Amazon Fine Food Reviews Analysis

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

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

In [5]:
```
!ls
```

```
drive  sample_data
```

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [8]:
```
%cd ./drive/'My Drive'
```

```
/content/drive/My Drive
```

In [7]:
```
from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [9]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 L
IMIT 100000""", 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 (100000, 10)
```

Out[9]:

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

In [0]:

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

In [11]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[11]:

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

In [12]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[12]:

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

In [13]:
```
display['COUNT(*)'].sum()
```

Out[13]: 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 [14]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[14]:

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

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

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

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

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

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

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

In [16]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
}, keep='first', inplace=False)
final.shape
```

Out[16]:
```
(87775, 10)
```

In [17]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[17]:
```
87.775
```

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

In [18]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[18]:

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

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

In [20]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

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

```
(87773, 10)
```

Out[20]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [21]:
```
# 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 t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy
has little taste to it.  Very little of the 2 lbs that I bought were eaten an
d I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So
if you are afraid of the fishy smell, don't get it. But I think my dog likes
it because of the smell. These treats are really small in size. They are grea
t for training. You can give your dog several of these without worrying about
him over eating. Amazon's price was much more reasonable than any other retai
ler. You can buy a 1 pound bag on Amazon for almost the same price as a 6 oun
ce bag at other retailers. It's definitely worth it to buy a big bag if your
dog eats them a lot.
==================================================
```

In [22]:
```
# 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 t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
```

In [23]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
ove-all-tags-from-an-element
from bs4 import BeautifulSoup

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

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

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

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

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

In [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [25]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

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

In [26]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/408403
9
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
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 t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
```

In [27]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

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

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he'
, 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becau
se', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [29]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:50<00:00, 1751.91it/s]
```

In [30]:
```
preprocessed_reviews[1500]
```

Out[30]:
```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

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

[4] Featurization

[4.1] BAG OF WORDS

In [32]:
```
#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', 'aaaaaaaaa
aaaaaa', '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 [33]:
```
#bi-gram, tri-gram and n-gram

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

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

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

[4.3] TF-IDF

In [34]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_featu
re_names()[0:10])
print('='*50)

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

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'ab
andon', '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 [36]:
```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('fantastic', 0.8591887950897217), ('good', 0.8366801142692566), ('awesome',
0.8230043649673462), ('excellent', 0.8213452696800232), ('terrific', 0.801186
0847473145), ('wonderful', 0.7864222526550293), ('perfect', 0.77493870258331
3), ('amazing', 0.7338196039199829), ('nice', 0.7024457454681396), ('fabulou
s', 0.6941018104553223)]
==================================================
[('greatest', 0.7767603397369385), ('best', 0.7081184387207031), ('tastiest',
0.700896143913269), ('nastiest', 0.6746709942817688), ('disgusting', 0.654035
9258651733), ('terrible', 0.6392963528633118), ('horrible', 0.630312919616699
2), ('awful', 0.618974506855011), ('smoothest', 0.6104997992515564), ('ive',
0.5959529280662537)]
```

In [37]:
```
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  17386
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buyin
g', 'anymore', 'hard', 'find', 'products', 'made', 'usa', 'one', 'isnt', 'ba
d', 'good', 'take', 'chances', 'till', 'know', 'going', 'imports', 'love', 's
aw', 'pet', 'store', 'tag', 'attached', 'regarding', 'satisfied', 'safe', 'in
festation', 'literally', 'everywhere', 'flying', 'around', 'kitchen', 'bough
t', 'hoping', 'least', 'get', 'rid', 'weeks', 'fly', 'stuck', 'squishing', 'b
uggers', 'success', 'rate']
```

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

[4.4.1.1] Avg W2v

[4.4.1.2] TFIDF weighted W2v

In [38]:
```
"""# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might
need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))"""
```

Out[38]:
```
"# average Word2Vec\n# compute average word2vec for each review.\nsent_vector
s = []; # the avg-w2v for each sentence/review is stored in this list\nfor se
nt in tqdm(list_of_sentance): # for each review/sentence\n    sent_vec = np.z
eros(50) # as word vectors are of zero length 50, you might need to change th
is to 300 if you use google's w2v\n    cnt_words =0; # num of words with a va
lid vector in the sentence/review\n    for word in sent: # for each word in a
review/sentence\n        if word in w2v_words:\n            vec = w2v_model.w
v[word]\n            sent_vec += vec\n            cnt_words += 1\n    if cnt_
words != 0:\n        sent_vec /= cnt_words\n    sent_vectors.append(sent_vec)
\nprint(len(sent_vectors))\nprint(len(sent_vectors[0]))"
```

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 [40]:
```
"""# 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"""
```

