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

[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 [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 [4]:
```
# 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[4]:

	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 [6]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[6]:

	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 [7]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[7]:

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

In [8]:
```
display['COUNT(*)'].sum()
```

Out[8]: 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 [9]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[9]:

	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 [11]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
}, keep='first', inplace=False)
final.shape
```

Out[11]:
```
(87775, 10)
```

In [12]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[12]:
```
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 [13]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[13]:

	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 [15]:
```
#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[15]:
```
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 [16]:
```
# 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 [17]:
```
# 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 [18]:
```
# 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 [20]:
```
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 [21]:
```
#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 [22]:
```
#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 [24]:
```
# 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 [00:41<00:00, 2125.77it/s]
```

In [25]:
```
preprocessed_reviews[1500]
```

Out[25]: '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 [27]:
```
#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 [28]:
```
#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 [29]:
```
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 [31]:
```
# 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 ")
```

```
[('awesome', 0.8363437652587891), ('fantastic', 0.8354489803314209), ('good',
0.8099424242973328), ('excellent', 0.7995731830596924), ('terrific', 0.788858
17527771), ('perfect', 0.7617314457893372), ('wonderful', 0.759920239448547
4), ('amazing', 0.7317340970039368), ('nice', 0.7202351689338684), ('fabulou
s', 0.6965214610099792)]
==================================================
[('greatest', 0.8151950836181641), ('tastiest', 0.7468560338020325), ('best',
0.7303048372268677), ('nastiest', 0.7142689824104309), ('softest', 0.63703829
05006409), ('coolest', 0.6089449524879456), ('disgusting', 0.602662086486816
4), ('horrible', 0.5979315638542175), ('experienced', 0.585682213306427), ('i
ve', 0.5734379887580872)]
```

In [32]:
```
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

In [33]:
```
"""# 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[33]:
```
"# 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]))"
```

[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 [35]:
```
"""# 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[35]:
```
'# 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 7: SVM

1. **Apply SVM 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. **Procedure**

   - You need to work with 2 versions of SVM
     - Linear kernel
     - RBF kernel
   - When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
   - When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use CalibratedClassifierCV (https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html)
   - Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

   - 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

4. **Feature importance**

   - When you are working on the linear kernel with BOW or TFIDF please print the top 10 best features for each of the positive and negative classes.

5. **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.

6. **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)

7. **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 SVM

[5.1] Linear SVM

[5.1.1] Applying Linear SVM 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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25)
```

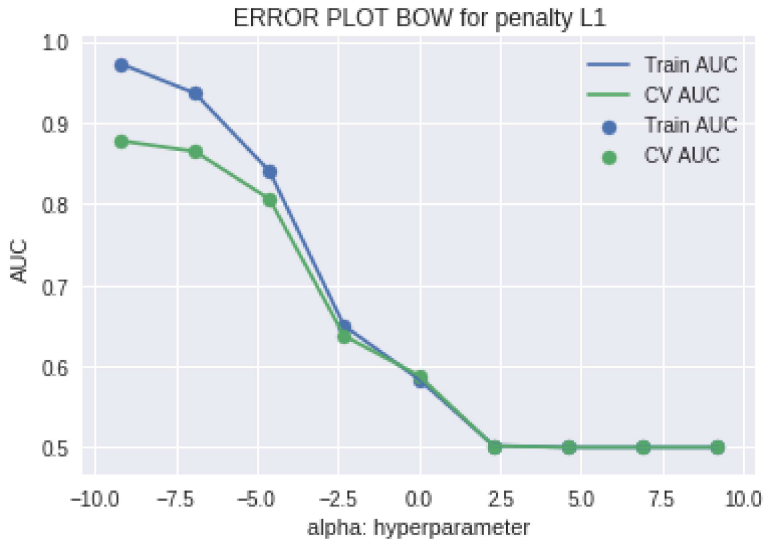
In [37]:
```
from sklearn.model_selection import learning_curve
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer

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_cv_bow = vectorizer.transform(X_cv)
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_cv_bow = scaler.transform(X_cv_bow)
X_test_bow = scaler.transform(X_test_bow)

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)


parameters = {'penalty':['l1','l2'],'alpha': [10**-4,10**-3, 10**-2,10**-1 ,10
**0,10**1, 10**2,10**3, 10**4]}

#Applying SGD classifier  on Grid search
clf=SGDClassifier(loss='hinge',class_weight='balanced',n_jobs=-1)

#Using GridSearchCV
model = GridSearchCV(clf,parameters, scoring = 'roc_auc', cv=5)
model.fit(X_train_bow, y_train)

#Finding the best C
print(model.best_estimator_)
print(model.best_params_)
print(model.score(X_cv_bow, y_cv))
```

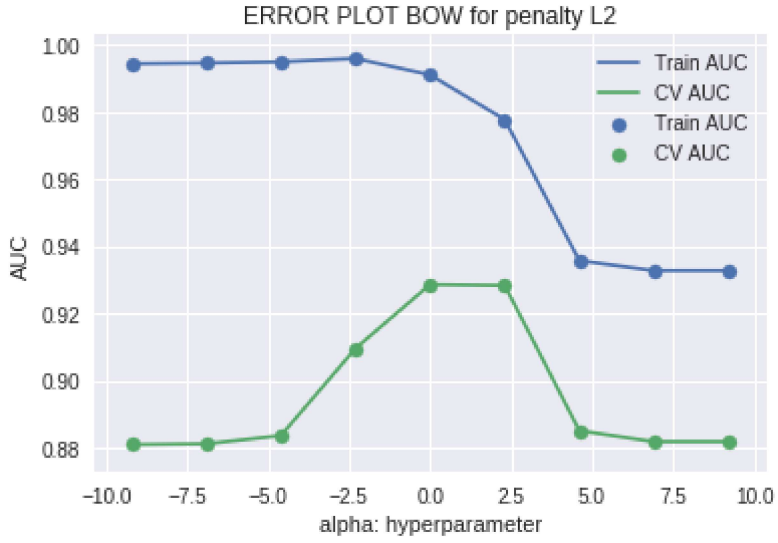
```
After vectorizations
(52663, 42848) (52663,)
(17555, 42848) (17555,)
(17555, 42848) (17555,)
==============================================================================
=======================
SGDClassifier(alpha=10, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=-1, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
{'alpha': 10, 'penalty': 'l2'}
0.927656091491788
```

Hyper parameter Tuning

In [38]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning where penalty =L1
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l1',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.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 =  cc.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  cc.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 vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW for penalty L1")
plt.show()


#penalty = L2


#Hyper parameter tuning where penalty =L2
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l2',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.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 =  cc.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  cc.predict_proba(X_cv_bow)[:,1]
```

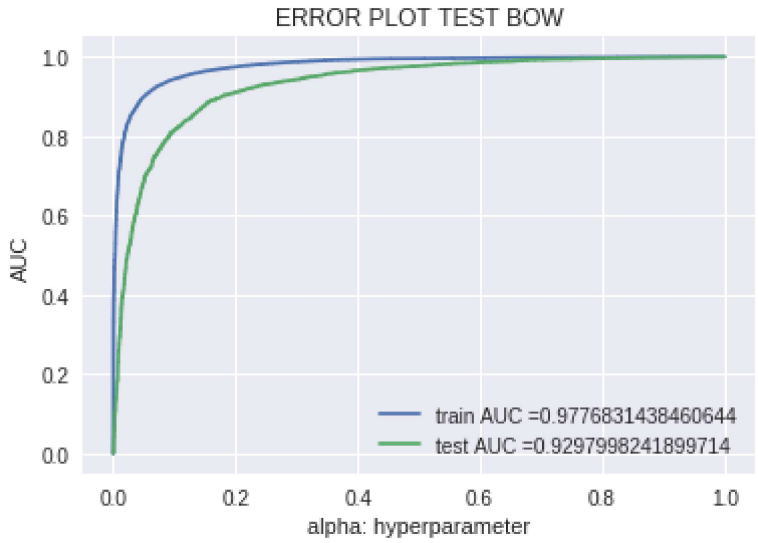
```
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

#plotting the vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW for penalty L2")
plt.show()
```





Testing with test data

In [39]:
```
#Best hyperparameters are penalty ='l2' and alpha =10

from sklearn.metrics import roc_curve, auc


clf=SGDClassifier(loss='hinge',alpha=10,penalty='l2',class_weight='balanced',n
_jobs=-1)
cc = CalibratedClassifierCV(clf,cv=5)
cc.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, cc.predict_proba(X_train
_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.predict_proba(X_test_bow
)[:,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)

#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, cc.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,cc.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
================================================================================
========================
Train confusion matrix
```

```
Test confusion matrix
```

In [40]:
```
clf.fit(X_train_bow,y_train)
#Finding feature log probability values and sorting them according to their in
dex
pos=np.argsort((clf.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' 'best' 'delicious' 'perfect' 'loves' 'favorite' 'highly'
 'easy' 'good']
```

In [41]:
```
#Finding feature log probability values and sorting them according to their in
dex
neg=np.argsort((clf.coef_)[0])[0:10]

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

```
['disappointed' 'not' 'money' 'bad' 'horrible' 'waste' 'awful' 'worst'
 'terrible' 'disappointing']
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

