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
```

```
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 [3]:

```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2
F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.co
m%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.re
adonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&respon
se_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [4]:

```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% |████████████████████████████████| 993kB 16.1MB/s
```

```
  Building wheel for PyDrive (setup.py) ... done
```

In [5]:
```
link="https://drive.google.com/open?id=1a_hWJTh8UJg1u23-DiaEQYvdGemXh9NA"
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1a_hWJTh8UJg1u23-DiaEQYvdGemXh9NA
```

In [0]:
```
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('database.sqlite')
con=sqlite3.connect('database.sqlite')
```

In [7]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 da
ta points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3
LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<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 (525814, 10)
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
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 [9]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[9]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUN
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 [10]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[10]:

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

In [11]:
```
display['COUNT(*)'].sum()
```

Out[11]: 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 [12]:

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

Out[12]:

	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 [14]:

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

Out[14]:

```
(364173, 10)
```

In [15]:

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

Out[15]:

```
69.25890143662969
```

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 [16]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[16]:

	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 [18]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

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

```
(364171, 10)
```

Out[18]:
```
1    307061
0     57110
```

```
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 [19]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
```

```
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks
is good, but I prefer bolder taste.... imagine my surprise when I ordered 2
boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit
that Amazon agreed to credit me for cost plus part of shipping, but geez, 2
years expired!!!  I'm hoping to find local San Diego area shoppe that carrie
s pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today's Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70's it
was poisonous until they figured out a way to fix that. I still like it but
it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the
excellence of this product.<br /><br />Thick, delicious.  Perfect.  3 ingred
ients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No ga
rbage.<br /><br />Have numerous friends & family members hooked on this stuf
f.  My husband & son, who do NOT like "sugar free" prefer this over major la
bel regular syrup.<br /><br />I use this as my SWEETENER in baking: cheeseca
kes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...
<br /><br />Can you tell I like it? :)
==================================================
```

In [20]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
```

In [21]:

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

```python
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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks
is good, but I prefer bolder taste.... imagine my surprise when I ordered 2
boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit
that Amazon agreed to credit me for cost plus part of shipping, but geez, 2
years expired!!!  I'm hoping to find local San Diego area shoppe that carrie
s pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today's Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70's it
was poisonous until they figured out a way to fix that. I still like it but
it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the
excellence of this product.Thick, delicious.  Perfect.  3 ingredients: Wate
r, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.Have
```

numerous friends & family members hooked on this stuff. My husband & son, w
ho do NOT like "sugar free" prefer this over major label regular syrup.I use
this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpki
n pies, etc... Unbelievably delicious...Can you tell I like it? :)

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 [23]:

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

```
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today is Food industrie
s have convinced the masses that Canola oil is a safe and even better oil th
an olive or virgin coconut, facts though say otherwise. Until the late 70 is
it was poisonous until they figured out a way to fix that. I still like it b
ut it could be better.
==================================================
```

In [24]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/408403
9
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
```

hales, India, drooping roses: i love all the new words this book introduce
s and the silliness of it all. this is a classic book i am willing to bet
my son will STILL be able to recite from memory when he is in college

In [25]:

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

Great ingredients although chicken should have been 1st rather than chicken
broth the only thing I do not think belongs in it is Canola oil Canola or ra
peseed is not someting a dog would ever find in nature and if it did find ra
peseed in nature and eat it it would poison them Today is Food industries ha
ve convinced the masses that Canola oil is a safe and even better oil than o
live or virgin coconut facts though say otherwise Until the late 70 is it wa
s poisonous until they figured out a way to fix that I still like it but it
could be better

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <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', 'ours',
'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he'
, 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becau
se', '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', 'how', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d'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', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [27]:
```
# 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 i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 364171/364171 [02:28<00:00, 2454.23it/s]
```

In [28]:
```
preprocessed_reviews[1500]
```

Out[28]:
```
'great ingredients although chicken rather chicken broth thing not think bel
ongs canola oil canola rapeseed not someting dog would ever find nature find
rapeseed nature eat would poison today food industries convinced masses cano
la oil safe even better oil olive virgin coconut facts though say otherwise
late poisonous figured way fix still like could better'
```

In [29]:
```
link="https://drive.google.com/open?id=1oLWdBJFQfO3Z0WbknKE2irIDynEQriD0"
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1oLWdBJFQfO3Z0WbknKE2irIDynEQriD0
```

In [0]:
```
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('final.sqlite')
con=sqlite3.connect('final.sqlite')
```

In [0]:
```
cleaned_reviews=pd.read_sql_query("select * from Reviews", con)
```

In [32]:
```
positive_reviews=cleaned_reviews[cleaned_reviews['Score']==1].sample(n=50000)
negitive_reviews=cleaned_reviews[cleaned_reviews['Score']==0].sample(n=50000)
final_reviews=pd.concat([positive_reviews, negitive_reviews])
final_reviews = final_reviews.sample(frac=1).reset_index(drop=True)
final_reviews.head()
```

Out[32]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumera
0	263728	285855	B000EVOSHG	A139URULUIF1V8	anonymous	0
1	220268	238762	B003P7U7J4	AVWX07T4NAFZ3	Art F. Jannicelli	1
2	164157	178046	B000YVGMAW	A3IJFRX2M22WD0	Marguerite Steffan "Harley55"	6
3	265435	287727	B004J402A6	A343ISEHQWVIS6	Peggy S. Lombardo	1
4	371426	401662	B000EUD6AM	A33QNTGCD2YUWE	Christophocles	24

In [33]:
```
final_reviews['Time']=pd.to_datetime(final_reviews['Time'], unit='s')
final_reviews=final_reviews.sort_values(by="Time")
final_reviews.head()
```

Out[33]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumera
83256	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
19145	346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19
22593	1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7
28924	121041	131217	B00004RAMX	A5NQLNC6QPGSI	Kim Nason	7
62731	138000	149768	B00004S1C5	A7P76IGRZZBFJ	E. Thompson	18

					"Sooooooper Genius"	

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott',
'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/sta
ble/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
```

```
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final
_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_featu
re_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final
_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able f
ind', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolut
ely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [0]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZP
Y
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negati
ve300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v =
True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderfu
l', 0.9946032166481018), ('excellent', 0.9944332838058472), ('especially',
0.9941144585609436), ('baked', 0.9940600395202637), ('salted', 0.99404722452
1637), ('alternative', 0.9937226176261902), ('tasty', 0.9936816692352295),
('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcor
```

```
n', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.999245107173
9197), ('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('ame
rican', 0.9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.9991
567134857178)]
```

