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
```

```python
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 [18]:

```python
# using SQLite Table to read data.
con = sqlite3.connect('/content/drive/My Drive/amazon 2/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 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 Score != 3 LIMIT 20000""", con
)
# for tsne assignment you can take 5k data points
1
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(1)
```

```
Number of data points in our data (525814, 10)
```

Out[18]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400

In [0]:

```python
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
```

```
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [20]:

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

(80668, 7)

Out[20]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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 [21]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[21]:

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

In [22]:

```
display['COUNT(*)'].sum()
```

Out[22]:

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 [23]:

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

Out[23]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(

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='qui
cksort', na_position='last')
```

In [25]:

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

Out[25]:

```
(364173, 10)
```

In [26]:

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

Out[26]:

```
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 [27]:

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

display.head()
```

Out[27]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892ɛ
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883ɛ

In [0]:

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

In [29]:

```
#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[29]:

```
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

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 [30]:

```python
# 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 a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he 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.  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, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat 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 fi
nd 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 v
irgin 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 ingredients: Water, Maltitol, Natural Maple
Flavor.  PERIOD.  No chemicals.  No garbage.<br /><br />Have numerous friends & family members
hooked on this stuff.  My husband & son, who do NOT like "sugar free" prefer this over major label
regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [31]:

```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 a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he 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 [32]:

```
# 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 car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he 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.  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, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat 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 fi
nd 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 v
irgin 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: Water, Maltitol, Natural Maple Flavor.
PERIOD.  No chemicals.  No garbage.Have numerous friends & family members hooked on this stuff.  M
y husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.I use thi
s as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin 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 [34]:

```
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 fi
nd 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 v
irgin 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 but it could be better.
==================================================
```

In [35]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
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 car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he 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 [36]:

```
#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 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 indu
stries have convinced the masses that Canola oil is a safe and even better oil than olive or virgi
n 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 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', "y
ou'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', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "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', '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', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', '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', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn'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 [38]:

```
# 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 [02:24<00:00, 2513.02it/s]
```

In [39]:

```
preprocessed_reviews[360000]
```

Out[39]:

```
'ok excited try product using shampoo conditioner clear obsessed naturally thought would love
nourishing scalp hair oil negative not distant smell pretty overwhelming greasy use many hair oils
believe not not greasy really disappointed crowning moment top bun hair product bun ran house star
ted driving work minutes away minute placed car park bun slid hair hair loose never save money'
```


Out[39]:

```
'ok excited try product using shampoo conditioner clear obsessed naturally thought would love
nourishing scalp hair oil negative not distant smell pretty overwhelming greasy use many hair oils
believe not not greasy really disappointed crowning moment top bun hair product bun ran house star
ted driving work minutes away minute placed car park bun slid hair hair loose never save money'
```


[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', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaaa',
'aaaaaaahhhhhh', 'aaaaaaarrrrrggghhh', 'aaaaaawwwwwwwwww', 'aaaaah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 54904)
the number of unique words  54904
```

[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/stable/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_s
hape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[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_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) ['aa', 'aafco', 'aback', 'abandon', 'abandoned',
'abdominal', 'ability', 'able', 'able add', 'able brew']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 51709)
the number of unique words including both unigrams and bigrams  51709
```

[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/#.W17SRFAzZPY
# 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=Tr
ue)
        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 ")
```

```
C:\Users\Rohith\Anaconda3\lib\site-packages\gensim\models\base_any2vec.py:743: UserWarning: C exte
nsion not loaded, training will be slow. Install a C compiler and reinstall gensim for fast traini
ng.
  "C extension not loaded, training will be slow. "
```

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])
```

[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, 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]))
```

[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 value
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 cell_val = tfidf
```

```
# 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
```

[5] Assignment 4: Apply Naive Bayes

Applying Multinomial Naive Bayes

In [41]:

```
#obtaining the cleaned_text from the preprocessed_reviews  for the given dataset.
final['cleaned_text']=preprocessed_reviews
#Applying the time based splitting for the sample 15k datapts.
final.sort_values(by='Time')
final1 = final.sample(n = 100000)

Y = final1['Score'].values
X = final1['cleaned_text'].values
print(X.shape,type(X))
print(Y.shape,type(Y))
```

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

In [0]:

```
#importing library.
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
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 [43]:

```
# performing training,CV & testing for performing splitting of the dataset.
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=12,shuffle=False)
X_train,X_cv,Y_train,Y_cv=train_test_split(X,Y,test_size=0.2,random_state=12,shuffle=False)
print("*"*10)
print("After splitting the data")
print(X_train.shape,Y_train.shape)
print(X_cv.shape,Y_cv.shape)
print(X_test.shape,Y_test.shape)
```