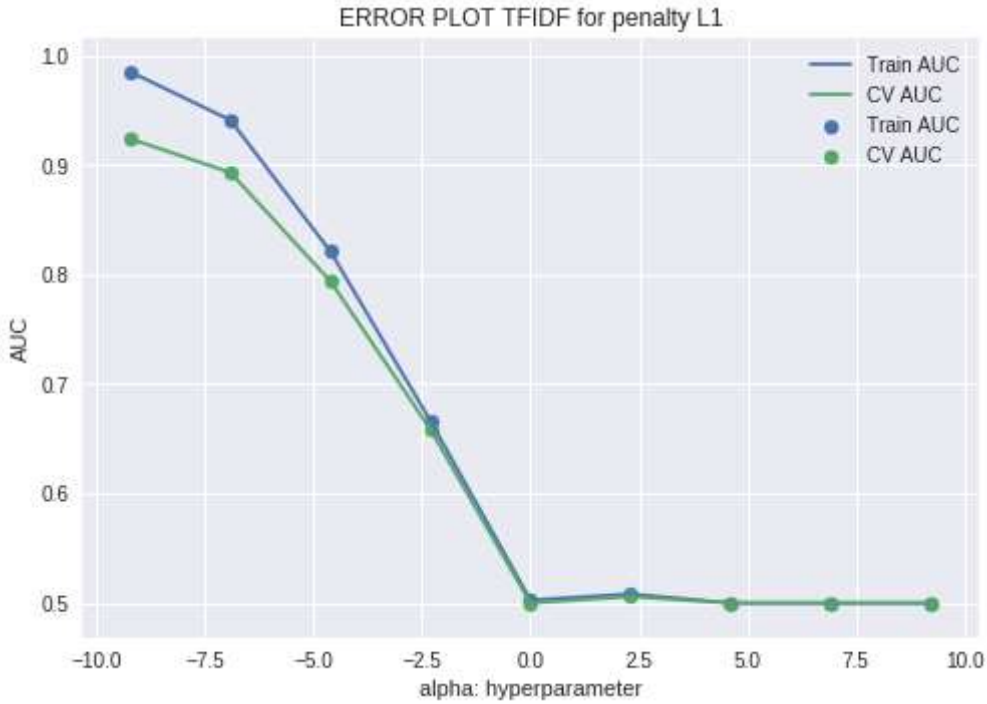
In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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_cv_tf=tf_idf_vect.transform(X_cv)
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_cv_tf = scaler.transform(X_cv_tf)
X_test_tf = scaler.transform(X_test_tf)

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)


parameters = {'penalty':['l1','l2'],'alpha': [10**-4,10**-3, 10**-2,10**-1 ,10
**0,10**1, 10**2,10**3, 10**4]}

#Applying SGD classifier  on Grid search
clf=SGDClassifier(loss='hinge',class_weight='balanced',n_jobs=-1)

#Using GridSearchCV
model = GridSearchCV(clf,parameters, scoring = 'roc_auc', cv=5)
model.fit(X_train_tf, y_train)

#Finding the best C
print(model.best_estimator_)
print(model.best_params_)
print(model.score(X_cv_tf, y_cv))
```

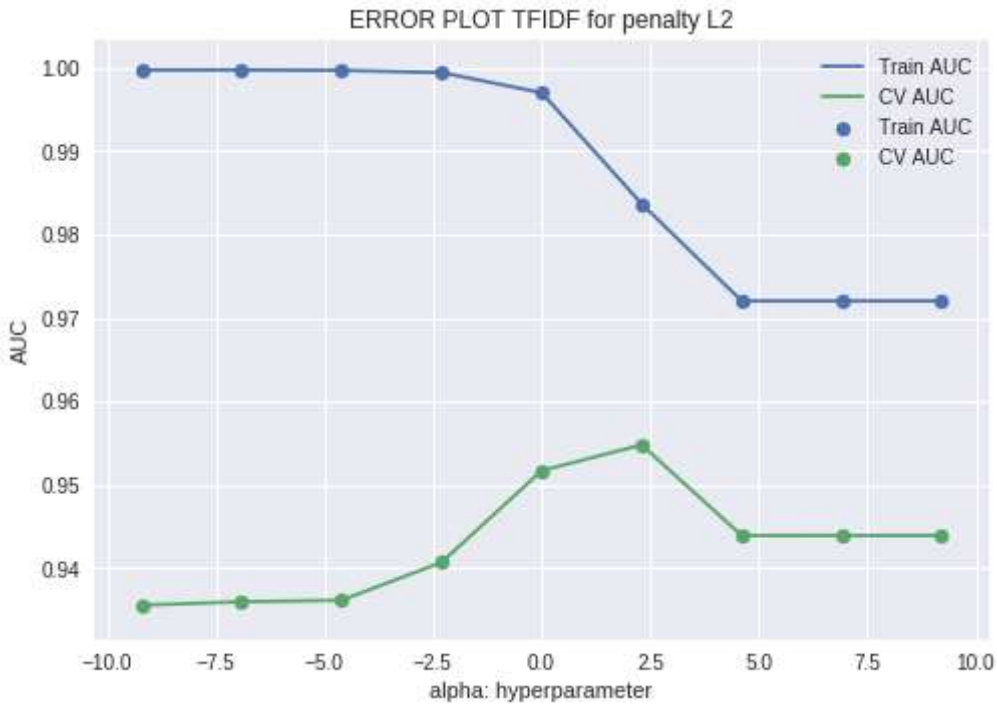
```
After vectorizations
(52663, 31134) (52663,)
(17555, 31134) (17555,)
(17555, 31134) (17555,)
================================================================================
=======================
SGDClassifier(alpha=10, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=-1, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
{'alpha': 10, 'penalty': 'l2'}
0.95469083953397
```

Hyper Parameter Tuning

In [43]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning where penalty =L1
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l1',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.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 =  cc.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  cc.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 vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF for penalty L1")
plt.show()


#penalty = L2


#Hyper parameter tuning where penalty =L2
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l2',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.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 =  cc.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  cc.predict_proba(X_cv_tf)[:,1]
```

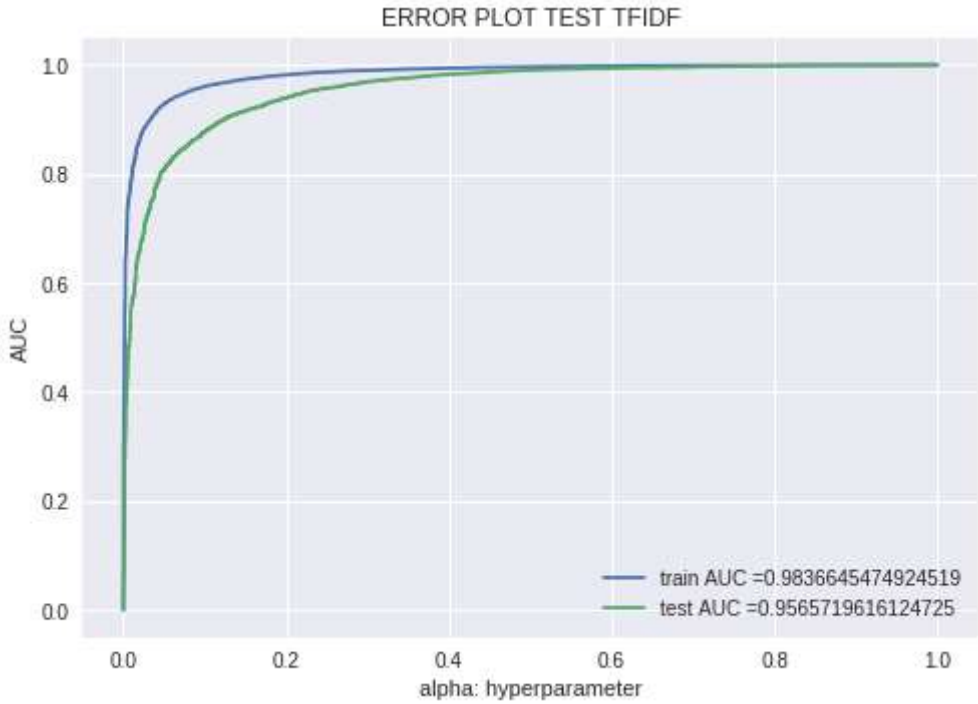
```
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

#plotting the vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF for penalty L2")
plt.show()
```



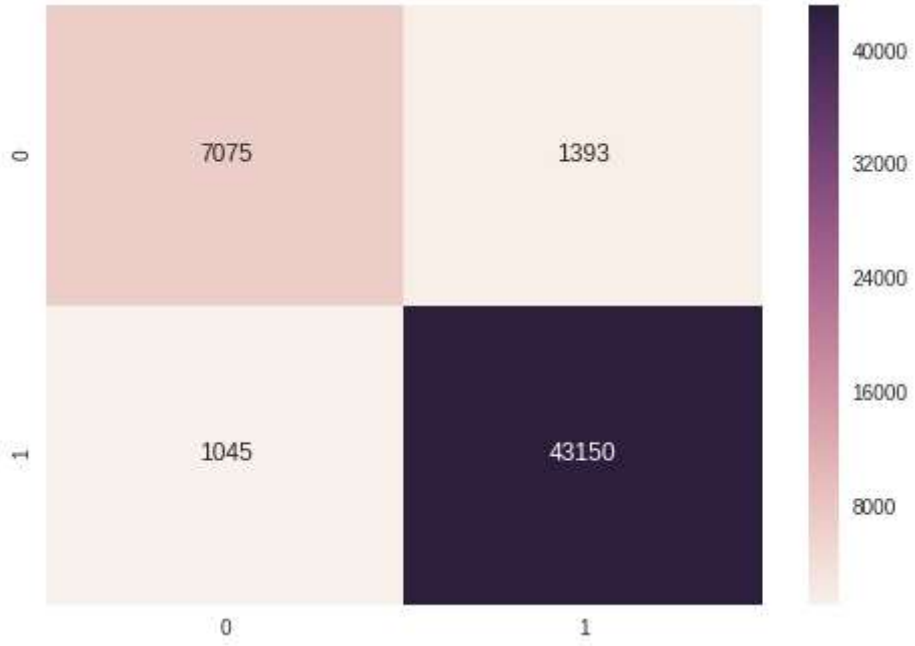


Testing with Test Data

In [44]:
```
#Best hyperparameters are penalty ='l2' and alpha =10

from sklearn.metrics import roc_curve, auc


clf=SGDClassifier(loss='hinge',alpha=10,penalty='l2',class_weight='balanced',n
_jobs=-1)
cc = CalibratedClassifierCV(clf,cv=5)
cc.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, cc.predict_proba(X_train
_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.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)

#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, cc.predict(X_train_tf)))
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,cc.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
==============================================================================
========================
Train confusion matrix
```




```
Test confusion matrix
```

