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
```

```python
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]:

```python
# using SQLite Table to read data.
```

```
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 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 500
000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 11500
0""", con)

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

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

```
Number of data points in our data (115000, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summa
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality D Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertise
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

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)
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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 [5]:

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

	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 [6]:

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

```
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()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sum
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER

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='f
irst', inplace=False)
final.shape
```

```
(99724, 10)
```

In [10]:

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

```
86.71652173913044
```

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()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure co taste wi crunchy almonds inside

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()
```

```
(99722, 10)


1    83711
0    16011
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
==================================================
IF YOU LIKE SALMON YOU WILL LOVE THESE OMAHA STEAKS SALMON VERY VERY GOOD
==================================================
OK....I thought I'd put a bit of punch to hubby's sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!<br />
One bite and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!<br />I guess it was that-bad!<br />I'll see if my neighbo
r will be able to use it w/her family.<br />If you  are a BEST FOODS lover---walk away---do NOT purchase this product!
==================================================
These people from Bavaria really know how to make this stuff. The Landjagers are super (you have to let them dry for them to
develop their full, intended flavor), and it is worth any sausage fan's time to check out their complete offering of German
style sausages and hams.  Due to the perishability of some of their products their S&H charges appear outrageous but I guess
that sending frozen or refrigerated foods costs money.  Personally, I recommend their coarse grind liverwurst but that is a
matter of personal taste.
==================================================
```

In [15]:

```
# 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)
```

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

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-ta
gs-from-an-element
from bs4 import BeautifulSoup

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
```

```
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
==================================================
IF YOU LIKE SALMON YOU WILL LOVE THESE OMAHA STEAKS SALMON VERY VERY GOOD
==================================================
OK....I thought I'd put a bit of punch to hubby's sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!One bi
te and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!I guess it was that-bad!I'll see if my neighbor will be able to
use it w/her family.If you  are a BEST FOODS lover---walk away---do NOT purchase this product!
==================================================
These people from Bavaria really know how to make this stuff. The Landjagers are super (you have to let them dry for them to
develop their full, intended flavor), and it is worth any sausage fan's time to check out their complete offering of German
style sausages and hams.  Due to the perishability of some of their products their S&H charges appear outrageous but I guess
that sending frozen or refrigerated foods costs money.  Personally, I recommend their coarse grind liverwurst but that is a
matter of personal taste.
```

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]:
```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
OK....I thought I would put a bit of punch to hubby is sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!<
br />One bite and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!<br />I guess it was that-bad!<br />I will see if my
neighbor will be able to use it w/her family.<br />If you  are a BEST FOODS lover---walk away---do NOT purchase this produc
t!
==================================================
```

In [19]:
```python
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

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

In [20]:
```python
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
OK I thought I would put a bit of punch to hubby is sandwich instead of the ho hum Best Foods Mayo ohOoooOh FAILURE br One b
ite and he said Please DO NOT EVER SERVE THIS TO ME AGAIN br I guess it was that bad br I will see if my neighbor will be ab
le to use it w her family br If you are a BEST FOODS lover walk away do NOT purchase this product
```

In [21]:
```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step
```

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'th
ey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
t'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ove
r', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any'
, 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too',
 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'might
n', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wa
sn', "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 stopword
s)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 99722/99722 [00:38<00:00, 2587.50it/s]
```

In [23]:

```
from sklearn.cross_validation import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.cross_validation import cross_val_score
from collections import Counter
import collections
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve,confusion_matrix,auc
from sklearn import cross_validation
from scipy.sparse import csr_matrix,hstack
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in ver
sion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note th
at the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [24]:
```
# Splitting our Data into Train , validation and test
X_1, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed_reviews,final[
'Score'], test_size=0.2, random_state=0)
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.25)
print(np.asarray(X_1).shape,np.asarray(X_test).shape,np.asarray(X_tr).shape,np.asarray(X
_test).shape,np.asarray(X_cv).shape)
```

```
(79777,) (19945,) (59832,) (19945,) (19945,)
```

[3.2] Preprocessing Review Summary

In [25]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [26]:
```
count_vect = CountVectorizer() #in scikit-learn
BOW_Train = count_vect.fit_transform(X_tr)
BOW_test = count_vect.transform(X_test)
BOW_CV = count_vect.transform(X_cv)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)
print("the type of count vectorizer ",type(BOW_Train))
print("the shape of out text BOW vectorizer ",BOW_Train.get_shape())
print("the number of unique words ", BOW_Train.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaaa', 'aaaaah', 'aaah', 'aaahs', 'aachen']
```