Out[40]:
```
'# TF-IDF weighted Word2Vec\ntfidf_feat = model.get_feature_names() # tfidf w
ords/col-names\n# final_tf_idf is the sparse matrix with row= sentence, col=w
ord and cell_val = tfidf\n\ntfidf_sent_vectors = []; # the tfidf-w2v for each
sentence/review is stored in this list\nrow=0;\nfor sent in tqdm(list_of_sent
ance): # for each review/sentence \n    sent_vec = np.zeros(50) # as word vec
tors are of zero length\n    weight_sum =0; # num of words with a valid vecto
r in the sentence/review\n    for word in sent: # for each word in a review/s
entence\n        if word in w2v_words and word in tfidf_feat:\n            ve
c = w2v_model.wv[word]\n#             tf_idf = tf_idf_matrix[row, tfidf_feat.
index(word)]\n            # to reduce the computation we are \n            #
dictionary[word] = idf value of word in whole courpus\n            # sent.cou
nt(word) = tf valeus of word in this review\n            tf_idf = dictionary
[word]*(sent.count(word)/len(sent))\n            sent_vec += (vec * tf_idf)\n
weight_sum += tf_idf\n    if weight_sum != 0:\n        sent_vec /= weight_sum
\n    tfidf_sent_vectors.append(sent_vec)\n    row += 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

   - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using absolute values of `coef_` parameter of MultinomialNB (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
   - (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

6. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

- 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

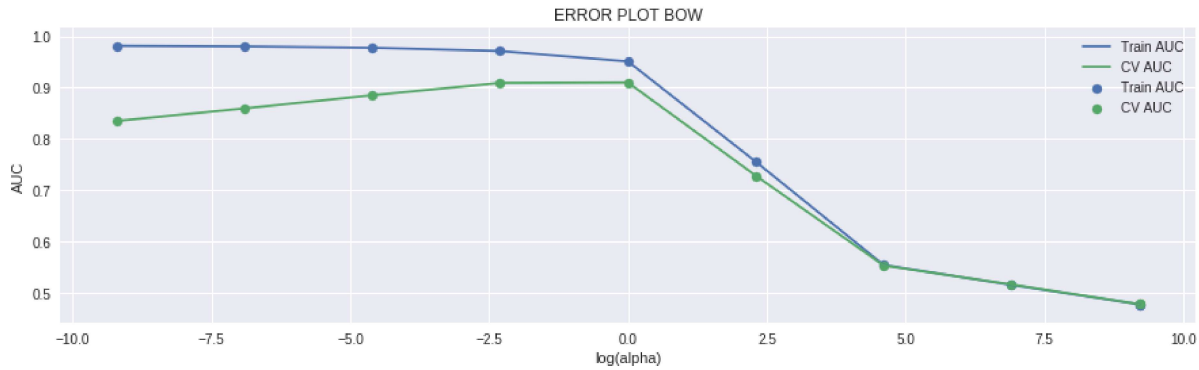
Applying Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

In [66]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
Y = final['Score'].values

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#print(X_train.shape, Y_train.shape)
#print(X_cv.shape, Y_cv.shape)
#print(X_test.shape, Y_test.shape)

print("="*100)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)


#printing the shapes of train,cv and test matrices

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

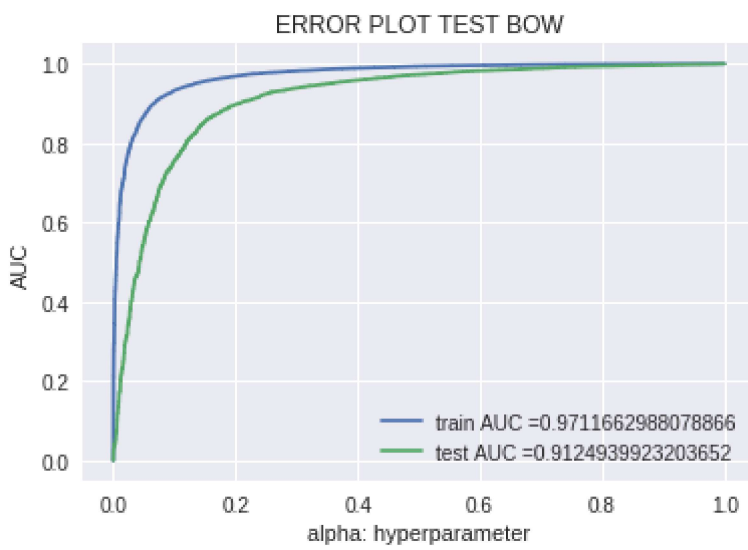
```
==============================================================================
=======================
After vectorizations
(52663, 43059) (52663,)
(17555, 43059) (17555,)
(17555, 43059) (17555,)
==============================================================================
=======================
```

In [67]:
```
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from math import log
import numpy as np
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
b = [10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]

#https://stackoverflow.com/questions/47582264/python-how-to-convert-a-list-to-
loglist
#converting list to log list so that it is easy to plot with log list values
a=list(np.log([10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]))

for i in b:
    mnb = MultinomialNB(alpha=i)
    mnb.fit(X_train_bow, y_train)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred = mnb.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  mnb.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))

# plotting the error plot log(list) vs AUC a=log(list) of both train and cv cu
rves
plt.figure(figsize=(15,4))
plt.plot(a, train_auc, label='Train AUC')
plt.scatter(a, train_auc, label='Train AUC')
plt.plot(a, cv_auc, label='CV AUC')
plt.scatter(a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log(alpha)")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW")
plt.show()
```



In [0]:
```
best_alpha =0.1
```

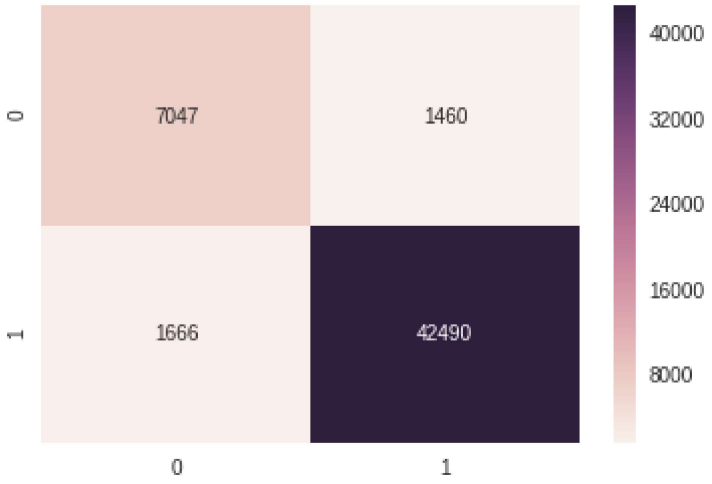
In [69]:
```
from sklearn.metrics import roc_curve, auc


#applying mnb on best_alpha
mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class


train_fpr, train_tpr, thresholds = roc_curve(y_train, mnb.predict_proba(X_trai
n_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, mnb.predict_proba(X_test_bo
w)[:,1])

#plotting the results
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST BOW")
plt.show()

print("="*100)
```



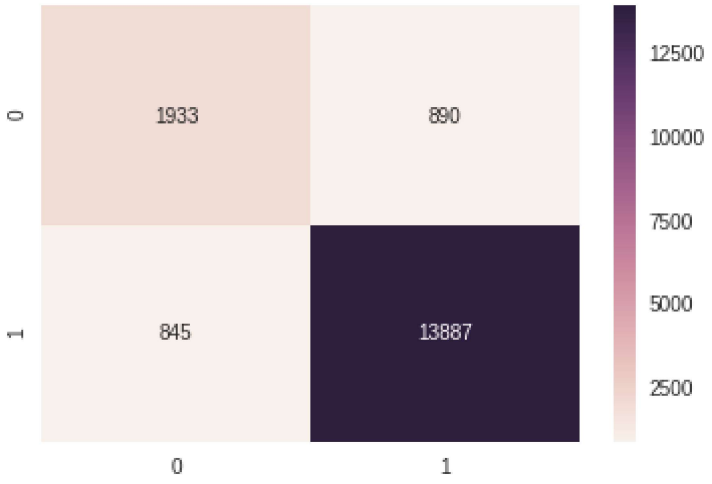

```
====================================================================================
========================
```

In [70]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
%matplotlib inline

#printing train confusion matrix
print("Train confusion matrix")
print(type(X_train_bow))
df_train=pd.DataFrame(confusion_matrix(y_train,mnb.predict(X_train_bow)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

#printing test confusion matrix
print("Test confusion matrix")
df_test=pd.DataFrame(confusion_matrix(y_test,mnb.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
<class 'scipy.sparse.csr.csr_matrix'>
```




```
Test confusion matrix
```



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

In [77]:
```
mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_test_bow, y_test)




# https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-i
n-naive-bayes

#Finding feature log probability values and sorting them according to their in
dex
pos=np.argsort((mnb.feature_log_prob_)[1])[::-1][0:10]

#Mapping the top 10 positive  values against respective feature names
print(np.take(vectorizer.get_feature_names(), pos))
```

```
['not' 'like' 'good' 'great' 'one' 'taste' 'coffee' 'flavor' 'love' 'tea']
```

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

In [76]:
```
# https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-i
n-naive-bayes

#Finding feature log probability values and sorting them according to their in
dex
neg=np.argsort((mnb.feature_log_prob_)[0])[::-1][0:10]


#Mapping the top 10 negative  values against respective feature names
print(np.take(vectorizer.get_feature_names(),neg))
```

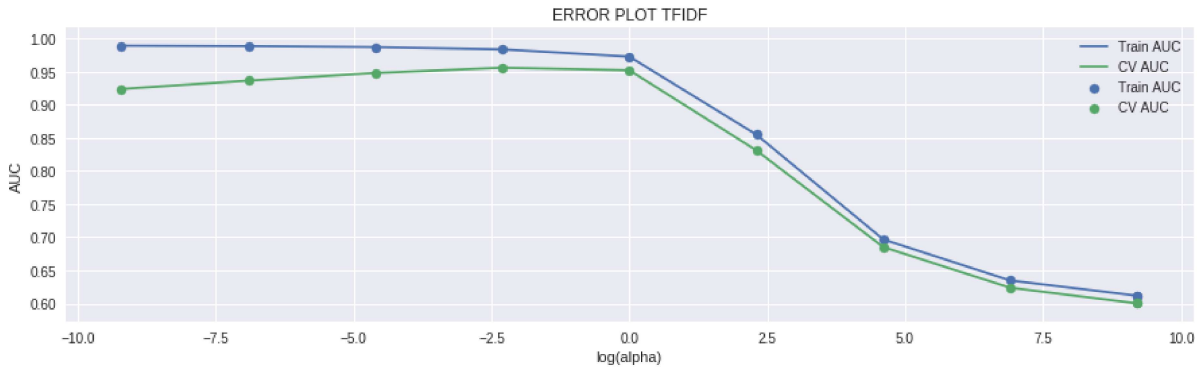
```
['not' 'like' 'would' 'product' 'taste' 'one' 'no' 'good' 'coffee'
 'flavor']
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [78]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#print(X_train.shape, Y_train.shape)
#print(X_cv.shape, Y_cv.shape)
#print(X_test.shape, Y_test.shape)

print("="*100)
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf = tf_idf_vect.transform(X_train)
X_cv_tf = tf_idf_vect.transform(X_cv)
X_test_tf = tf_idf_vect.transform(X_test)

print("After vectorizations")
print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)
print("="*100)
```

```
==============================================================================
=======================
After vectorizations
(52663, 31103) (52663,)
(17555, 31103) (17555,)
(17555, 31103) (17555,)
==============================================================================
=======================
```

In [79]:
```
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from math import log
import numpy as np
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
b = [10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]

#https://stackoverflow.com/questions/47582264/python-how-to-convert-a-list-to-
loglist
#converting list to log list so that it is easy to plot with log list values
a=list(np.log([10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]))

for i in b:
    mnb = MultinomialNB(alpha=i)
    mnb.fit(X_train_tf, y_train)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred = mnb.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  mnb.predict_proba(X_cv_tf)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

# plotting the error plot log(list) vs AUC a=log(list) of both train and cv cu
rves
plt.figure(figsize=(15,4))
plt.plot(a, train_auc, label='Train AUC')
plt.scatter(a, train_auc, label='Train AUC')
plt.plot(a, cv_auc, label='CV AUC')
plt.scatter(a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log(alpha)")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF")
plt.show()
```



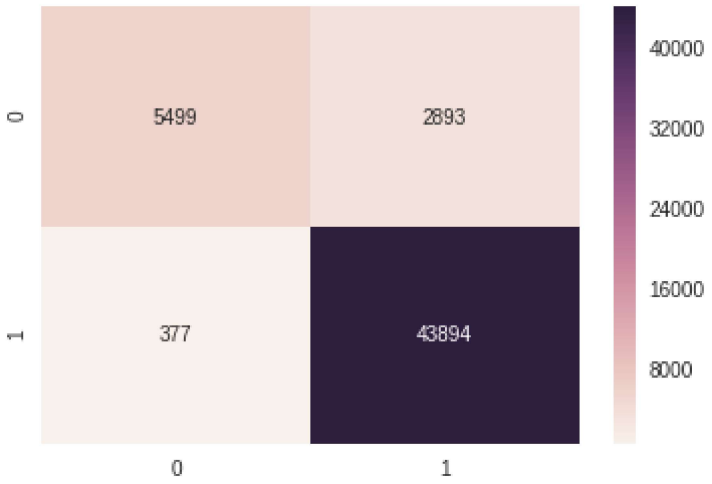
In [0]:
```
best_alpha  = 0.1
```

In [81]:
```
from sklearn.metrics import roc_curve, auc



mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, mnb.predict_proba(X_trai
n_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, mnb.predict_proba(X_test_tf
)[:,1])

#plotting the results
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDF")
plt.show()

print("="*100)
```

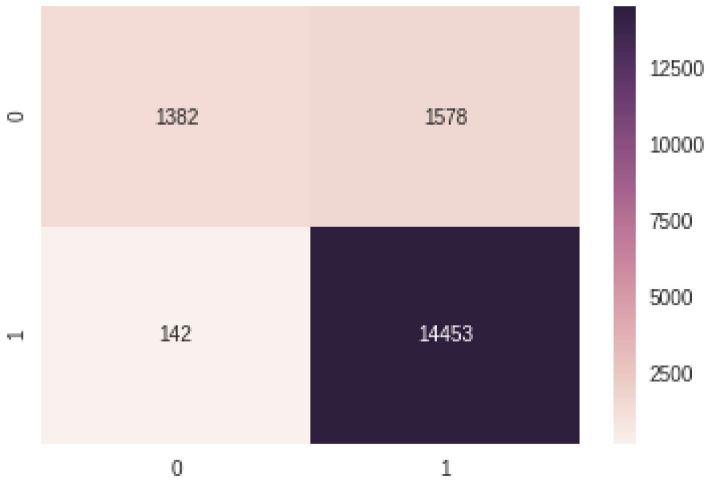
```
====================================================================================
========================
```

In [82]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
%matplotlib inline

#printing train confusion matrix
print("Train confusion matrix")
df_train=pd.DataFrame(confusion_matrix(y_train,mnb.predict(X_train_tf)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

#printing test confusion matrix
print("Test confusion matrix")
df_test=pd.DataFrame(confusion_matrix(y_test,mnb.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



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

In [86]:
```
mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_test_tf, y_test)


# https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-i
n-naive-bayes

#Finding feature log probability values and sorting them according to their in
dex
pos=np.argsort((mnb.feature_log_prob_)[1])[::-1][0:10]

#Mapping the top 10 positive values against respective feature names
print(np.take(tf_idf_vect.get_feature_names(),pos))
```

```
['not' 'great' 'good' 'like' 'coffee' 'tea' 'love' 'one' 'product' 'taste']
```

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

In [85]:
```
# https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-i
n-naive-bayes

#Finding feature log probability values and sorting them according to their in
dex
neg=np.argsort((mnb.feature_log_prob_)[0])[::-1][0:10]

#Mapping the top 10 negative  values against respective feature names
print(np.take(tf_idf_vect.get_feature_names(),neg))
```

```
['not' 'like' 'product' 'would' 'taste' 'coffee' 'one' 'no' 'flavor' 'tea']
```

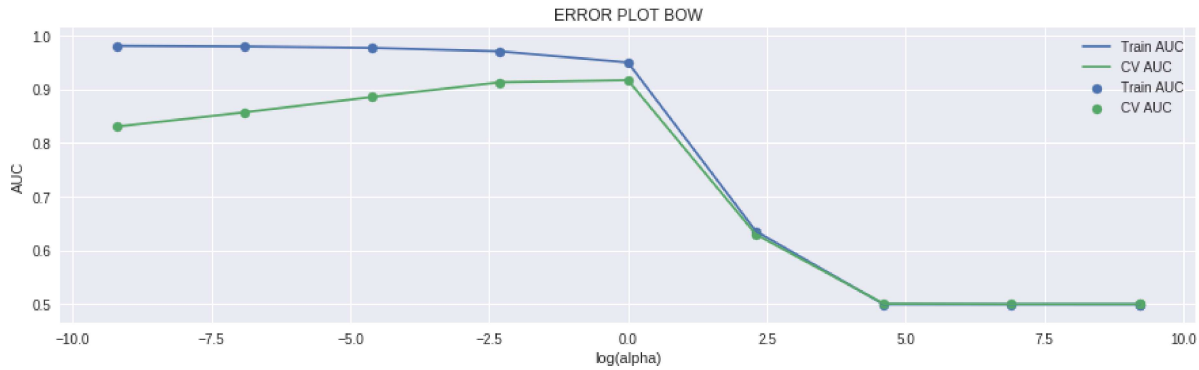
Feature Engineering

In [55]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix
from scipy.sparse import hstack
from scipy.sparse import coo_matrix

X = preprocessed_reviews
Y = final['Score'].values

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#X_tr_rl is lengths of reviews of X_train
i=0
X_tr_rl=[]
for sentance in X_train:
    X_tr_rl.append(len(sentance))


#X_cv_rl is lengths of reviews of X_cv
i=0
X_cv_rl=[]
for sentance in X_cv:
    X_cv_rl.append(len(sentance))

#X_test_rl is lengths of reviews of X_test
i=0
X_test_rl=[]
for sentance in X_test:
    X_test_rl.append(len(sentance))



print("="*100)



from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

```
#Adding review length column to X_train
X_train_bow=coo_matrix(X_train_bow)
X_tr_rl=coo_matrix(X_tr_rl)
X_train_bow=hstack([X_train_bow,X_tr_rl.T])


#Adding review length column to X_cv
X_cv_bow=coo_matrix(X_cv_bow)
X_cv_rl=coo_matrix(X_cv_rl)
X_cv_bow=hstack([X_cv_bow,X_cv_rl.T])

#Adding review length column to X_test
X_test_bow=coo_matrix(X_test_bow)
X_test_rl=coo_matrix(X_test_rl)
X_test_bow=hstack([X_test_bow,X_test_rl.T])



print("After adding the review length column")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

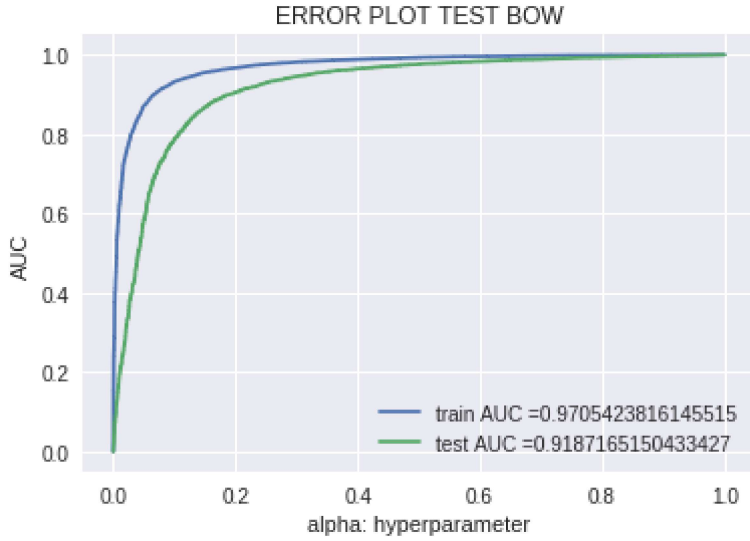
```
==============================================================================
=======================
(52663, 43187) (52663,)
(17555, 43187) (17555,)
(17555, 43187) (17555,)
==============================================================================
=======================
After adding the review length column
(52663, 43188) (52663,)
(17555, 43188) (17555,)
(17555, 43188) (17555,)
==============================================================================
=======================
```

In [56]:
```
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from math import log
import numpy as np
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
b = [10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]

#https://stackoverflow.com/questions/47582264/python-how-to-convert-a-list-to-
loglist
#converting list to log list so that it is easy to plot with log list values
a=list(np.log([10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]))

for i in b:
    mnb = MultinomialNB(alpha=i)
    mnb.fit(X_train_bow, y_train)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred = mnb.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  mnb.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))

# plotting the error plot log(list) vs AUC a=log(list) of both train and cv cu
rves
plt.figure(figsize=(15,4))
plt.plot(a, train_auc, label='Train AUC')
plt.scatter(a, train_auc, label='Train AUC')
plt.plot(a, cv_auc, label='CV AUC')
plt.scatter(a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log(alpha)")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW")
plt.show()
```



In [0]:
```
best_alpha=0.1
```

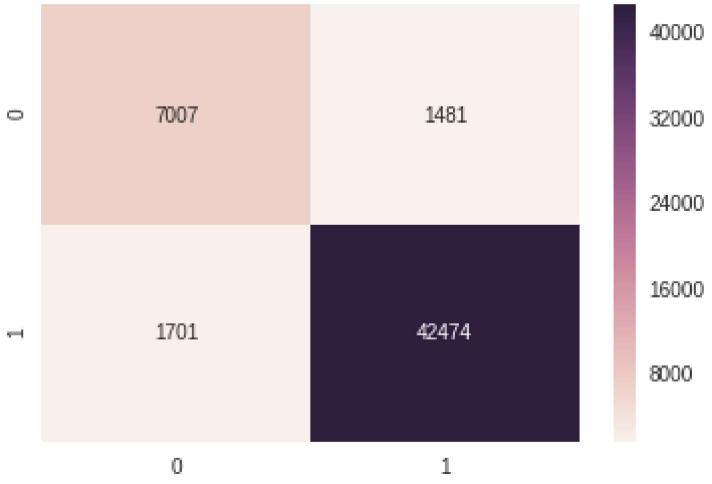
In [58]:
```
from sklearn.metrics import roc_curve, auc



mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, mnb.predict_proba(X_trai
n_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, mnb.predict_proba(X_test_bo
w)[:,1])

#plotting the results
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST BOW")
plt.show()

print("="*100)
```



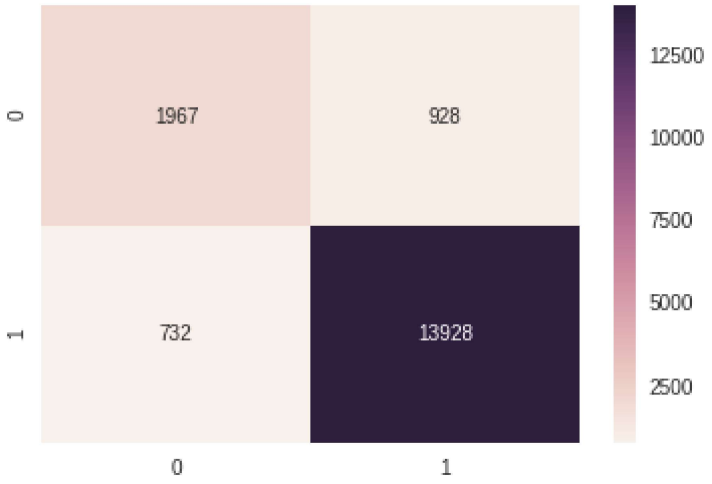

```
====================================================================================
========================
```

In [59]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
%matplotlib inline


print("Train confusion matrix")
print(type(X_train_bow))
df_train=pd.DataFrame(confusion_matrix(y_train,mnb.predict(X_train_bow)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")
df_test=pd.DataFrame(confusion_matrix(y_test,mnb.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
<class 'scipy.sparse.coo.coo_matrix'>
```




```
Test confusion matrix
```



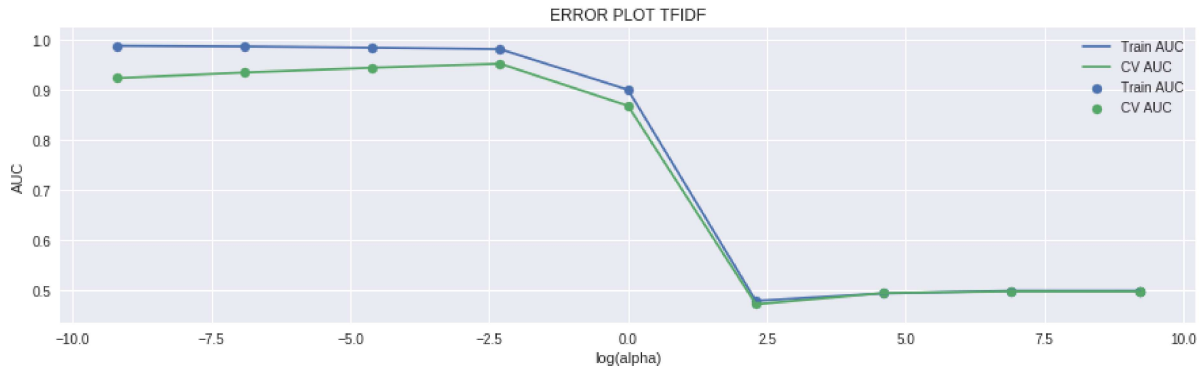
TFIDF

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#X_tf_tr_rl is lengths of reviews of X_train
i=0
X_tf_tr_rl=[]
for sentance in X_train:
    X_tf_tr_rl.append(len(sentance))


#X_tf_cv_rl is lengths of reviews of X_cv
i=0
X_tf_cv_rl=[]
for sentance in X_cv:
    X_tf_cv_rl.append(len(sentance))

#X_tf_test_rl is lengths of reviews of X_test
i=0
X_tf_test_rl=[]
for sentance in X_test:
    X_tf_test_rl.append(len(sentance))

print("="*100)
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf = tf_idf_vect.transform(X_train)
X_cv_tf = tf_idf_vect.transform(X_cv)
X_test_tf = tf_idf_vect.transform(X_test)

print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)

#Adding review length column to X_train
X_train_tf=coo_matrix(X_train_tf)
X_tf_tr_rl=coo_matrix(X_tf_tr_rl)
X_train_tf=hstack([X_train_tf,X_tf_tr_rl.T])
```

```
#Adding review length column to X_cv
X_cv_tf=coo_matrix(X_cv_tf)
X_tf_cv_rl=coo_matrix(X_tf_cv_rl)
X_cv_tf=hstack([X_cv_tf,X_tf_cv_rl.T])

#Adding review length column to X_test
X_test_tf=coo_matrix(X_test_tf)
X_tf_test_rl=coo_matrix(X_tf_test_rl)
X_test_tf=hstack([X_test_tf,X_tf_test_rl.T])




print("After adding review length")
print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)
print("="*100)
```

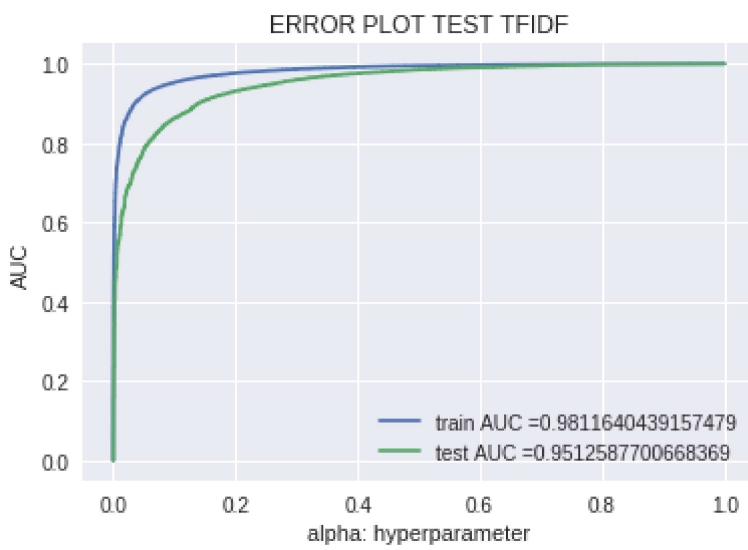
```
==============================================================================
=======================
(52663, 31036) (52663,)
(17555, 31036) (17555,)
(17555, 31036) (17555,)
After adding review length
(52663, 31037) (52663,)
(17555, 31037) (17555,)
(17555, 31037) (17555,)
==============================================================================
=======================
```

In [61]:
```
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from math import log
import numpy as np
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
b = [10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]

#https://stackoverflow.com/questions/47582264/python-how-to-convert-a-list-to-
loglist
#converting list to log list so that it is easy to plot with log list values
a=list(np.log([10**-4,10**-3,10**-2,10**-1,1,10,100,1000,10000,10000]))

for i in b:
    mnb = MultinomialNB(alpha=i)
    mnb.fit(X_train_tf, y_train)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred = mnb.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  mnb.predict_proba(X_cv_tf)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

# plotting the error plot log(list) vs AUC a=log(list) of both train and cv cu
rves
plt.figure(figsize=(15,4))
plt.plot(a, train_auc, label='Train AUC')
plt.scatter(a, train_auc, label='Train AUC')
plt.plot(a, cv_auc, label='CV AUC')
plt.scatter(a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log(alpha)")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF")
plt.show()
```



In [0]:
```
best_alpha=0.1
```

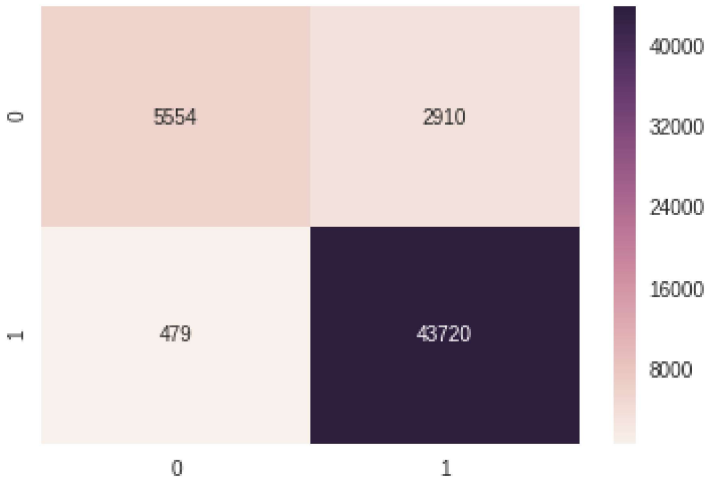
In [63]:
```
from sklearn.metrics import roc_curve, auc



mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, mnb.predict_proba(X_trai
n_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, mnb.predict_proba(X_test_tf
)[:,1])

#plotting the results
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDF")
plt.show()

print("="*100)
```



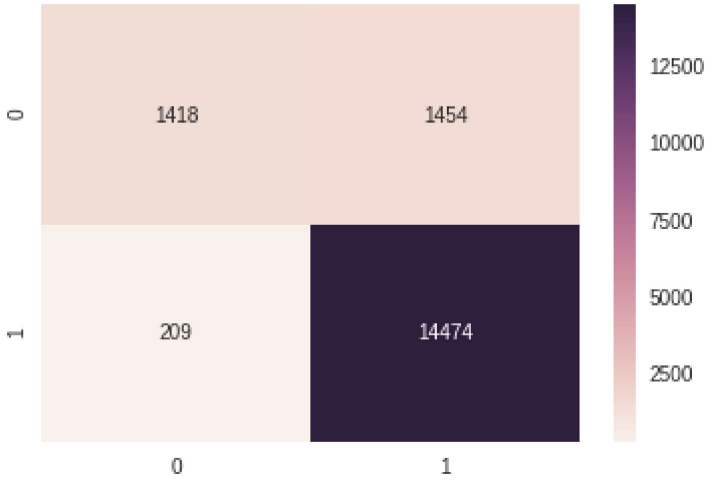

```
====================================================================================
========================
```

In [64]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
%matplotlib inline

#printing train confusion matrix
print("Train confusion matrix")
df_train=pd.DataFrame(confusion_matrix(y_train,mnb.predict(X_train_tf)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

#printing test confusion matrix
print("Test confusion matrix")
df_test=pd.DataFrame(confusion_matrix(y_test,mnb.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



[6] Pretty Table

In [65]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "F.E", "Hyper-Parameter", "AUC"]

#F.E =Feature Engineering

x.add_row(["BOW", "No", 0.1,0.9101 ])
x.add_row(["TFIDF", "No",0.1,0.9525])

x.add_row(["BOW", "Yes", 0.1,0.9084])
x.add_row(["TFIDF", "Yes",0.1,0.9522])



print(x)
```

```
+------------+-----+-----------------+--------+
| Vectorizer | F.E | Hyper-Parameter |  AUC   |
+------------+-----+-----------------+--------+
|    BOW     |  No |       0.1       | 0.9101 |
|   TFIDF    |  No |       0.1       | 0.9525 |
|    BOW     | Yes |       0.1       | 0.9084 |
|   TFIDF    | Yes |       0.1       | 0.9522 |
+------------+-----+-----------------+--------+
```

Conclusions

1.Naive bayes algorithm is far quicker than knn algorithm.

2.Adding Review length as another feature doesnot cause much difference to AUC.

3.Applying multinomial naive bayes on TFIDF vectorizer produces best AUC.