[5.1.3] Applying Linear SVM on AVG W2V, SET 3

In [45]:
```
# 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



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])
```

```
number of words that occured minimum 5 times  13729
sample words  ['eating', 'every', 'day', 'love', 'flavor', 'easy', 'make', 'a
dd', 'water', 'put', 'microwave', 'taste', 'good', 'healthy', 'win', 'tried',
'whim', 'hearing', 'veggie', 'booty', 'long', 'time', 'knew', 'not', 'ready',
'could', 'certainly', 'try', 'new', 'kind', 'cheese', 'puff', 'previously',
'thought', 'without', 'cheetos', 'wasted', 'big', 'change', 'best', 'things',
'ever', 'like', 'real', 'variation', 'intensity', 'single', 'one', 'exactly',
'crunchy']
```

In [46]:
```
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%|██████████| 52663/52663 [01:34<00:00, 558.04it/s]

(52663, 50)
[ 0.96576822 -0.43752158 -0.44794767  0.0637415  -0.65821839  0.40489613
  0.92900494  0.44667491 -0.17406099 -0.05067648  0.05719058  0.54813132
 -0.59372037 -0.15531867 -0.29203636  0.20759405 -0.44609896  0.27774461
  1.29643654  0.53448989  0.87705793  1.53874631  0.17394387  0.9274957
  0.57971082 -0.67787898  0.50671512 -0.61072905  0.5665995   0.11113851
  0.40492572  0.37169417 -0.59116112  0.32855988  0.88448226 -0.33974134
  0.26361434  0.95026051  1.15300811  0.60175055  0.47699589 -0.75999661
  0.18075355 -0.26654792 -0.66982254  0.20958446 -0.46633263  0.4637106
 -0.08387663  0.22141082]
```

In [47]:
```
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this
list
for sent in tqdm(list_of_sentance_cv): # 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_cv.append(sent_vec)
sent_vectors_cv = np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
100%|██████████| 17555/17555 [00:31<00:00, 550.37it/s]

(17555, 50)
[-0.04193683  0.44440936 -0.30254084 -0.09989631 -0.03903614  0.7419258
  0.30649623  0.62899745 -0.74874206 -0.12437969  0.03745604 -0.30008031
 -0.13071016 -0.36479306  0.7282525   0.07946717  0.64728391 -0.27873861
  0.6318314   0.34623912  0.11766641  0.39689962  0.20472318 -0.02931873
 -0.19492626  0.32487794  0.34527142 -0.01542353  0.45603267  0.21763602
 -0.34844246 -0.90554547 -0.36977667 -0.61720845  0.00900773  0.36684113
 -0.23524491  0.61801973  0.73528782 -0.07359309  0.36611014 -0.66022227
  1.21481745 -0.26247982 -0.61904328  0.13602058  0.10598151 -0.03579052
  0.05847896  0.21024156]
```

In [48]:
```
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%|██████████| 17555/17555 [00:31<00:00, 556.06it/s]

(17555, 50)
[ 1.06241283 -0.92974193  0.14605167 -0.07011041  0.27095147 -0.20545299
  0.49731433 -0.4056612   0.22018987  0.065493   -0.09727637  0.41335295
  0.14320006 -0.56901672  0.04513327 -0.05925039 -0.58894875 -0.28496291
  1.10273023  0.86370196  0.98571309  1.19931106 -0.44820515  0.47969336
  0.89946052  0.32043234  0.08232993 -0.1489272   0.64283266  0.97640314
  0.06867161  0.37400832 -0.04679913  0.2665291   0.60072201 -0.76270252
 -0.07708319  0.34710141  0.92296481  0.22750823  0.26430534 -0.29815373
  0.29432175  0.26409567 -0.53879847 -0.05470269 -0.20702002  0.46020535
  0.59488636 -0.13611348]
```

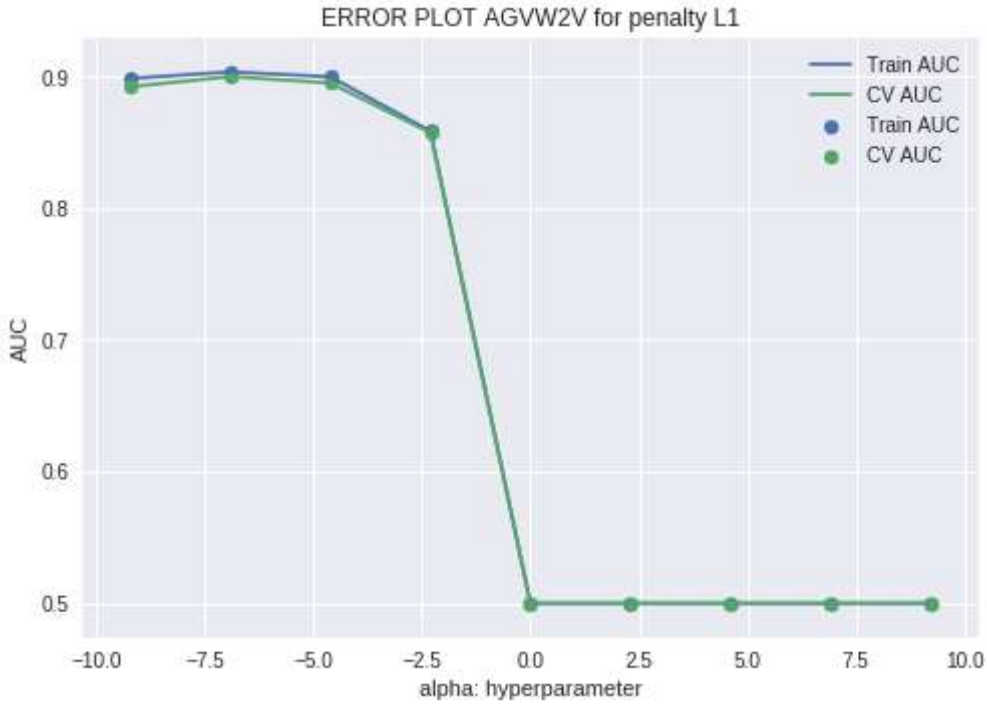
In [49]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


#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_cv= scaler.transform(sent_vectors_cv)
sent_vectors_test = scaler.transform(sent_vectors_test)




print("After vectorizations")
print(sent_vectors_train.shape, y_train.shape)
print(sent_vectors_cv.shape, y_cv.shape)
print(sent_vectors_test.shape, y_test.shape)
print("="*100)


parameters = {'penalty':['l1','l2'],'alpha': [10**-4,10**-3, 10**-2,10**-1 ,10
**0,10**1, 10**2,10**3, 10**4]}

#Applying SGD classifier  on Grid search
clf=SGDClassifier(loss='hinge',class_weight='balanced',n_jobs=-1)

#Using GridSearchCV
model = GridSearchCV(clf,parameters, scoring = 'roc_auc', cv=5)
model.fit(sent_vectors_train, y_train)

#Finding the best C
print(model.best_estimator_)
print(model.best_params_)
print(model.score(sent_vectors_cv, y_cv))
```

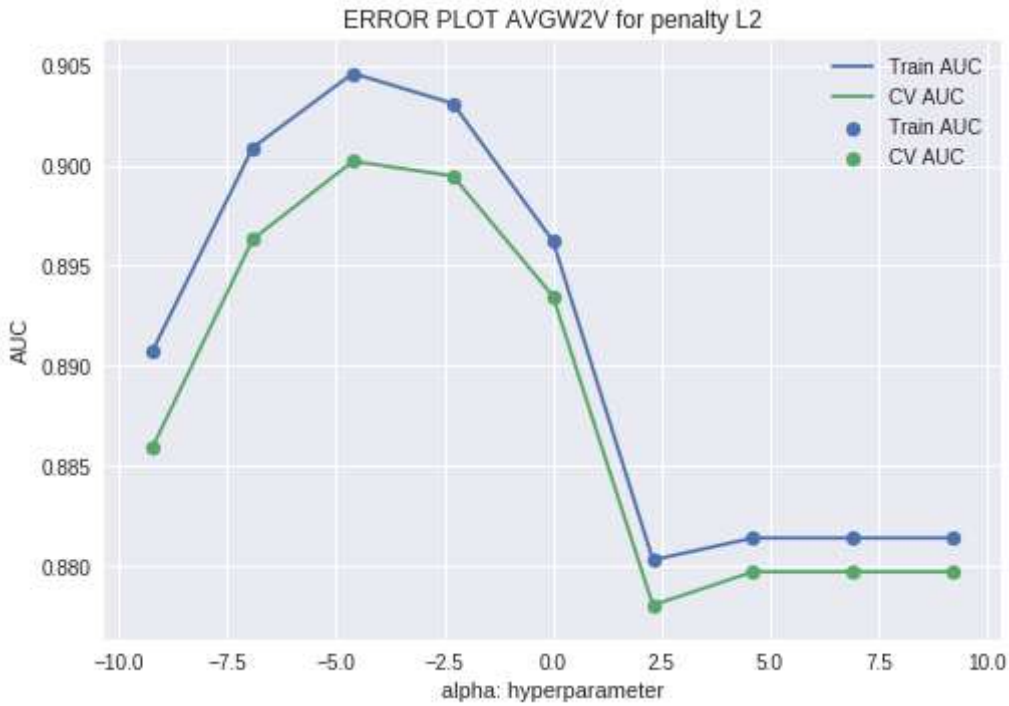
```
After vectorizations
(52663, 50) (52663,)
(17555, 50) (17555,)
(17555, 50) (17555,)
================================================================================
=======================
SGDClassifier(alpha=0.01, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=-1, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
{'alpha': 0.01, 'penalty': 'l2'}
0.900337134488324
```

Hyper parameter Tuning

In [50]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning where penalty =L1
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l1',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.fit(sent_vectors_train,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 =  cc.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  cc.predict_proba(sent_vectors_cv)[:,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 vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AGVW2V for penalty L1")
plt.show()


#penalty = L2


#Hyper parameter tuning where penalty =L2
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l2',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.fit(sent_vectors_train,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 =  cc.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  cc.predict_proba(sent_vectors_cv)[:,1]
```