```
===================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (59832, 45714)
the number of unique words  45714
```

[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/module
s/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_co
unts.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 [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
TFIDF_Train = tf_idf_vect.fit_transform(X_tr)
```

```
TFIDF_Test = tf_idf_vect.transform(X_test)
TFIDF_Validation = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(TFIDF_Train))
print("the shape of out text TFIDF vectorizer ",TFIDF_Train.get_shape())
print("the number of unique words including both unigrams and bigrams ", TFIDF_Train.get
_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (59832, 35266)
the number of unique words including both unigrams and bigrams  35266
```

In [28]:

```
print(type(TFIDF_Train))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

[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=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
 train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.9946032166481018), ('excellent', 0.99443328
```

```
38058472), ('especially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted', 0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.9992451071739197), ('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('american', 0.9991837739944458), ('beef', 0.9991780519485474), ('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', 'stinky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this 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]))
```

```
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 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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
```

```
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

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

Getting a new feature

In [29]:
```
flength = []
for i in range(len(preprocessed_reviews)):
    flength.append(len(preprocessed_reviews[i]))
print(flength[0:10])
print(len(flength))
```

```
[162, 72, 249, 127, 109, 207, 53, 194, 595, 129]
99722
```

In [30]:
```
## splitting it
f1, ftest, y1, ytest = cross_validation.train_test_split(flength,final['Score'], test_si
```

```
ze=0.2, random_state=0)
ftr, fcv, ytr, ycv = cross_validation.train_test_split(f1, y1, test_size=0.25)
print(np.asarray(f1).shape,np.asarray(ftest).shape,np.asarray(ftr).shape,np.asarray(ftes
t).shape,np.asarray(fcv).shape)
```

```
(79777,) (19945,) (59832,) (19945,) (19945,)
```

In [31]:
```
New_BOW_TRAIN = hstack((BOW_Train,np.asarray(ftr).reshape(-1,1)))
New_BOW_CV = hstack((BOW_CV,np.asarray(fcv).reshape(-1,1)))
New_BOW_Test = hstack((BOW_test,np.asarray(ftest).reshape(-1,1)))
```

In [32]:
```
New_TFIDF_TRAIN = hstack((TFIDF_Train,np.asarray(ftr).reshape(-1,1)))
New_TFIDF_CV = hstack((TFIDF_Validation,np.asarray(fcv).reshape(-1,1)))
New_TFIDF_Test = hstack((TFIDF_Test,np.asarray(ftest).reshape(-1,1)))
```

[5] Assignment 4: Apply Naive Bayes

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

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

3. **Feature importance**

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

4. **Feature engineering**

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

5. **Representation of results**

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

6. **Conclusion**

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

Note: Data Leakage

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

Applying Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

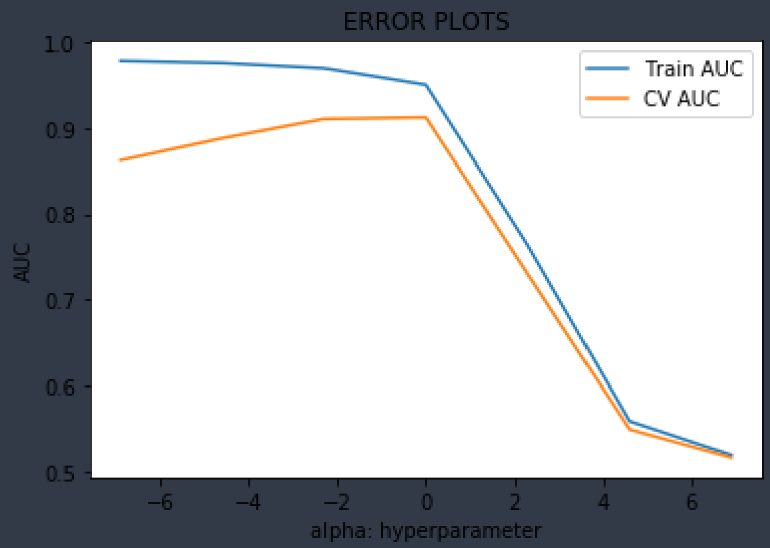
In [54]:

```
# Please write all the code with proper documentation
alph = [10**(-3),10**(-2),10**(-1),1,10,100,1000]
BOW_Train_Accuracy = []
BOW_CV_Accuracy = []
for i in alph:
    model = MultinomialNB(alpha=i)
    model.fit(BOW_Train,y_tr)
    train_pred = model.predict_log_proba(BOW_Train)[:,1]
    val_pred=model.predict_log_proba(BOW_CV)[:,1]
    BOW_Train_Accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_pred)))
    BOW_CV_Accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_pred)))
```

In [55]:

```
plt.plot(np.log(np.asarray(alph)), BOW_Train_Accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(alph)), BOW_CV_Accuracy, label='CV AUC')
plt.legend()
```

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

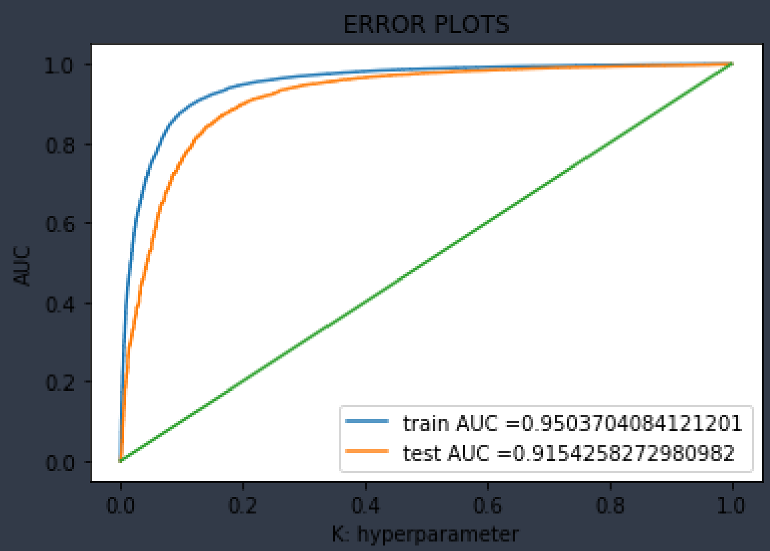


In [56]:
```
best_alpha = 1
```

Testing on test data

In [57]:
```
model = MultinomialNB(alpha=best_alpha)
model.fit(BOW_Train,y_tr)
test_pred = model.predict_log_proba(BOW_test)[:,1]
train_pred =model.predict_log_proba(BOW_Train)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)
```

```
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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

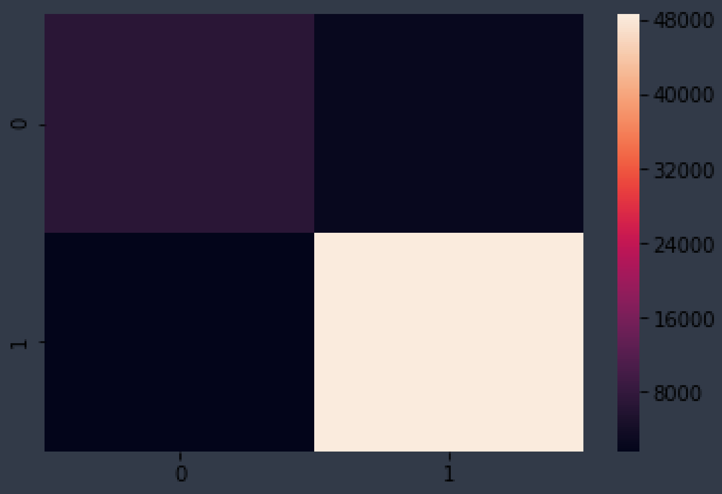


Confusion Matrix

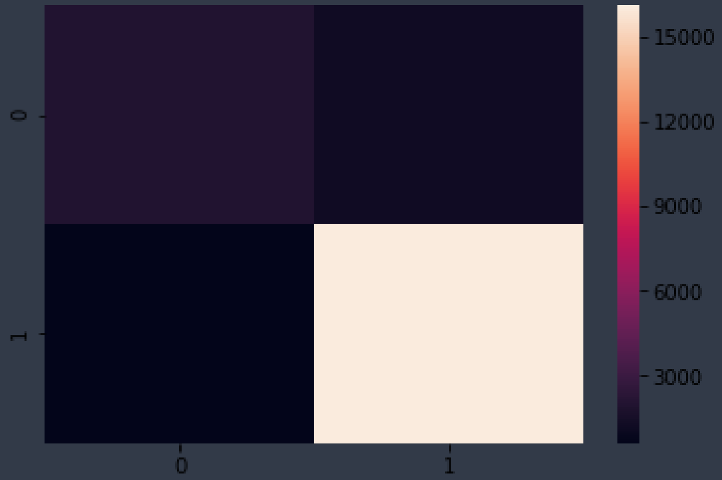
In [58]:
```
ytrain = model.predict(BOW_Train)
ytest = model.predict(BOW_test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
```

```python
print(ctrain)
plt.show()
sns.heatmap(ctest)
print(ctrain)
plt.show()
```

```
[[ 6895  2700]
 [ 1662 48575]]
```




```
[[ 6895  2700]
 [ 1662 48575]]
```