```
**********
After splitting the data
(80000,) (80000,)
(20000,) (20000,)
```

```
(20000,) (20000,)
```

[5.1] Applying Naive Bayes on BOW.

In [44]:

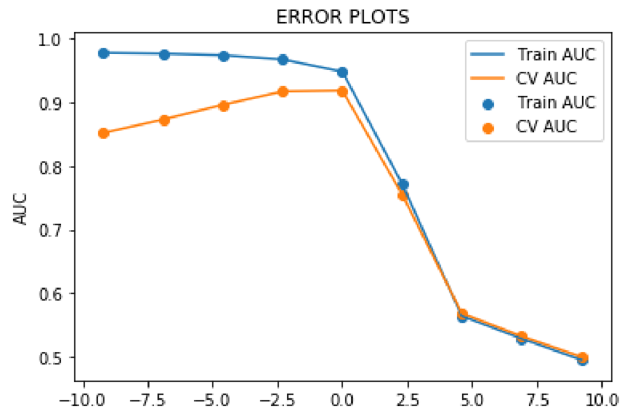
```
vectorizer=CountVectorizer()
vectorizer=vectorizer.fit(X_train)

X_train_bow=vectorizer.transform(X_train)
X_cv_bow=vectorizer.transform(X_cv)
X_test_bow=vectorizer.transform(X_test)

print("After transforming the data")
print(X_train_bow.shape,Y_train.shape)
print(X_cv_bow.shape,Y_cv.shape)
print(X_test_bow.shape,Y_cv.shape)
```

```
After transforming the data
(80000, 54695) (80000,)
(20000, 54695) (20000,)
(20000, 54695) (20000,)
```

In [45]:

```
#performing roc_auc_score on NaiveBayes assumption using probablity distribution
import math
logalpha=[]
train_auc = []
cv_auc = []
alpha = [10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for i in tqdm(alpha):
    naive = MultinomialNB(alpha=i, class_prior=None, fit_prior=True)
    naive.fit(X_train_bow, Y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    Y_train_pred =  naive.predict_proba(X_train_bow)[:,1]
    Y_cv_pred =  naive.predict_proba(X_cv_bow)[:,1]

    train_auc.append(roc_auc_score(Y_train,Y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, Y_cv_pred))
    logalpha.append(math.log(i))

plt.plot(logalpha, train_auc, label='Train AUC')
plt.scatter(logalpha, train_auc, label='Train AUC')
plt.plot(logalpha, cv_auc, label='CV AUC')
plt.scatter(logalpha, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log-alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 9/9 [00:01<00:00,  7.97it/s]
```



log-alpha: hyperparameter

Observations:-

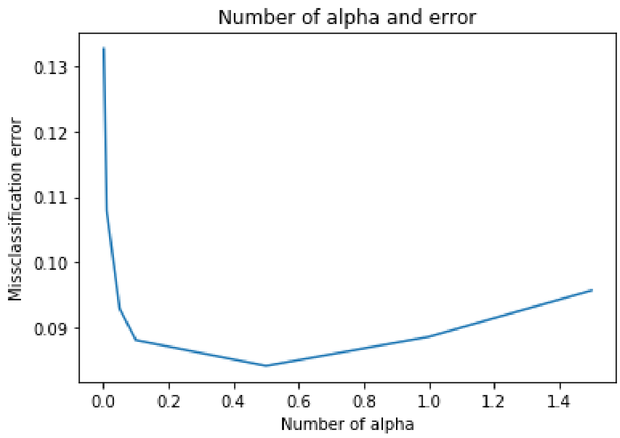
From the above plot the gap between Train and Test Curve is very less in between 0 & 10. so, applying the crossvalidation at this less range.

In [46]:

```
#finding the CV_scorees for the MUltiNomial NaiveBayes.
cv_score = []
alpha=[0.001,0.01,0.05,0.1,0.5,1,1.5]
for k in tqdm(alpha):
    NB = MultinomialNB(alpha=k, class_prior=None, fit_prior=True)
    scores = cross_val_score(NB, X_train_bow, Y_train, cv=10, scoring='roc_auc')
    cv_score.append(scores.mean())

    #finding the missclassification error for the given graph.
MSE = [1 - x for x in cv_score]
optimal_alpha_1 = alpha[MSE.index(min(MSE))]
print ("Optimal number alpha: ", optimal_alpha_1)
plt.plot(alpha, MSE)
plt.title("Number of alpha and error")
plt.xlabel("Number of alpha")
plt.ylabel("Missclassification error")
plt.show()
```

```
100%|██████████| 7/7 [00:04<00:00,  1.53it/s]
```

```
Optimal number alpha:  0.5
```

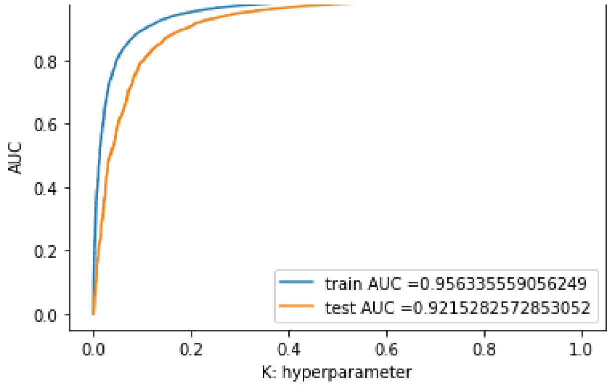


In [0]:

```
#finding the optimal_model for the Multinomial NaiveBayes.
optimal_model = MultinomialNB(alpha=optimal_alpha_1)
optimal_model.fit(X_train_bow,Y_train)
prediction = optimal_model.predict(X_test_bow)
```

In [48]:

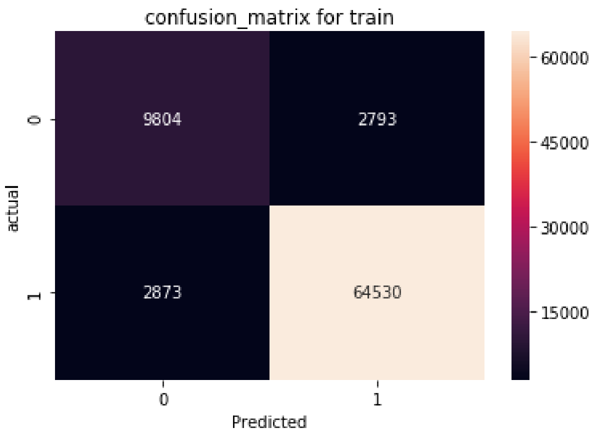
```
#Plotting the train & test methods
train_fpr, train_tpr, thresholds = roc_curve(Y_train, optimal_model.predict_proba(X_train_bow)[:,1]
)
test_fpr, test_tpr, thresholds = roc_curve(Y_test, optimal_model.predict_proba(X_test_bow)[:,1])
AUC1=str(auc(test_fpr, test_tpr))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



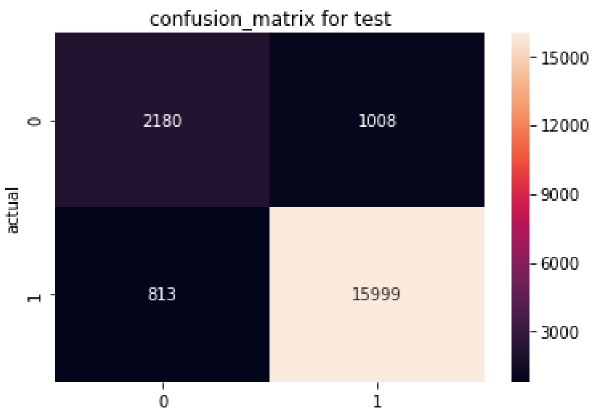
In [49]:

```
print("confusion_matrix for train_data")
conf_matrix = confusion_matrix(Y_train,optimal_model.predict(X_train_bow))
class_label =[0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("confusion_matrix for train")
plt.xlabel("Predicted")
plt.ylabel('actual')
plt.show()
print("*"*20)

print("confusion_matrix for test_data.")
conf_matrix = confusion_matrix(Y_test,optimal_model.predict(X_test_bow))
class_label =[0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix,annot=True,fmt='d')
plt.title("confusion_matrix for test")
plt.xlabel("Predicted")
plt.ylabel('actual')
plt.show()
```

```
confusion_matrix for train_data
```




```
********************
confusion_matrix for test_data.
```



Predicted

In [50]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.73      0.68      0.71      3188
           1       0.94      0.95      0.95     16812

    accuracy                           0.91     20000
   macro avg       0.83      0.82      0.83     20000
weighted avg       0.91      0.91      0.91     20000
```

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

In [51]:

```
#Sorting the feature names through the Count_Vectorizer.
feature_names = vectorizer.get_feature_names()
print(feature_names[:5])
```

```
['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa']
```

In [52]:

```
#Sorting all the logaritham probabilites of a feature new variable
feature_log_prob=(optimal_model.feature_log_prob_[:])
#Creating a new DataFrame with Feature names and their log probabilities
probability = pd.DataFrame(feature_log_prob,columns=feature_names)

print(probability.shape)
```

```
(2, 54695)
```

In [53]:

```
#Determing the probability values for the given data.
print(probability.head(3))
probability1 = probability.T
```

```
          aa        aaa       aaaa  ...      zzzzz    zzzzzzz  zzzzzzzzzzz
0 -12.327971 -13.937409 -13.937409  ... -13.937409 -12.838796   -13.937409
1 -12.339294 -13.076893 -14.376176  ... -14.376176 -15.474788   -14.376176

[2 rows x 54695 columns]
```

In [54]:

```
print(probability1[1].sort_values(ascending=False)[:10])
```

```
not       -3.730140
like      -4.576180
good      -4.683430
great     -4.759637
one       -4.918272
taste     -4.961019
tea       -5.072436
love      -5.084163
product   -5.084532
flavor    -5.089152
Name: 1, dtype: float64
```

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

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

In [55]:

```
#Similarly from the +ve class finding for Top 10 important features for the negative_class
print(probability1[0].sort_values(ascending=False)[:10])
```