In [0]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'sh
ipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'remov
ed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows',
'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'lik
e', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'win
dow', 'everybody', 'asks', 'bought', 'made']
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might
need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|
```

```
█████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:
```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```python
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████
██████| 4986/4986 [00:20<00:00, 245.63it/s]
```

[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 absolute values of `coef_` 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]:
```
new_reviews=final_reviews['cleaned_text']
labels=final_reviews['Score']
```

In [35]:
```
from sklearn.model_selection import train_test_split
x_train,x_train2,y_train,y_train2=train_test_split(new_reviews, labels, test_size=0.4, random_state=0, shuffle=True)
print(len(x_train), len(x_train2), len(y_train), len(y_train2))
```

```
60000 40000 60000 40000
```

In [36]:
```
x_cv,x_test,y_cv,y_test=train_test_split(x_train2,y_train2, test_size=0.5,random_state=0, shuffle=True)
print(len(x_cv), len(y_cv), len(x_test), len(y_test))
```

```
20000 20000 20000 20000
```

In [0]:
```
count_vect=CountVectorizer(ngram_range=(1,1))
```

```
train_reviews=count_vect.fit_transform(x_train)
cv_reviews=count_vect.transform(x_cv)
test_reviews=count_vect.transform(x_test)
```

In [61]:

```
from sklearn.naive_bayes import MultinomialNB
import math
from tqdm import tqdm_notebook as tqdm
alpha1=[math.pow(10,-4), math.pow(10,-3), math.pow(10,-2),math.pow(10,-1), mat
h.pow(10,0), math.pow(10,1), math.pow(10,2), math.pow(10,3),math.pow(10,4)]
train_aucs=[]
for i in tqdm(alpha1):
  clf=MultinomialNB(alpha=i)
  clf.fit(train_reviews, y_train)
  prob=(clf.predict_proba(train_reviews))[:,1]
  fpr, tpr, thre=roc_curve(y_train,prob)
  roc_auc=auc(fpr, tpr)
  train_aucs.append(roc_auc)
print(train_aucs)
```

```
[0.97451312600819, 0.973400336423939, 0.9710618041251198, 0.966002309444985
7, 0.9561061624469005, 0.9447045835279724, 0.9265829247460396, 0.89368764318
73219, 0.8245712863563817]
```

In [62]:

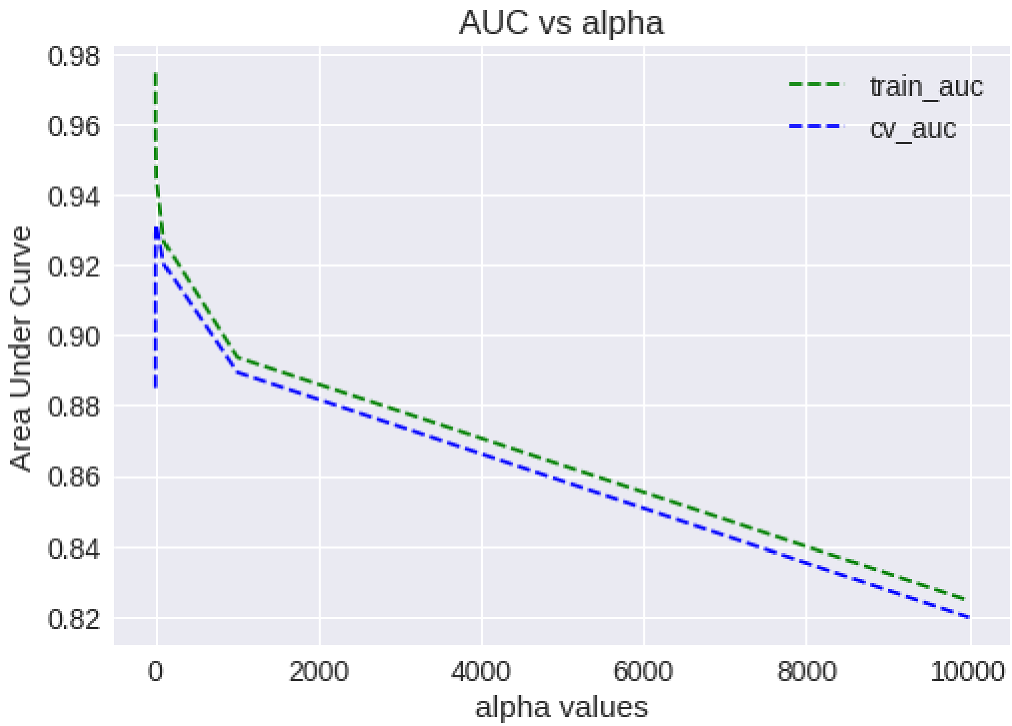
```
from sklearn.naive_bayes import MultinomialNB
import math
from tqdm import tqdm_notebook as tqdm
alpha1=[math.pow(10,-4), math.pow(10,-3), math.pow(10,-2),math.pow(10,-1), mat
h.pow(10,0), math.pow(10,1), math.pow(10,2), math.pow(10,3),math.pow(10,4)]
aucs=[]
for i in tqdm(alpha1):
  clf=MultinomialNB(alpha=i)
  clf.fit(train_reviews, y_train)
  prob=(clf.predict_proba(cv_reviews))[:,1]
  fpr, tpr, thre=roc_curve(y_cv,prob)
  roc_auc=auc(fpr, tpr)
  aucs.append(roc_auc)
print(aucs)
```

```
[0.8849899878809718, 0.8980298388668468, 0.9108197618964566, 0.9219461286873
112, 0.9288493565506436, 0.9316482566456246, 0.9201893026134473, 0.889423237
7006425, 0.8197882478227672]
```

In [63]:

```
import seaborn as sns
```

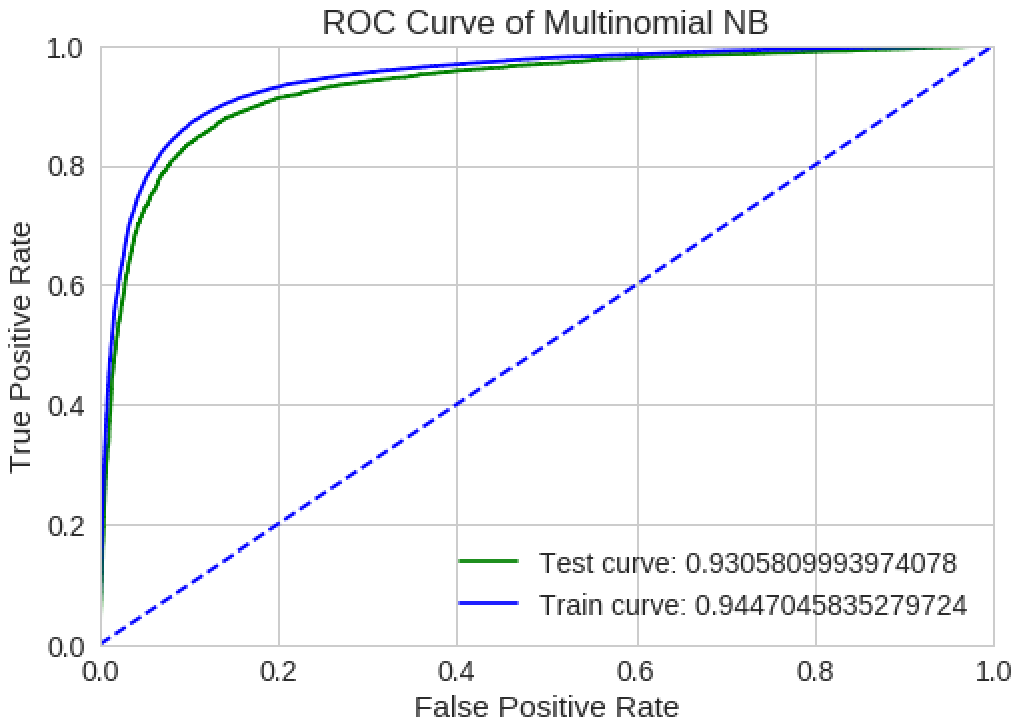
```
plt.plot(alpha1,train_aucs,'g--', label='train_auc')
plt.plot(alpha1,aucs,'b--', label='cv_auc')
sns.set_style('whitegrid')
plt.xlabel('alpha values')
plt.ylabel('Area Under Curve')
plt.legend()
plt.title('AUC vs alpha')
plt.show()
```