Applying the whole model again by using a new feature which is review length

In [59]:

```
alph = [10**(-3),10**(-2),10**(-1),1,10,100,1000]
BOW_Train_Accuracy = []
BOW_CV_Accuracy = []
for i in alph:
    model = MultinomialNB(alpha=i)
    model.fit(New_BOW_TRAIN,y_tr)
    train_pred = model.predict_log_proba(New_BOW_TRAIN)[:,1]
    val_pred=model.predict_log_proba(New_BOW_CV)[:,1]
    BOW_Train_Accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_pred)))
    BOW_CV_Accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_pred)))
plt.plot(np.log(np.asarray(alph)), BOW_Train_Accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(alph)), BOW_CV_Accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

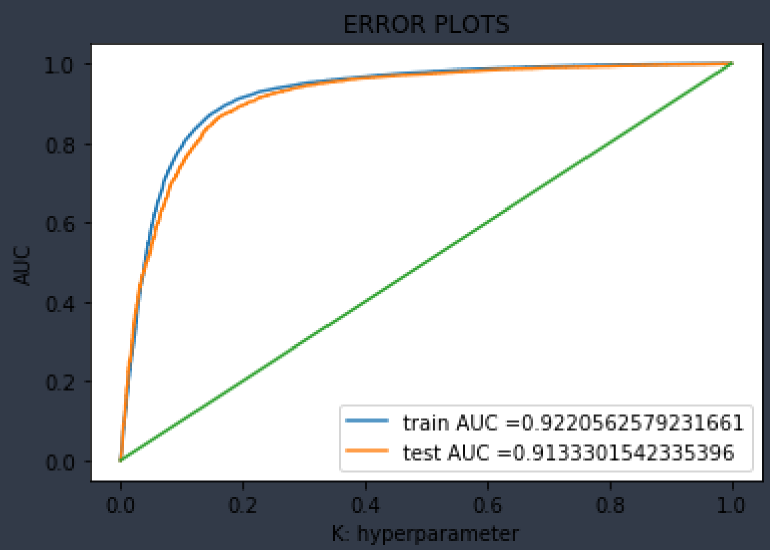
In [60]:
```
best_alpha = 10**(-1)
```

Testing on our test data

In [61]:
```
model = MultinomialNB(alpha=best_alpha)
model.fit(New_BOW_TRAIN,y_tr)
test_pred = model.predict_log_proba(New_BOW_Test)[:,1]
train_pred =model.predict_log_proba(New_BOW_TRAIN)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

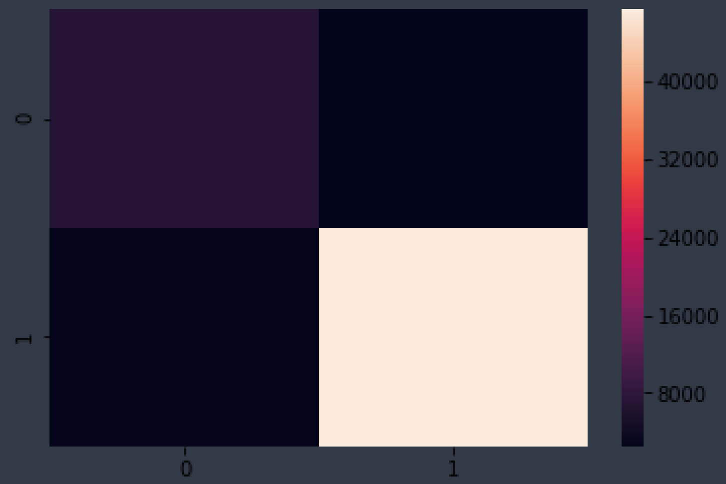


Confusion Matrix

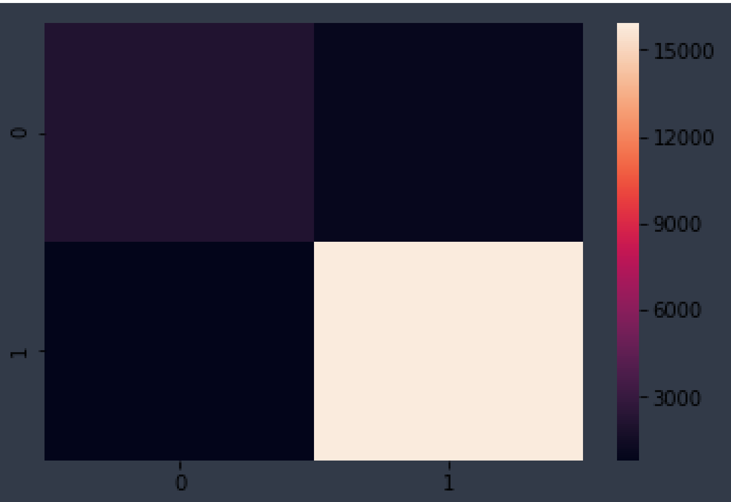
In [62]:
```
ytrain = model.predict(New_BOW_TRAIN)
ytest = model.predict(New_BOW_Test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
print(ctrain)
```

```
plt.show()
sns.heatmap(ctest)
print(ctrain)
plt.show()
```

```
[[ 7028  2567]
 [ 2923 47314]]
```




```
[[ 7028  2567]
 [ 2923 47314]]
```