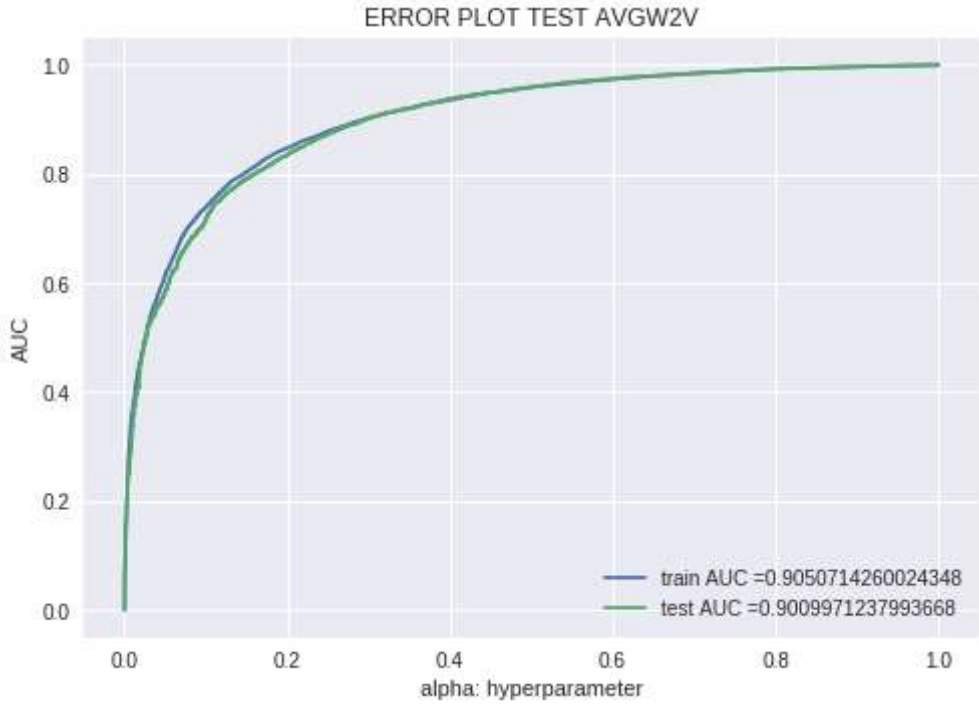
```
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

#plotting the vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AVGW2V for penalty L2")
plt.show()
```





Testing with Test Data

In [51]:
```
#Best hyperparameters are penalty ='l2' and alpha =0.01

from sklearn.metrics import roc_curve, auc


clf=SGDClassifier(loss='hinge',alpha=0.01,penalty='l2',class_weight='balanced'
,n_jobs=-1)
cc = CalibratedClassifierCV(clf,cv=5)
cc.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, cc.predict_proba(sent_ve
ctors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.predict_proba(sent_vecto
rs_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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST AVGW2V")
plt.show()

print("="*100)

#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, cc.predict(sent_vectors_train
)))
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,cc.predict(sent_vectors_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
==============================================================================
========================
Train confusion matrix
```

```
Test confusion matrix
```

[5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

In [52]:
```
#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])
```

```
number of words that occured minimum 5 times  13729
sample words  ['eating', 'every', 'day', 'love', 'flavor', 'easy', 'make', 'a
dd', 'water', 'put', 'microwave', 'taste', 'good', 'healthy', 'win', 'tried',
'whim', 'hearing', 'veggie', 'booty', 'long', 'time', 'knew', 'not', 'ready',
'could', 'certainly', 'try', 'new', 'kind', 'cheese', 'puff', 'previously',
'thought', 'without', 'cheetos', 'wasted', 'big', 'change', 'best', 'things',
'ever', 'like', 'real', 'variation', 'intensity', 'single', 'one', 'exactly',
'crunchy']
```

In [53]:
```
#Train Data

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%|          | 5/52663 [00:00<20:47, 42.21it/s]

<class 'list'>

100%|██████████| 52663/52663 [19:57<00:00, 43.97it/s]
```

In [54]:
```
#CV Data
i=0
list_of_sentance_tf_cv=[]
for sentance in X_cv:
    list_of_sentance_tf_cv.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_cv= []; # the tfidf-w2v for each sentence/review is stored in this
 list
row=0;
for sent in tqdm(list_of_sentance_tf_cv): # 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_cv.append(sent_vec)
    row += 1
```

```
100%|██████████| 17555/17555 [06:42<00:00, 43.60it/s]
```

In [55]:
```
#Test Data
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%|██████████| 17555/17555 [06:47<00:00, 43.08it/s]
```

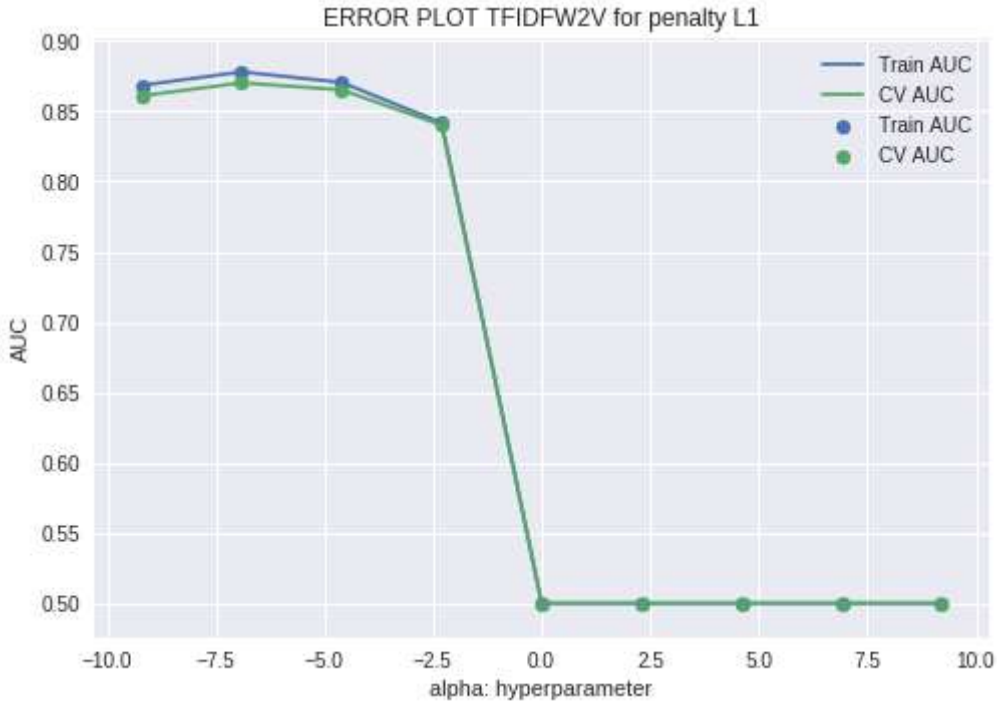
In [56]:
```
from sklearn.model_selection import learning_curve
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer




#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_cv = scaler.transform(tfidf_sent_cv)
tfidf_sent_test = scaler.transform(tfidf_sent_test)


parameters = {'penalty':['l1','l2'],'alpha': [10**-4,10**-3, 10**-2,10**-1 ,10
**0,10**1, 10**2,10**3, 10**4]}

#Applying SGD classifier  on Grid search
clf=SGDClassifier(loss='hinge',class_weight='balanced',n_jobs=-1)

#Using GridSearchCV
model = GridSearchCV(clf,parameters, scoring = 'roc_auc', cv=5)
model.fit(tfidf_sent_train, y_train)

#Finding the best C
print(model.best_estimator_)
print(model.best_params_)
print(model.score(tfidf_sent_cv, y_cv))
```

