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 [1]:

```
%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 [2]:

```
# 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 50
0000 data 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 Sco
re != 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 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 (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:
```
display['COUNT(*)'].sum()
```

Out[6]: 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 [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	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 [8]:
```
#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 [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[9]:
```
(364173, 10)
```

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]:
```
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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)
```

```
display.head()
```

Out[11]:

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

In [12]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:

```
#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()
```

```
(364171, 10)
```

Out[13]:

```
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 [14]:

```
# 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 c
ar as we're driving along and he always can sing the refrain. he's lear
ned about whales, India, drooping roses:  i love all the new words this
```

```
book  introduces 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.  Starb
ucks 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 s
hipping, but geez, 2 years expired!!!  I'm hoping to find local San Die
go area shoppe that carries pods so that I can try something different
than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than ch
icken 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. Tod
ay's Food industries have convinced the masses that Canola oil is a saf
e and even better oil than olive or virgin coconut, facts though say ot
herwise. 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 touc
h the excellence of this product.<br /><br />Thick, delicious.  Perfec
t.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No
chemicals.  No garbage.<br /><br />Have numerous friends & family membe
rs hooked on this stuff.  My husband & son, who do NOT like "sugar fre
e" prefer this over major label regular syrup.<br /><br />I use this as
my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin p
ies, etc... Unbelievably delicious...<br /><br />Can you tell I like i
t? :)
==================================================
```

In [15]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the c
ar as we're driving along and he always can sing the refrain. he's lear
ned about whales, India, drooping roses:  i love all the new words this
book  introduces 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 [16]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the c
ar as we're driving along and he always can sing the refrain. he's lear
ned about whales, India, drooping roses:  i love all the new words this
book  introduces 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.  Starb
ucks 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 s
hipping, but geez, 2 years expired!!!  I'm hoping to find local San Die
go area shoppe that carries pods so that I can try something different
than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than ch
icken 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. Tod
ay's Food industries have convinced the masses that Canola oil is a saf
e and even better oil than olive or virgin coconut, facts though say ot
herwise. 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 touc
h the excellence of this product.Thick, delicious.  Perfect.  3 ingredi
ents: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  N
o garbage.Have numerous friends & family members hooked on this stuff.
My husband & son, who do NOT like "sugar free" prefer this over major l
abel regular syrup.I use this as my SWEETENER in baking: cheesecakes, w
hite brownies, muffins, pumpkin pies, etc... Unbelievably delicious...C
an you tell I like it? :)
```

In [17]:

```python
# 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 [18]:

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

```
Great ingredients although, chicken should have been 1st rather than ch
icken 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. Tod
ay is Food industries have convinced the masses that Canola oil is a sa
fe and even better oil than olive or virgin coconut, facts though say o
therwise. Until the late 70 is it was poisonous until they figured out
a way to fix that. I still like it but it could be better.
==================================================
```

In [19]:

```
#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)
```

```
this witty little book makes my son laugh at loud. i recite it in the c
ar as we're driving along and he always can sing the refrain. he's lear
ned about whales, India, drooping roses:  i love all the new words this
book  introduces 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 [20]:

```
#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 chi
cken broth the only thing I do not think belongs in it is Canola oil Ca
```

```
nola 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 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 is it was poisonous until they figured out a way to f
ix that I still like it but it could be better
```

In [21]:

```
# 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"])
```

In [22]:

```
# 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%|██████████| 364171/364171 [02:44<00:00, 2218.90it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great ingredients although chicken rather chicken broth thing not thin
k belongs canola oil canola rapeseed not someting dog would ever find n
ature find rapeseed nature eat would poison today food industries convi
nced masses canola oil safe even better oil olive virgin coconut facts
though say otherwise late poisonous figured way fix still like could be
tter'
```

In [24]:

```
final['Cleaned_Text']=preprocessed_reviews
```

[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', 'abb
ott', '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-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html
```

```
# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
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.ge
t_feature_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', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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
-negative300.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_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

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', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

[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 list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u 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/re
view
    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]:
```
# 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 v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
# 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 ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored 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/r
eview
    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 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

In [25]:
```
final["Score"].value_counts()
```

Out[25]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

In [26]:
```
po_p = final[final['Score']==1].sample(n=50000)
po_n = final[final["Score"]==0].sample(n=50000)
final_p =  pd.concat([po_p,po_n])
final_p.head(3)
```