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

In [64]:
```
# Please write all the code with proper documentation
a = model.feature_log_prob_
b =  a[1].argsort()[-11:-1][::-1]
print(" So the top 10 features of positive class are--")
for i in range(10):
    print("feature name : %s , value : %f"%(count_vect.get_feature_names()[b[i]],float(a
[1,b[i]])))
```

```
 So the top 10 features of positive class are--
feature name : not , value : -5.751573
feature name : like , value : -6.559459
feature name : good , value : -6.704799
feature name : great , value : -6.780951
feature name : one , value : -6.916424
feature name : taste , value : -7.006706
feature name : coffee , value : -7.029793
feature name : flavor , value : -7.101451
```

```
feature name : would , value : -7.105299
feature name : love , value : -7.108833
```

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

In [68]:

```
# Please write all the code with proper documentation
a = model.feature_log_prob_
b =  a[0].argsort()[-11:-1][::-1]
print(" So the top 10 features of negative class are--")
for i in range(10):
    print("feature name : %s , value : %f"%(count_vect.get_feature_names()[b[i]],float(a
[0,b[i]])))
```

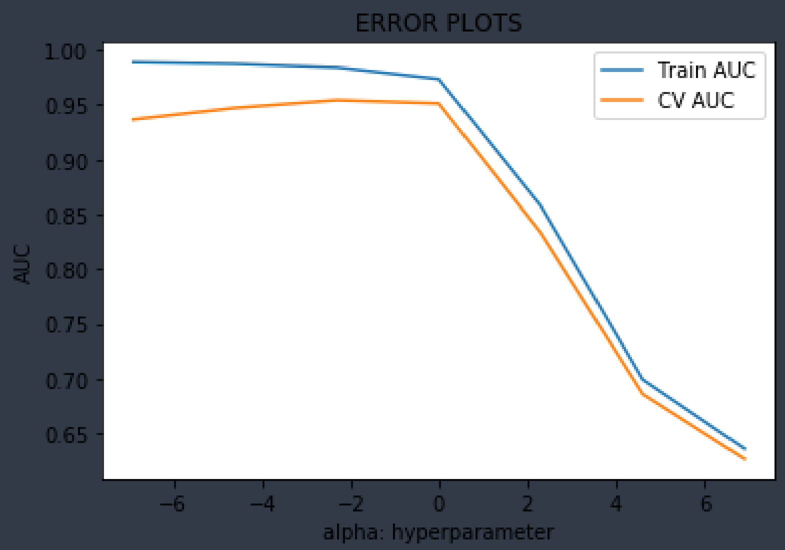
```
 So the top 10 features of negative class are--
feature name : not , value : -5.178029
feature name : like , value : -6.299056
feature name : product , value : -6.577599
feature name : taste , value : -6.581648
feature name : would , value : -6.583170
feature name : one , value : -6.786632
feature name : good , value : -7.022847
feature name : no , value : -7.073785
feature name : coffee , value : -7.082115
feature name : flavor , value : -7.106671
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [69]:

```
# Please write all the code with proper documentation
alph = [10**(-3),10**(-2),10**(-1),1,10,100,1000]
TFIDF_Train_Accuracy = []
TFIDF_CV_Accuracy = []
for i in alph:
```

```
    model = MultinomialNB(alpha=i)
    model.fit(TFIDF_Train,y_tr)
    train_pred = model.predict_log_proba(TFIDF_Train)[:,1]
    val_pred=model.predict_log_proba(TFIDF_Validation)[:,1]
    TFIDF_Train_Accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_pred)))
    TFIDF_CV_Accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_pred)))
plt.plot(np.log(np.asarray(alph)), TFIDF_Train_Accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(alph)), TFIDF_CV_Accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