```
not        -3.318694
like       -4.469253
product    -4.703645
would      -4.724571
taste      -4.749825
one        -4.933969
good       -5.168523
coffee     -5.182302
no         -5.187042
flavor     -5.222841
Name: 0, dtype: float64
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

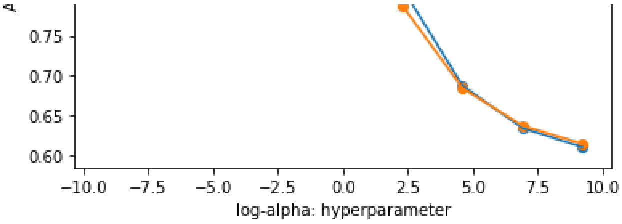
In [0]:

```
#Performing TF-IDF Vectorizer for the NaiveBayes model
Tfidf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=5)
Tfidf_vect.fit(X_train)

X_train_tfidf = Tfidf_vect.transform(X_train)
X_cv_tfidf = Tfidf_vect.transform(X_cv)
X_test_tfidf = Tfidf_vect.transform(X_test)
```

In [57]:

```
#performing roc_auc_score on NaiveBayes assumption using probablity distribution
import math
logalpha=[]
train_auc = []
cv_auc = []
alpha = [10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for i in tqdm(alpha):
    naive = MultinomialNB(alpha=i, class_prior=None, fit_prior=True)
    naive.fit(X_train_tfidf,Y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    Y_train_pred =  naive.predict_proba(X_train_tfidf)[:,1]
    Y_cv_pred =  naive.predict_proba(X_cv_tfidf)[:,1]

    train_auc.append(roc_auc_score(Y_train,Y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, Y_cv_pred))
    logalpha.append(math.log(i))

plt.plot(logalpha, train_auc, label='Train AUC')
plt.scatter(logalpha, train_auc, label='Train AUC')
plt.plot(logalpha, cv_auc, label='CV AUC')
plt.scatter(logalpha, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log-alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 9/9 [00:01<00:00,  6.35it/s]
```



Observations:-

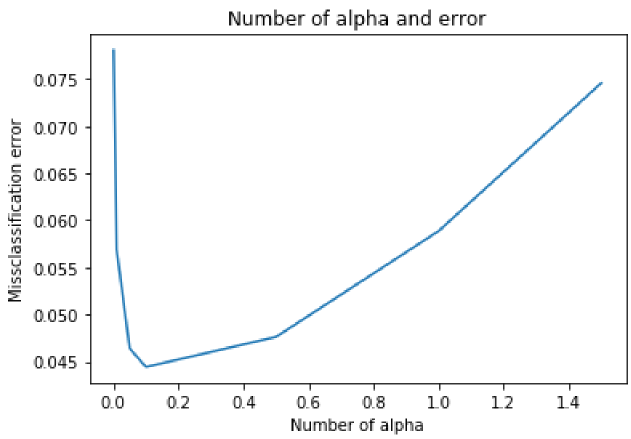
From the above plot the gap between Train and Test Curve is very less in between 0.0 & 10.0. so, applying the crossvalidation at this less range.

In [58]:

```
#finding the CV_scorees for the MUltiNomial NaiveBayes.
cv_score = []
alpha=[0.001,0.01,0.05,0.1,0.5,1,1.5]
for k in tqdm(alpha):
    NB = MultinomialNB(alpha=k, class_prior=None, fit_prior=True)
    scores = cross_val_score(NB, X_train_tfidf, Y_train, cv=10, scoring='roc_auc')
    cv_score.append(scores.mean())

#finding the missclassification error for the given graph.
MSE = [1 - x for x in cv_score]
optimal_alpha_2 = alpha[MSE.index(min(MSE))]

print("Optimal number alpha: ", optimal_alpha_2)
plt.plot(alpha, MSE)
plt.title("Number of alpha and error")
plt.xlabel("Number of alpha")
plt.ylabel("Missclassification error")
plt.show()
```

```
100%|██████████| 7/7 [00:06<00:00,  1.17it/s]
```

```
Optimal number alpha:  0.1
```



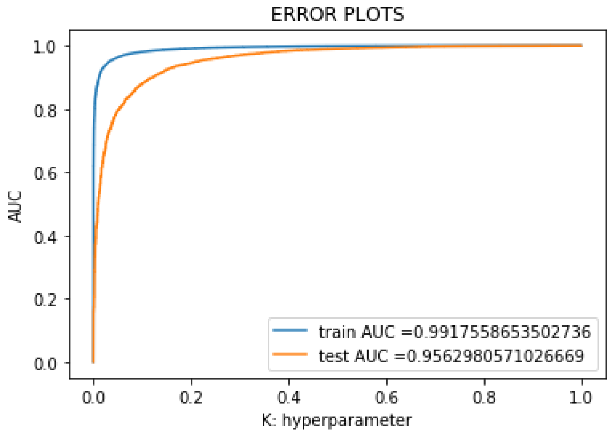
In [0]:

```
#finding the optimal_model for the Multinomial NaiveBayes.
optimal_model = MultinomialNB(alpha=optimal_alpha_2)
optimal_model.fit(X_train_tfidf,Y_train)
prediction = optimal_model.predict(X_test_tfidf)
```