From the above above observation we can consider alpha value as 1 or 10

In [64]:
```
clf=MultinomialNB(alpha=10)
clf.fit(train_reviews, y_train)
predict=clf.predict(test_reviews)
prob=(clf.predict_proba(test_reviews))[:,1]
fpr, tpr, thre=roc_curve(y_test,prob)
roc_auc_test=auc(fpr, tpr)
prob2=(clf.predict_proba(train_reviews))[:,1]
fpr1, tpr1, thre1=roc_curve(y_train, prob2)
roc_auc_train=auc(fpr1, tpr1)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'Test curve: '+str(roc_auc_test))
```

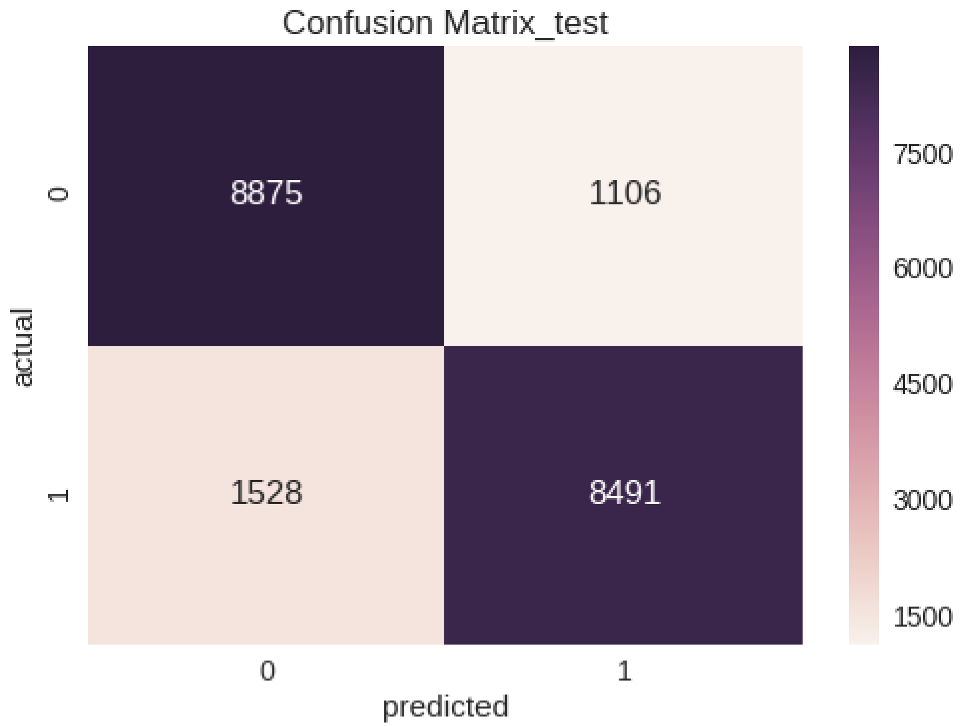
```
plt.plot(fpr1, tpr1, 'b', label = 'Train curve: '+str(roc_auc_train))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
sns.set_style('whitegrid')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of Multinomial NB')
plt.show()
```