Out[26]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
295563	320168	B003Z6W32E	A1E1T8Y5QVQCIX	Reviewer "TheMax"	0	
87673	95423	B00395DVWM	A2L7M1TT7T843O	Linda Fox	2	
48877	53101	B000EEK4OO	A1TKEUXQNN4BBI	James A. Saksa "CP viewer"	0	

In [27]:

```
# Sorting data based on time
final_p["Time"] = pd.to_datetime(final_p["Time"],unit ="s")
final_p = final_p.sort_values(by = "Time")
final_p.head(5)
```

Out[27]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessI
346116	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	
346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessI
346141	374450	B00004CI84	ACJR7EQF9S6FP	Jeremy Robertson	2	
346102	374408	B00004CI84	A1GB1Q193DNFGR	Bruce Lee Pullen	5	
346078	374383	B00004CI84	A34NBH479RB0E	"dmab6395"	0	

In [28]:
```
x=final_p["Cleaned_Text"].values
y=final_p["Score"].values
print(type(x),type(y))
print(x.shape,y.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
(100000,) (100000,)
```

In [29]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
```

In [30]:
```
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.3, random_stat
e = 42,shuffle = False)
```

```
x_tr,x_cv,y_tr,y_cv = train_test_split(x,y,test_size = 0.3, random_stat
e = 42,shuffle = False)
print('='*50)
print(x_tr.shape,y_tr.shape)
print(x_te.shape,y_te.shape)
print(x_cv.shape,y_cv.shape)
print('='*50)
```

```
==================================================
(70000,) (70000,)
(30000,) (30000,)
(30000,) (30000,)
==================================================
```

[5.1] Applying Naive Bayes on BOW, SET 1

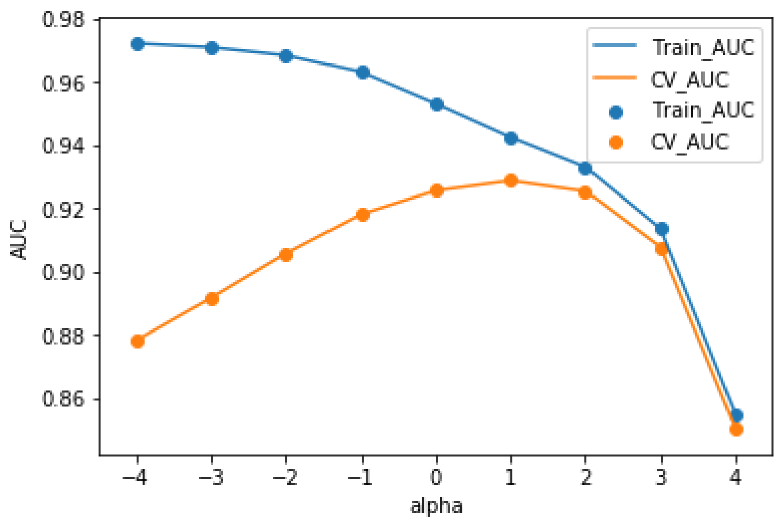
In [32]:
```
# Please write all the code with proper documentation
vec = CountVectorizer()
vec = vec.fit(x_tr)
x_tr_bow = vec.transform(x_tr)
x_cv_bow = vec.transform(x_cv)
x_te_bow = vec.transform(x_te)
print('='*50)
print(x_tr_bow.shape,y_tr.shape)
print(x_te_bow.shape,y_te.shape)
print(x_cv_bow.shape,y_cv.shape)
print('='*50)
```

```
==================================================
(70000, 51616) (70000,)
(30000, 51616) (30000,)
(30000, 51616) (30000,)
==================================================
```

In [34]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