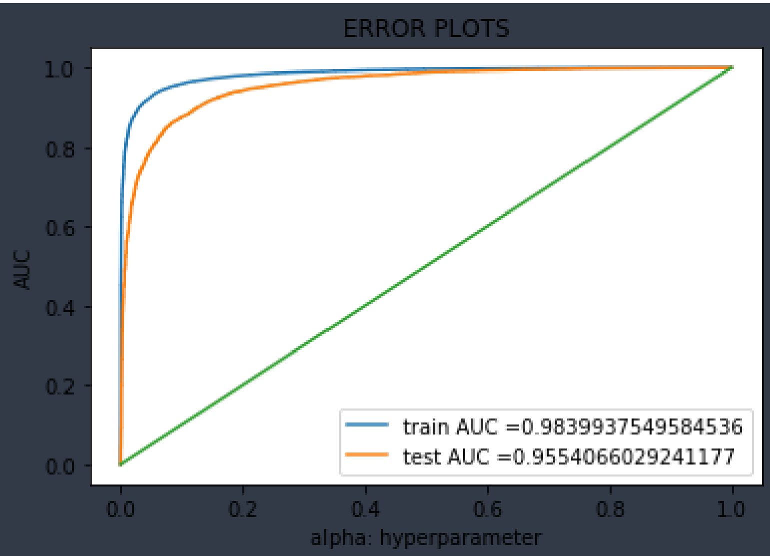


In [70]:
```
best_alpha = 10**(-1)
```

Testing on test data

In [71]:
```
model = MultinomialNB(alpha=best_alpha)
model.fit(TFIDF_Train,y_tr)
test_pred = model.predict_log_proba(TFIDF_Test)[:,1]
train_pred =model.predict_log_proba(TFIDF_Train)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

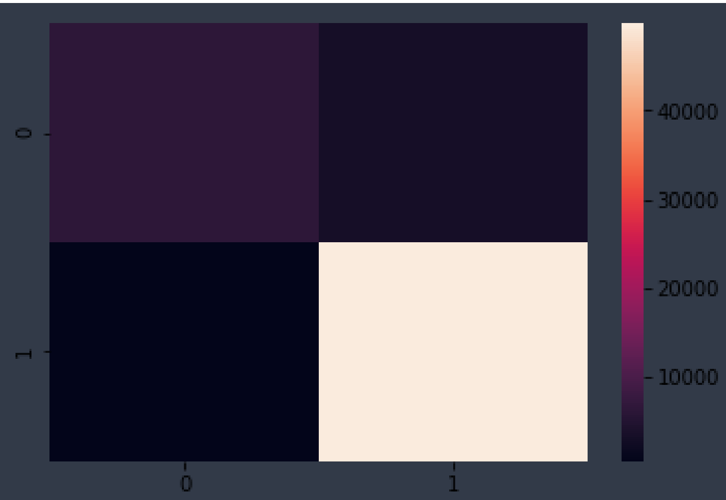


Confusion Matrix

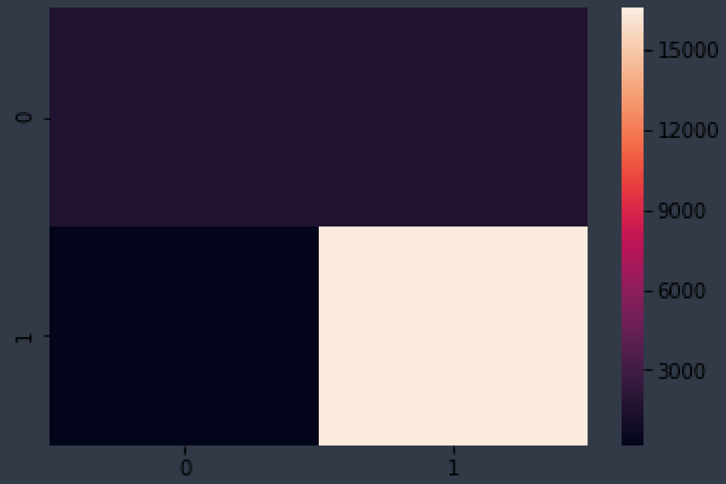
In [72]:

```
ytrain = model.predict(TFIDF_Train)
ytest = model.predict(TFIDF_Test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
print(ctrain)
plt.show()
sns.heatmap(ctest)
print(ctrain)
plt.show()
```