In [60]:

```
#Plotting the train & test methods
train_fpr, train_tpr, thresholds = roc_curve(Y_train, optimal_model.predict_proba(X_train_tfidf)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, optimal_model.predict_proba(X_test_tfidf)[:,1])
AUC2=str(auc(test_fpr, test_tpr))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
```

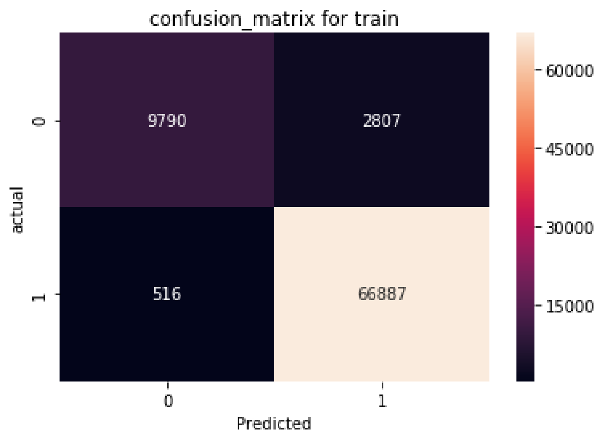
```
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



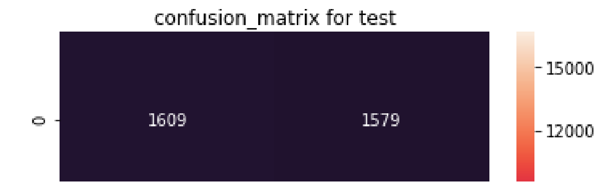
In [61]:

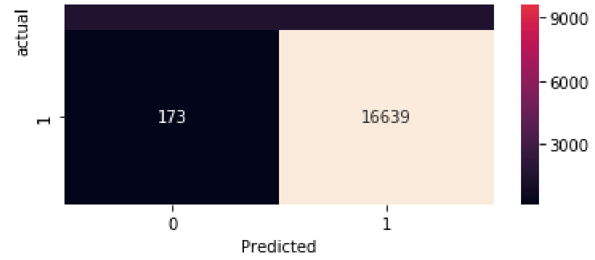
```
print("confusion_matrix for train_data")
conf_matrix = confusion_matrix(Y_train,optimal_model.predict(X_train_tfidf))
class_label =[0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("confusion_matrix for train")
plt.xlabel("Predicted")
plt.ylabel('actual')
plt.show()
print("*"*20)

print("confusion_matrix for test_data.")
conf_matrix = confusion_matrix(Y_test,optimal_model.predict(X_test_tfidf))
class_label =[0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix,annot=True,fmt='d')
plt.title("confusion_matrix for test")
plt.xlabel("Predicted")
plt.ylabel('actual')
plt.show()
```

```
confusion_matrix for train_data
```




```
********************
confusion_matrix for test_data.
```





In [62]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.90      0.50      0.65      3188
           1       0.91      0.99      0.95     16812

    accuracy                           0.91     20000
   macro avg       0.91      0.75      0.80     20000
weighted avg       0.91      0.91      0.90     20000
```

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

In [63]:

```
#Sorting the feature names through the Count_Vectorizer.
feature_names = Tfidf_vect.get_feature_names()
print(feature_names[:5])
```

```
['aa', 'aaa', 'aafco', 'ab', 'aback']
```

In [64]:

```
#Sorting all the logaritham probabilites of a feature new variable
feature_log_prob=(optimal_model.feature_log_prob_[:])
#Creating a new DataFrame with Feature names and their log probabilities
probability = pd.DataFrame(feature_log_prob,columns=feature_names)

print(probability.shape)
```

```
(2, 103091)
```

In [65]:

```
#Determing the probability values for the given data.
print(probability.head(3))
probability1 = probability.T
```

```
          aa        aaa      aafco  ...      zukes  zukes mini   zwieback
0 -12.202962 -13.698241 -12.931316  ... -12.367136  -12.744377 -12.718915
1 -12.104181 -12.910632 -13.518201  ... -11.321668  -13.204316 -12.959297

[2 rows x 103091 columns]
```

In [68]:

```
print(probability1[1].sort_values(ascending=False)[:10])
```

```
not       -5.492606
great     -5.852561
good      -5.910871
like      -5.980797
tea       -6.057768
coffee    -6.059638
```

```
love       -6.083920
product    -6.174258
taste      -6.194361
one        -6.223081
Name: 1, dtype: float64
```

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

In [69]:

```
print(probability1[0].sort_values(ascending=False)[:10])
```

```
not        -5.019329
like       -5.854766
product    -5.907259
taste      -5.973095
would      -5.979686
coffee     -6.217961
one        -6.245886
no         -6.338963
flavor     -6.402012
buy        -6.467045
Name: 0, dtype: float64
```

[5.3] Feature Engineering Model for Length of reviews using BOW.