```
import math
tr_auc =[]
cv_auc =[]
log_alpha =[]
alpha =[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for i in tqdm(alpha):
    nai = MultinomialNB(alpha=i,class_prior = [0.5,0.5])
    nai.fit(x_tr_bow,y_tr)
    y_tr_pre = nai.predict_proba(x_tr_bow)[:,1]
    y_cv_pre = nai.predict_proba(x_cv_bow)[:,1]
    tr_auc.append(roc_auc_score(y_tr,y_tr_pre))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pre))
    log_alpha.append(np.log10(i))
plt.plot(log_alpha,tr_auc,label="Train_AUC")
plt.scatter(log_alpha,tr_auc,label="Train_AUC")
plt.plot(log_alpha,cv_auc,label="CV_AUC")
plt.scatter(log_alpha,cv_auc,label="CV_AUC")
plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.show()
```

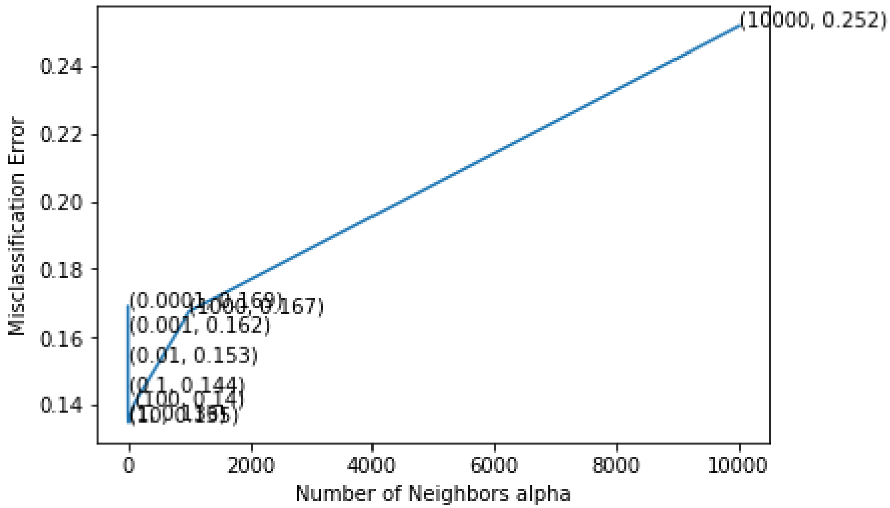
```
100%|██████████| 9/9 [00:01<00:00,  6.57it/s]
```



In [35]:
```
cv_score = []
alpha_val = [10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for alpha in tqdm(alpha_val):
        nai = MultinomialNB(alpha = alpha,class_prior = [0.5,0.5])
        scores = cross_val_score(nai, x_tr_bow, y_tr, cv = 10, scoring
= 'accuracy')
        cv_score.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_score]
# determining the best alpha
optimal_alpha1 = alpha_val[MSE.index(min(MSE))]

print('\nThe optimal number of alpha is %d.' % optimal_alpha1)

plt.plot(alpha_val, MSE)

for xy in zip(alpha_val, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_alpha1)
print("the misclassification error for each alpha value is : ", np.roun
d(MSE,3))
plt.xlabel('Number of Neighbors alpha')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████| 9/9 [00:13<00:00,  1.52s/it]
```

```
The optimal number of alpha is 10.
==================================================

The optimal number of neighbors is 10.
the misclassification error for each alpha value is :  [0.169 0.162 0.1
53 0.144 0.136 0.135 0.14  0.167 0.252]
```



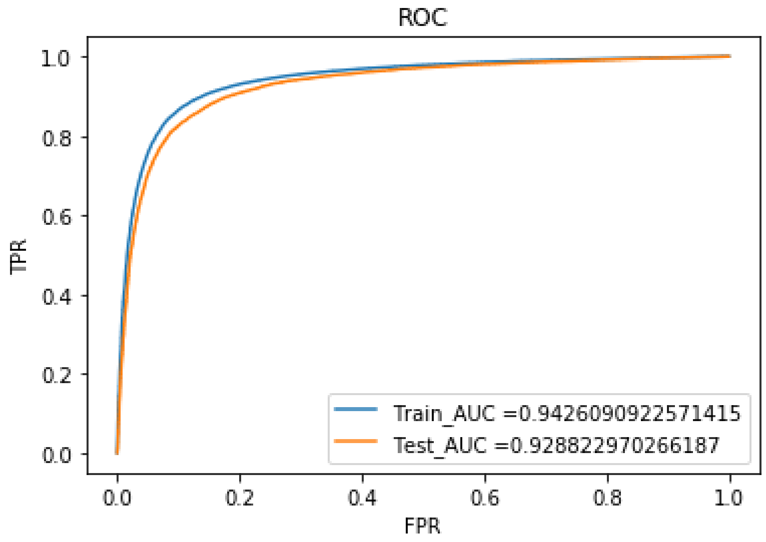
==

In [36]:
```
opt_model1 = MultinomialNB(alpha = optimal_alpha1,class_prior = [0.5,0.
5])
opt_model1.fit(x_tr_bow,y_tr)
pred1=opt_model1.predict(x_te_bow)
# evaluate accuracy
acc_bow = accuracy_score(y_te, pred1) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_alpha1, acc_bow))
```