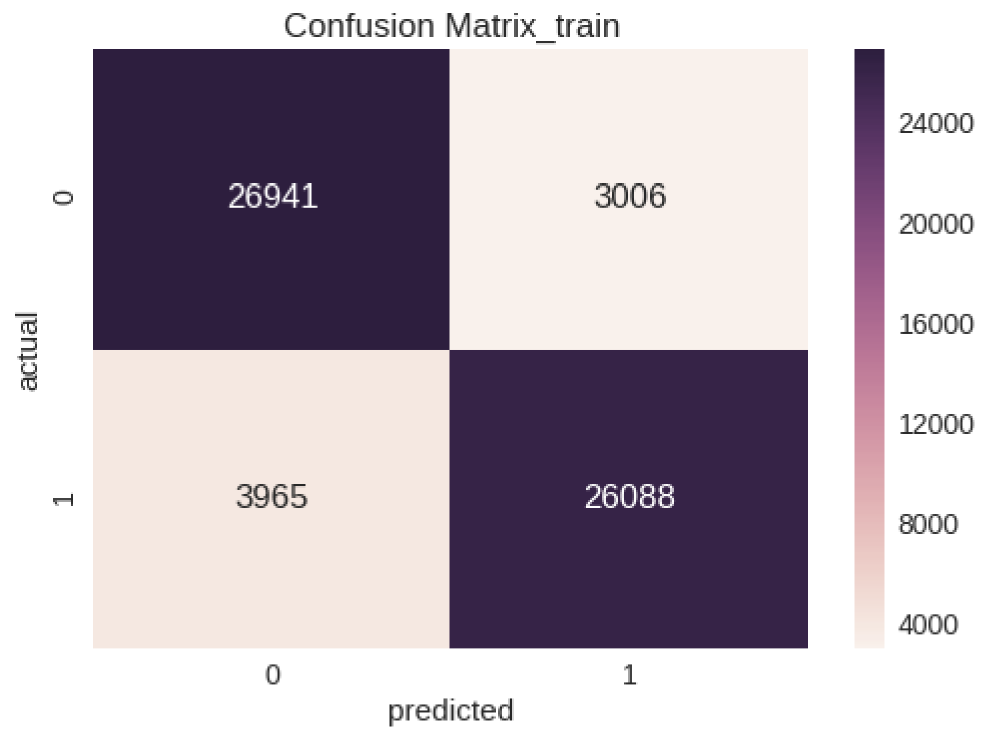


In [65]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
lab=[0,1]
predict2=clf.predict(train_reviews)
cm2=confusion_matrix(y_train, predict2, labels=lab)
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
df_cm2=pd.DataFrame(cm2)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, fmt='g')
plt.xlabel('predicted')
plt.ylabel('actual')
```

```
plt.title('Confusion Matrix_test')
plt.show()
sns.set(font_scale=1.4)
sns.heatmap(df_cm2, annot=True, fmt='g')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix_train')
plt.show()
```





In [66]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      9981
           1       0.88      0.85      0.87     10019

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.87      0.87      0.87     20000
weighted avg       0.87      0.87      0.87     20000
```

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

In [68]:
```python
count_features=count_vect.get_feature_names()
y_train.value_counts()
```

Out[68]:
```
1    30053
0    29947
Name: Score, dtype: int64
```

In [77]:
```
new_df=pd.DataFrame(data=clf.feature_log_prob_, columns=count_features)
new_df=new_df.T
new_df.shape
new_df.head()
```

Out[77]:

	0	1
aa	-11.670692	-11.735480
aaa	-12.076158	-11.735480
aaaaa	-11.980847	-11.997844
aaaaaaarrrrrggghhh	-11.980847	-11.997844
aaaaaahhhhhyaaaaaa	-11.980847	-11.997844

In [78]:
```
new_df_pos=new_df[1].sort_values(ascending=False)
new_df_pos[0:10]
```

Out[78]:
```
not        -4.071061
like       -4.910939
good       -5.027208
great      -5.076679
one        -5.247211
taste      -5.303901
product    -5.405074
love       -5.414850
coffee     -5.420539
flavor     -5.429626
Name: 1, dtype: float64
```

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

In [79]:
```
new_df_neg=new_df[0].sort_values(ascending=False)
new_df_neg[0:10]
```

Out[79]:
```
not        -3.579371
like       -4.702218
product    -4.966850
would      -4.978691
taste      -5.008497
one        -5.179362
good       -5.426655
```

```
no         -5.450500
flavor     -5.475063
coffee     -5.480650
Name: 0, dtype: float64
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [0]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,1))
train_reviews=tf_idf_vect.fit_transform(x_train)
cv_reviews=tf_idf_vect.transform(x_cv)
test_reviews=tf_idf_vect.transform(x_test)
```

In [81]:
```
from sklearn.naive_bayes import MultinomialNB
import math
from tqdm import tqdm_notebook as tqdm
alpha1=[math.pow(10,-4), math.pow(10,-3), math.pow(10,-2),math.pow(10,-1), mat
h.pow(10,0), math.pow(10,1), math.pow(10,2), math.pow(10,3),math.pow(10,4)]
train_aucs=[]
for i in tqdm(alpha1):
  clf=MultinomialNB(alpha=i)
  clf.fit(train_reviews, y_train)
  prob=(clf.predict_proba(train_reviews))[:,1]
  fpr, tpr, thre=roc_curve(y_train,prob)
  roc_auc=auc(fpr, tpr)
  train_aucs.append(roc_auc)
print(train_aucs)
```

```
[0.9802634189554931, 0.9795193477442752, 0.9775577971776136, 0.9719468729986
291, 0.9606836045113833, 0.9502773675879173, 0.9402081850497687, 0.925209829
349345, 0.9115085121415674]
```