In Feature Engineering model adding a length of reviews for feature to increasing the auc model.

Creating a list af length of the words from the preprocessed reviews

Performing 15k data points from the given dataset.

In [70]:

```
# Creating a list af length of  the words in preprocessed reviews
lengths=[]
for sentence in preprocessed_reviews:
    lengths.append(len(sentence.split()))
print(lengths[:5])
lengths1=np.asarray(lengths)
print(lengths1.shape)
```

```
[35, 27, 15, 53, 38]
(364171,)
```

In [71]:

```
#Taking 15kdatapoints for the length of reviews using bow.
final['lengths']=lengths1
final2 = final.sample(n = 15000)


X = final2['cleaned_text'].values
Y = final2['Score'].values
Z = final2['lengths'].values

print(X.shape)
print(Y.shape)
print(Z.shape)
```

```
(15000,)
(15000,)
(15000,)
```

In [72]:

```
#Similarly perfroming train,cv & test for length of the reviews concept
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=12,shuffle=False)
```

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=12,shuffle=False)
X_train,X_cv,Y_train,Y_cv=train_test_split(X,Y,test_size=0.2,random_state=12,shuffle=False)

print("After splitting the data.")
print(X_train.shape,  Y_train.shape)
print(X_cv.shape,     Y_cv.shape)
print(X_test.shape,   Y_test.shape)
```

```
After splitting the data.
(12000,) (12000,)
(3000,) (3000,)
(3000,) (3000,)
```

In [73]:

```
vectorizer=CountVectorizer()
vectorizer=vectorizer.fit(X_train)

X_train_bow=vectorizer.transform(X_train)
X_cv_bow=vectorizer.transform(X_cv)
X_test_bow=vectorizer.transform(X_test)

print("After transform")
print(X_train_bow.shape,  Y_train.shape)
print(X_cv_bow.shape,     Y_cv.shape)
print(X_test_bow.shape,   Y_test.shape)
```

```
After transform
(12000, 21841) (12000,)
(3000, 21841) (3000,)
(3000, 21841) (3000,)
```

In [74]:

```
A_train,A_test,B_train,B_test = train_test_split(Z,Y,test_size =0.2,random_state=12,shuffle=False)
A_train,A_cv,B_train,B_cv= train_test_split(Z,Y,test_size=0.2,random_state=12,shuffle= False)
print(A_train.shape,  B_train.shape)
print(A_cv.shape,     B_cv.shape)
print(A_test.shape,  B_test.shape)
```

```
(12000,) (12000,)
(3000,) (3000,)
(3000,) (3000,)
```

In [0]:

```
from scipy import sparse
from scipy.sparse import hstack
```

In [76]:

```
#Training model

print("Feature Engineering model for Train")

A_train1=sparse.csr_matrix(A_train)
print("X_train_bow:",X_train_bow.shape,type(X_train_bow))
print("A_train1:",A_train1.shape)

train = hstack([X_train_bow,A_train1.T]).toarray()
print(train)
##################################################################################################
print("*"*50)
print("Feature Engineering model for CV")

# Cross Validation model
A_cv1=sparse.csr_matrix(A_cv)
print("X_cv_bow:",X_cv_bow.shape)
print("A_cv1:",A_cv1.shape)

cv = hstack([X_cv_bow,A_cv1.T]).toarray()
print(cv)
```

```
#########################################################################################

print("*"*50)
print("Feature Engineering model for Test")

# Testing model.
A_test1=sparse.csr_matrix(A_test)

print("X_test_bow:",X_test_bow.shape)
print("A_test1:",A_test1.shape)

test = hstack([X_test_bow,A_test1.T]).toarray()
print(test)
```

```
Feature Engineering model for Train
X_train_bow: (12000, 21841) <class 'scipy.sparse.csr.csr_matrix'>
A_train1: (1, 12000)
[[  0   0   0 ...   0   0  13]
 [  0   0   0 ...   0   0  15]
 [  0   0   0 ...   0   0  83]
 ...
 [  0   0   0 ...   0   0  13]
 [  0   0   0 ...   0   0 298]
 [  0   0   0 ...   0   0  15]]
**************************************************
Feature Engineering model for CV
X_cv_bow: (3000, 21841)
A_cv1: (1, 3000)
[[ 0  0  0 ...  0  0  7]
 [ 0  0  0 ...  0  0 23]
 [ 0  0  0 ...  0  0 23]
 ...
 [ 0  0  0 ...  0  0 58]
 [ 0  0  0 ...  0  0 26]
 [ 0  0  0 ...  0  0 17]]
**************************************************
Feature Engineering model for Test
X_test_bow: (3000, 21841)
A_test1: (1, 3000)
[[ 0  0  0 ...  0  0  7]
 [ 0  0  0 ...  0  0 23]
 [ 0  0  0 ...  0  0 23]
 ...
 [ 0  0  0 ...  0  0 58]
 [ 0  0  0 ...  0  0 26]
 [ 0  0  0 ...  0  0 17]]
```

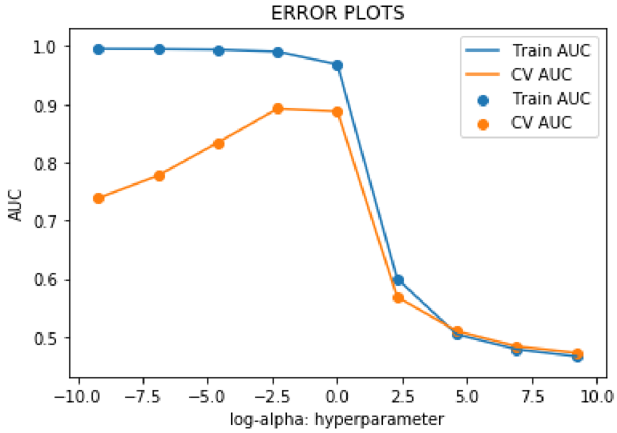
Plotting the error plots

In [77]:

```
#performing roc_auc_score on NaiveBayes assumption using probablity distribution
import math
logalpha=[]
train_auc = []
cv_auc = []
alpha = [10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]
for i in tqdm(alpha):
    naive = MultinomialNB(alpha=i, class_prior=None, fit_prior=True)
    naive.fit(train,Y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    Y_train_pred =  naive.predict_proba(train)[:,1]
    Y_cv_pred =  naive.predict_proba(cv)[:,1]

    train_auc.append(roc_auc_score(Y_train,Y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, Y_cv_pred))
    logalpha.append(math.log(i))

plt.plot(logalpha, train_auc, label='Train AUC')
plt.scatter(logalpha, train_auc, label='Train AUC')
plt.plot(logalpha, cv_auc, label='CV AUC')
plt.scatter(logalpha, cv_auc, label='CV AUC')
```

```
plt.legend()
plt.xlabel("log-alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 9/9 [00:30<00:00,  3.36s/it]
```



Observations:-

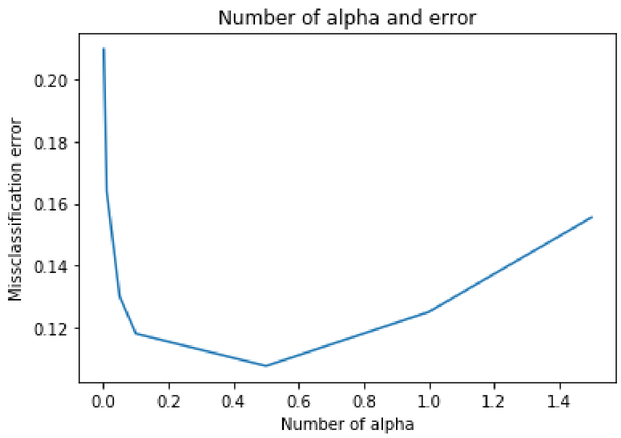
From the above plot the gap between Train and Test Curve is very less in between 0.0 & 10.0. so, applying the crossvalidation at this less range.

In [78]:

```
#finding the CV_scorees for the MUltiNomial NaiveBayes.
cv_score = []
alpha=[0.001,0.01,0.05,0.1,0.5,1,1.5]
for k in tqdm(alpha):
    NB = MultinomialNB(alpha=k, class_prior=None, fit_prior=True)
    scores = cross_val_score(NB, train, Y_train, cv=10, scoring='roc_auc')
    cv_score.append(scores.mean())

#finding the missclassification error for the given graph.
MSE = [1 - x for x in cv_score]
optimal_alpha_3 = alpha[MSE.index(min(MSE))]

print("Optimal number alpha: ", optimal_alpha_3)
plt.plot(alpha, MSE)
plt.title("Number of alpha and error")
plt.xlabel("Number of alpha")
plt.ylabel("Missclassification error")
plt.show()
```

```
100%|██████████| 7/7 [02:37<00:00, 22.42s/it]
```

```
Optimal number alpha:  0.5
```

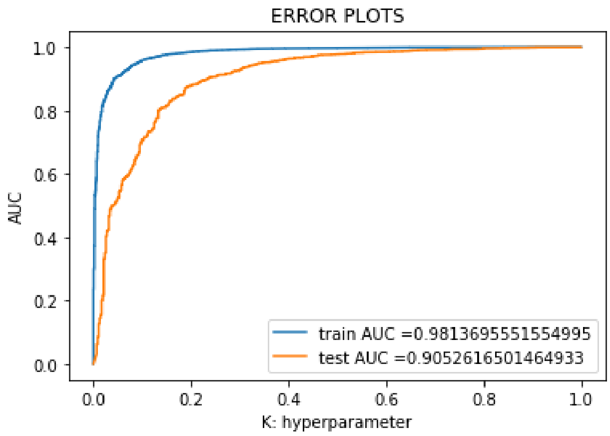


In [0]:

```
#finding the optimal_model for the Multinomial NaiveBayes.
optimal_model = MultinomialNB(alpha=optimal_alpha_3)
optimal_model.fit(train,Y_train)
prediction = optimal_model.predict(test)
```

In [80]:

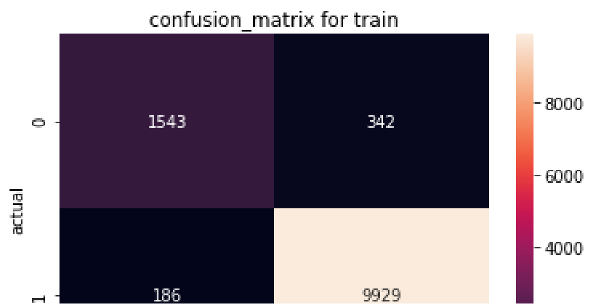
```
#Plotting the train & test methods
train_fpr, train_tpr, thresholds = roc_curve(Y_train, optimal_model.predict_proba(train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, optimal_model.predict_proba(test)[:,1])
AUC3=str(auc(test_fpr, test_tpr))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



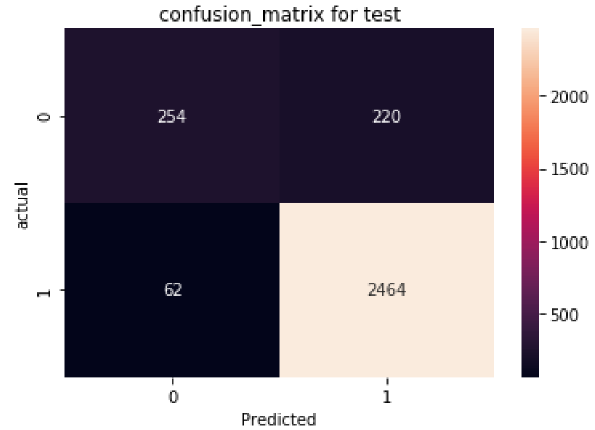
In [81]:

```
print("confusion_matrix for train_data")
conf_matrix = confusion_matrix(Y_train,optimal_model.predict(train))
class_label =[0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("confusion_matrix for train")
plt.xlabel("Predicted")
plt.ylabel('actual')
plt.show()
print("*"*20)

print("confusion_matrix for test_data.")
conf_matrix = confusion_matrix(Y_test,optimal_model.predict(test))
class_label =[0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix,annot=True,fmt='d')
plt.title("confusion_matrix for test")
plt.xlabel("Predicted")
plt.ylabel('actual')
plt.show()
```

confusion_matrix for train_data




```
*******************
confusion_matrix for test_data.
```



In [82]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.80      0.54      0.64       474
           1       0.92      0.98      0.95      2526

    accuracy                           0.91      3000
   macro avg       0.86      0.76      0.79      3000
weighted avg       0.90      0.91      0.90      3000
```

The following steps for brute force & kd_tree

1.) USing 15k dataset point from the total dataset.

2.) Splitting the dataset in to train_data,CV_data & test_data.

3.) Applying Brute force algoritham on BOW,TFIDF &Length of reviews using BOW by using NaiveBayes.

4.)Plot(train) the ROC_AUC_curve for both the train & CV data.now,Applying the CV_score from the selected range.

5.)plotting MSE(MissClassificationError) to get optimal_aplha from the cv_score.

6.) Taking an Optimal_alpha value so that it should not Overfit or Underfit.

7.)Plot(test) AUC_ROC_curve for train & test (tpr_fpr).

7.)Plotting confusion matrix for both Train & Test data.

8.)from all the above process obtaining the average classification report.

9.)finding the Top 10 features names for positive & Negative class.

>>>>> from the step 2 to step 8 all these steps are repeated simalarly for (BOW,TFIDF) using brute force and

1.)Similarly Applying the feature engineering model for the Length of review using BOW.(Same steps are followed for feature Engineering model also.)

[6] Conclusions

In [84]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
comparison = PrettyTable()
comparison.field_names = ["Vectorizer", "Model", "Hyperparameter(C)","Hyperparameter(alpha)", "AUC"
]

comparison.add_row(["BOW",    'Brute',  optimal_alpha_1,  (1/optimal_alpha_1), np.round(float(AUC1)
,3)])
comparison.add_row(["TFIDF",   'Brute', optimal_alpha_2,  (1/optimal_alpha_2), np.round(float(AUC2)
,3)])

comparison.add_row(["Length of review using BOW", 'Brute', optimal_alpha_3, (1/optimal_alpha_3), n
p.round(float(AUC3),3)])

print(comparison)
```

```
+----------------------------+-------+-------------------+-----------------------+-------+
|         Vectorizer         | Model | Hyperparameter(C) | Hyperparameter(alpha) |  AUC  |
+----------------------------+-------+-------------------+-----------------------+-------+
|            BOW             | Brute |        0.5        |          2.0          | 0.922 |
|           TFIDF            | Brute |        0.1        |          10.0         | 0.956 |
| Length of review using BOW | Brute |        0.5        |          2.0          | 0.905 |
+----------------------------+-------+-------------------+-----------------------+-------+
```