```
The accuracy of the knn classifier for k = 10 is 86.513333%
```

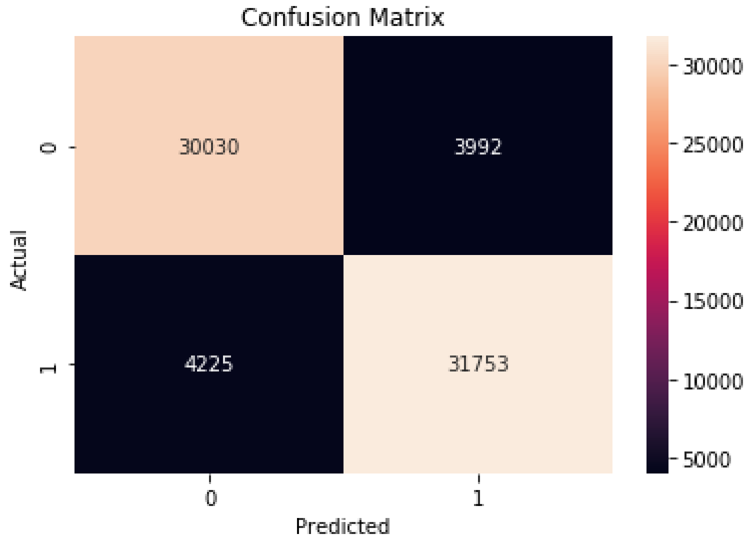
In [38]:
```
#plotting auc curve
tr_fpr,tr_tpr,threshold = roc_curve(y_tr,opt_model1.predict_proba(x_tr_
bow)[:,1])
te_fpr,te_tpr,threshold = roc_curve(y_te,opt_model1.predict_proba(x_te_
bow)[:,1])
Auc1=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr, tr_tpr, label="Train_AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="Test_AUC ="+str(auc(te_fpr, te_tpr)))
```

```
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



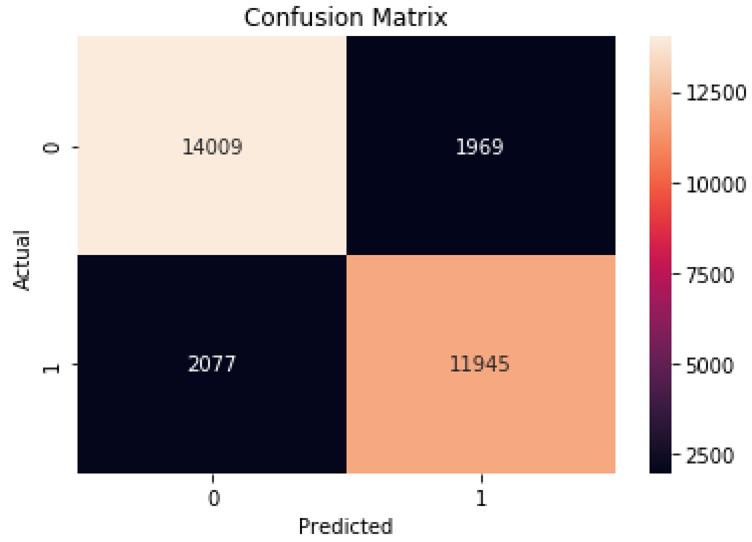
In [39]:

```
#Confusion Matrix
import seaborn as sb
con_mat = confusion_matrix(y_tr,opt_model1.predict(x_tr_bow))
c_l = [0,1]
df_con_matrix = pd.DataFrame(con_mat, index=c_l, columns=c_l)
sb.heatmap(df_con_matrix, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [40]:

```
#Confusion Matrix
import seaborn as sb
con_mat = confusion_matrix(y_te,opt_model1.predict(x_te_bow))
c_l = [0,1]
df_con_matrix = pd.DataFrame(con_mat, index=c_l, columns=c_l)
sb.heatmap(df_con_matrix, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [41]:
```
from sklearn.metrics import classification_report
print(classification_report(y_te, pred1))
```

```
              precision    recall  f1-score   support

           0       0.87      0.88      0.87     15978
           1       0.86      0.85      0.86     14022

   micro avg       0.87      0.87      0.87     30000
   macro avg       0.86      0.86      0.86     30000
weighted avg       0.87      0.87      0.87     30000
```