```
[[ 6348  3247]
 [  455 49782]]
```

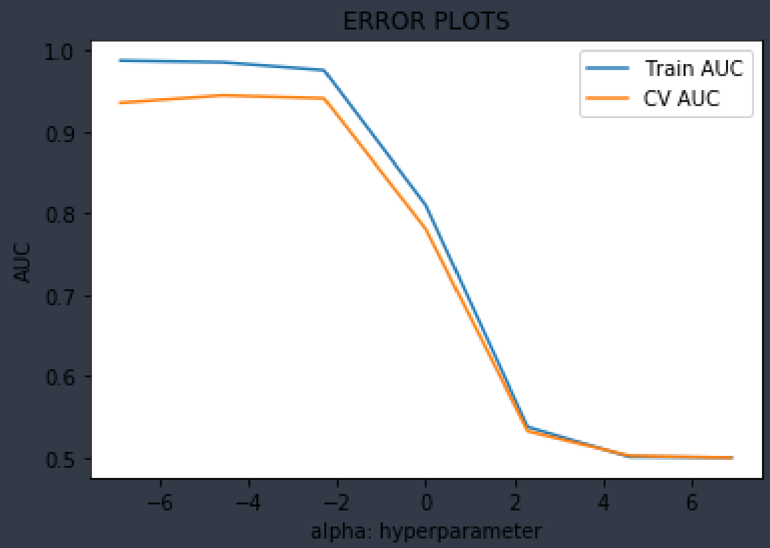



```
[[ 6348  3247]
 [  455 49782]]
```



In [73]:
```
alph = [10**(-3),10**(-2),10**(-1),1,10,100,1000]
```

```python
TFIDF_Train_Accuracy = []
TFIDF_CV_Accuracy = []
for i in alph:
    model = MultinomialNB(alpha=i)
    model.fit(New_TFIDF_TRAIN,y_tr)
    train_pred = model.predict_log_proba(New_TFIDF_TRAIN)[:,1]
    val_pred=model.predict_log_proba(New_TFIDF_CV)[:,1]
    TFIDF_Train_Accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_pred)))
    TFIDF_CV_Accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_pred)))
plt.plot(np.log(np.asarray(alph)), TFIDF_Train_Accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(alph)), TFIDF_CV_Accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