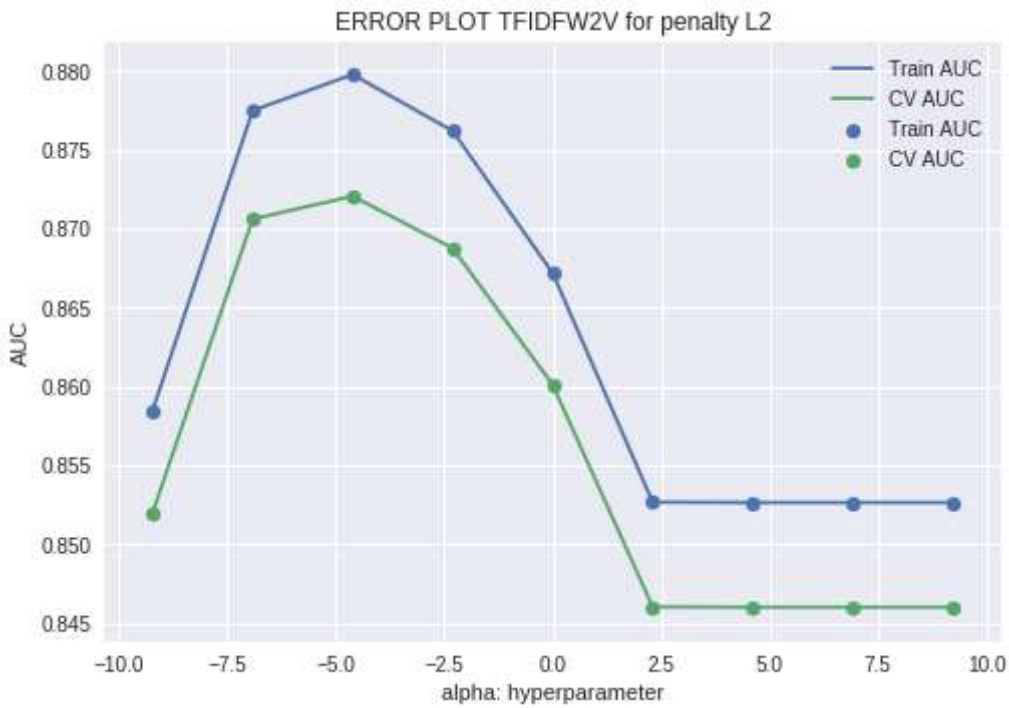
```
SGDClassifier(alpha=0.01, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=-1, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
{'alpha': 0.01, 'penalty': 'l2'}
0.8699195258782842
```

Hyper parameter Tuning

In [57]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning where penalty =L1
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l1',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.fit(tfidf_sent_train,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 =  cc.predict_proba(tfidf_sent_train)[:,1]
    y_cv_pred =  cc.predict_proba(tfidf_sent_cv)[:,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 vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V for penalty L1")
plt.show()


#penalty = L2


#Hyper parameter tuning where penalty =L2
train_auc = []
cv_auc = []
C=[10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SGDClassifier(loss='hinge',alpha=i,penalty='l2',class_weight='balance
d',n_jobs=-1)
    cc = CalibratedClassifierCV(clf,cv=5)
    cc.fit(tfidf_sent_train,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 =  cc.predict_proba(tfidf_sent_train)[:,1]
    y_cv_pred =  cc.predict_proba(tfidf_sent_cv)[:,1]
```

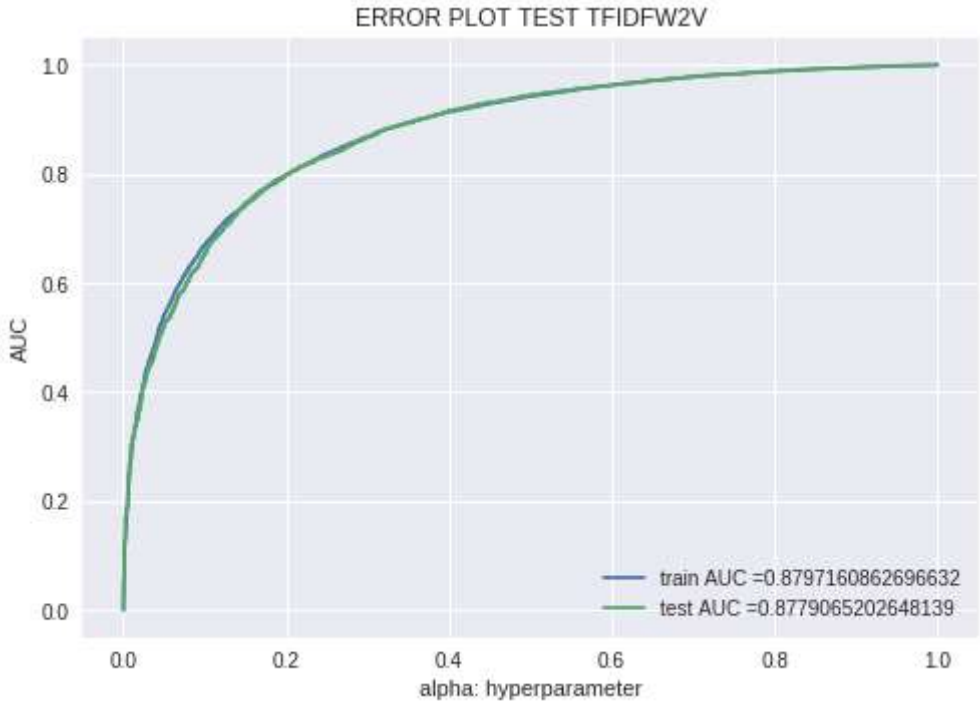
```
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

#plotting the vaalues to see the best hyperparameter
C=list(np.log([10**-4,10**-3, 10**-2,10**-1 ,10**0,10**1, 10**2,10**3, 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='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V for penalty L2")
plt.show()
```



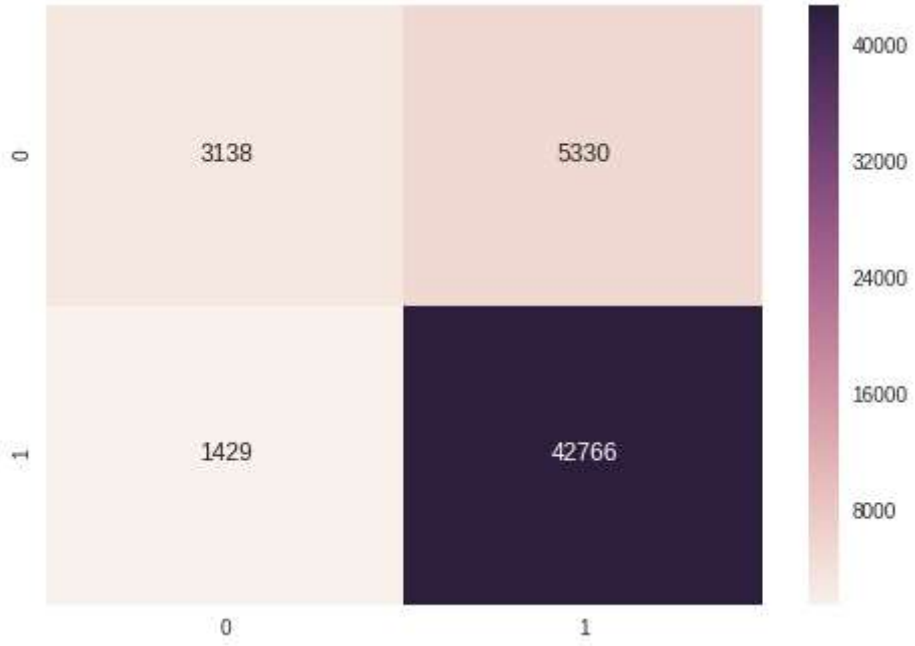


Testing with Test Data

In [58]:
```
#Best hyperparameters are penalty ='l2' and alpha =0.01

from sklearn.metrics import roc_curve, auc


clf=SGDClassifier(loss='hinge',alpha=0.01,penalty='l2',class_weight='balanced'
,n_jobs=-1)
cc = CalibratedClassifierCV(clf,cv=5)
cc.fit(tfidf_sent_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, cc.predict_proba(tfidf_s
ent_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.predict_proba(tfidf_sent
_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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDFW2V")
plt.show()

print("="*100)

#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, cc.predict(tfidf_sent_train)))
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,cc.predict(tfidf_sent_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
================================================================================
========================
Train confusion matrix
```




```
Test confusion matrix
```

[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

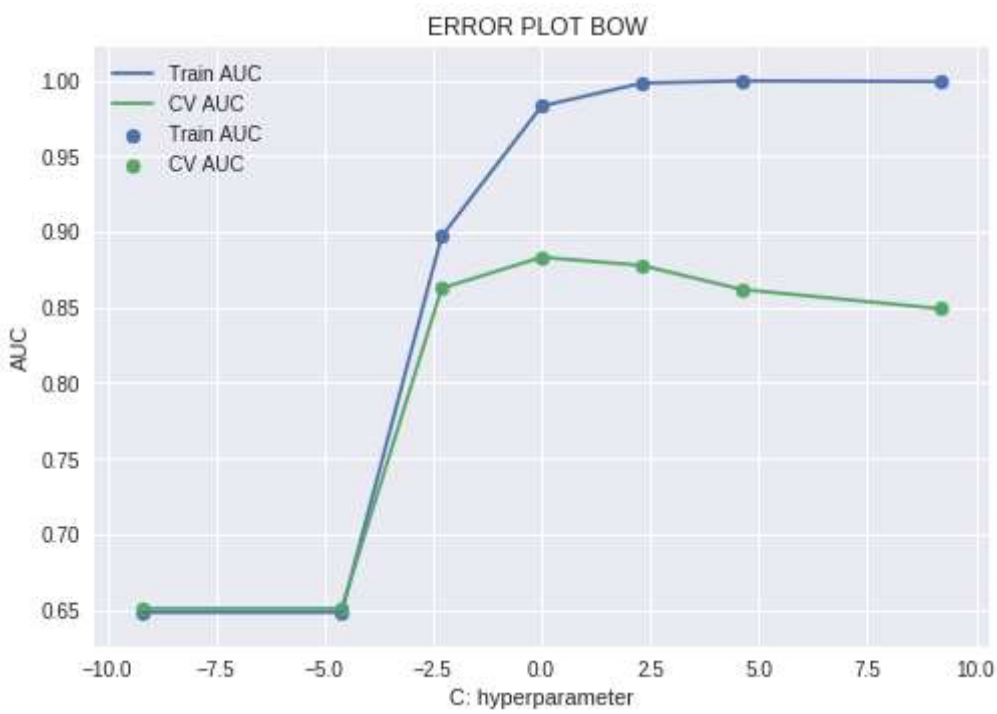
In [0]:
```
X = preprocessed_reviews[0:20000]
Y = final['Score'].values[0:20000]