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

In [42]:
```
#Get all feature names
fe_names = vec.get_feature_names()
print(fe_names[:10])
```

```
['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
aaaaa', 'aaaaaaaaagghh', 'aaaaaaarrrrrggghhh', 'aachen', 'aachener', 'a
ad']
```

In [43]:
```
#log probabilites for feature names
log_proba = (opt_model1.feature_log_prob_)[:]
#dataframe contains feature_names and probabilities
prob_df = pd.DataFrame(log_proba,columns = fe_names)
print(prob_df.shape)
prob_df
```

```
(2, 51616)
```

Out[43]:

	aa	aaa	aaaa	aaaaa	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
0	-11.493357	-12.091194	-12.186504	-12.091194	-12.186504
1	-11.572055	-12.064531	-11.977520	-11.897477	-12.064531

2 rows × 51616 columns

In [44]:
```
#Transpose matrix
prob_t = prob_df.T
print(prob_t.shape)
```

```
(51616, 2)
```

In [45]:
```
#Top 10 postive important features
print(prob_t[1].sort_values(ascending = False)[:10])
```

```
not        -4.041873
like       -4.900725
good       -4.994039
great      -5.051597
one        -5.231401
taste      -5.288231
tea        -5.332212
flavor     -5.393995
product    -5.422163
```

```
love       -5.424536
Name: 1, dtype: float64
```

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

In [46]:
```
#Top 10 Negative Important features
print(prob_t[0].sort_values(ascending = False)[:10])
```

```
not        -3.578301
like       -4.688909
would      -4.962188
taste      -4.983099
product    -4.984439
one        -5.155028
good       -5.405106
no         -5.439857
flavor     -5.440327
coffee     -5.460391
Name: 0, dtype: float64
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

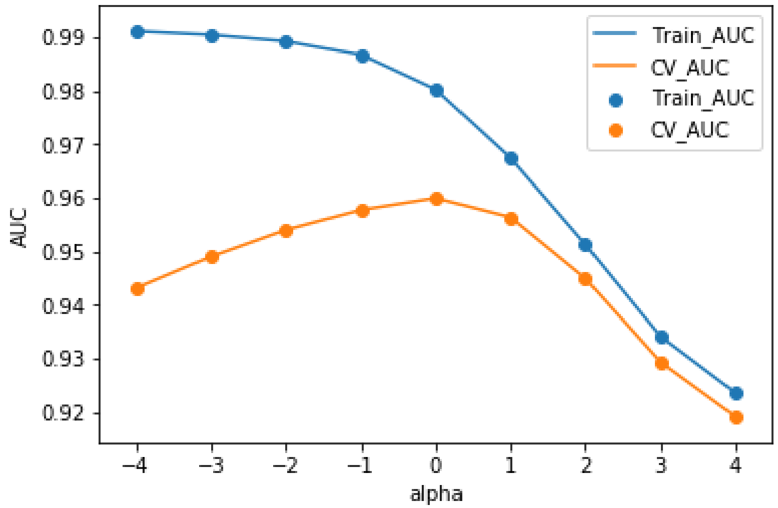
In [47]:
```
tfidf_vec = TfidfVectorizer(ngram_range=(1,2),min_df=5)
tfidf_vec.fit(x_tr)
x_tr_tfidf = tfidf_vec.transform(x_tr)
x_cv_tfidf = tfidf_vec.transform(x_cv)
x_te_tfidf = tfidf_vec.transform(x_te)
print('='*50)
print(x_tr_tfidf.shape,y_tr.shape)
print(x_cv_tfidf.shape,y_cv.shape)
print(x_te_tfidf.shape,y_te.shape)
print('='*50)
```

```
==================================================
(70000, 93535) (70000,)
(30000, 93535) (30000,)
```