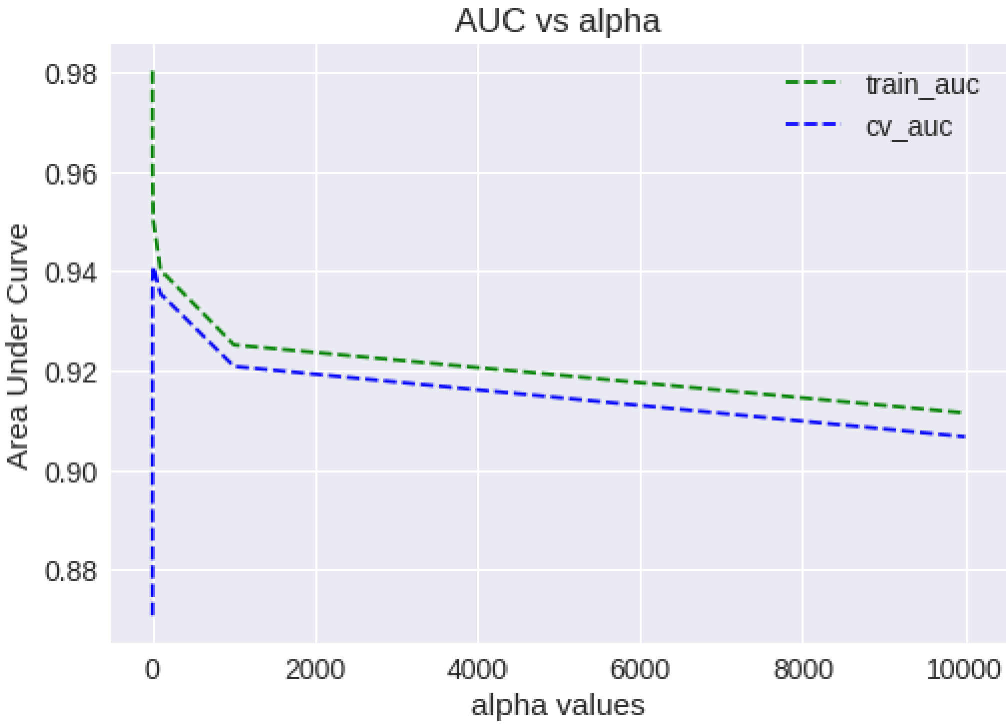
In [82]:
```
from sklearn.naive_bayes import MultinomialNB
import math
from tqdm import tqdm_notebook as tqdm
alpha1=[math.pow(10,-4), math.pow(10,-3), math.pow(10,-2),math.pow(10,-1), mat
h.pow(10,0), math.pow(10,1), math.pow(10,2), math.pow(10,3),math.pow(10,4)]
aucs=[]
for i in tqdm(alpha1):
  clf=MultinomialNB(alpha=i)
  clf.fit(train_reviews, y_train)
  prob=(clf.predict_proba(cv_reviews))[:,1]
  fpr, tpr, thre=roc_curve(y_cv,prob)
```

```
  roc_auc=auc(fpr, tpr)
  aucs.append(roc_auc)
print(aucs)
```

```
[0.8708536650539963, 0.8884970396865374, 0.9089346391716947, 0.9272968910708
332, 0.938020316973232, 0.9410272528527879, 0.9354565240662076, 0.9208825085
492433, 0.9067175842395669]
```

In [83]:

```
import seaborn as sns
plt.plot(alpha1,train_aucs,'g--', label='train_auc')
plt.plot(alpha1,aucs,'b--', label='cv_auc')
sns.set_style('whitegrid')
plt.xlabel('alpha values')
plt.ylabel('Area Under Curve')
plt.legend()
plt.title('AUC vs alpha')
plt.show()
```

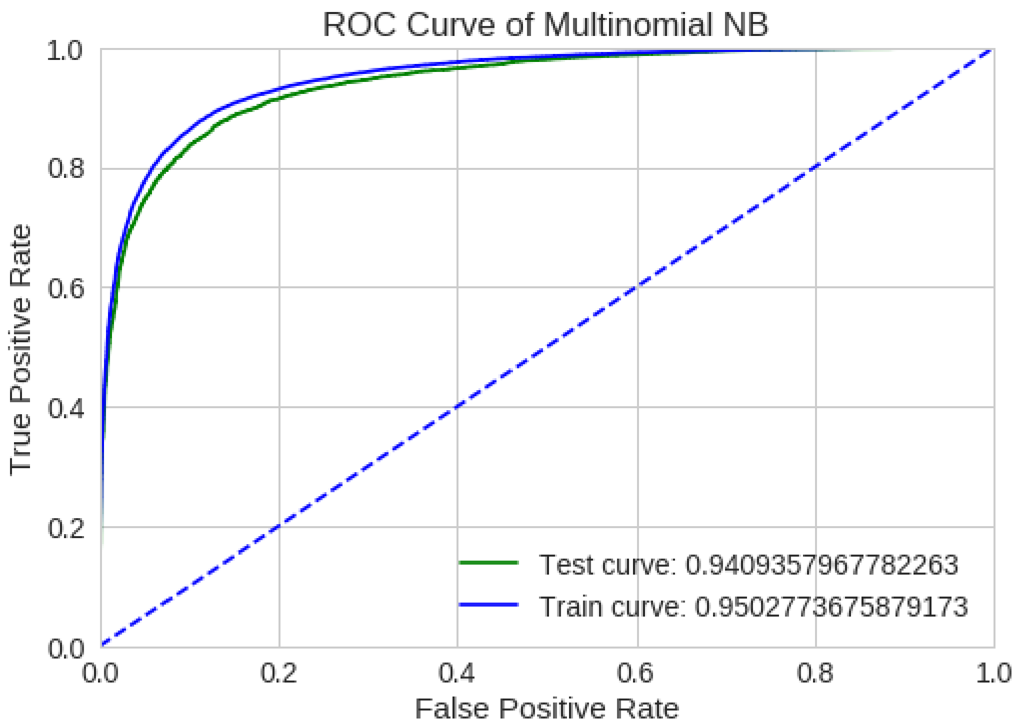


Here consider alpha is 10

In [84]:

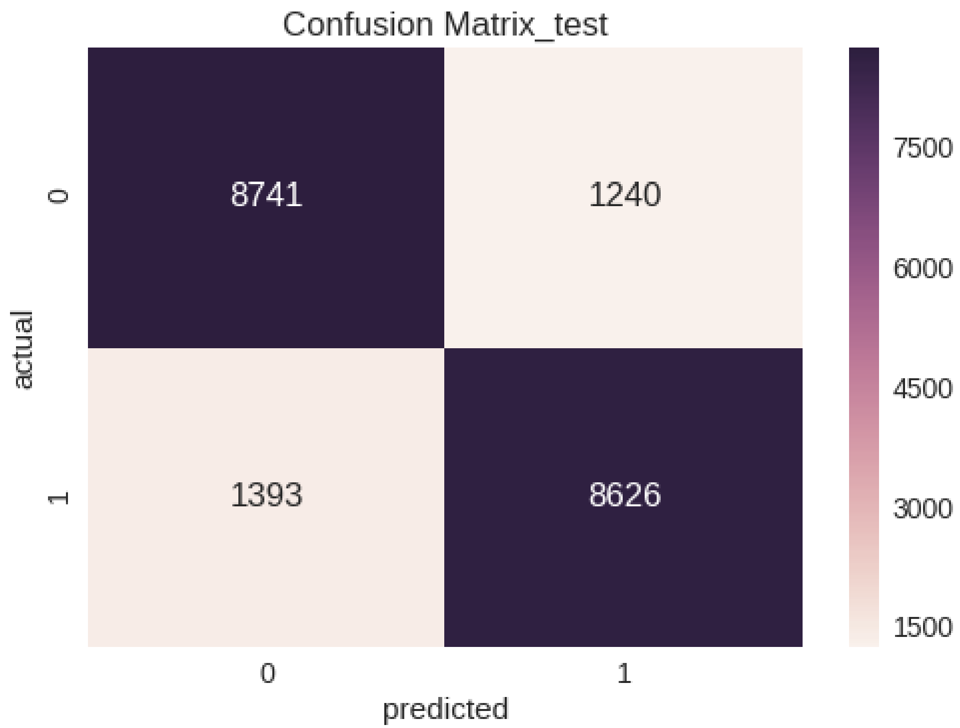
```
clf=MultinomialNB(alpha=10)
clf.fit(train_reviews, y_train)
```

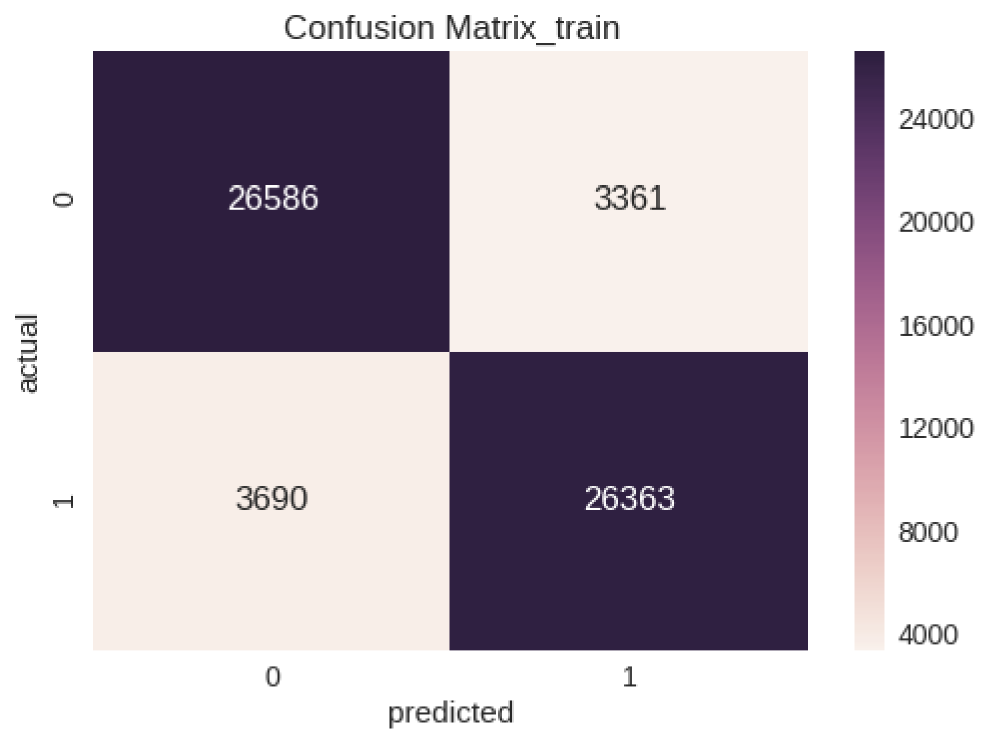
```
predict=clf.predict(test_reviews)
prob=(clf.predict_proba(test_reviews))[:,1]
fpr, tpr, thre=roc_curve(y_test,prob)
roc_auc_test=auc(fpr, tpr)
prob2=(clf.predict_proba(train_reviews))[:,1]
fpr1, tpr1, thre1=roc_curve(y_train, prob2)
roc_auc_train=auc(fpr1, tpr1)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'Test curve: '+str(roc_auc_test))
plt.plot(fpr1, tpr1, 'b', label = 'Train curve: '+str(roc_auc_train))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
sns.set_style('whitegrid')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of Multinomial NB')
plt.show()
```



In [85]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
lab=[0,1]
```

```
predict2=clf.predict(train_reviews)
cm2=confusion_matrix(y_train, predict2, labels=lab)
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
df_cm2=pd.DataFrame(cm2)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, fmt='g')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix_test')
plt.show()
sns.set(font_scale=1.4)
sns.heatmap(df_cm2, annot=True, fmt='g')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix_train')
plt.show()
```





In [86]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      9981
           1       0.87      0.86      0.87     10019

   micro avg       0.87      0.87      0.87     20000
   macro avg       0.87      0.87      0.87     20000
weighted avg       0.87      0.87      0.87     20000
```

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

In [88]:
```
tfidf_features=tf_idf_vect.get_feature_names()
new_df=pd.DataFrame(data=clf.feature_log_prob_, columns=tfidf_features)
new_df=new_df.T
new_df.shape
new_df.head()
```

Out[88]:

	0	1
aa	-10.918469	-10.998070
aaa	-11.048958	-10.960121
aaaaa	-11.013016	-11.041407
aaaaaaarrrrrggghhh	-11.034132	-11.041407
aaaaaahhhhhyaaaaaa	-11.044057	-11.041407

In [89]:

```
new_df_pos=new_df[1].sort_values(ascending=False)
new_df_pos[0:10]
```

Out[89]:

```
not        -6.403970
great      -6.504923
good       -6.666297
love       -6.782427
coffee     -6.809501
like       -6.812651
tea        -6.838717
product    -7.010814
taste      -7.014976
flavor     -7.031430
Name: 1, dtype: float64
```

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

In [90]:

```
new_df_neg=new_df[0].sort_values(ascending=False)
new_df_neg[0:10]
```

Out[90]:

```
not        -5.806498
like       -6.549219
product    -6.669252
taste      -6.675804
would      -6.740129
coffee     -6.902300
one        -6.954017
flavor     -7.050860
no         -7.110457
good       -7.123248
Name: 0, dtype: float64
```

[6] Conclusions

In [92]:
```
data=[['BOW',10, 0.93],['TF-IDF',10, 0.940]]
new_df=pd.DataFrame(data=data, columns=['vectorizer','Alpha(roc)', 'AUC'])
new_df
```

Out[92]:

	vectorizer	Alpha(roc)	AUC
0	BOW	10	0.93
1	TF-IDF	10	0.94