#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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25)
```

In [60]:
```
from sklearn.model_selection import learning_curve
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import TruncatedSVD

vectorizer = CountVectorizer(min_df=10, max_features=500)
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)


#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_cv_bow=scaler.transform(X_cv_bow)
X_test_bow = scaler.transform(X_test_bow)

#Reducing the dimensions
svd = TruncatedSVD(n_components=10, n_iter=7, random_state=42)
svd.fit(X_train_bow)

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
(12000, 500) (12000,)
(4000, 500) (4000,)
(4000, 500) (4000,)
==============================================================================
=======================
```

Hyperparameter Tuning

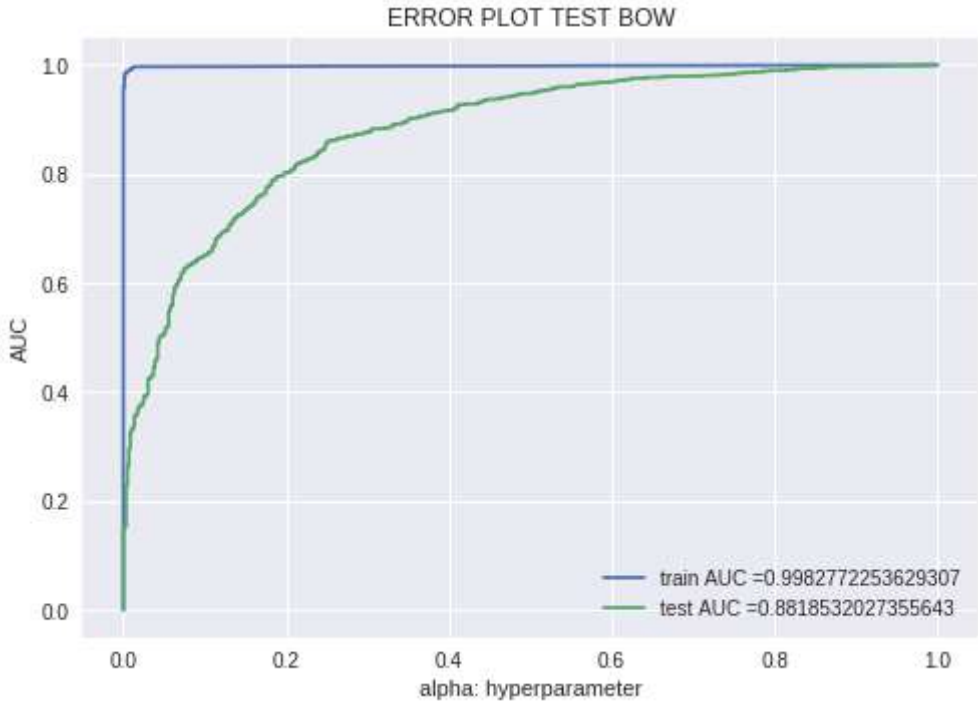
In [61]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning
train_auc = []
cv_auc = []
C=[10**-4, 10**-2,10**-1 ,10**0,10**1, 10**2, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SVC(kernel='rbf',C=i,class_weight='balanced')
    cc = CalibratedClassifierCV(clf)
    cc.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 =  cc.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  cc.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 vaalues to see the best hyperparameter
C=list(np.log([10**-4, 10**-2,10**-1 ,10**0,10**1, 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='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT BOW ")
plt.show()
```



In [0]:
```
best_C=10
```

Testing with Test Data

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


clf=SVC(kernel='rbf',C=best_C,class_weight='balanced')
cc = CalibratedClassifierCV(clf)
cc.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, cc.predict_proba(X_train
_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.predict_proba(X_test_bow
)[:,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)

#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, cc.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,cc.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
================================================================================
========================
Train confusion matrix
```

```
Test confusion matrix
```

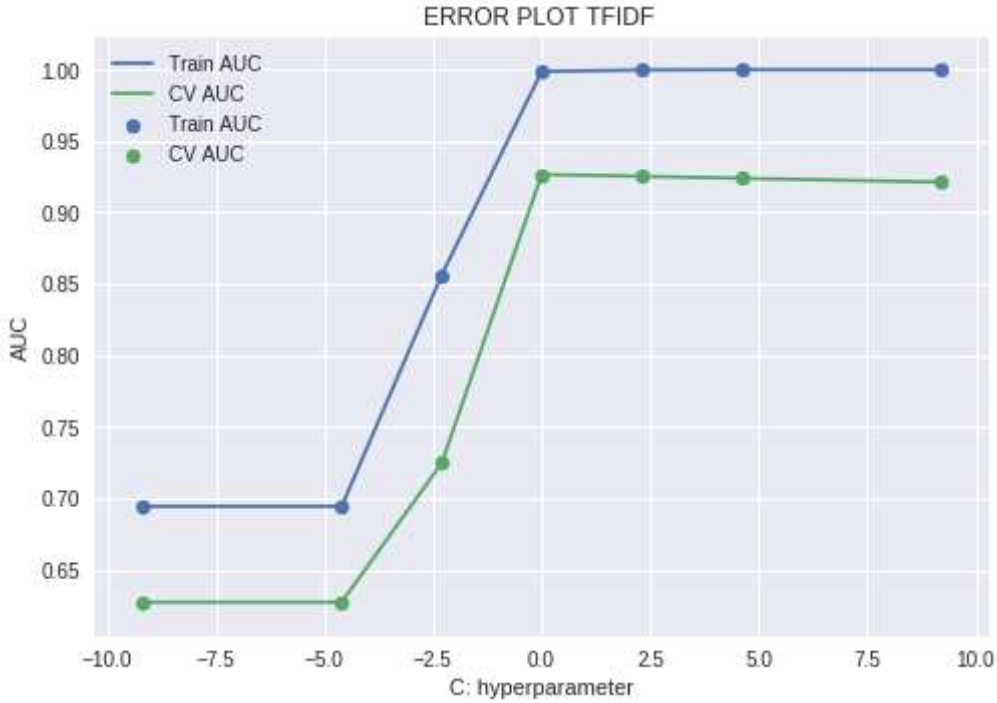
[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [64]:
```
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_cv_tf=tf_idf_vect.transform(X_cv)
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_cv_tf = scaler.transform(X_cv_tf)
X_test_tf = scaler.transform(X_test_tf)


#Reducing the dimensions
svd = TruncatedSVD(n_components=10, n_iter=7, random_state=42)
svd.fit(X_train_tf)

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
(12000, 6994) (12000,)
(4000, 6994) (4000,)
(4000, 6994) (4000,)
================================================================================
=======================
```

Hyper Parameter Tuning

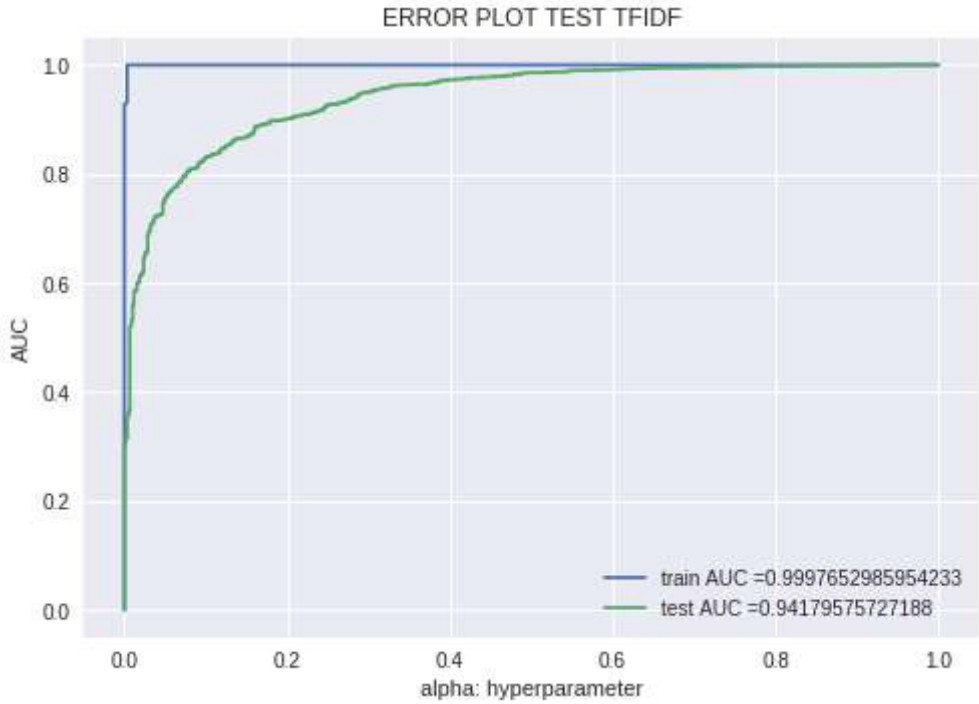
In [65]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning
train_auc = []
cv_auc = []
C=[10**-4, 10**-2,10**-1 ,10**0,10**1, 10**2, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SVC(kernel='rbf',C=i,class_weight='balanced')
    cc = CalibratedClassifierCV(clf)
    cc.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 =  cc.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  cc.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 values to see the best hyperparameter
C=list(np.log([10**-4, 10**-2,10**-1 ,10**0,10**1, 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='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDF ")
plt.show()
```