```
(30000, 93535) (30000,)
==================================================
```

In [48]:
```
tr_auc =[]
cv_auc =[]
log_alpha1 =[]
alpha =[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for i in tqdm(alpha):
    nai = MultinomialNB(alpha=i,class_prior = [0.5,0.5])
    nai.fit(x_tr_tfidf,y_tr)
    y_tr_pre = nai.predict_proba(x_tr_tfidf)[:,1]
    y_cv_pre = nai.predict_proba(x_cv_tfidf)[:,1]
    tr_auc.append(roc_auc_score(y_tr,y_tr_pre))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pre))
    log_alpha1.append(np.log10(i))
plt.plot(log_alpha1,tr_auc,label="Train_AUC")
plt.scatter(log_alpha1,tr_auc,label="Train_AUC")
plt.plot(log_alpha1,cv_auc,label="CV_AUC")
plt.scatter(log_alpha1,cv_auc,label="CV_AUC")
plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.show()
```

```
100%|██████████| 9/9 [00:01<00:00,  7.16it/s]
```

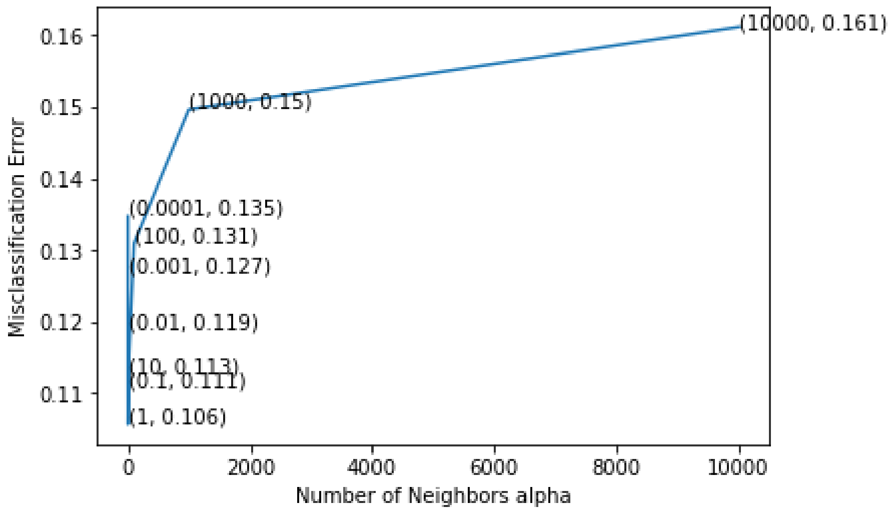


In [49]:
```
cv_score = []
alpha_val = [10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for alpha in tqdm(alpha_val):
        nai = MultinomialNB(alpha = alpha , class_prior = [0.5,0.5])
        scores = cross_val_score(nai, x_tr_tfidf, y_tr, cv = 10, scorin
g = 'accuracy')
        cv_score.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_score]
# determining the best alpha
optimal_alpha2 = alpha_val[MSE.index(min(MSE))]

print('\nThe optimal number of alpha is %d.' % optimal_alpha2)

plt.plot(alpha_val, MSE)

for xy in zip(alpha_val, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_alpha2)
```

```
print("the misclassification error for each alpha value is : ", np.roun
d(MSE,3))
plt.xlabel('Number of Neighbors alpha')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████| 9/9 [00:16<00:00,  1.82s/it]
```

```
The optimal number of alpha is 1.
==================================================

The optimal number of neighbors is 1.
the misclassification error for each alpha value is :  [0.135 0.127 0.1
19 0.111 0.106 0.113 0.131 0.15  0.161]
```




```
==================================================
```

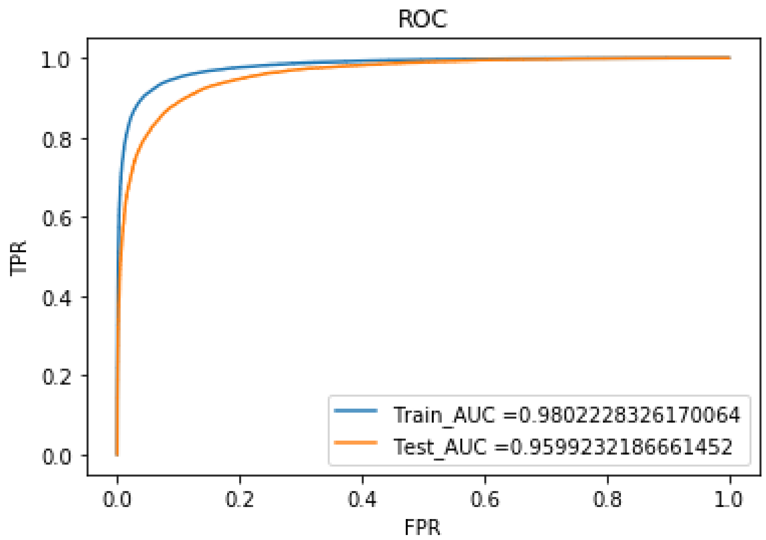
In [50]:
```
opt_model2 = MultinomialNB(alpha=optimal_alpha2,class_prior = [0.5,0.5
])
opt_model2.fit(x_tr_tfidf,y_tr)
pred2 = opt_model2.predict(x_te_tfidf)
# evaluate accuracy
```