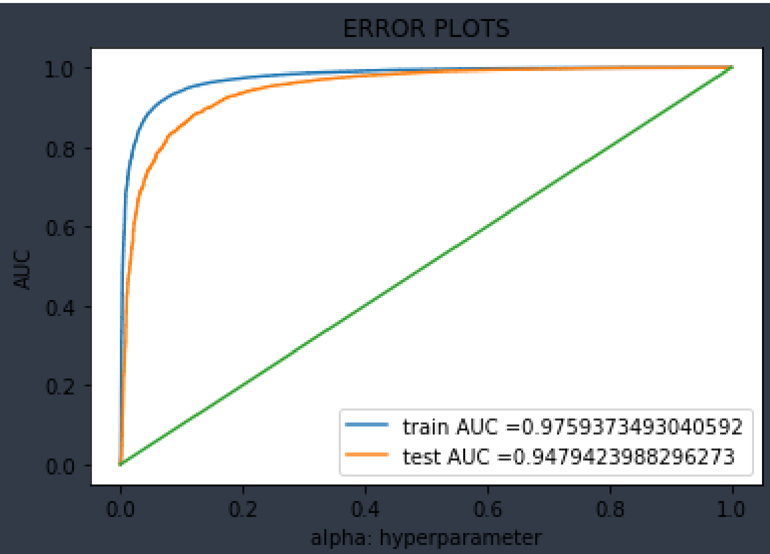


In [74]:
```
best_alpha = 10**(-1)
```

Testing on test data

In [75]:
```
model = MultinomialNB(alpha=best_alpha)
model.fit(New_TFIDF_TRAIN,y_tr)
test_pred = model.predict_log_proba(New_TFIDF_Test)[:,1]
train_pred =model.predict_log_proba(New_TFIDF_TRAIN)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

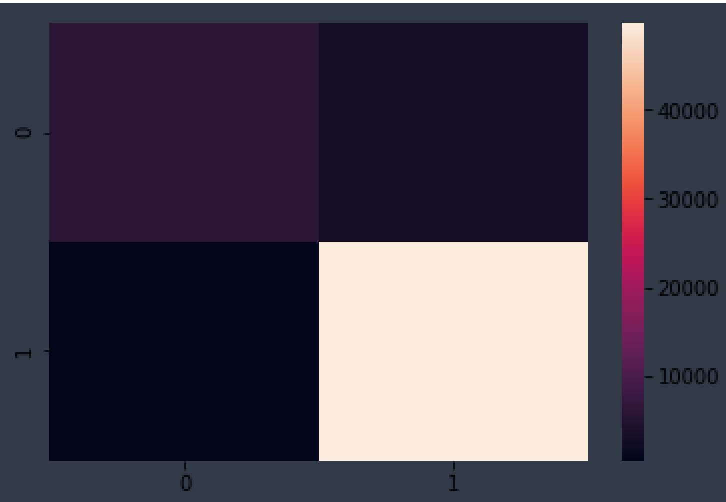


Confusion Matrix

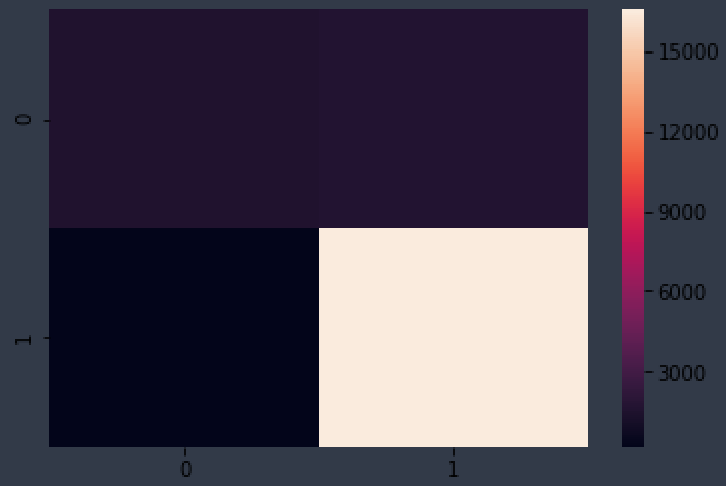
In [76]:

```
ytrain = model.predict(New_TFIDF_TRAIN)
ytest = model.predict(New_TFIDF_Test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
print(ctrain)
plt.show()
sns.heatmap(ctest)
print(ctrain)
plt.show()
```

```
[[ 6265  3330]
 [  566 49671]]
```




```
[[ 6265  3330]
 [  566 49671]]
```



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

In [80]:

```
# Please write all the code with proper documentation
a = model.feature_log_prob_
b =  a[1].argsort()[-11:-1][::-1]
print(" So the top 10 features of positive class are--")
for i in range(10):
    print("feature name : %s , value : %f"%(count_vect.get_feature_names()[b[i]],float(a
[1,b[i]])))
```

```
 So the top 10 features of positive class are--
feature name : intensely , value : -9.105273
feature name : esteemed , value : -9.454600
feature name : emfirst , value : -9.520135
feature name : get , value : -9.563814
feature name : caducity , value : -9.599920
feature name : gruesome , value : -9.693915
feature name : postal , value : -9.699837
feature name : kanin , value : -9.803803
feature name : pleasedand , value : -9.818764
feature name : marine , value : -9.826945
```

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

In [81]:

```
# Please write all the code with proper documentation
a = model.feature_log_prob_
b =  a[0].argsort()[-11:-1][::-1]
print(" So the top 10 features of negative class are--")
for i in range(10):
    print("feature no : %s , value : %f"%(count_vect.get_feature_names()[b[i]],float(a[0
,b[i]])))
```

```
 So the top 10 features of negative class are--
feature no : intensely , value : -8.472209
```

```
feature no : get , value : -9.299468
feature no : marine , value : -9.378093
feature no : pleasedand , value : -9.405943
feature no : samoasserving , value : -9.449215
feature no : caducity , value : -9.703515
feature no : kanin , value : -9.714579
feature no : ingrid , value : -9.849424
feature no : desires , value : -9.863774
feature no : emfirst , value : -9.956270
```

[6] Conclusions

Hence after analyzing we can conclude that tfidf gives heighest auc value value on our data with alpha=0.1 , one more thing that we onserved is when we apply Multinomial Naive Bayes after doing some feature engineering that our model is tend to overfit quickly although it impact much on our model and our model didn't improve after applying feature engineering

In [82]:

```python
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","value of alpha","Train AUC","Test AUC"]

x.add_row(["BOW",1,0.95,0.91])
x.add_row(["Featurized_BOW",0.1,0.92,0.91])
x.add_row(["TFIDF",0.1,0.98,0.95])
x.add_row(["Featurized_TFIDF",0.1,0.97,0.94])
print(x)
```

```
+------------------+----------------+-----------+----------+
|      Model       | value of alpha | Train AUC | Test AUC |
+------------------+----------------+-----------+----------+
|       BOW        |       1        |    0.95   |   0.91   |
|  Featurized_BOW  |      0.1       |    0.92   |   0.91   |
```

```
|      TFIDF       |      0.1       |    0.98   |   0.95   |
| Featurized_TFIDF |      0.1       |    0.97   |   0.94   |
+------------------+----------------+-----------+----------+
```