[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".

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.

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.linear_model import LogisticRegression

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]:
```
from google.colab import drive


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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2F
www.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fau
th%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=
code

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

In [3]:
```
%cd ./drive/My Drive
```

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

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 [8]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[8]:

	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 [6]:
```
# 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 10000""", 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 (10000, 10)
```

Out[6]:

	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 [9]:
```
display['COUNT(*)'].sum()
```

Out[9]: 393063

In [10]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[10]:

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

[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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[11]:

	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 [18]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
}, keep='first', inplace=False)
final.shape
```

Out[18]: (9564, 10)

In [19]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[19]: 95.64

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 [20]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[20]:

	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 [22]:
```
#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()
```

```
(9564, 10)
```

Out[22]:
```
1    7976
0    1588
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 [23]:
```
# 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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great produc
t!
==================================================
15 month old loves to eat them on the go! They seem great for a healthy, quic
k, and easy snack!
==================================================
These chips are truly amazing. They have it all. They're light, crisp, great
tasting, nice texture, AND they're all natural... AND low in fat and sodium!
Need I say more? I recently bought a bag of them at a regular grocery store,
and couldn't belive my taste buds. That's why I excited why I saw them here o
n Amazon, and decided to buy a case!
==================================================
These tablets definitely made things sweeter -- like lemons, limes, and grape
fruit.  But it wasn't to the point of sheer amazement.  They also had an inte
resting effect on cheeses and vinegar, but still did virtually nothing for be
er and wine. The tablets are a bit pricey but they do work.  If you've got ex
tra money, sure, give them a try, but if you're looking for some amazing way
to get your kids to eat broccoli or something along those lines then this is
not the answer. Fun experiment, but not life-changing. :)
==================================================
```

In [24]:
```
# 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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great produc
t!
```

In [25]:
```
# 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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great produc
t!
==================================================
15 month old loves to eat them on the go! They seem great for a healthy, quic
k, and easy snack!
==================================================
These chips are truly amazing. They have it all. They're light, crisp, great
tasting, nice texture, AND they're all natural... AND low in fat and sodium!
Need I say more? I recently bought a bag of them at a regular grocery store,
and couldn't belive my taste buds. That's why I excited why I saw them here o
n Amazon, and decided to buy a case!
==================================================
These tablets definitely made things sweeter -- like lemons, limes, and grape
fruit.  But it wasn't to the point of sheer amazement.  They also had an inte
resting effect on cheeses and vinegar, but still did virtually nothing for be
er and wine. The tablets are a bit pricey but they do work.  If you've got ex
tra money, sure, give them a try, but if you're looking for some amazing way
to get your kids to eat broccoli or something along those lines then this is
not the answer. Fun experiment, but not life-changing. :)
```

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 [27]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
These chips are truly amazing. They have it all. They are light, crisp, great
tasting, nice texture, AND they are all natural... AND low in fat and sodium!
Need I say more? I recently bought a bag of them at a regular grocery store,
and could not belive my taste buds. That is why I excited why I saw them here
on Amazon, and decided to buy a case!
==================================================
```

In [28]:
```
#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)
```

```
We have used the Victor fly bait for  seasons.  Can't beat it.  Great produc
t!
```

In [29]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
These chips are truly amazing They have it all They are light crisp great tas
ting nice texture AND they are all natural AND low in fat and sodium Need I s
ay more I recently bought a bag of them at a regular grocery store and could
not belive my taste buds That is why I excited why I saw them here on Amazon
and decided to buy a case
```

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 [31]:
```
# 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%|██████████| 9564/9564 [00:04<00:00, 2170.39it/s]
```

In [32]:
```
preprocessed_reviews[1500]
```

Out[32]:
```
'chips truly amazing light crisp great tasting nice texture natural low fat s
odium need say recently bought bag regular grocery store could not belive tas
te buds excited saw amazon decided buy case'
```

[3.2] Preprocessing Review Summary

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

[4] Featurization

[4.1] BAG OF WORDS

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

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

[4.2] Bi-Grams and n-Grams.

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

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

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

[4.3] TF-IDF

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

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

[4.4] Word2Vec

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

In [0]:
```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

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

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

[4.4.1.1] Avg W2v

In [0]:
```
"""# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this 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]))"""
```

[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"""
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression 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)
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

   - 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
   - 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. **Pertubation Test**

   - Get the weights W after fit your model with the data X.
   - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
   - Fit the model again on data X' and get the weights W'
   - Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
   - Now find the % change between W and W' (| (W-W') / (W) |)*100)
   - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
   - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
   - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

   - Calculate sparsity on weight vector obtained after using L1 regularization

   NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

   - Get top 10 important features for both positive and negative classes separately.

6. **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.

7. **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.
   - 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)

8. **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 Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [0]:
```
from sklearn.model_selection import train_test_split
#Loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values

#splitting the data into Train,Cross validate and test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

In [35]:
```
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
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_test_bow = vectorizer.transform(X_test)


#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(X_train_bow)
X_train_bow = scaler.transform(X_train_bow)
X_test_bow = scaler.transform(X_test_bow)

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


tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

#Applying logistic regression on Grid search
log=LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(X_train_bow, y_train)

#Finding the best C
print(model.best_estimator_)
print(model.score(X_test_bow, y_test))
```

```
After vectorizations
(7651, 16426) (7651,)
(1913, 16426) (1913,)
==============================================================================
=======================
LogisticRegression(C=1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.9035109474907964
```

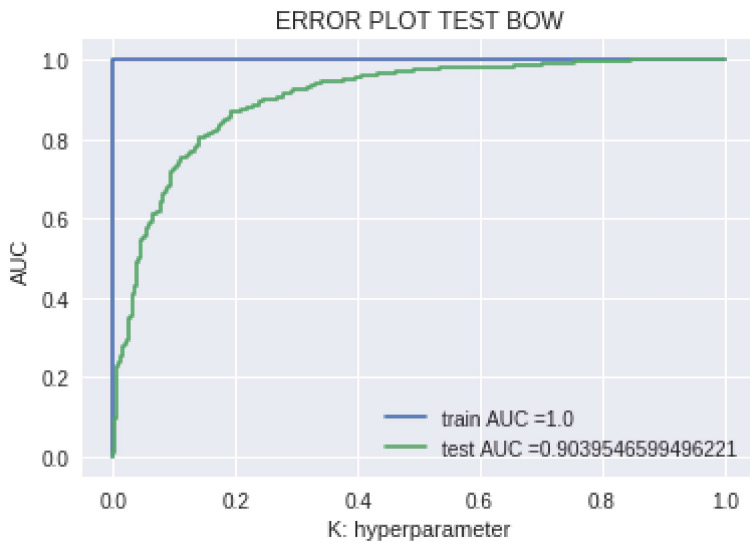
Hyper parameter tuning BOW L1

In [36]:
```
#TO find best C we use the below parameters  in Gridsearch
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')


#Fitting the grid search logistic regression model on train data
model.fit(X_train_bow, y_train)

#These are the AUC values
train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

#plotting the AUC values against C to find best hyperparameter
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW")
plt.show()
```

In [0]:
```
best_c=1
```

Testing with test data

In [38]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1,C=best
_c)
log.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, log.predict_proba(X_trai
n_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST BOW")
plt.show()

print("="*100)
```




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

In [39]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

#printing train_confusion matrix using pandas dataframe and seaborn

df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(X_train_bow)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

#printing train_confusion matrix using pandas dataframe and seaborn
print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```

[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

In [40]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (de
creasing C)

import numpy as np

#Finding no of nonzero weights in weight vector w for c =10
log = LogisticRegression(C=10,class_weight='balanced', penalty='l1')
log.fit(X_train_bow, y_train)
w = log.coef_
print(np.count_nonzero(w))
```

```
2894
```

In [41]:
```
import numpy as np


#Finding no of nonzero weights in weight vector w for c =1
log = LogisticRegression(C=1,class_weight='balanced', penalty='l1')
log.fit(X_train_bow, y_train)
w = log.coef_
print(np.count_nonzero(w))
```

```
2557
```

In [42]:
```
import numpy as np


#Finding no of nonzero weights in weight vector w for c =0.1
log = LogisticRegression(C=0.1,class_weight='balanced', penalty='l1')
log.fit(X_train_bow, y_train)
w = log.coef_
print(np.count_nonzero(w))
```

```
2068
```

In [43]:
```
import numpy as np


#Finding no of nonzero weights in weight vector w for c =0.01
log = LogisticRegression(C=0.01,class_weight='balanced', penalty='l1')
log.fit(X_train_bow, y_train)
w = log.coef_
print(np.count_nonzero(w))

import numpy as np
```

```
546
```

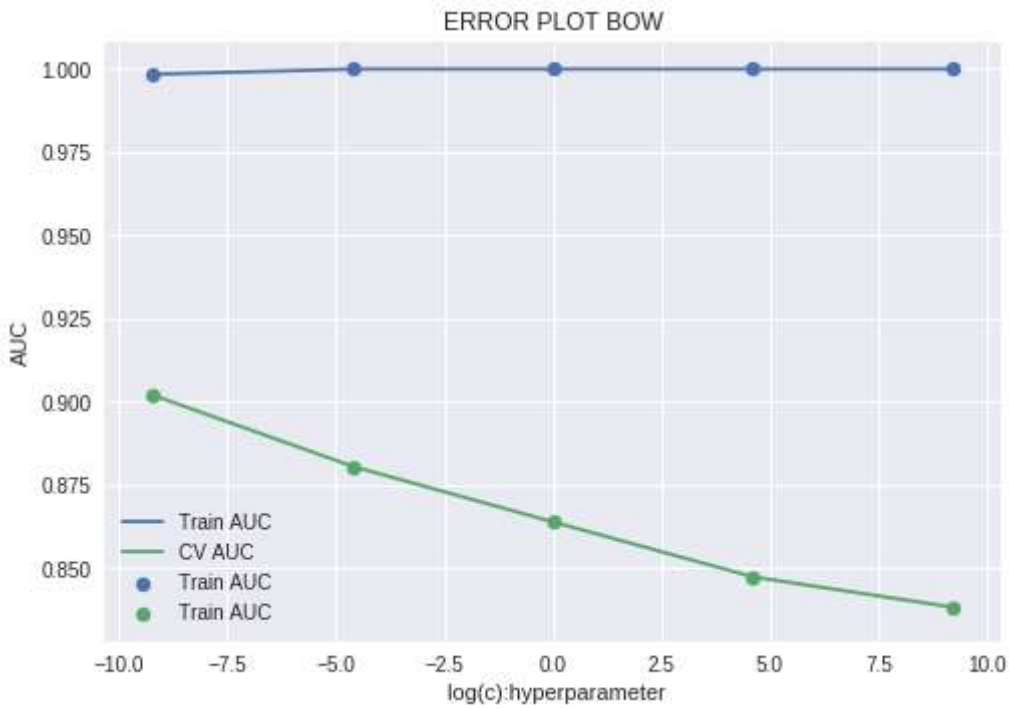
[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [44]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.linear_model import LogisticRegression


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_test_bow = vectorizer.transform(X_test)

#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(X_train_bow)
X_train_bow = scaler.transform(X_train_bow)
X_test_bow = scaler.transform(X_test_bow)

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


tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(X_train_bow, y_train)

print(model.best_estimator_)
print(model.score(X_test_bow, y_test))
```

```
After vectorizations
(7651, 16426) (7651,)
(1913, 16426) (1913,)
====================================================================================
=======================
LogisticRegression(C=0.0001, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.8985681069560163
```

Hyper parameter tuning BOW L2

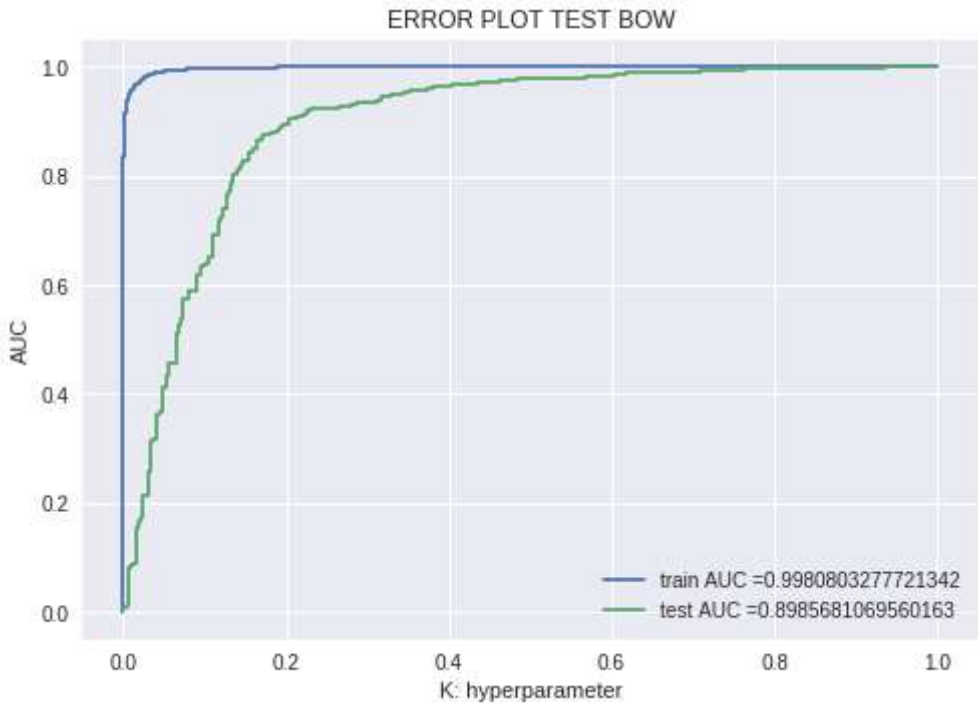
In [45]:
```
#TO find best C we use the below parameters  in Gridsearch

tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

#Fitting the grid search logistic regression model on train data
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(X_train_bow, y_train)

#These are the AUC values
train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']


#plotting the AUC values against C to find best hyperparameter
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW")
plt.show()
```



In [0]:
```
best_c=0.0001
```

Testing with Test Data

In [47]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1,C=best
_c)
log.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, log.predict_proba(X_trai
n_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST BOW")
plt.show()

print("="*100)
```




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

In [48]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

#Using seaborn and pandas dataframe for plotting train_confusion matrix
df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(X_train_bow)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")
#Using seaborn and pandas dataframe for plotting test_confusion matrix
df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```

[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [49]:
```
log = LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1,C=1.0)
log.fit(X_train_bow, y_train)
#Finding the weight vector
w=log.coef_
print(w)
print(w.shape)
```

```
[[ 5.53361661e-03  2.06375309e-02  1.05908253e-05 ...  6.32100933e-03
  -2.84520739e-02  2.69148209e-03]]
(1, 16426)
```

In [0]:
```
import numpy as np

#Adding a random noise to X_train_bow
e= np.random.normal(0,0.1,1)
X_train_bow+=e

#X_train_bow=X_train_bow+e
```

In [52]:
```
#Applying logistic regression on bow on which random noise is added
log2 = LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1,C=1.0
)
log2.fit(X_train_bow, y_train)
```

Out[52]:
```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [53]:
```
w_new=log2.coef_
print(w_new)
```

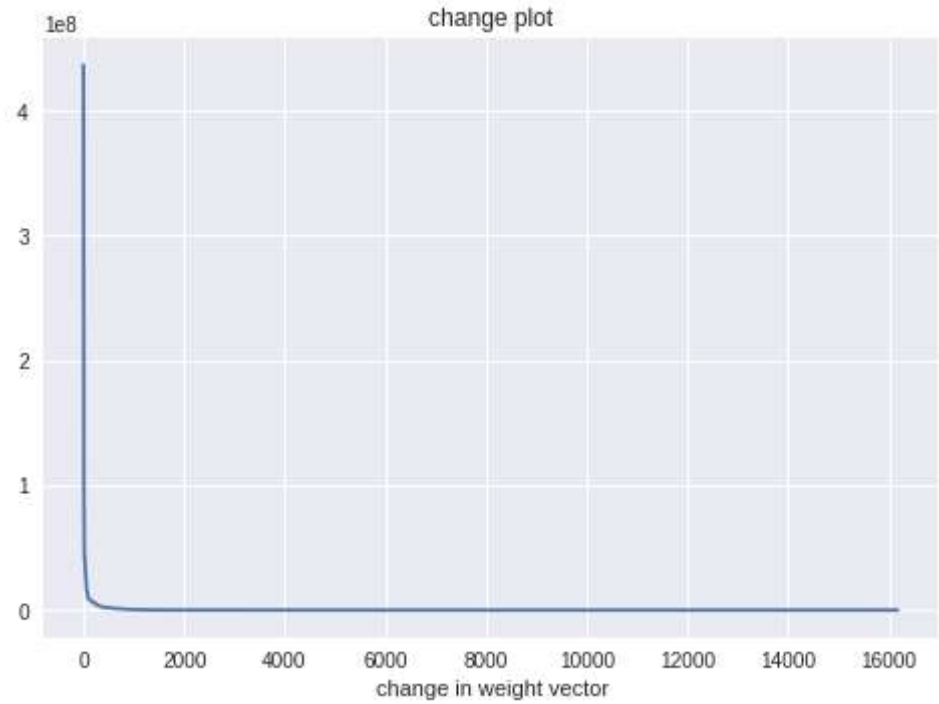
```
[[-0.00532064  0.0118378  -0.01859805 ... -0.00704221 -0.02878196
  -0.00200324]]
```

In [0]:
```
w=w+10**-6
w_new=w_new+10**-6
```

In [0]:
```
change=[]
for i in range (0,16176) :
     a=(abs(w[0][i]-w_new[0][i])/abs(w[0][i]))*100
     change.append(a)
```

In [0]:
```
change.sort(reverse=True)
```

In [57]:
```
#plotting the results

plt.plot(change)


plt.xlabel(" change in weight vector")
#plt.ylabel("no of dimensions")
plt.title("change plot")
plt.show()
```



In [58]:
```
print(np.percentile(change,0))
print(np.percentile(change,10))
print(np.percentile(change,20))
print(np.percentile(change,30))
print(np.percentile(change,40))
print(np.percentile(change,50))
print(np.percentile(change,60))
print(np.percentile(change,70))
print(np.percentile(change,80))
print(np.percentile(change,90))
print(np.percentile(change,100))
```

```
3.884571366400856e-06
6.242734542128228
12.885047367066901
21.849811680698828
34.51892805697244
56.7950324078063
114.16238204996658
303.02240863637473
1480.1087168225577
70318.93026643129
436604142.5479932
```

In [59]:
```
print(np.percentile(change,91))
print(np.percentile(change,92))
print(np.percentile(change,93))
print(np.percentile(change,94))
print(np.percentile(change,95))
print(np.percentile(change,96))
print(np.percentile(change,97))
print(np.percentile(change,98))
print(np.percentile(change,99))
```

```
115027.02157957216
185152.33943338884
280767.04163496156
479143.0116824928
766600.6918983769
1261893.9003941764
2029547.9474265256
3243180.128037035
6979701.534395505
```

In [0]:
```
percent_change=97
```

In [0]:
```
df = pd.DataFrame(data = percent_change, index = vectorizer.get_feature_names
() , columns = ['percent_change']).sort_values(by = 'percent_change',ascending
= False)[:10]
```

In [0]:
```
print(df)
```

```
           percent_change
aa                     97
pointless              97
powell                 97
power                  97
powerbar               97
powerbars              97
powered                97
powerful               97
powering               97
practical              97
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [61]:
```
#Finding feature log probability values and sorting them according to their in
dex
pos=np.argsort((log.coef_)[0])[::-1][0:10]

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

```
['great' 'love' 'good' 'best' 'delicious' 'nice' 'find' 'favorite'
 'perfect' 'smooth']
```

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

In [62]:
```
#Finding feature log probability values and sorting them according to their in
dex
neg=np.argsort((log.coef_)[0])[0:10]

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

```
['not' 'weak' 'grocery' 'disappointing' 'disappointed' 'horrible' 'salty'
 'okay' 'sour' 'thought']
```

[5.2] Logistic Regression on TFIDF, SET 2

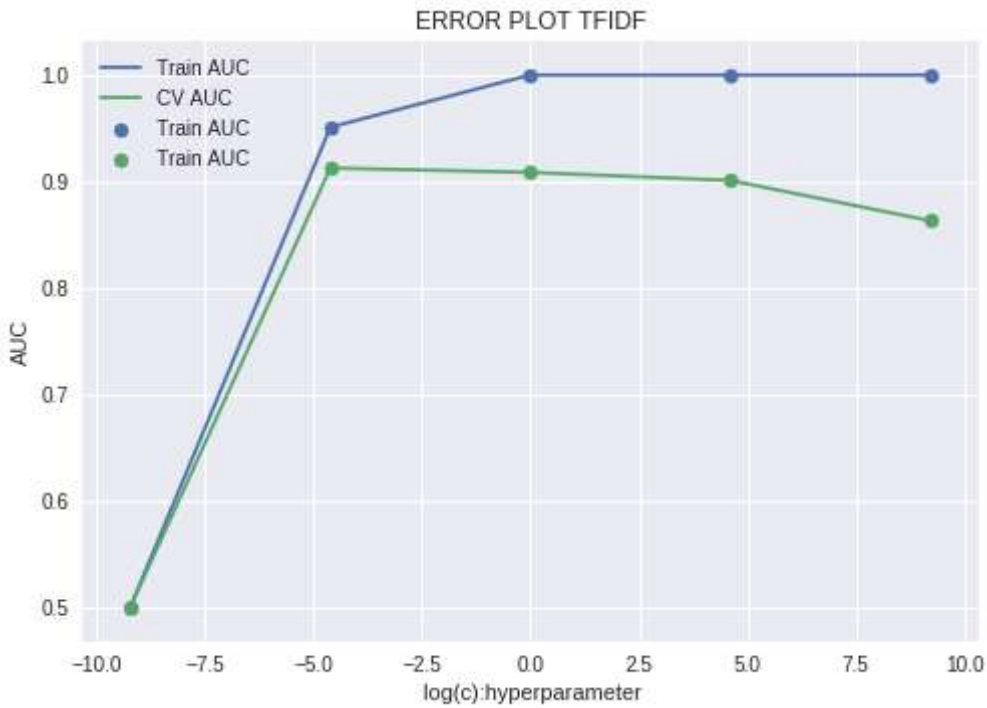
[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [63]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
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_test_tf = tf_idf_vect.transform(X_test)

#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(X_train_tf)
X_train_tf = scaler.transform(X_train_tf)
X_test_tf = scaler.transform(X_test_tf)

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


tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(X_train_tf, y_train)

print(model.best_estimator_)
print(model.score(X_test_tf, y_test))
```

```
After vectorizations
(7651, 4665) (7651,)
(1913, 4665) (1913,)
==============================================================================
=======================
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.9267816314667701
```

Hyper parameter tuning TFIDF L1

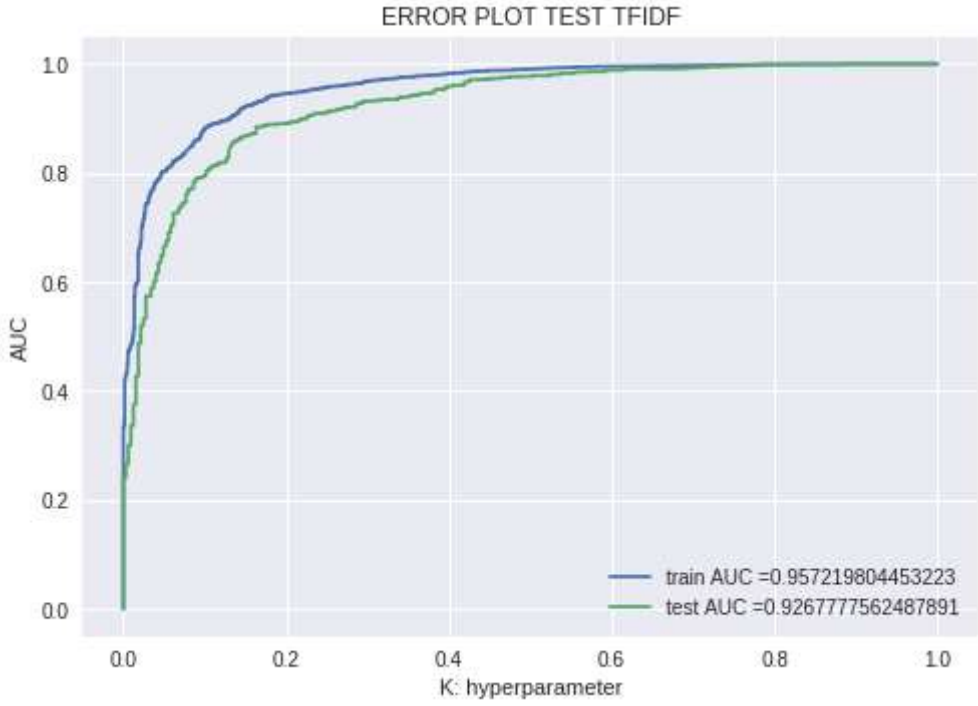
In [64]:
```
#Finding the best hyperparameter using gridsearch cv
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(X_train_tf, y_train)

train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

#plotting the AUC plot for different hyperparameters
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF")
plt.show()
```



In [0]:
```
best_c=0.01
```

Testing With Test Data

In [66]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1,C=best
_c)
log.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, log.predict_proba(X_trai
n_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDF")
plt.show()

print("="*100)
```




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

In [67]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(X_train_tf)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```

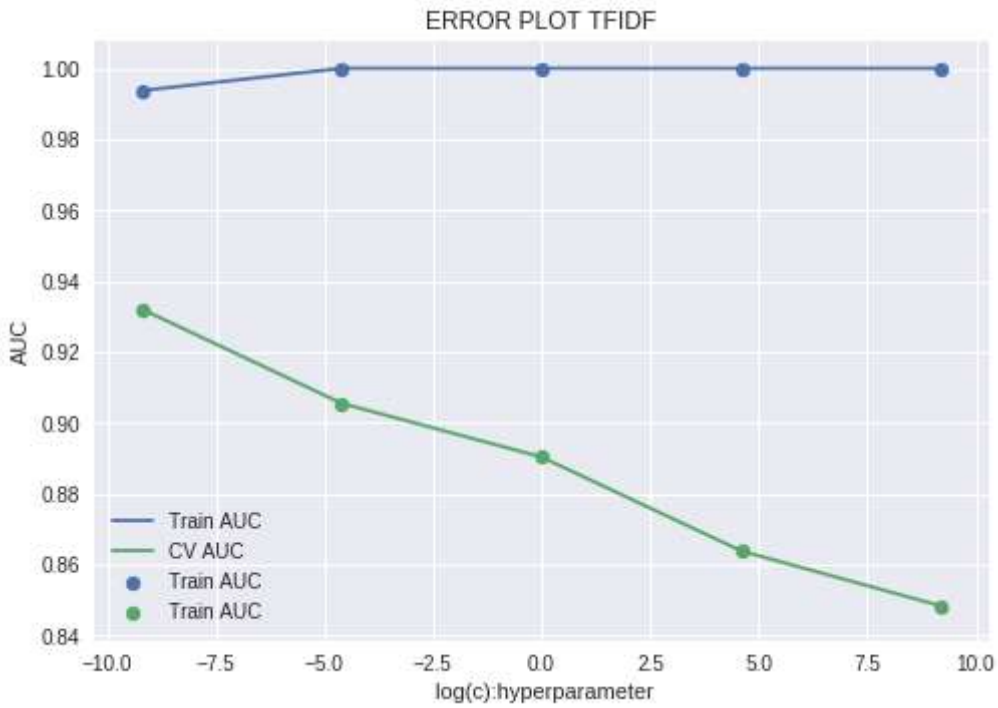
[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [68]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
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_test_tf = tf_idf_vect.transform(X_test)

#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(X_train_tf)
X_train_tf = scaler.transform(X_train_tf)
X_test_tf = scaler.transform(X_test_tf)

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


tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(X_train_tf, y_train)

print(model.best_estimator_)
print(model.score(X_test_tf, y_test))
```

```
After vectorizations
(7651, 4665) (7651,)
(1913, 4665) (1913,)
==============================================================================
=======================
LogisticRegression(C=0.0001, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.9466614997093585
```

Hyper parameter tuning TFIDF L2

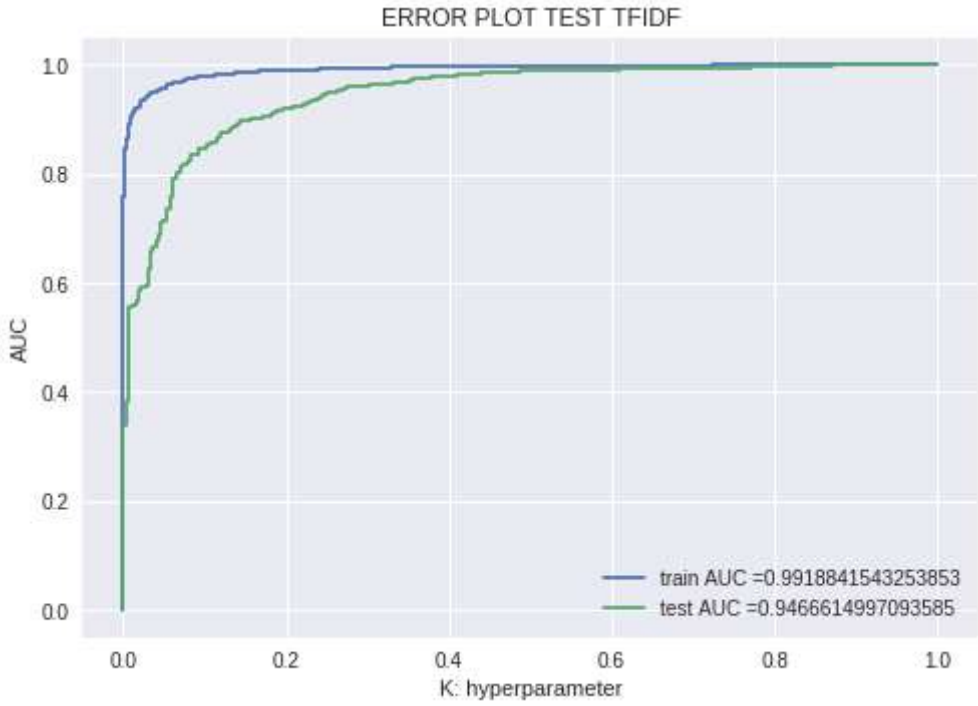
In [69]:
```
#Finding best hyperparameter using gridsearch cv

tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(X_train_tf, y_train)

train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

#plotting AUC for different hyper parameters
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF")
plt.show()
```



In [0]:
```
best_c=0.0001
```

Testing with Test Data

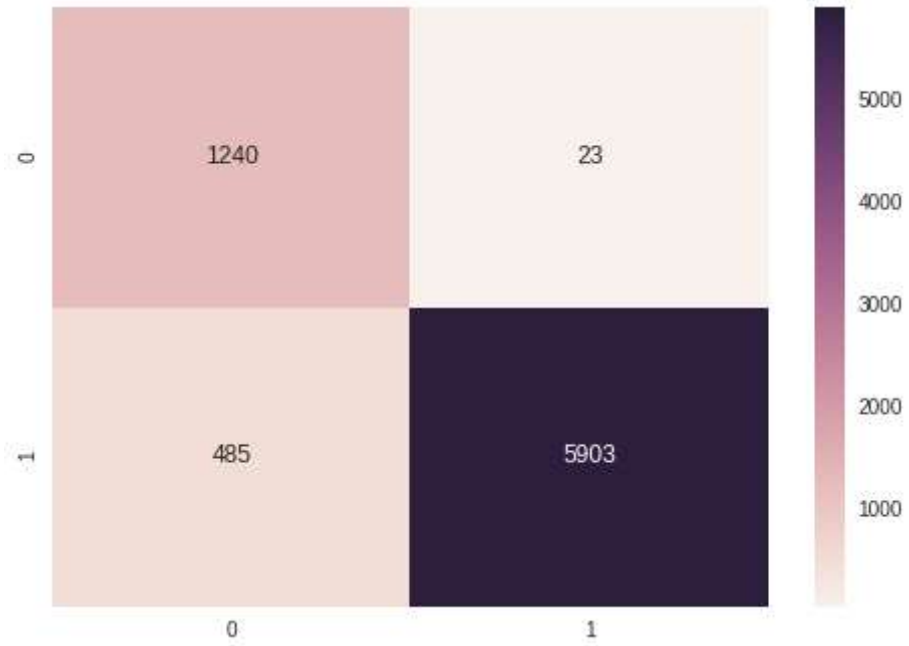
In [71]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1,C=best
_c)
log.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, log.predict_proba(X_trai
n_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDF")
plt.show()

print("="*100)
```




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

In [72]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

#printing train_confusion matrix using seaborn
df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(X_train_tf)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```

[5.2.3] Feature Importance on TFIDF, SET 2

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

In [73]:
```
#Finding feature log probability values and sorting them according to their in
dex
pos=np.argsort((log.coef_)[0])[::-1][0:10]

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

```
['bros' 'cheapand' 'breastfeed' 'airline' 'baggie' 'clementine'
 'consistant' 'cheese' 'birch' 'bergamot']
```

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

In [74]:
```
#Finding feature log probability values and sorting them according to their in
dex
neg=np.argsort((log.coef_)[0])[0:10]

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

```
['closures' 'barfing' 'bargained' 'ecstatic' 'coccidia' 'discontinuing'
 'caffeine' 'effects' 'affected' 'cold']
```

[5.3] Logistic Regression on AVG W2V, SET 3

[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

In [116]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.linear_model import LogisticRegression




i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

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])
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple
of 4 for greater performance

number of words that occured minimum 5 times  5029
sample words  ['overpriced', 'package', 'three', 'boxes', 'take', 'account',
'dollar', 'two', 'per', 'box', 'pay', 'shipping', 'sorry', 'would', 'skip',
'local', 'grocery', 'store', 'carries', 'anywhere', 'although', 'healthier',
'cookies', 'still', 'watch', 'fat', 'sugar', 'salt', 'contents', 'whole', 'fo
ods', 'much', 'better', 'moment', 'fit', 'bill', 'price', 'site', 'ridiculou
s', 'ask', 'grocer', 'start', 'carrying', 'good', 'deal', 'vanilla', 'beans',
'mostly', 'use', 'making']
```

In [117]:
```
from tqdm import tqdm
import numpy as np

# converting train data text

# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
his list
for sent in tqdm(list_of_sentance_train): # 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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|██████████| 7651/7651 [00:07<00:00, 1012.46it/s]

(7651, 50)
[ 0.38715666  0.41082704 -0.01748877 -0.14746943 -0.50185985 -0.11107954
  0.25340161  0.1063067  -0.09876043  0.05303632  0.04423909 -0.17199383
  1.11469213 -0.21782215 -0.21129312 -0.09374576 -0.17323078 -0.04417362
 -0.6990044  -0.0365729  -0.43748418  0.63027236 -0.36704096  0.13760217
 -0.09774711 -0.30998467  0.31693645 -0.0637576  -0.10119082  0.14548774
 -0.46945522  0.15441322  0.64083516  0.14111494  0.09599695 -0.04739214
  0.43488115 -0.61306582 -0.13268769 -0.16234623  0.06792334  0.34111169
 -0.09136568  0.74398549 -0.34882308  0.79826055  0.5423763  -0.27435734
  0.72107451 -0.32730327]
```

In [118]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in th
is list
for sent in tqdm(list_of_sentance_test): # 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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|██████████| 1913/1913 [00:02<00:00, 953.06it/s]

(1913, 50)
[ 0.23248787  0.36742907  0.03822075 -0.07692887 -0.41593676 -0.09317758
  0.23729453  0.10803283 -0.08463274  0.13291253  0.11175882 -0.17986271
  0.96789925 -0.34716445 -0.13115    -0.05553325 -0.14995364  0.04116283
 -0.5633841  -0.00921919 -0.38197194  0.52351305 -0.42148027  0.05539457
 -0.15902617 -0.37171446  0.31205882 -0.07324575 -0.1859896   0.1650085
 -0.42605012  0.1770681   0.48388122  0.05496427 -0.0133636  -0.05976571
  0.37075155 -0.47071387 -0.14523056  0.00251878  0.06417985  0.34276141
 -0.17477717  0.53877416 -0.32886825  0.65077296  0.44966953 -0.14265452
  0.63649135 -0.1372253 ]
```

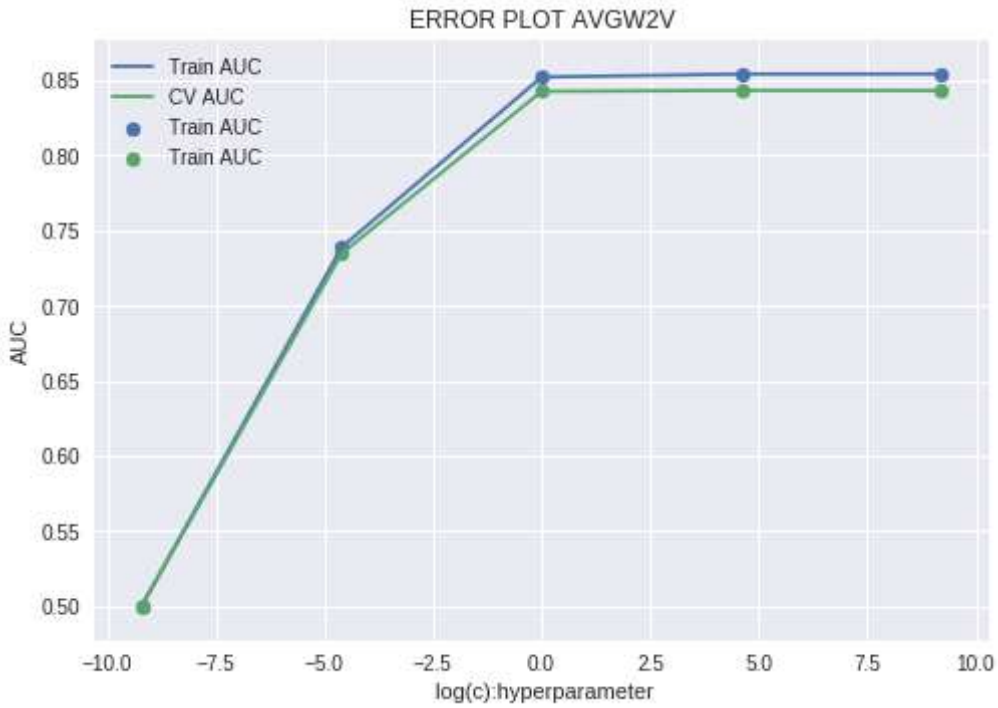
In [0]:
```
#Standardaizing the data on both Train and test datasets

from sklearn import preprocessing
scaler = preprocessing.StandardScaler(with_mean=False).fit(sent_vectors_train)
sent_vectors_train = scaler.transform(sent_vectors_train)
sent_vectors_test = scaler.transform(sent_vectors_test)
```

In [120]:
```
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(sent_vectors_train, y_train)

print(model.best_estimator_)
print(model.score(sent_vectors_test, y_test))
```

```
LogisticRegression(C=100, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.8573086611121876
```

Hyperparameter Tuning AVGword2vec L1

In [122]:
```
#Finding the best hyperparameter
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(sent_vectors_train, y_train)

train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

#plotting to find the AUC of hyperparameters
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AVGW2V")
plt.show()
```



In [0]:
```
best_c=100
```

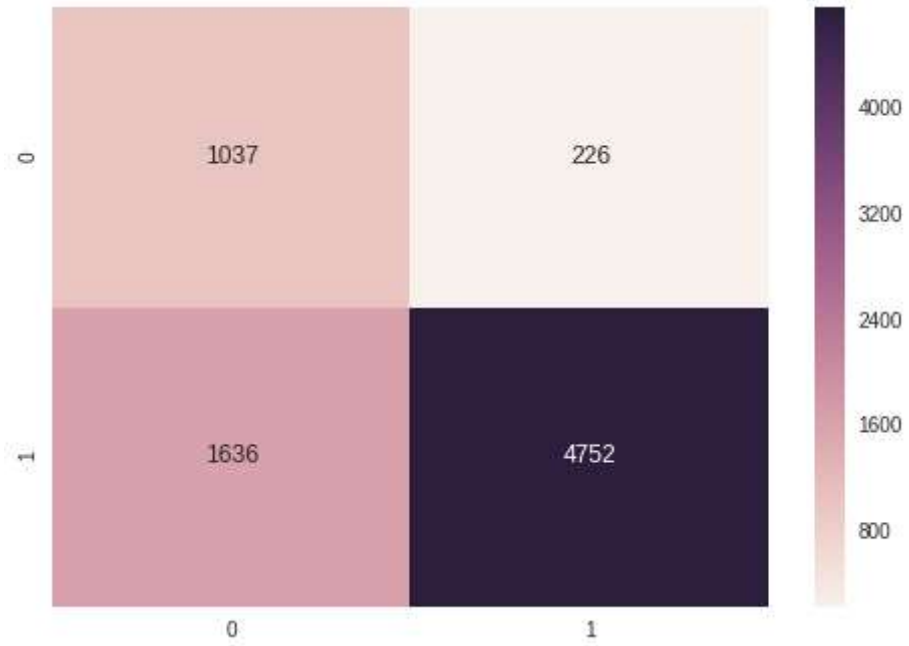
Testing with Test Data

In [124]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1,C=best
_c)
log.fit(sent_vectors_train, 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, log.predict_proba(sent_v
ectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.predict_proba(sent_vect
ors_test)[:,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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST AVGW2V")
plt.show()

print("="*100)
```

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

In [125]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(sent_vectors_train
)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(sent_vectors_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```

[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [126]:
```
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

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])
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple
of 4 for greater performance

number of words that occured minimum 5 times  5029
sample words  ['overpriced', 'package', 'three', 'boxes', 'take', 'account',
'dollar', 'two', 'per', 'box', 'pay', 'shipping', 'sorry', 'would', 'skip',
'local', 'grocery', 'store', 'carries', 'anywhere', 'although', 'healthier',
'cookies', 'still', 'watch', 'fat', 'sugar', 'salt', 'contents', 'whole', 'fo
ods', 'much', 'better', 'moment', 'fit', 'bill', 'price', 'site', 'ridiculou
s', 'ask', 'grocer', 'start', 'carrying', 'good', 'deal', 'vanilla', 'beans',
'mostly', 'use', 'making']
```

In [127]:
```
from tqdm import tqdm
import numpy as np

# converting train data text

# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
his list
for sent in tqdm(list_of_sentance_train): # 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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|██████████| 7651/7651 [00:07<00:00, 1019.48it/s]

(7651, 50)
[ 0.46085809  0.49813172 -0.00250837 -0.23249507 -0.50594874 -0.13688212
  0.35249491  0.23991603 -0.02955176  0.12012863  0.12218556 -0.16164128
  1.33711714 -0.28347277 -0.11566599 -0.15314722 -0.15946637 -0.01317924
 -0.65176376  0.02866535 -0.40349412  0.54692956 -0.37193421  0.28916931
 -0.09672172 -0.32612483  0.31751227 -0.09251533 -0.07196056  0.21231617
 -0.38418349  0.25053256  0.63318579  0.10024225  0.13231028 -0.03024692
  0.34672315 -0.51466152 -0.00448228 -0.20640331 -0.06460736  0.24004921
 -0.09618174  0.67953179 -0.2239729   0.69036891  0.59270237 -0.2670583
  0.61570821 -0.33063516]
```

In [128]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in th
is list
for sent in tqdm(list_of_sentance_test): # 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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|██████████| 1913/1913 [00:01<00:00, 971.70it/s]

(1913, 50)
[ 0.28142291  0.41138057  0.00377285 -0.15276306 -0.38578007 -0.13442358
  0.26035768  0.20804834  0.01646869  0.18772904  0.149494   -0.16596055
  1.11646519 -0.36698827 -0.06080655 -0.12051629 -0.12078402  0.06437
 -0.4833762   0.04308838 -0.35594596  0.40729215 -0.40833717  0.16332746
 -0.18167836 -0.38477939  0.25471377 -0.08366962 -0.16247964  0.21881858
 -0.33349653  0.21433743  0.5229512   0.05034585  0.00496523 -0.04690657
  0.34920155 -0.44686366 -0.07596261 -0.09172501 -0.04128297  0.29018598
 -0.21327039  0.49723904 -0.24227183  0.60033188  0.46966103 -0.10169595
  0.60267113 -0.11758999]
```

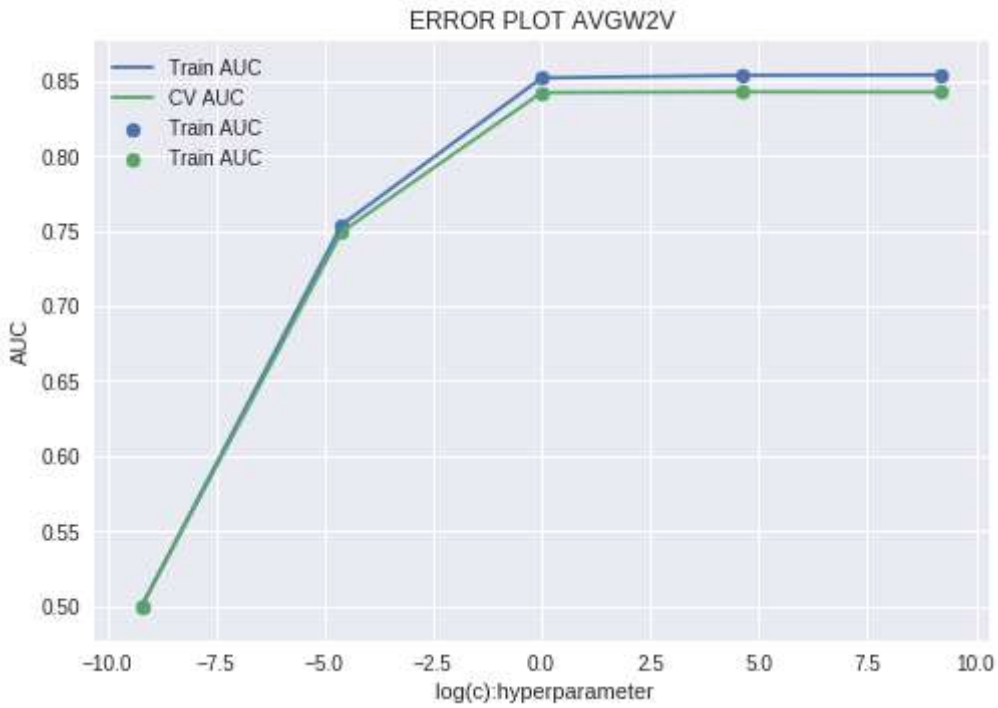
In [0]:
```
#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(sent_vectors_train)
sent_vectors_train = scaler.transform(sent_vectors_train)
sent_vectors_test = scaler.transform(sent_vectors_test)
```

In [130]:
```
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(sent_vectors_train, y_train)

print(model.best_estimator_)
print(model.score(sent_vectors_test, y_test))
```

```
LogisticRegression(C=10000, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.8524336368920751
```

Hyperparameter Tuning AVGword2vec L2

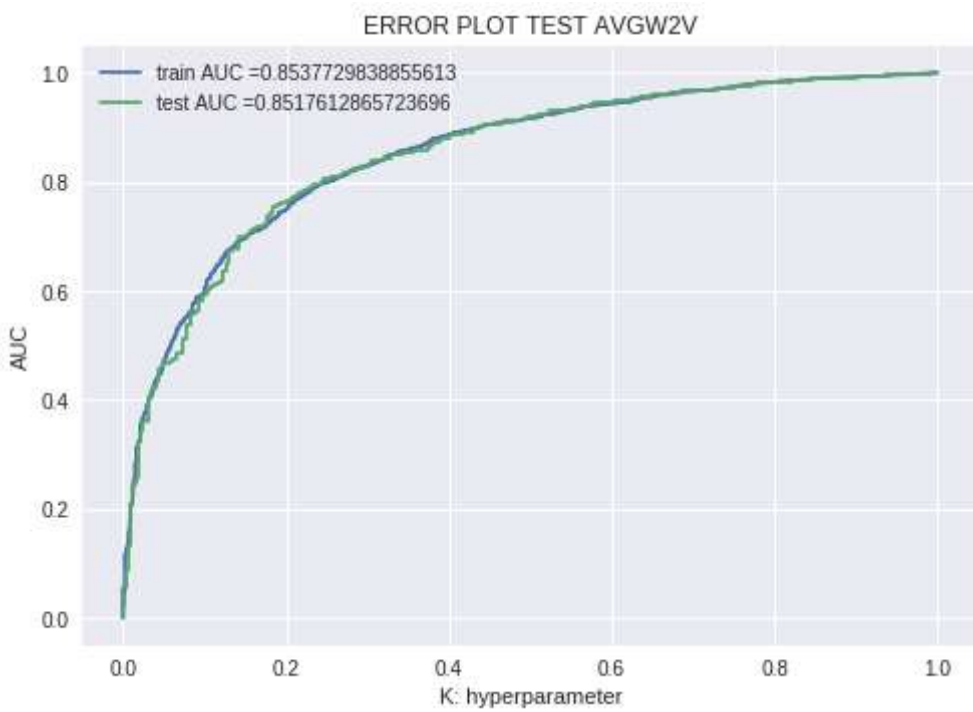
In [131]:
```
#FInding best hyper parameter using gridsearch
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(sent_vectors_train, y_train)

train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AVGW2V")
plt.show()
```



In [0]:
```
best_c=10000
```

Testing with Test Data

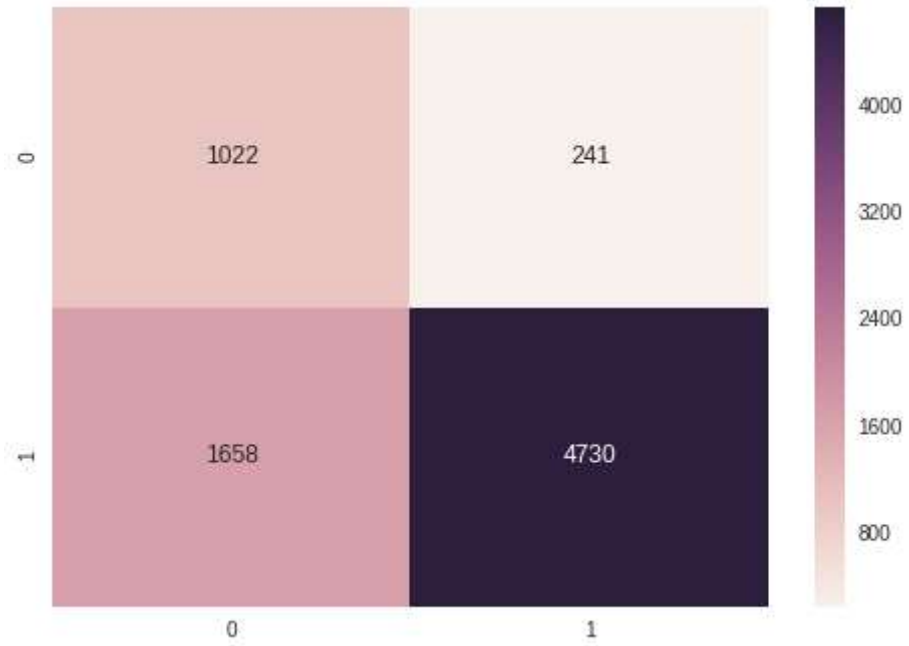
In [133]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1,C=best
_c)
log.fit(sent_vectors_train, 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, log.predict_proba(sent_v
ectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.predict_proba(sent_vect
ors_test)[:,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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST AVGW2V")
plt.show()

print("="*100)
```



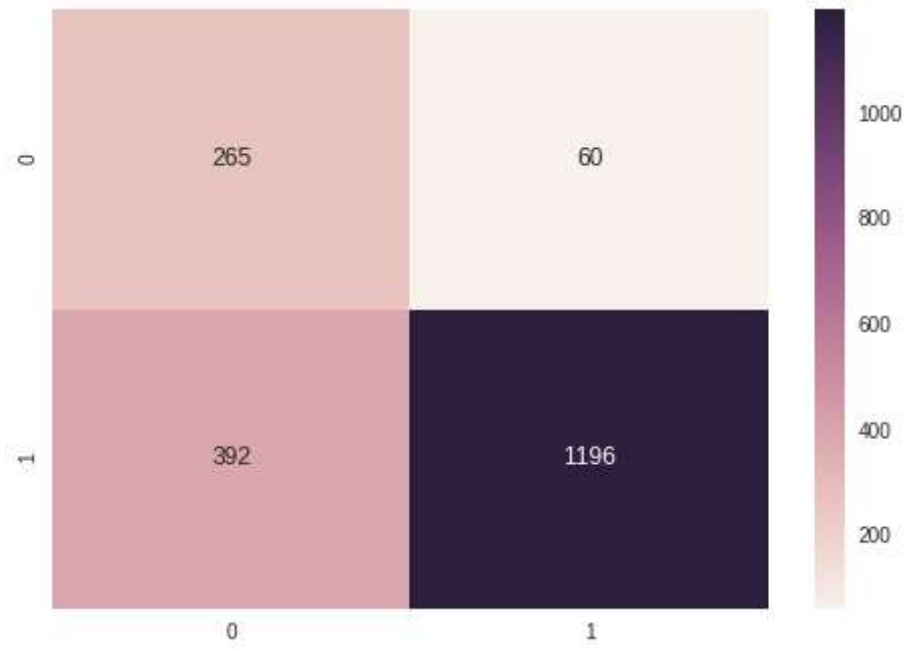

```
==================================================================================
=======================
```

In [134]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

#printing the train confusion matrix
df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(sent_vectors_train
)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(sent_vectors_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



[5.4] Logistic Regression on TFIDF W2V, SET 4

[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

In [93]:
```
#preparing reviews for genism model
i=0
list_of_sentance_tf_train=[]
for sentance in X_train:
    list_of_sentance_tf_train.append(sentance.split())


#Training w2v models

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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tf_train,min_count=5,size=50, workers=4)

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])
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple
of 4 for greater performance
```

```
number of words that occured minimum 5 times  5029
sample words  ['overpriced', 'package', 'three', 'boxes', 'take', 'account',
'dollar', 'two', 'per', 'box', 'pay', 'shipping', 'sorry', 'would', 'skip',
'local', 'grocery', 'store', 'carries', 'anywhere', 'although', 'healthier',
'cookies', 'still', 'watch', 'fat', 'sugar', 'salt', 'contents', 'whole', 'fo
ods', 'much', 'better', 'moment', 'fit', 'bill', 'price', 'site', 'ridiculou
s', 'ask', 'grocer', 'start', 'carrying', 'good', 'deal', 'vanilla', 'beans',
'mostly', 'use', 'making']
```

Converting Train data text into TFIDFW2V vectors

In [94]:
```
from tqdm import tqdm
import numpy as np

model = TfidfVectorizer()
model.fit(X_train)
tf_idf_matrix = model.transform(X_train)
# 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_)))


# 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_train= []; # the tfidf-w2v for each sentence/review is stored in th
is list
row=0;
for sent in tqdm(list_of_sentance_tf_train): # 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_train.append(sent_vec)
    row += 1
```

```
100%|██████████| 7651/7651 [01:01<00:00, 123.68it/s]
```

Converting Test data text into TFIDFW2V vectors

In [95]:
```
i=0
list_of_sentance_tf_test=[]
for sentance in X_test:
    list_of_sentance_tf_test.append(sentance.split())

# 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_test= []; # the tfidf-w2v for each sentence/review is stored in thi
s list
row=0;
for sent in tqdm(list_of_sentance_tf_test): # 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_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 1913/1913 [00:15<00:00, 122.16it/s]
```

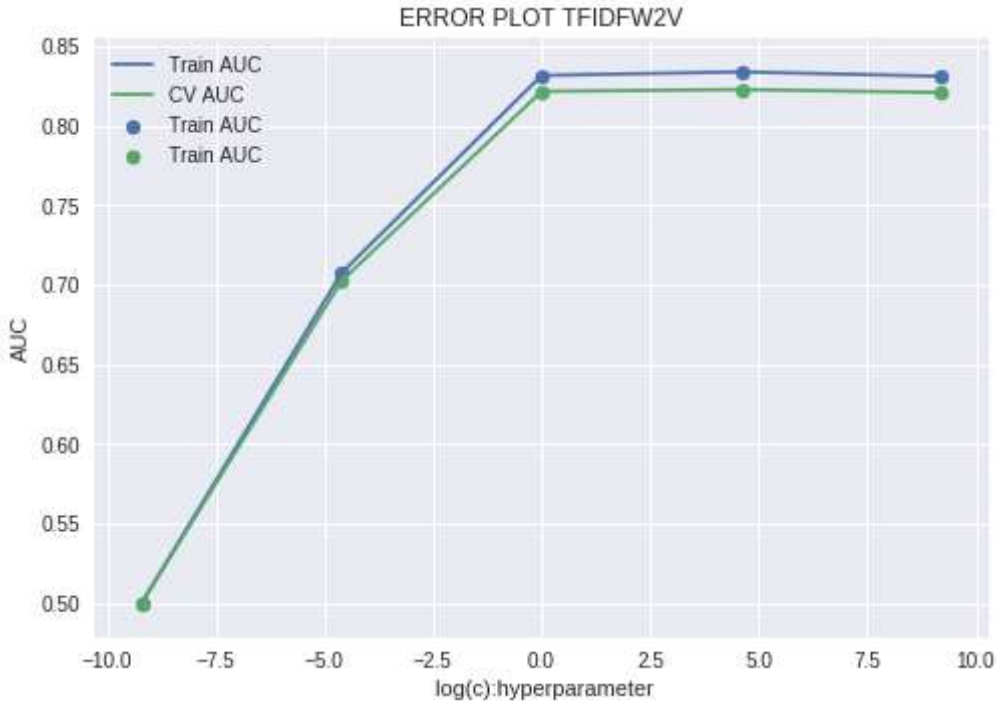
In [0]:
```
#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(tfidf_sent_train)
tfidf_sent_train = scaler.transform(tfidf_sent_train)
tfidf_sent_test = scaler.transform(tfidf_sent_test)
```

In [97]:
```
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(tfidf_sent_train, y_train)

print(model.best_estimator_)
print(model.score(tfidf_sent_test, y_test))
```

```
LogisticRegression(C=10000, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.8420015500871925
```

Hyper parameter tuning TFIDFWord2VEC L1

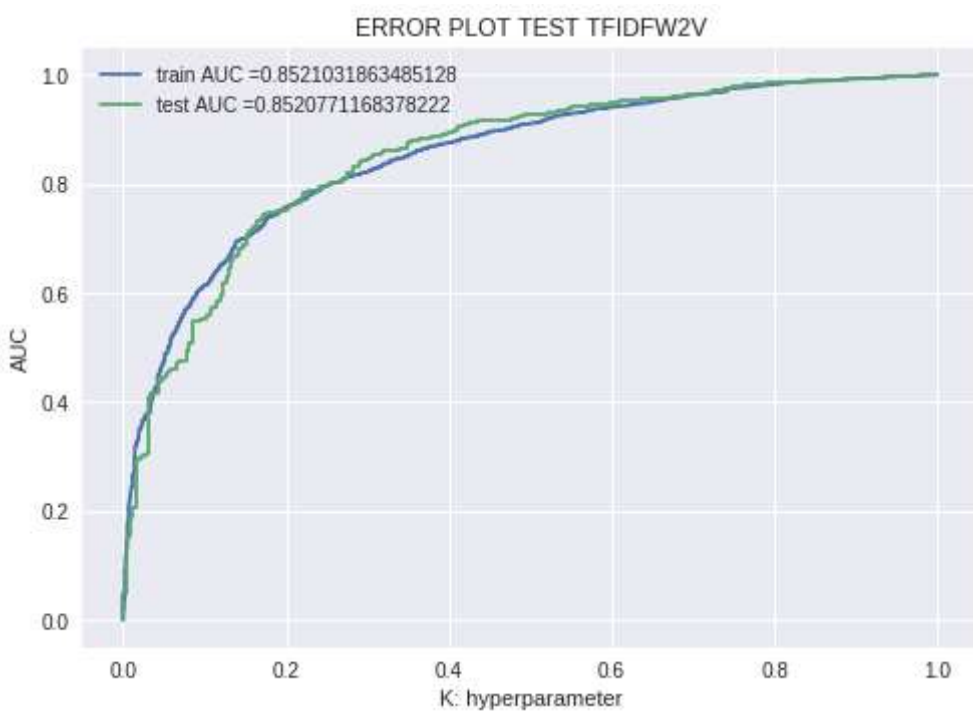
In [98]:
```
#finding the best hyper parameter using gridsearch cv
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(tfidf_sent_train, y_train)

train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

# Plotting AUC for Hyperparameters
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V")
plt.show()
```



In [0]:
```
best_c=10000
```

Testing with Test Data

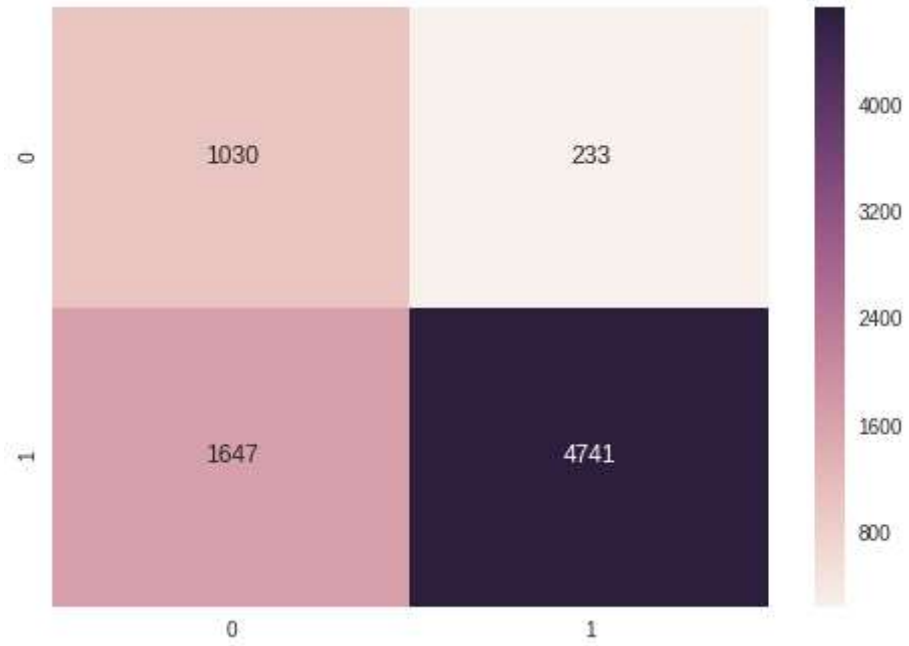
In [100]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1,C=best
_c)
log.fit(sent_vectors_train, 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, log.predict_proba(sent_v
ectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.predict_proba(sent_vect
ors_test)[:,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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDFW2V")
plt.show()

print("="*100)
```



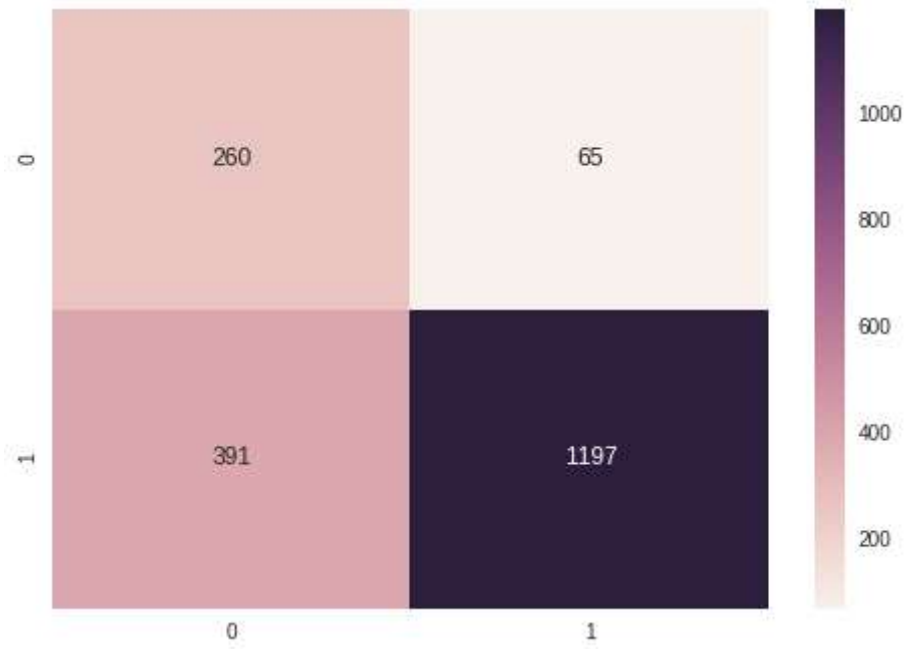

```
====================================================================================
=======================
```

In [102]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(sent_vectors_train
)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(sent_vectors_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [103]:
```
#preparing reviews for genism model
i=0
list_of_sentance_tf_train=[]
for sentance in X_train:
    list_of_sentance_tf_train.append(sentance.split())


#Training w2v models

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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tf_train,min_count=5,size=50, workers=4)

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])
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple
of 4 for greater performance
```

```
number of words that occured minimum 5 times  5029
sample words  ['overpriced', 'package', 'three', 'boxes', 'take', 'account',
'dollar', 'two', 'per', 'box', 'pay', 'shipping', 'sorry', 'would', 'skip',
'local', 'grocery', 'store', 'carries', 'anywhere', 'although', 'healthier',
'cookies', 'still', 'watch', 'fat', 'sugar', 'salt', 'contents', 'whole', 'fo
ods', 'much', 'better', 'moment', 'fit', 'bill', 'price', 'site', 'ridiculou
s', 'ask', 'grocer', 'start', 'carrying', 'good', 'deal', 'vanilla', 'beans',
'mostly', 'use', 'making']
```

Converting Train data text into TFIDFW2V vectors

In [104]:
```
from tqdm import tqdm
import numpy as np

model = TfidfVectorizer()
model.fit(X_train)
tf_idf_matrix = model.transform(X_train)
print(type(X_train))
# 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_)))


# 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_train= []; # the tfidf-w2v for each sentence/review is stored in th
is list
row=0;
for sent in tqdm(list_of_sentance_tf_train): # 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_train.append(sent_vec)
    row += 1
```

```
  0%|          | 15/7651 [00:00<00:55, 136.58it/s]

<class 'list'>

100%|██████████| 7651/7651 [01:09<00:00, 109.61it/s]
```

Converting Test data text into TFIDFW2V vectors

In [105]:
```
i=0
list_of_sentance_tf_test=[]
for sentance in X_test:
    list_of_sentance_tf_test.append(sentance.split())

# 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_test= []; # the tfidf-w2v for each sentence/review is stored in thi
s list
row=0;
for sent in tqdm(list_of_sentance_tf_test): # 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_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 1913/1913 [00:18<00:00, 105.65it/s]
```

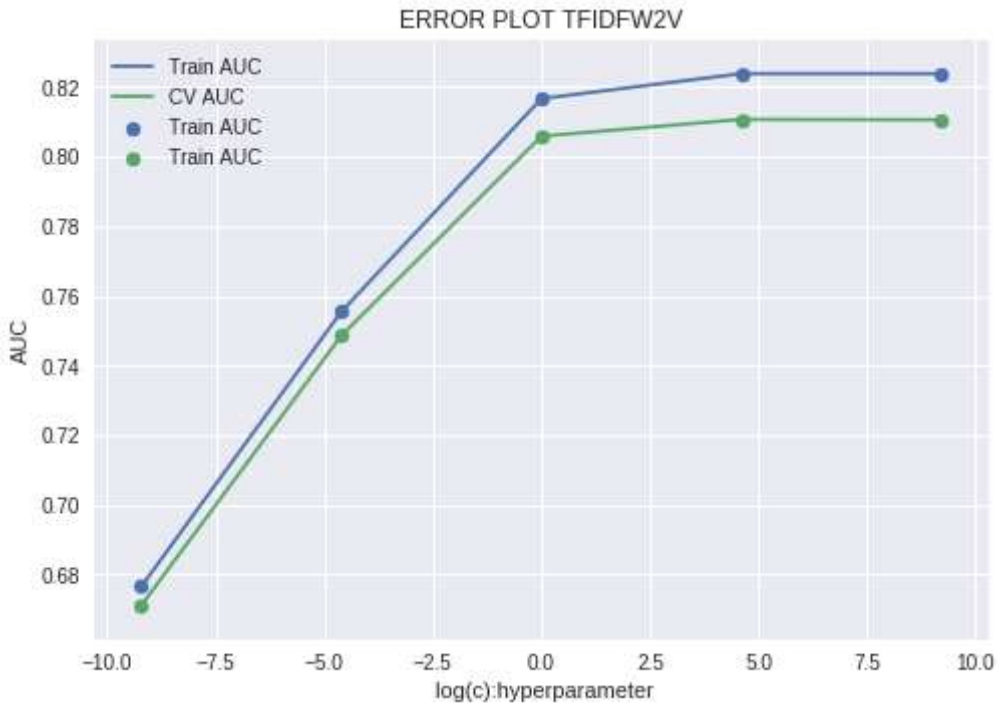
In [0]:
```
#Standardaizing the data on both Train and test datasets
scaler = preprocessing.StandardScaler(with_mean=False).fit(tfidf_sent_train)
tfidf_sent_train = scaler.transform(tfidf_sent_train)
tfidf_sent_test = scaler.transform(tfidf_sent_test)
```

In [107]:
```
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]


log=LogisticRegression(penalty="l2",class_weight='balanced',n_jobs=-1)
#Using GridSearchCV
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5)
model.fit(tfidf_sent_train, y_train)

print(model.best_estimator_)
print(model.score(tfidf_sent_test, y_test))
```

```
LogisticRegression(C=100, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.8284809145514436
```

Hyper parameter tuning TFIDFWord2VEC L2

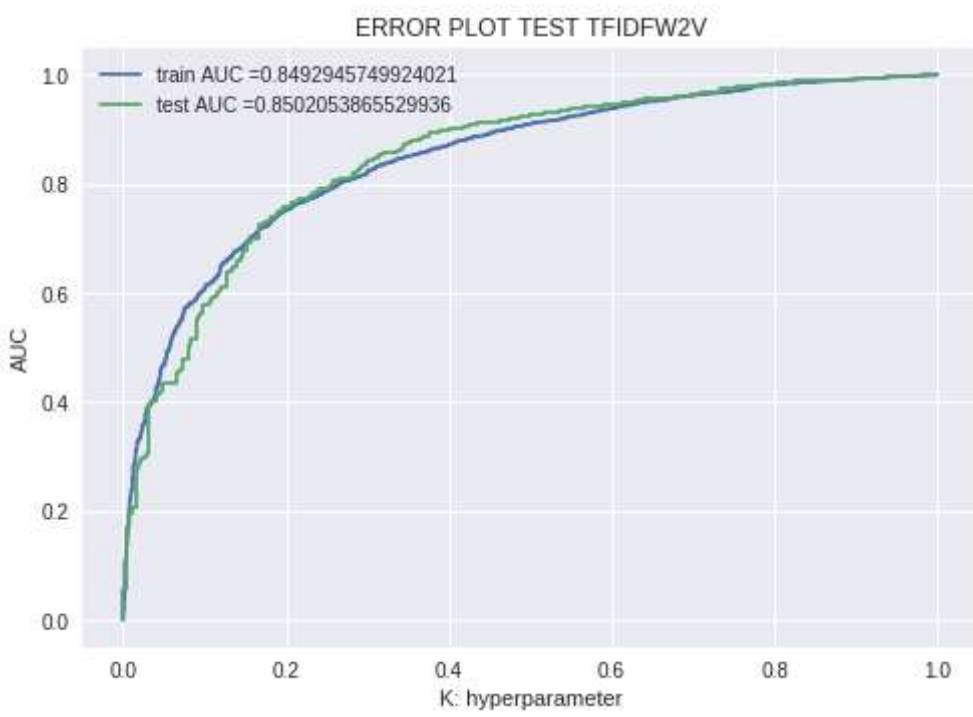
In [108]:
```
#finding best hyperparameter
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(log, tuned_parameters, scoring = 'roc_auc', cv=5,return_t
rain_score='warn')
model.fit(tfidf_sent_train, y_train)

train_auc=model.cv_results_['mean_train_score']
cv_auc = model.cv_results_['mean_test_score']

#plotting AUC of hyper parameters
C=list(np.log([10**-4, 10**-2, 10**0, 10**2, 10**4]))
plt.plot(C, train_auc, label='Train AUC')
plt.scatter(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.scatter(C, cv_auc, label='Train AUC')
plt.legend()
plt.xlabel(" log(c):hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V")
plt.show()
```



In [0]:
```
best_c=100
```

Testing with Test Data

In [110]:
```
from sklearn.metrics import roc_curve, auc


log = LogisticRegression(penalty="l1",class_weight='balanced',n_jobs=-1,C=best
_c)
log.fit(sent_vectors_train, 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, log.predict_proba(sent_v
ectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, log.predict_proba(sent_vect
ors_test)[:,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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDFW2V")
plt.show()

print("="*100)
```




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

In [112]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, log.predict(sent_vectors_train
)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,log.predict(sent_vectors_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```

[6] Conclusions

In [135]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper-Parameter(C)", "AUC"]

x.add_row(["BOW", "L1", 1,0.9039])
x.add_row(["BOW", "L2",0.0001,0.8985])
x.add_row(["TFIDF", "L1", 0.01,0.9267])
x.add_row(["TFIDF", "L2",0.0001,0.9466])
x.add_row(["W2V", "L1", 100,0.8573])
x.add_row(["W2v", "L2",10000,0.8517])
x.add_row(["TFIDFW2V", "L1", 10000, 0.8520])
x.add_row(["TFIDFW2V", "L2",100,0.8502])


print(x)
```

```
+------------+-------+--------------------+--------+
| Vectorizer | Model | Hyper-Parameter(C) |  AUC   |
+------------+-------+--------------------+--------+
|    BOW     |   L1  |         1          | 0.9039 |
|    BOW     |   L2  |       0.0001       | 0.8985 |
|   TFIDF    |   L1  |        0.01        | 0.9267 |
|   TFIDF    |   L2  |       0.0001       | 0.9466 |
|    W2V     |   L1  |        100         | 0.8573 |
|    W2v     |   L2  |       10000        | 0.8517 |
|  TFIDFW2V  |   L1  |       10000        | 0.852  |
|  TFIDFW2V  |   L2  |        100         | 0.8502 |
+------------+-------+--------------------+--------+
```

Conclusions

1.TFIDF Vectorizer with L1 regularizer will yield the best AUC.

2.Top 10 positive and negative features in BOW and TFIDF differ.

3.Sparsity of weight vector decrases as C decreases.