In [0]:
```
best_c=10
```

Testing with Test Data

In [67]:
```
from sklearn.metrics import roc_curve, auc


clf=SVC(kernel='rbf',C=best_C,class_weight='balanced')
cc = CalibratedClassifierCV(clf)
cc.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, cc.predict_proba(X_train
_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.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)

#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, cc.predict(X_train_tf)))
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,cc.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
==========================================================================
========================
Train confusion matrix
```

```
Test confusion matrix
```

[5.2.3] Applying RBF SVM on AVG W2V, SET 3

In [68]:
```
# 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



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])
```

```
number of words that occured minimum 5 times  6476
sample words  ['dog', 'years', 'old', 'jack', 'terrier', 'fed', 'solid', 'gol
d', 'barking', 'moon', 'since', 'puppy', 'no', 'issue', 'may', 'year', 'start
ed', 'weekly', 'episode', 'bloody', 'took', 'vet', 'special', 'doctor', 'kind
s', 'testing', 'cost', 'not', 'figure', 'root', 'cause', 'searched', 'web',
'sites', 'found', 'many', 'owners', 'experiences', 'due', 'change', 'foods',
'therefore', 'tried', 'another', 'salmon', 'potato', 'chicken', 'rice', 'succ
essful', 'still']
```

In [69]:
```
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%|██████████| 12000/12000 [00:14<00:00, 835.21it/s]

(12000, 50)
[-0.08291401  0.3699944   0.26851166  0.30173113  0.09510536  0.39280695
 -0.17527677  0.68120248  0.0975246   0.00735864 -0.21446062  0.04740411
 -0.13981778 -0.7914513   0.07232243  0.40044194  0.5620122  -0.93921644
 -0.35199888  0.27369439  0.5497119   0.08793421 -0.18540088  0.18687924
 -0.39522714  0.51507658  0.46520364  0.06867894  0.64277493  0.60970556
 -0.45146925 -0.58111916 -0.11950749 -0.20414288  0.10045563 -0.03714844
 -0.16890444  0.52661271  0.26901891 -0.25087542  0.18403862 -0.44908794
  0.65976617 -0.2452204  -0.06027025  0.3019052   0.48398942  0.21654085
 -0.46366312 -0.47106151]
```

In [70]:
```
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this
list
for sent in tqdm(list_of_sentance_cv): # 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_cv.append(sent_vec)
sent_vectors_cv = np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
100%|██████████| 4000/4000 [00:04<00:00, 820.65it/s]

(4000, 50)
[ 0.10049232 -0.32505345  0.03639861  0.19198918  0.17365173  0.54061327
  0.03155979  0.39388392  0.0328611   0.05878075 -0.5622218   0.29502815
 -0.17827996 -0.21399289 -0.17777523  0.66749556  0.0026984  -0.79950212
 -0.33849582  0.09843663  0.4610862   0.10548058 -0.32604629  0.50654018
  0.47165591  0.60816921  0.46126142 -0.38081653  0.89310999  0.69601586
 -0.09763021 -0.38491303  0.07151018  0.04484542  0.4055275   0.00525674
 -0.39722092  0.49000483  0.13892414 -0.06335016  0.35636254 -0.86271764
  0.2784194  -0.23770679  0.00753077  0.12960763  0.16509253  0.22551919
 -0.74455976 -0.0467272 ]
```

In [71]:
```
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])
```

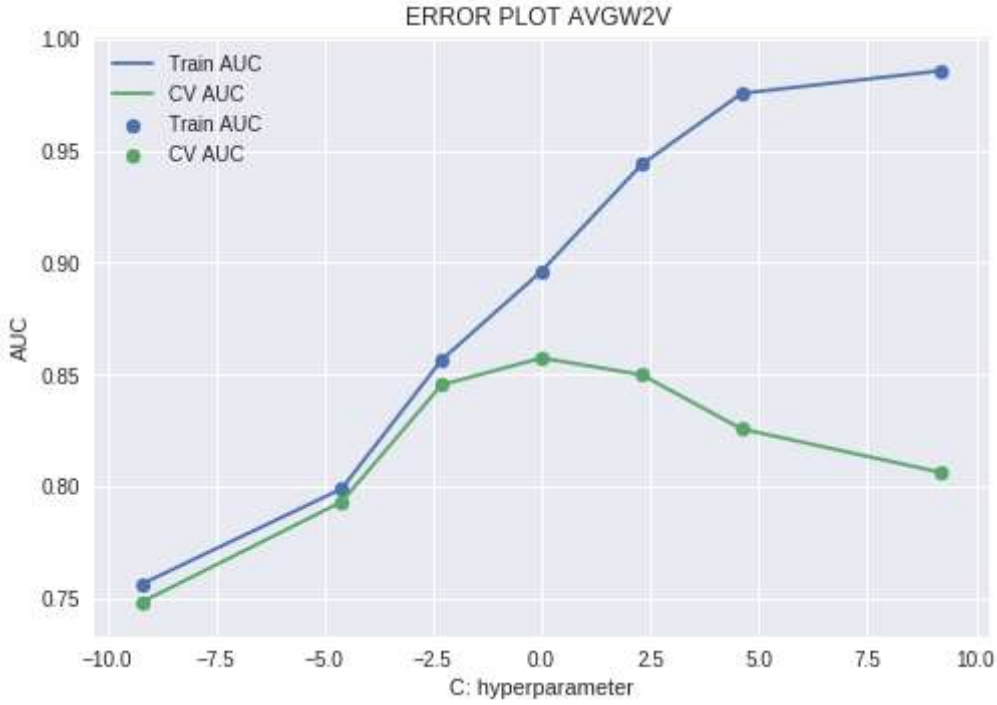
```
100%|██████████| 4000/4000 [00:04<00:00, 806.09it/s]

(4000, 50)
[ 0.10230216 -0.34003865  0.03267605  0.15563538  0.10524591  0.46084356
 -0.0292371   0.45949519  0.09672472 -0.00505164 -0.40219684  0.36158409
 -0.20300847 -0.18622412 -0.10007658  0.64374568 -0.00962435 -0.71481948
 -0.39635538  0.17447181  0.44136978  0.1490059  -0.27466816  0.48088556
  0.19337241  0.5642669   0.42299395 -0.41589155  0.92546692  0.74969323
 -0.21448519 -0.36882937  0.07214456  0.01575615  0.46905775  0.04016492
 -0.39548923  0.45801235  0.12911635 -0.08774683  0.23970843 -0.79864039
  0.19896008 -0.2654042   0.07146419  0.14312986  0.07709897  0.2532438
 -0.78192115 -0.0386465 ]
```

In [72]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


#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_cv= scaler.transform(sent_vectors_cv)
sent_vectors_test = scaler.transform(sent_vectors_test)




print("After vectorizations")
print(sent_vectors_train.shape, y_train.shape)
print(sent_vectors_cv.shape, y_cv.shape)
print(sent_vectors_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(12000, 50) (12000,)
(4000, 50) (4000,)
(4000, 50) (4000,)
==============================================================================
=======================
```

Hyper parameter Tuning

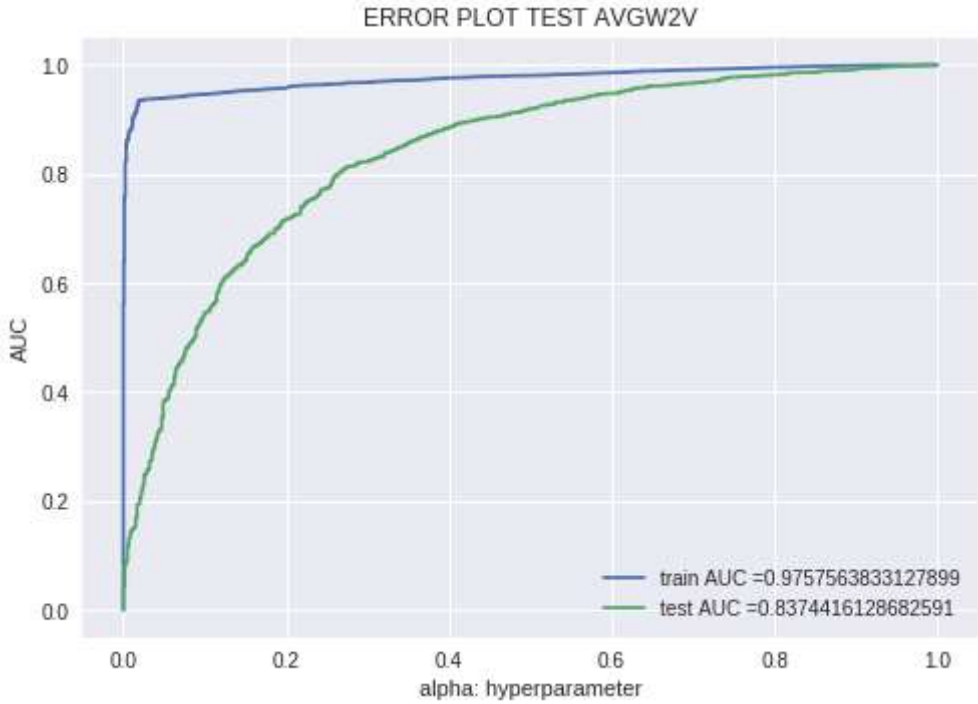
In [73]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning
train_auc = []
cv_auc = []
C=[10**-4, 10**-2,10**-1 ,10**0,10**1, 10**2, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SVC(kernel='rbf',C=i,class_weight='balanced')
    cc = CalibratedClassifierCV(clf)
    cc.fit(sent_vectors_train,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 =  cc.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  cc.predict_proba(sent_vectors_cv)[:,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 vaalues to see the best hyperparameter
C=list(np.log([10**-4, 10**-2,10**-1 ,10**0,10**1, 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='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AVGW2V ")
plt.show()
```



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

Testing with Test data

In [75]:
```
from sklearn.metrics import roc_curve, auc


clf=SVC(kernel='rbf',C=best_C,class_weight='balanced')
cc = CalibratedClassifierCV(clf)
cc.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, cc.predict_proba(sent_ve
ctors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.predict_proba(sent_vecto
rs_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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST AVGW2V")
plt.show()

print("="*100)

#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, cc.predict(sent_vectors_train
)))
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,cc.predict(sent_vectors_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
==============================================================================
========================
Train confusion matrix
```

```
Test confusion matrix
```

[5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

In [76]:
```
#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])
```

```
number of words that occured minimum 5 times  6476
sample words  ['dog', 'years', 'old', 'jack', 'terrier', 'fed', 'solid', 'gol
d', 'barking', 'moon', 'since', 'puppy', 'no', 'issue', 'may', 'year', 'start
ed', 'weekly', 'episode', 'bloody', 'took', 'vet', 'special', 'doctor', 'kind
s', 'testing', 'cost', 'not', 'figure', 'root', 'cause', 'searched', 'web',
'sites', 'found', 'many', 'owners', 'experiences', 'due', 'change', 'foods',
'therefore', 'tried', 'another', 'salmon', 'potato', 'chicken', 'rice', 'succ
essful', 'still']
```

In [77]:
```
#Train Data

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%|          | 9/12000 [00:00<02:23, 83.57it/s]

<class 'list'>

100%|██████████| 12000/12000 [02:06<00:00, 95.15it/s]
```

In [78]:
```
#CV Data
i=0
list_of_sentance_tf_cv=[]
for sentance in X_cv:
    list_of_sentance_tf_cv.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_cv= []; # the tfidf-w2v for each sentence/review is stored in this
 list
row=0;
for sent in tqdm(list_of_sentance_tf_cv): # 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_cv.append(sent_vec)
    row += 1
```

```
100%|██████████| 4000/4000 [00:41<00:00, 96.80it/s]
```

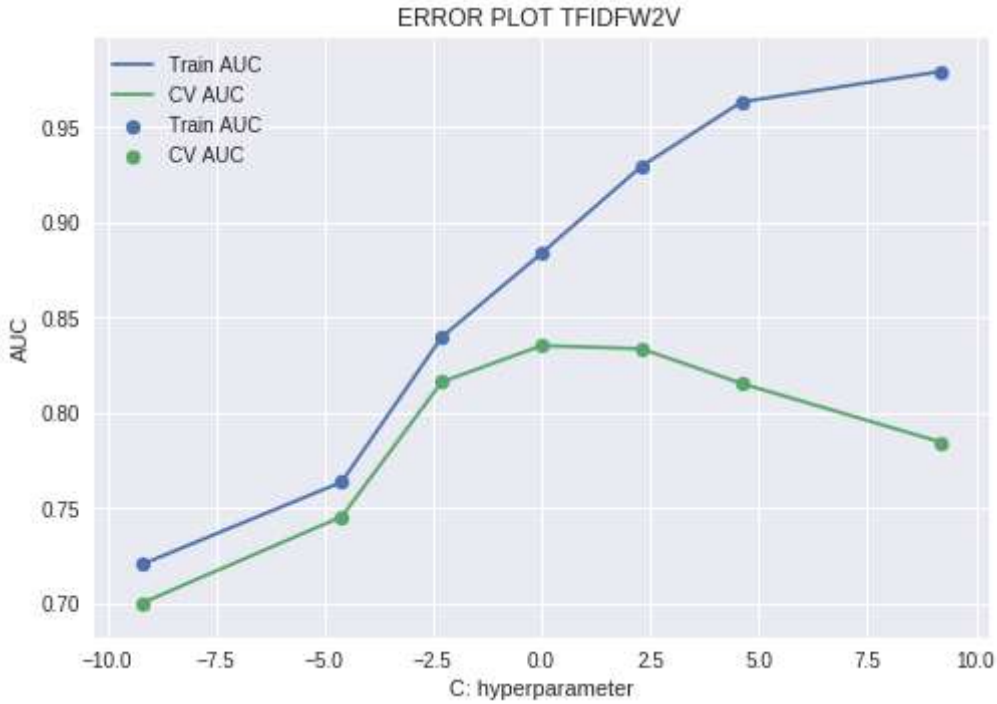
In [79]:
```
#Test Data
i=0
list_of_sentance_tf_test=[]
for sentance in X_test:
    list_of_sentance_tf_test.append(sentance.split())

# 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%|██████████| 4000/4000 [00:41<00:00, 95.46it/s]
```

In [0]:
```
from sklearn.model_selection import learning_curve
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer




#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_cv = scaler.transform(tfidf_sent_cv)
tfidf_sent_test = scaler.transform(tfidf_sent_test)
```

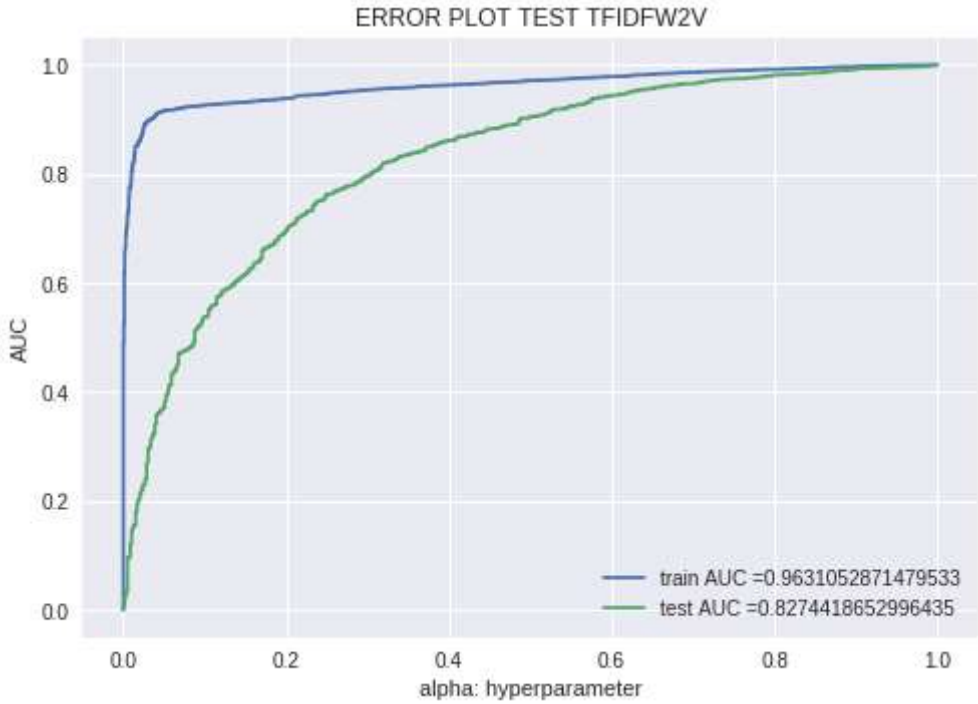
Hyperparameter Tuning

In [81]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


#Hyper parameter tuning
train_auc = []
cv_auc = []
C=[10**-4, 10**-2,10**-1 ,10**0,10**1, 10**2, 10**4]
#For loop to find train_auc,cv_auc
for i in C:
    clf=SVC(kernel='rbf',C=i,class_weight='balanced')
    cc = CalibratedClassifierCV(clf)
    cc.fit(tfidf_sent_train,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 =  cc.predict_proba(tfidf_sent_train)[:,1]
    y_cv_pred =  cc.predict_proba(tfidf_sent_cv)[:,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 values to see the best hyperparameter
C=list(np.log([10**-4, 10**-2,10**-1 ,10**0,10**1, 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='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V ")
plt.show()
```



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

In [83]:
```
from sklearn.metrics import roc_curve, auc


clf=SVC(kernel='rbf',C=best_C,class_weight='balanced')
cc = CalibratedClassifierCV(clf)
cc.fit(tfidf_sent_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, cc.predict_proba(tfidf_s
ent_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cc.predict_proba(tfidf_sent
_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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TEST TFIDFW2V")
plt.show()

print("="*100)

#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, cc.predict(tfidf_sent_train)))
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,cc.predict(tfidf_sent_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```




```
================================================================================
========================
Train confusion matrix
```

```
Test confusion matrix
```

[6] Conclusions

In [89]:
```
from prettytable import PrettyTable

x = PrettyTable()

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

x.add_row(["BOW", "Linear", 10,0.9297 ])
x.add_row(["TFIDF", "Linear",10,0.9565])
x.add_row(["W2V", "Linear", 0.01,0.9009])
x.add_row(["TFIDFW2V", "Linear",0.01,0.8779])
x.add_row(["BOW", "RBF", 10,0.8818])
x.add_row(["TFIDF", "RBF",10,0.9417])
x.add_row(["W2V", "RBF", 100, 0.8374])
x.add_row(["TFIDFW2V", "RBF",100,0.8274])


print(x)
```

```
+------------+--------+-----------------+--------+
| Vectorizer | Kernel | Hyper-Parameter |  AUC   |
+------------+--------+-----------------+--------+
|    BOW     | Linear |        10       | 0.9297 |
|   TFIDF    | Linear |        10       | 0.9565 |
|    W2V     | Linear |       0.01      | 0.9009 |
|  TFIDFW2V  | Linear |       0.01      | 0.8779 |
|    BOW     |  RBF   |        10       | 0.8818 |
|   TFIDF    |  RBF   |        10       | 0.9417 |
|    W2V     |  RBF   |       100       | 0.8374 |
|  TFIDFW2V  |  RBF   |       100       | 0.8274 |
+------------+--------+-----------------+--------+
```

1.Applying Linear kernel on TFIDF vectorizer will yield best AUC.