1]

# Getting all feature names
feature_names_tfidf = tfidf_vect.get_feature_names()

# Sorting 'positive_features' in descending order using argsort() funct
ion
sorted_positive_features = np.argsort(positive_features)[::-1]


print("\n\nTop 10 Important Features and their log probabilities For Po
sitive Class :\n\n")
for i in list(sorted_positive_features[0:10]):
    print("%s\t -->\t%f  "%(feature_names_tfidf[i],positive_features[i
]))
```

```
Top 10 Important Features and their log probabilities For Positive Clas
s :


not      -->     -4.113322
great    -->     -4.442705
good     -->     -4.501926
coffee   -->     -4.531867
like     -->     -4.581567
tea      -->     -4.619725
love     -->     -4.685195
```

```
product  -->    -4.753869
taste    -->    -4.796337
flavor   -->    -4.804070
```

[5.2.2] Top 10 important features of negative class from SET 2

In [0]:
```
# Please write all the code with proper documentation
```

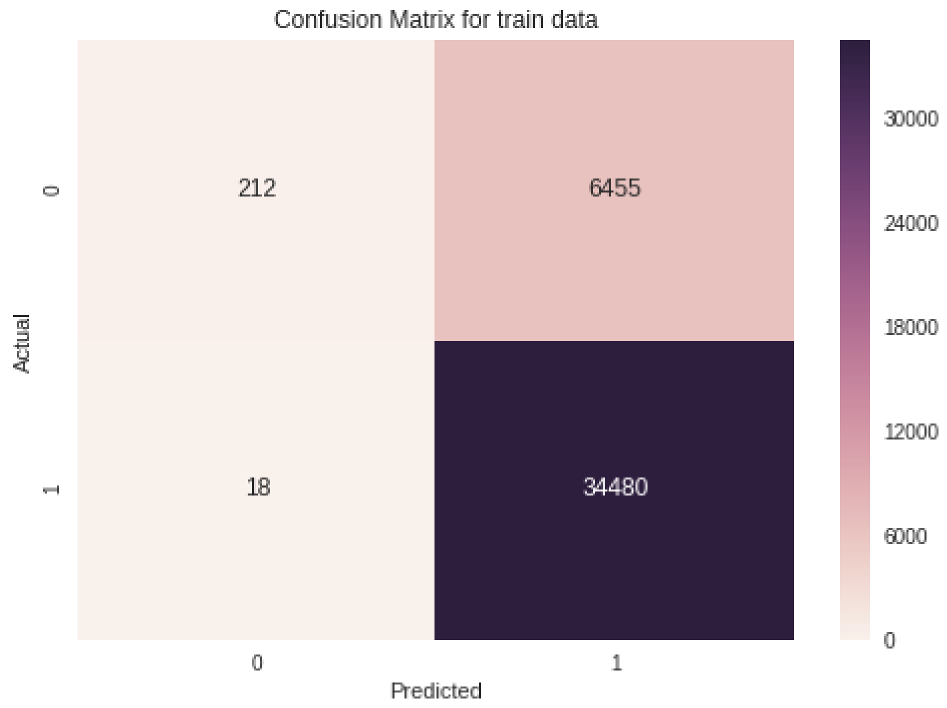
In [49]:
```
# Please write all the code with proper documentation
sorted_negative_features = np.argsort(negative_features)[::-1]
print("Top 10 Important Features and their log probabilities For Negati
ve Class :\n\n")
for i in list(sorted_negative_features[0:10]):
    print("%s\t -->\t%f  "%(feature_names_tfidf[i],negative_features[i
]))
```

```
Top 10 Important Features and their log probabilities For Negative Clas
s :


not      -->    -3.465390
like     -->    -4.271101
taste    -->    -4.395500
product  -->    -4.400324
would    -->    -4.437066
coffee   -->    -4.657750
one      -->    -4.735264
flavor   -->    -4.831041
no       -->    -4.832025
good     -->    -4.881943
```

In [50]:
```
print(y_train_cv.shape)
print(X_train_cv_tfidf.shape)
```

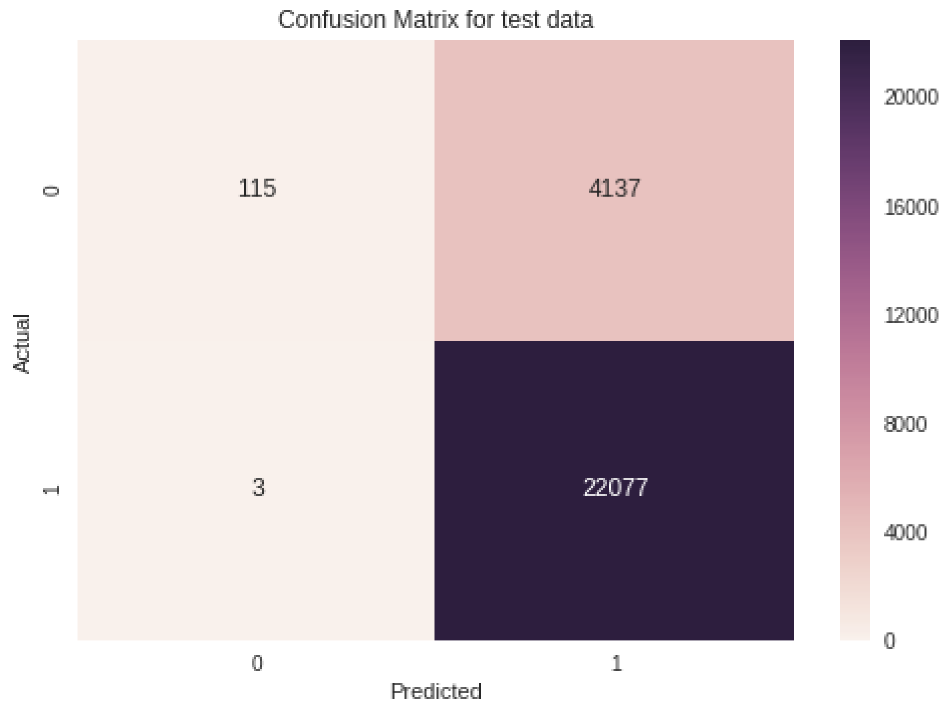
```
(41165,)
(41165, 500)
```

In [51]:
```
conf_matrix = confusion_matrix(y_train_cv, nb_optimal_tfidf.predict(X_t
rain_cv_tfidf))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix for train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [52]:
```
conf_matrix = confusion_matrix(y_test, nb_optimal_tfidf.predict(X_test_
tfidf))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
```

```
plt.title("Confusion Matrix for test data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



[6] Conclusions

In [0]:
```
# Please compare all your models using Prettytable library
```

refered some GitHub sites

In [59]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
```

```
names = ["MultinomialNB for BoW", "MultinomialNB for TFIDF"]

optimal_alpha = [0.1,1]


train_acc_bow = auc(train_fpr, train_tpr)
test_acc_bow = auc(test_fpr, test_tpr)

train_acc = [train_acc_bow,train_acc_tfidf]

test_acc = [test_acc_bow,test_acc_tfidf]

numbering = [1,2]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Best Alpha",optimal_alpha)
ptable.add_column("Training Accuracy",train_acc)
ptable.add_column("Test Accuracy",test_acc)

# Printing the Table
print(ptable)
```

```
+-------+-------------------------+------------+--------------------+--
------------------+
| S.NO. |          MODEL          | Best Alpha | Training Accuracy  |
Test Accuracy    |
+-------+-------------------------+------------+--------------------+--
------------------+
|   1   |  MultinomialNB for BoW  |    0.1     | 0.8943742164448389 |
0.8888928494433992 |
|   2   | MultinomialNB for TFIDF |     1      | 0.8943742164448389 |
0.8888928494433992 |
+-------+-------------------------+------------+--------------------+--
------------------+
```