```
acc_tfidf = accuracy_score(y_te, pred2) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_alpha2, acc_tfidf))
```

```
The accuracy of the knn classifier for k = 1 is 89.466667%
```

In [52]:

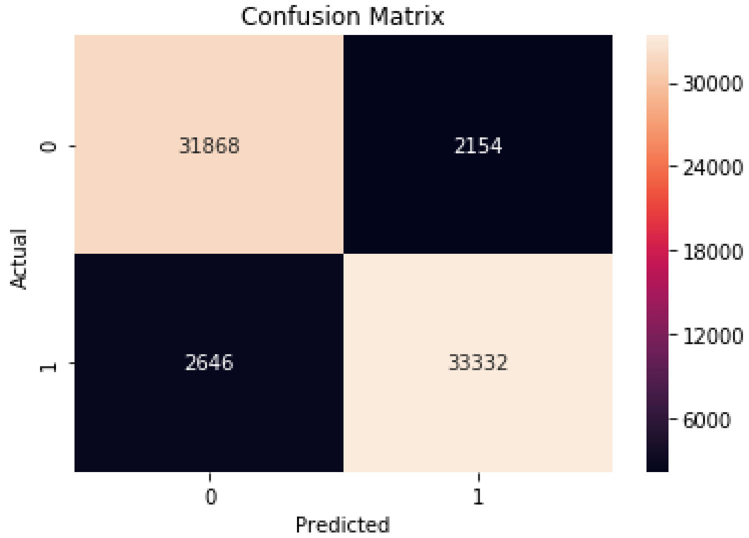
```
#plotting auc curve
tr_fpr,tr_tpr,threshold = roc_curve(y_tr,opt_model2.predict_proba(x_tr_
tfidf)[:,1])
te_fpr,te_tpr,threshold = roc_curve(y_te,opt_model2.predict_proba(x_te_
tfidf)[:,1])
Auc2=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr, tr_tpr, label="Train_AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="Test_AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



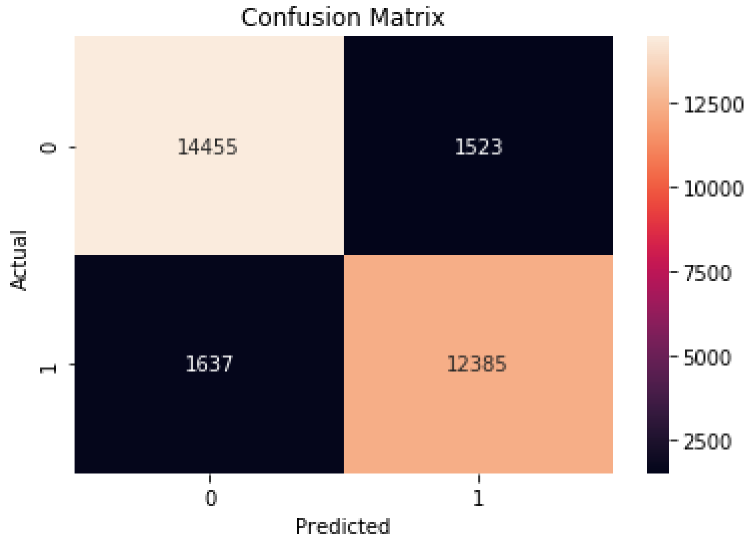
In [53]:

```
#Confusion Matrix
import seaborn as sb
```

```
con_mat = confusion_matrix(y_tr,opt_model2.predict(x_tr_tfidf))
c_l = [0,1]
df_con_matrix = pd.DataFrame(con_mat, index=c_l, columns=c_l)
sb.heatmap(df_con_matrix, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [54]:
```
#Confusion Matrix
import seaborn as sb
con_mat = confusion_matrix(y_te,pred2)
c_l = [0,1]
df_con_matrix = pd.DataFrame(con_mat, index=c_l, columns=c_l)
sb.heatmap(df_con_matrix, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [55]:
```
from sklearn.metrics import classification_report
print(classification_report(y_te, pred2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90     15978
           1       0.89      0.88      0.89     14022

   micro avg       0.89      0.89      0.89     30000
   macro avg       0.89      0.89      0.89     30000
weighted avg       0.89      0.89      0.89     30000
```

In [56]:
```
#Get all feature names
fe_names = tfidf_vec.get_feature_names()
print(fe_names[:10])
```

```
['aa', 'aafco', 'aback', 'abandon', 'abandoned', 'abc', 'abdomen', 'abd
ominal', 'abdominal pain', 'abilities']
```

In [57]:
```
#log probabilites for feature names
```

```
log_proba = (opt_model2.feature_log_prob_)[:]
#dataframe contains feature_names and probabilities
prob_df = pd.DataFrame(log_proba,columns = fe_names)
print(prob_df.shape)
prob_df
```

```
(2, 93535)
```

Out[57]:

	aa	aafco	aback	abandon	abandoned	abc	abdomen	abdominal
0	-11.655973	-12.351628	-11.971863	-12.186593	-11.527043	-11.500020	-12.186218	-11.459725
1	-12.021482	-12.401782	-11.730568	-12.055329	-11.835986	-12.453093	-12.529363	-12.234017

2 rows × 93535 columns

In [58]:

```
#Transpose matrix
prob_t = prob_df.T
print(prob_t.shape)
```

```
(93535, 2)
```

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

In [59]:

```
#Top 10 Postive Important features
print(prob_t[1].sort_values(ascending = False)[:10])
```

```
not       -5.887862
great     -6.036509
good      -6.191399
tea       -6.309932
like      -6.338005
love      -6.342960
coffee    -6.386179
one       -6.541557
flavor    -6.544262
```

```
product   -6.549592
Name: 1, dtype: float64
```

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

In [60]:

```
#Top 10 Negative Important features
print(prob_t[0].sort_values(ascending = False)[:10])
```

```
not        -5.340676
like       -6.081634
product    -6.214238
taste      -6.218731
would      -6.263527
coffee     -6.454798
one        -6.465922
flavor     -6.600242
no         -6.618348
good       -6.667402
Name: 0, dtype: float64
```

[6] Conclusions

In [61]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
comparison = PrettyTable()
comparison.field_names = ["Vectorizer", "Model", "Hyperparameter", "AU
C","Accuracy"]
comparison.add_row(["BOW", 'brute', optimal_alpha1, np.round(float(Auc1
),3),acc_bow])
comparison.add_row(["TFIDF", 'brute', optimal_alpha2, np.round(float(Au
c2),3),acc_tfidf])
print(comparison)
```

```
+------------+-------+----------------+-------+------------------+
| Vectorizer | Model | Hyperparameter |  AUC  |     Accuracy     |
+------------+-------+----------------+-------+------------------+
```

```
|    BOW     | brute |       10       | 0.929 | 86.51333333333334 |
|   TFIDF    | brute |       1        |  0.96 | 89.46666666666667 |
+------------+-------+----------------+-------+-------------------+
```

-> Firstly, naive bayes works with the positive values so we used bow and tfidf -> In BOW we splitted data into three parts train,test and cv -> After using Count_vectorizer we used to fit the train data and applied transform for the train,test and cv -> Then we applied multinomial naive bayes for different alpha values and store the result in an array. -> By using the acu score and log of alpha values we are able to get the maximum hyperparameter value(ALPHA). -> And by using the alpha value we are able to train our model and also able to getting ROC Curve and confusion matrix -> Finally, at the end we are able to get top 10 positive and negative features using feature_log_prob_ ___________ -> In TFIDF by using TFIDFVECTORIZER we fitted the train data and then applied transform for the train,test,cv. -> And same steps have been taken place for tfidf also. -> Finally , at the end we are able to get top 10 positive and negative features using feature_log_prob_ ___________ -> Last by seeing the above preety table. for both vectorizers the model accuracy is similar.

In [ ]: