Amazon Fine Food Reviews Analysis

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

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

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. **Id**
2. **ProductId - unique identifier for the product**
3. **UserId - unqiue identifier for the user**
4. **ProfileName**
5. **HelpfulnessNumerator - number of users who found the review helpful**
6. **HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not**
7. **Score - rating between 1 and 5**
8. **Time - timestamp for the review**
9. **Summary - brief summary of the review**
10. **Text - text of the review**

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. **.csv file**
2. **SQLite Database**

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

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

In [69]:

```
%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 [70]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 400000 """, 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 (400000, 10)
```

Out[70]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	S

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	I
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	A
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	

In [71]:

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

In [72]:

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

```
(80668, 7)
```

Out[72]:

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

In [73]:

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

Out[73]:

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

In [74]:

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

Out[74]:

```
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 [75]:

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

Out[75]:

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

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 [76]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [77]:

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

Out[77]:

```
(286837, 10)
```

In [78]:

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

Out[78]:

```
71.70925
```

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 [79]:

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

display.head()
```

Out[79]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200

In [80]:

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

In [81]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(286835, 10)
```

Out[81]:

```
1    241601
0     45234
Name: Score, dtype: int64
```

segragating datapoints w.r.t calss labels and sampling optimum number of data points for diffrent implementations of SVM

In [82]:

```
zero_class=final[final.Score==0]
print(zero_class['Score'].value_counts())
print(zero_class.shape)
one_class=final[final.Score==1]
print(one_class['Score'].value_counts())
print(one_class.shape)
```

```
0    45234
Name: Score, dtype: int64
(45234, 10)
1    241601
Name: Score, dtype: int64
(241601, 10)
```

In [83]:

```
one_class1=one_class.sample(n=45234)
print(one_class1.shape)
```

```
(45234, 10)
```

In [84]:

```
print(zero_class.shape)
print(one_class1.shape)
combined_frame=pd.concat([zero_class,one_class1])
print(combined_frame.shape)
```

```
(45234, 10)
(45234, 10)
(90468, 10)
```

In [85]:

```
final_new_frame_90 =combined_frame.sample(frac=1)
```

In [86]:

```
print(type(final_new_frame_90))
print(final_new_frame_90.shape)
print(final_new_frame_90['Score'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
(90468, 10)
1    45234
0    45234
Name: Score, dtype: int64
```

final_new_frame_90 to be used while operating with linear kernel

In [87]:

```
one_class2=one_class.sample(n=10000)
print(one_class2.shape)
zero_class2=zero_class.sample(n=10000)
print(zero_class2.shape)
combined_frame1=pd.concat([zero_class2,one_class2])
print(combined_frame1.shape)
```

```
(10000, 10)
(10000, 10)
(20000, 10)
```

In [88]:

```
final_new_frame_20 =combined_frame1.sample(frac=1)
print(type(final_new_frame_20))
print(final_new_frame_20.shape)
print(final_new_frame_20['Score'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
(20000, 10)
1    10000
0    10000
Name: Score, dtype: int64
```

final_new_frame_20 to be used while operating with RBF kernel

[3] Preprocessing

In [89]:

```
# 1.11 -this here cotinuation https://stackoverflow.com/a/47091490/4084039
import re
from bs4 import BeautifulSoup

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


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


from tqdm import tqdm
preprocessed_reviews_90 = []
# tqdm is for printing the status bar
for sentance in tqdm(final_new_frame_90['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopword
s)
    preprocessed_reviews_90.append(sentance.strip())

j=0
for i in tqdm(preprocessed_reviews_90):
    j=j+1
print(j)


from tqdm import tqdm
preprocessed_reviews_20 = []
# tqdm is for printing the status bar
for sentance in tqdm(final_new_frame_20['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopword
s)
    preprocessed_reviews_20.append(sentance.strip())



j=0
for i in tqdm(preprocessed_reviews_20):
    j=j+1
print(j)
```

```
100%|██████████████████████████████████████████████████| 90468/90
468 [00:33<00:00, 2694.48it/s]
100%|████████████████████████████████████████████████| 90468/90468
[00:00<00:00, 2179508.75it/s]
```

```
90468
```

```
100%|██████████████████████████████████████████████████| 20000/20
000 [00:07<00:00, 2716.42it/s]
100%|████████████████████████████████████████████████| 20000/20000
[00:00<00:00, 1671270.50it/s]
```

```
20000
```

In [90]:

```
print(len(preprocessed_reviews_90))
print(len(preprocessed_reviews_20))
print(type(final_new_frame_90))
print(type(final_new_frame_20))
print(final_new_frame_90.shape)
print(final_new_frame_20.shape)
```

```
90468
20000
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
(90468, 10)
(20000, 10)
```

In [24]:

```
#below preprocessing is not used
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.am
azon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><br />The Victor M380 and M502
traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  T
he best thing was that there were a lot of "brown" chips in the bsg (my favorite), so I b
ought some more through amazon and shared with family and friends.  I am a little disappo
inted that there are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are thicker and cr
```

```
unchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; t
he other wants crispy cookies.  Hey, I'm sorry; but these reviews do nobody any good beyo
nd reminding us to look  before ordering.<br /><br />These are chocolate-oatmeal cookies.
If you don't like that combination, don't order this type of cookie.  I find the combo qu
ite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the co
okie sort of a coconut-type consistency.  Now let's also remember that tastes differ; so,
I've given my opinion.<br /><br />Then, these are soft, chewy cookies -- as advertised.
They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy."  I h
appen to like raw cookie dough; however, I don't see where these taste like raw cookie do
ugh.  Both are soft, however, so is this the confusion?  And, yes, they stick together.
Soft cookies tend to do that.  They aren't individually wrapped, which would add to the c
ost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you wa
nt something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie that'
s soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try.
I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great c
offee.  dcaf is very good as well
==================================================
```

In [15]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The
Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, bu
t only right nearby.
```

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tag
s-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)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380
and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right
nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  T
he best thing was that there were a lot of "brown" chips in the bsg (my favorite), so I b
ought some more through amazon and shared with family and friends.  I am a little disappo
inted that there are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are thicker and cr
```

```
unchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; t
he other wants crispy cookies.  Hey, I'm sorry; but these reviews do nobody any good beyo
nd reminding us to look  before ordering.These are chocolate-oatmeal cookies.  If you don
't like that combination, don't order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort
of a coconut-type consistency.  Now let's also remember that tastes differ; so, I've give
n my opinion.Then, these are soft, chewy cookies -- as advertised.  They are not "crispy"
cookies, or the blurb would say "crispy," rather than "chewy."  I happen to like raw cook
ie dough; however, I don't see where these taste like raw cookie dough.  Both are soft, h
owever, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to d
o that.  They aren't individually wrapped, which would add to the cost.  Oh yeah, chocola
te chip cookies tend to be somewhat sweet.So, if you want something hard and crisp, I sug
gest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy and tastes like a co
mbination of chocolate and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.
dcaf is very good as well
```

In [17]:

```python
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [18]:

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

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; t
he other wants crispy cookies.  Hey, I am sorry; but these reviews do nobody any good bey
ond reminding us to look  before ordering.<br /><br />These are chocolate-oatmeal cookies
.  If you do not like that combination, do not order this type of cookie.  I find the com
bo quite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives t
he cookie sort of a coconut-type consistency.  Now let is also remember that tastes diffe
r; so, I have given my opinion.<br /><br />Then, these are soft, chewy cookies -- as adve
rtised.  They are not "crispy" cookies, or the blurb would say "crispy," rather than "che
wy."  I happen to like raw cookie dough; however, I do not see where these taste like raw
cookie dough.  Both are soft, however, so is this the confusion?  And, yes, they stick to
gether.  Soft cookies tend to do that.  They are not individually wrapped, which would ad
d to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So,
if you want something hard and crisp, I suggest Nabiso is Ginger Snaps.  If you want a co
okie that is soft, chewy and tastes like a combination of chocolate and oatmeal, give the
se a try.  I am here to place my second order.
==================================================
```

In [19]:

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

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The
Victor  and  traps are unreal, of course -- total fly genocide. Pretty stinky, but only r
```

```
ight nearby.
```

In [20]:

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

```
Wow So far two two star reviews One obviously had no idea what they were ordering the oth
er wants crispy cookies Hey I am sorry but these reviews do nobody any good beyond remind
ing us to look before ordering br br These are chocolate oatmeal cookies If you do not lik
e that combination do not order this type of cookie I find the combo quite nice really Th
e oatmeal sort of calms the rich chocolate flavor and gives the cookie sort of a coconut 
type consistency Now let is also remember that tastes differ so I have given my opinion b
r br Then these are soft chewy cookies as advertised They are not crispy cookies or the b
lurb would say crispy rather than chewy I happen to like raw cookie dough however I do no
t see where these taste like raw cookie dough Both are soft however so is this the confus
ion And yes they stick together Soft cookies tend to do that They are not individually wr
apped which would add to the cost Oh yeah chocolate chip cookies tend to be somewhat swee
t br br So if you want something hard and crisp I suggest Nabiso is Ginger Snaps If you w
ant a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal gi
ve these a try I am here to place my second order
```

In [21]:

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

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

In [22]:

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

```
100%|██████████████████████████████████████████████████████████████████████████████| 4986/4
986 [00:01<00:00, 3137.37it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sor
ry reviews nobody good beyond reminding us look ordering chocolate oatmeal cookies not li
ke combination not order type cookie find combo quite nice really oatmeal sort calms rich
chocolate flavor gives cookie sort coconut type consistency let also remember tastes diff
er given opinion soft chewy cookies advertised not crispy cookies blurb would say crispy
rather chewy happen like raw cookie dough however not see taste like raw cookie dough sof
t however confusion yes stick together soft cookies tend not individually wrapped would a
dd cost oh yeah chocolate chip cookies tend somewhat sweet want something hard crisp sugg
est nabiso ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal
give try place second order'
```

[3.2] Preprocessing Review Summary

In [6]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#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', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abd
ominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [26]:

```
#bi-gram, tri-gram and n-gram

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

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [27]:

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

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able g
et', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely love', 'absolutely no'
, 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [28]:

```
# 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 [42]:

```
# Using Google News Word2Vectors

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


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
```

```
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321
66481018), ('excellent', 0.9944332838058472), ('especially', 0.9941144585609436), ('baked
', 0.9940600395202637), ('salted', 0.994047224521637), ('alternative', 0.9937226176261902
), ('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750
883102417), ('de', 0.9992610216140747), ('miss', 0.9992451071739197), ('melitta', 0.99921
8761920929), ('choice', 0.9992102384567261), ('american', 0.9991837739944458), ('beef', 0
.9991780519485474), ('finish', 0.9991567134857178)]
```

In [36]:

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

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'n
earby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly',
'wait', 'try', 'love', 'call', 'instead', 'removed', 'easily', 'daughter', 'designed', 'p
rinted', 'use', 'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun'
, 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding
', 'window', 'everybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [38]:

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

```
100%|████████████████████████████████████████████████████████████████| 4986/4
986 [00:03<00:00, 1330.47it/s]
```

```
986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [39]:

```python
# 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 [41]:

```python
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

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

[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**
    - **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 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 with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

7. **Conclusion**

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

Note: Data Leakage

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

Applying SVM

[5.1] Linear SVM

[5.1.1] Applying Linear SVM on BOW, SET 1

In [91]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)
```

In [92]:

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

X_train_bow = vectorizer.transform(X_train)
```

```
X_test_bow = vectorizer.transform(X_test)
```

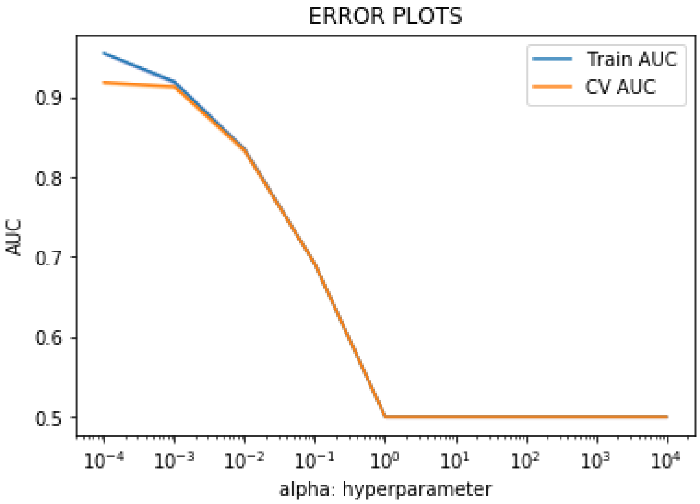
In [93]:

```
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
print(type(X_train_bow))
print(X_train_bow.get_shape())
```

```
After vectorizations
(60613, 48566) (60613,)
(29855, 48566) (29855,)
<class 'scipy.sparse.csr.csr_matrix'>
(60613, 48566)
```

In [101]:

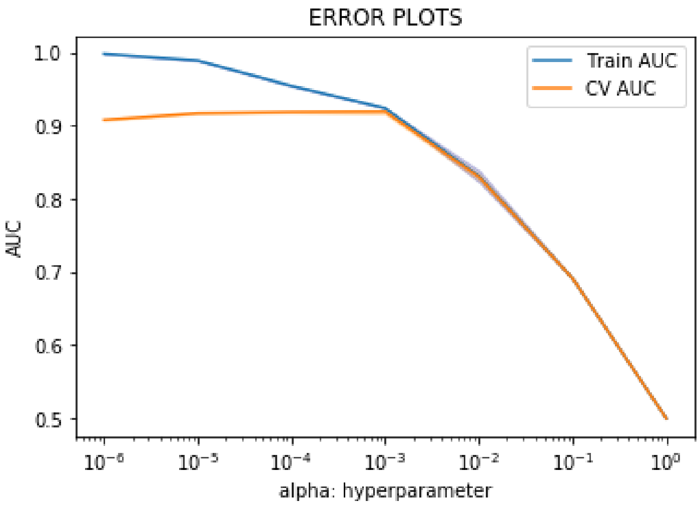
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [106]:

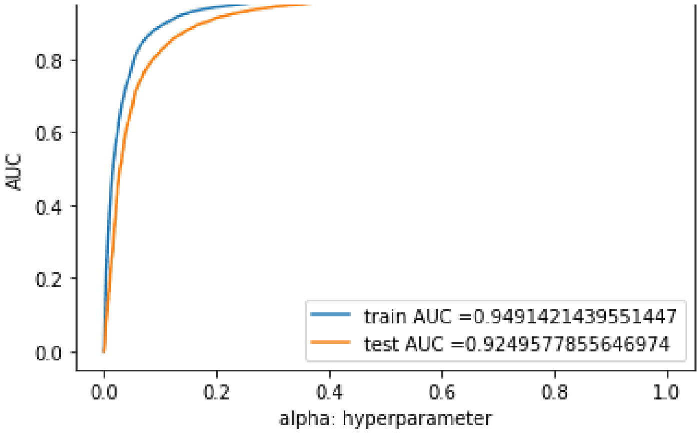
```
from sklearn.model_selection import GridSearchCV
```

```python
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [108]:

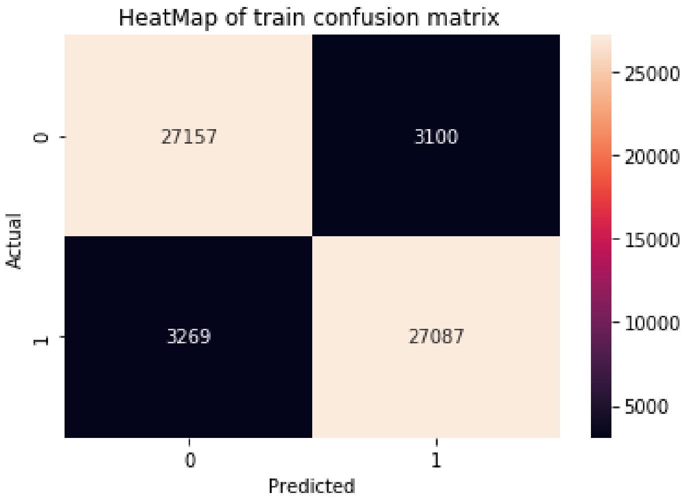
```python
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l1',alpha=0.0001,class_weight='balanced')
SGDclas.fit(X_train_bow, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(X_train_bow, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(X_train_bow)[:,1
])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(X_test_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

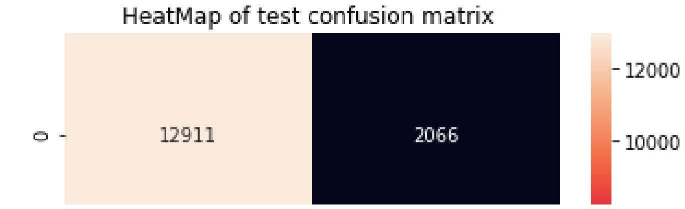


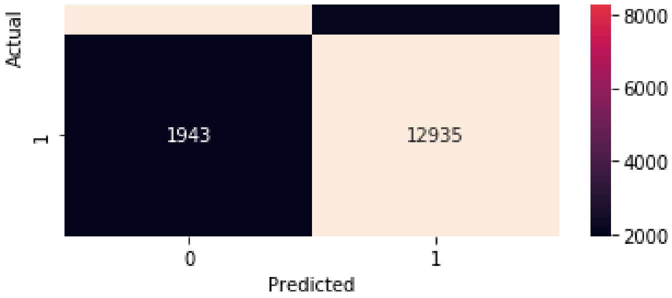
In [109]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(X_train_bow))
y=confusion_matrix(y_test, SGDclas.predict(X_test_bow))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[27157  3100]
 [ 3269 27087]]
Test confusion matrix
[[12911  2066]
 [ 1943 12935]]
```







In [114]:

```
SGD_Claf_Linear_Weights=SGDclas.coef_
```

In [115]:

```
cp5 = np.argsort(SGD_Claf_Linear_Weights[0])
```

In [116]:

```
l1=vectorizer.get_feature_names()
```

In [117]:

```
topnve=cp5[:10]
toppve=cp5[-10:]
```

top 10 best features for positive class.

In [118]:

```
pve=[]
for i in toppve:
    pve.append(l1[i])
print(pve)
```

```
['pleasantly', 'sustained', 'tbs', 'sifter', 'prepackaged', 'frangos', 'extendbar', 'rib'
, 'clementine', 'tonkotsu']
```

top 10 best features for negative class.

In [119]:

```
l1=vectorizer.get_feature_names()
nve=[]
for i in topnve:
    nve.append(l1[i])
print(nve)
```

```
['chlorhexidine', 'cancelled', 'discs', 'poison', 'deceptive', 'worst', 'haystacks', 'rui
ned', 'hopes', 'disappointing']
```

BOW vectorized data which contains Review length feature added

In [123]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)
```

In [124]:

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

X_train_bow = vectorizer.transform(X_train)
```

```
X_test_bow = vectorizer.transform(X_test)
```

In [125]:

```
from scipy import sparse
review_length_train=[]

for eachreview in X_train:
    x=len(eachreview)
    review_length_train.append(x)
```

In [126]:

```
review_length_test=[]

for eachreview in X_test:
    x=len(eachreview)
    review_length_test.append(x)

print(review_length_test[0])
```

```
428
```

In [127]:

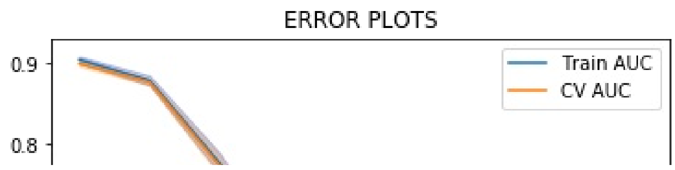
```
X_train_bow_f1=sparse.hstack((X_train_bow,np.array(review_length_train)[:,None]))
```

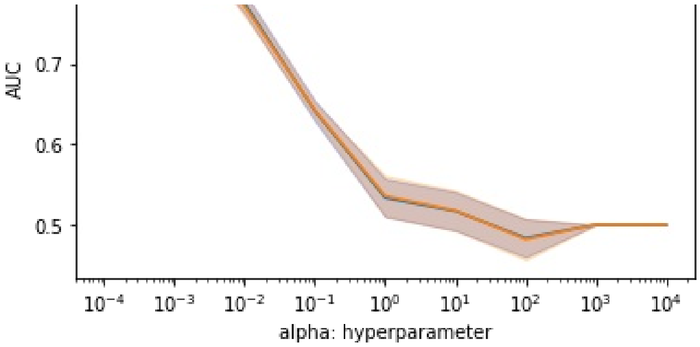
In [128]:

```
X_test_bow_f1=sparse.hstack((X_test_bow,np.array(review_length_test)[:,None]))
```

In [129]:

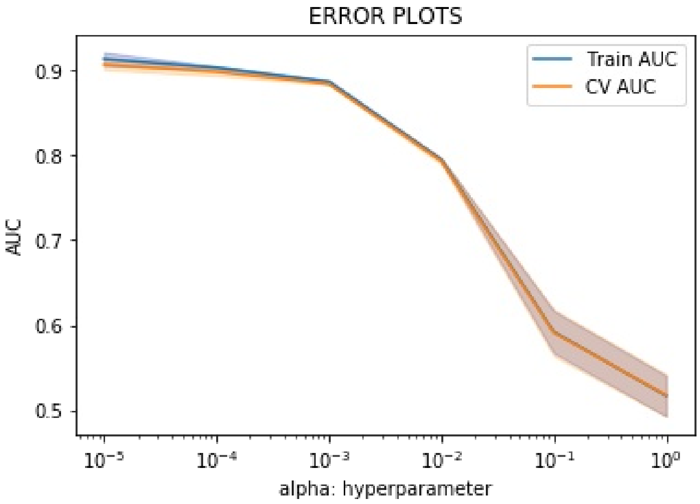
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_f1, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```





In [130]:

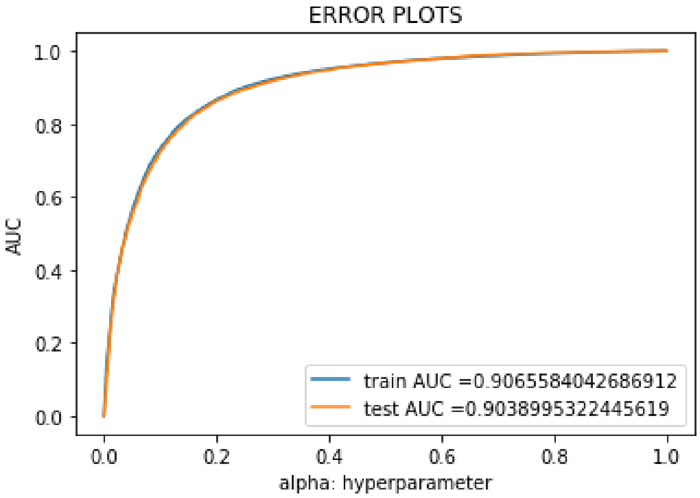
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.00001,0.0001, 0.001, 0.01, 0.1, 1]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_f1, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.00001,0.0001, 0.001, 0.01, 0.1, 1]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [131]:

```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l1',alpha=0.0001,class_weight='balanced')
SGDclas.fit(X_train_bow_f1, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
```

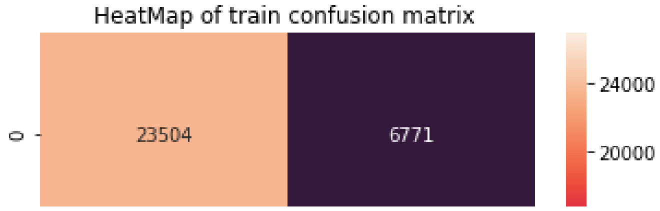
```
model.fit(X_train_bow_f1, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(X_train_bow_f1)[
:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(X_test_bow_f1)[:,1]
)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

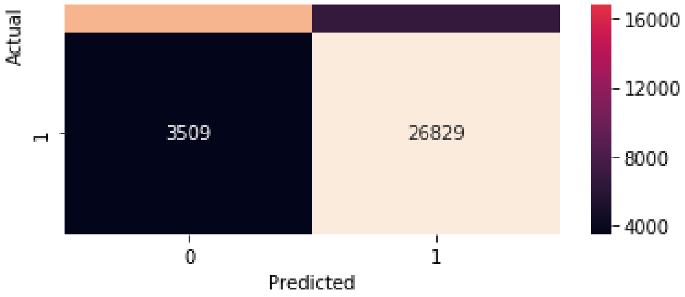


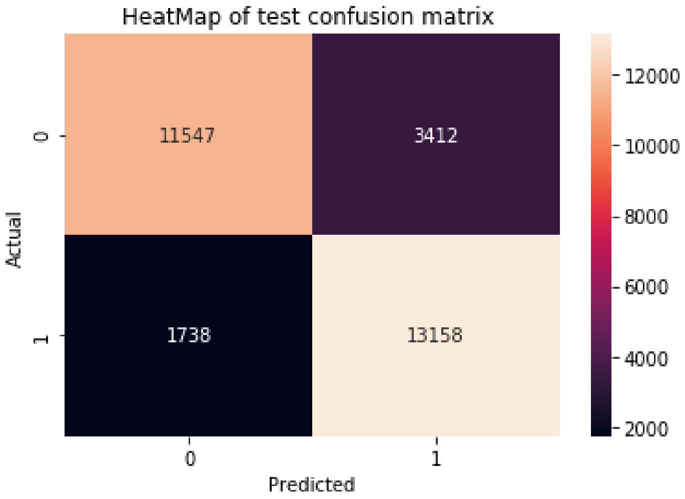
In [132]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(X_train_bow_f1))
y=confusion_matrix(y_test, SGDclas.predict(X_test_bow_f1))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[23504  6771]
 [ 3509 26829]]
Test confusion matrix
[[11547  3412]
 [ 1738 13158]]
```







Applying Linear SVM on BOW L2 regularizer

In [133]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)

vectorizer = CountVectorizer()
vectorizer.fit(X_train)

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

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
print(type(X_train_bow))
print(X_train_bow.get_shape())
```

```
After vectorizations
(60613, 48522) (60613,)
(29855, 48522) (29855,)
<class 'scipy.sparse.csr.csr_matrix'>
(60613, 48522)
```

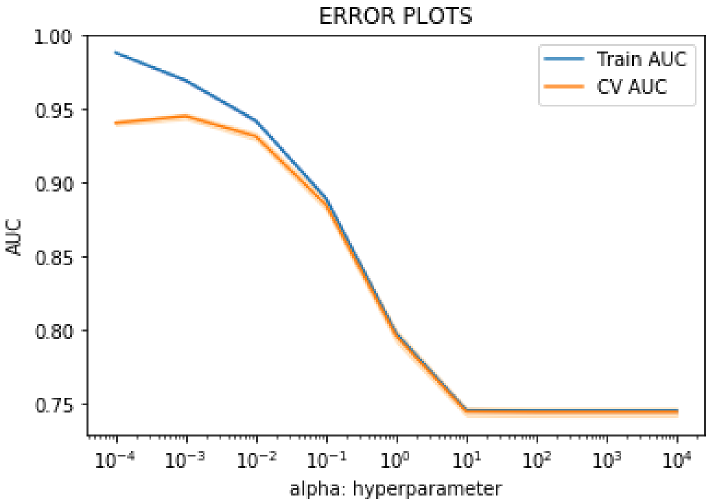
In [134]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',class_weight='balanced')
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],'penalty':['l1',
'l2']}
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

print(clf.best_estimator_)
```

```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [135]:

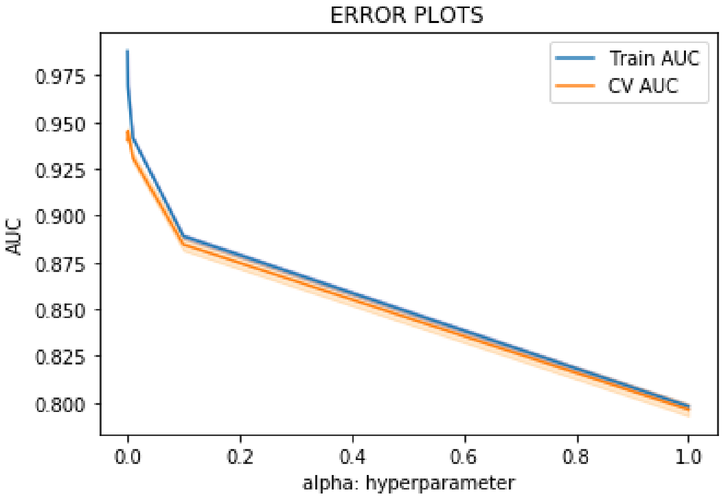
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [136]:

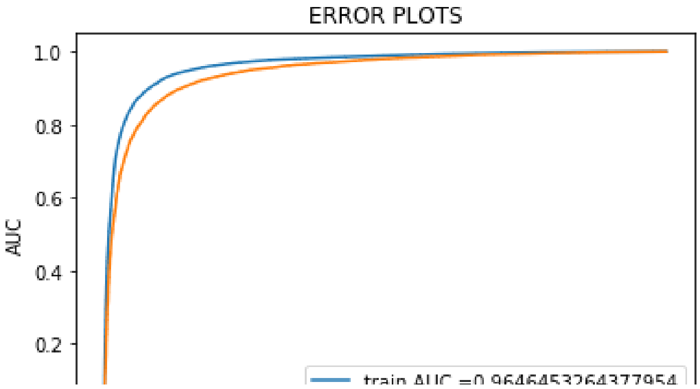
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
```

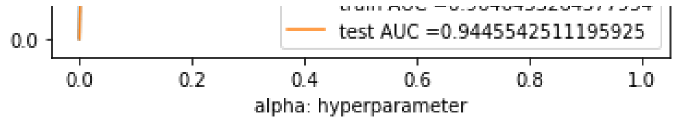
```
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1]


plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [137]:

```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l2',alpha=0.001,class_weight='balanced')
SGDclas.fit(X_train_bow, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(X_train_bow, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(X_train_bow)[:,1
])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(X_test_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

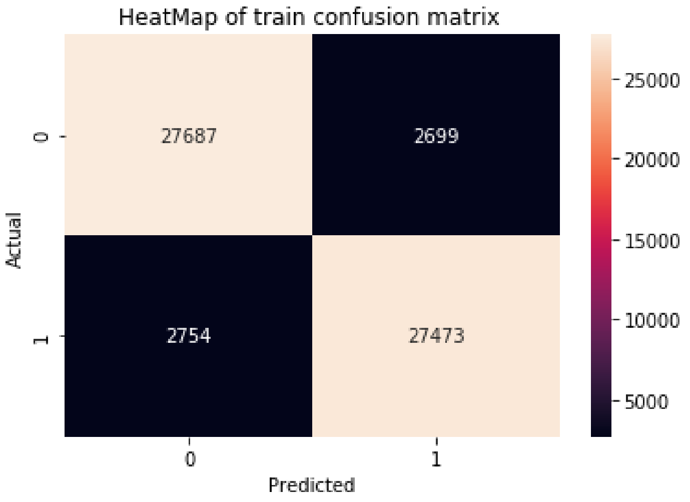


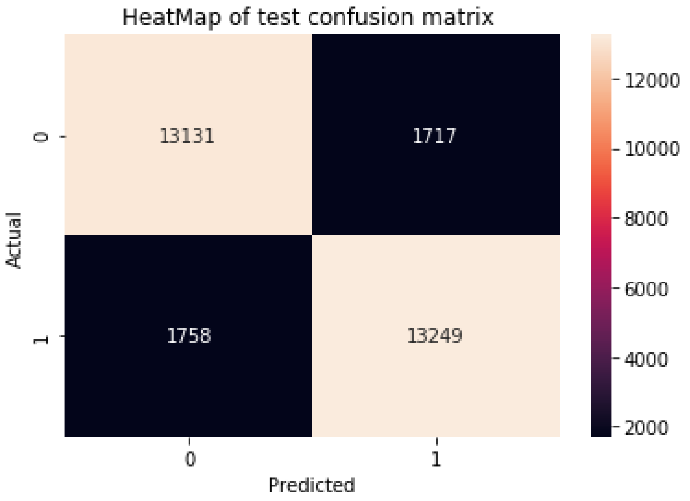


In [138]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(X_train_bow))
y=confusion_matrix(y_test, SGDclas.predict(X_test_bow))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[27687  2699]
 [ 2754 27473]]
Test confusion matrix
[[13131  1717]
 [ 1758 13249]]
```





In [142]:

```
SGD_Claf_Linear_Weights=SGDclas.coef_
cp5 = np.argsort(SGD_Claf_Linear_Weights[0])
l1=vectorizer.get_feature_names()
topnve=cp5[:10]
toppve=cp5[-10:]
```

Top 10 best features for positive class

In [143]:

```
pve=[]
for i in toppve:
    pve.append(l1[i])
print(pve)
```

```
['great', 'beat', 'wonderful', 'best', 'highly', 'loves', 'pleased', 'perfect', 'excellen
t', 'delicious']
```

Top 10 best features for negative class

In [144]:

```
l1=vectorizer.get_feature_names()
nve=[]
for i in topnve:
    nve.append(l1[i])
print(nve)
```

```
['worst', 'disappointing', 'terrible', 'awful', 'disappointed', 'unfortunately', 'disappo
intment', 'sorry', 'stale', 'bland']
```

BOW vectorized data which contains Review length feature added

In [145]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)


vectorizer = CountVectorizer()
vectorizer.fit(X_train)

X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)
```

In [146]:

```
from scipy import sparse
review_length_train=[]

for eachreview in X_train:
    x=len(eachreview)
    review_length_train.append(x)
```

In [147]:

```
review_length_test=[]

for eachreview in X_test:
    x=len(eachreview)
    review_length_test.append(x)
```

In [148]:

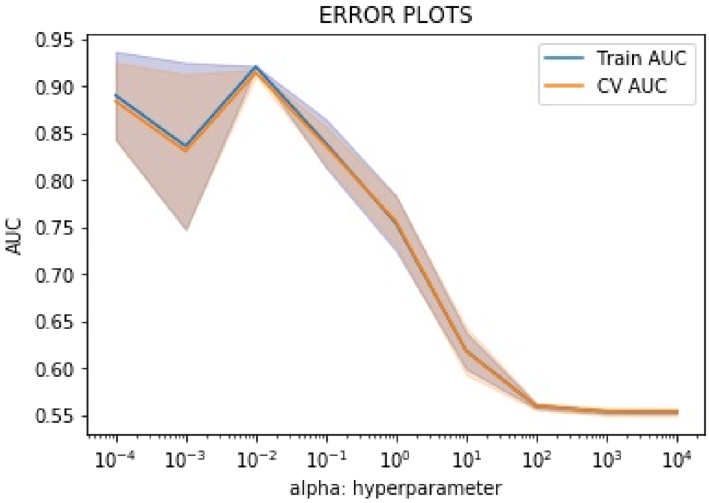
```
X_train_bow_f1=sparse.hstack((X_train_bow,np.array(review_length_train)[:,None]))
```

In [149]:

```
X_test_bow_f1=sparse.hstack((X_test_bow,np.array(review_length_test)[:,None]))
```

In [151]:

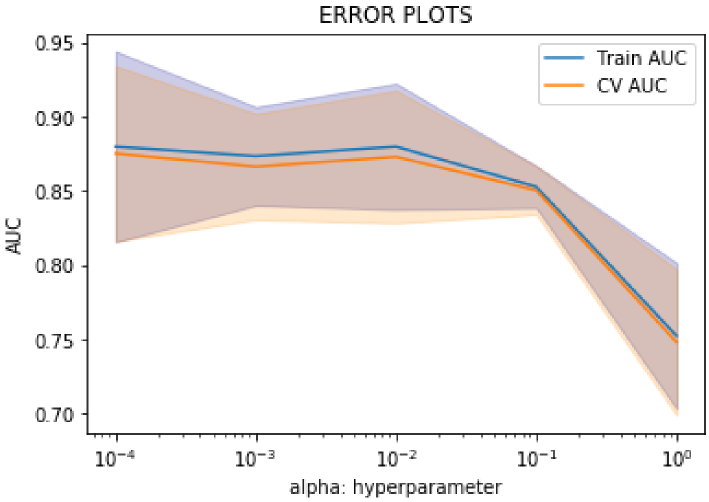
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_f1, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [153]:

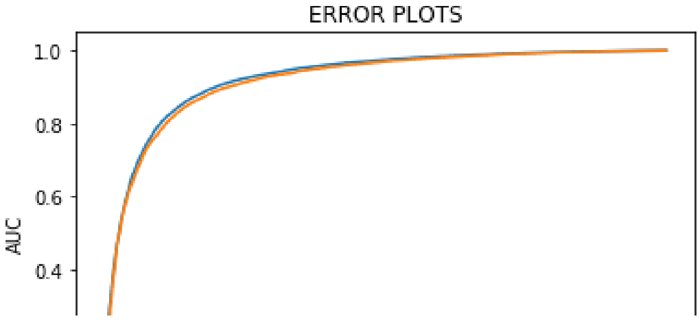
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_f1, y_train)
```

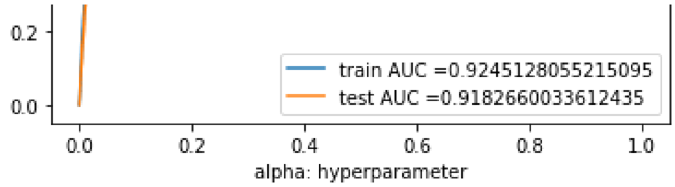
```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [155]:

```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l2',alpha=0.01,class_weight='balanced')
SGDclas.fit(X_train_bow_f1, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(X_train_bow_f1, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(X_train_bow_f1)[
:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(X_test_bow_f1)[:,1]
)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

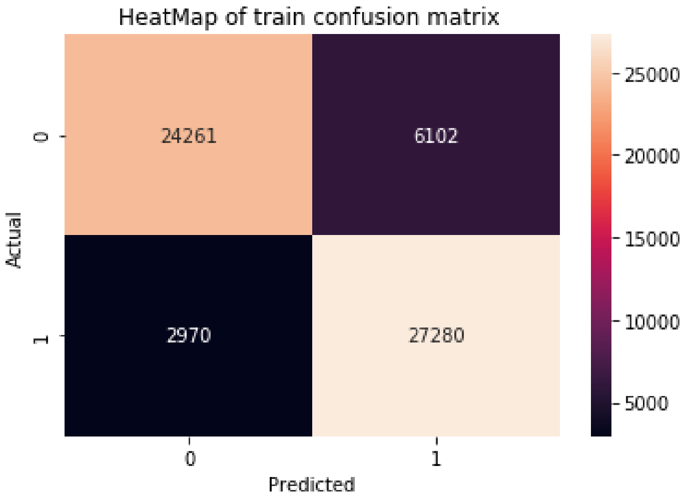


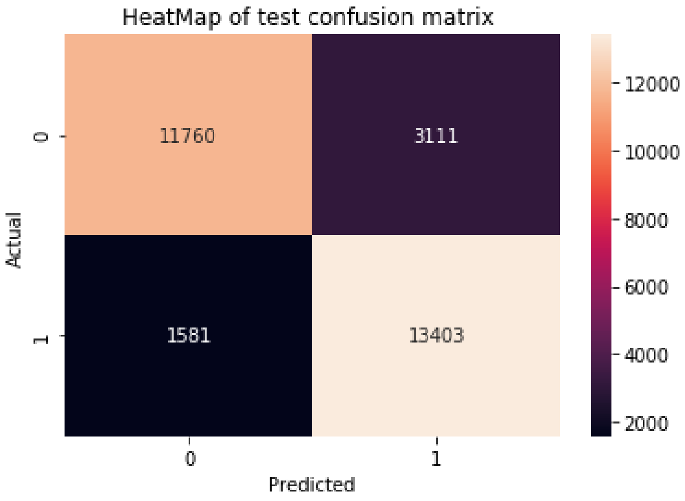


In [156]:

```
print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(X_train_bow_f1))
y=confusion_matrix(y_test, SGDclas.predict(X_test_bow_f1))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[24261  6102]
 [ 2970 27280]]
Test confusion matrix
[[11760  3111]
 [ 1581 13403]]
```





[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [157]:

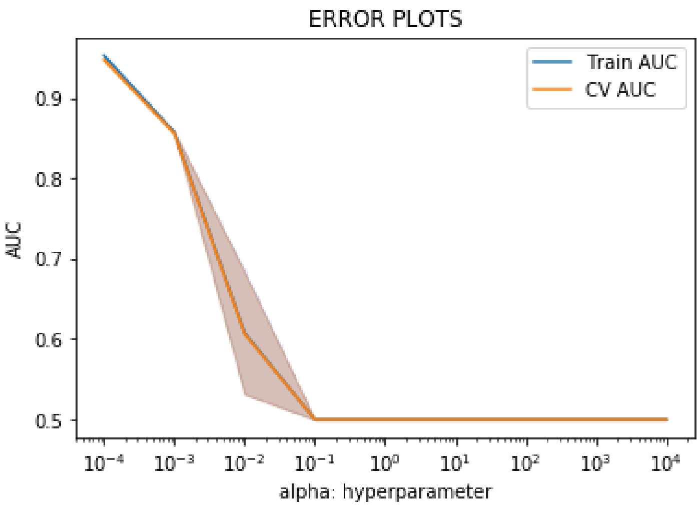
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)
```

In [159]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)


X_train_tfidf = tf_idf_vect.transform(X_train)
X_test_tfidf = tf_idf_vect.transform(X_test)
```

In [160]:

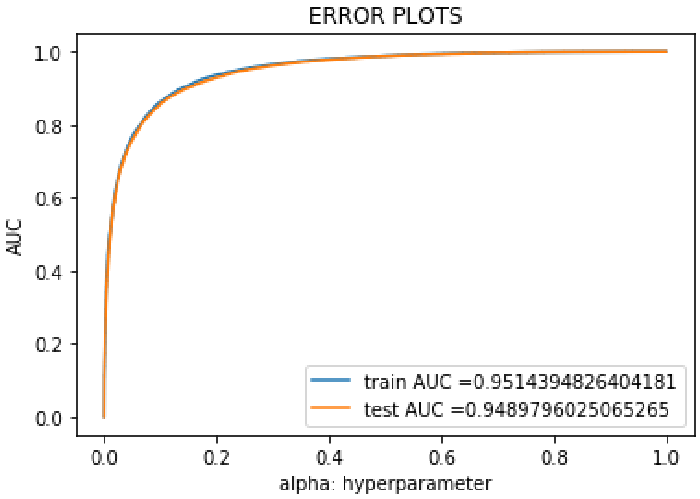
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [161]:

In [161]:

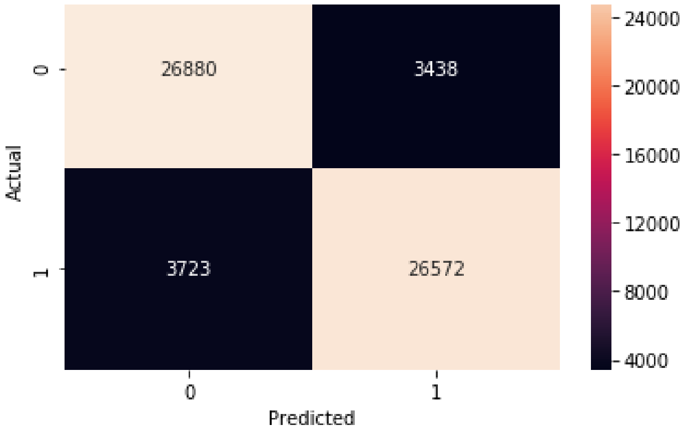
```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l1',alpha=0.0001,class_weight='balanced')
SGDclas.fit(X_train_tfidf, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(X_train_tfidf, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(X_train_tfidf)[:
,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

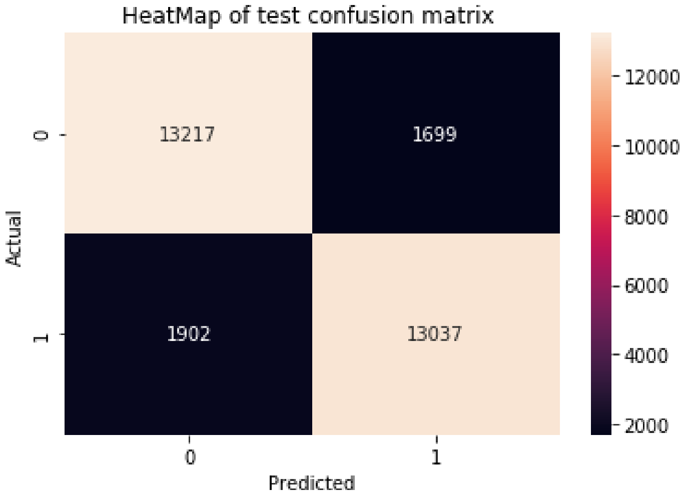


In [162]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(X_train_tfidf))
y=confusion_matrix(y_test, SGDclas.predict(X_test_tfidf))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[26880  3438]
 [ 3723 26572]]
Test confusion matrix
[[13217  1699]
 [ 1902 13037]]
```





In [165]:

```
W1=SGDclas.coef_
l1=tf_idf_vect.get_feature_names()
```

Top 10 Positive and Negative features

In [166]:

```
cp5=np.argsort(W1[0])
topnve=cp5[:10]
toppve=cp5[-10:]
print("top 10 positive")

pve=[]
for i in toppve:
    pve.append(l1[i])
print(pve)

print("\n")

print("top 10 negative")
print("\n")
nve=[]
for i in topnve:
    nve.append(l1[i])
print(nve)
```

```
top 10 positive
['wonderful', 'love', 'excellent', 'loves', 'good', 'perfect', 'not disappointed', 'best'
, 'delicious', 'great']


top 10 negative


['disappointed', 'worst', 'disappointing', 'not worth', 'awful', 'terrible', 'not good',
'not', 'unfortunately', 'horrible']
```

Applying Linear SVM on TFIDF vectorized data using Sgd classifier with l2 regularization

In [167]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)


X_train_tfidf = tf_idf_vect.transform(X_train)
X_test_tfidf = tf_idf_vect.transform(X_test)
```

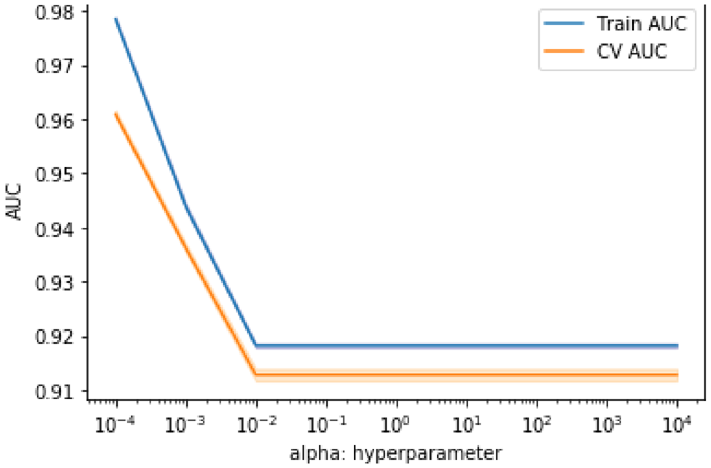
In [168]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',class_weight='balanced')
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],'penalty':['l1',
'l2']}
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

print(clf.best_estimator_)
```

```
SGDClassifier(alpha=0.0001, 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=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

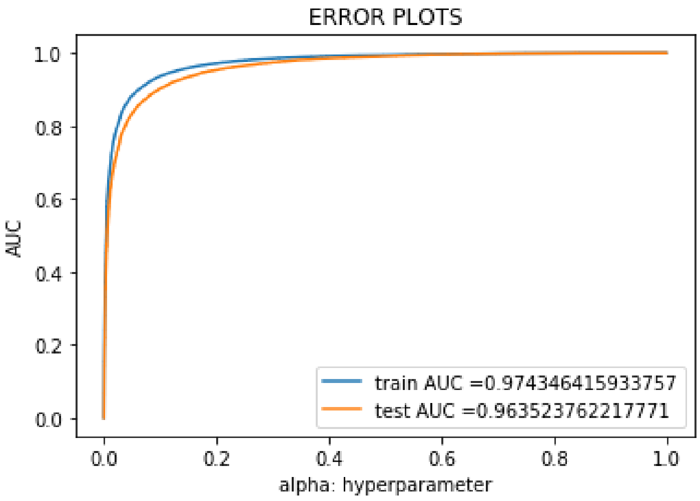
In [169]:

```
SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [170]:

```
SGDclas = SGDClassifier(loss='hinge',penalty='l2',alpha=0.0001,class_weight='balanced')
SGDclas.fit(X_train_tfidf, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(X_train_tfidf, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(X_train_tfidf)[:
,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

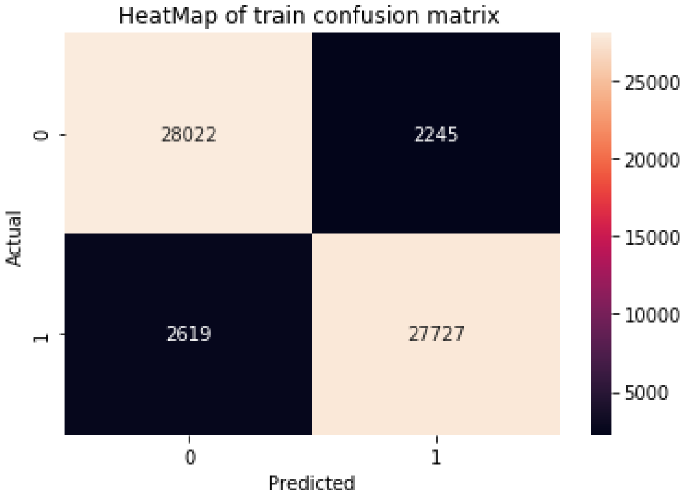


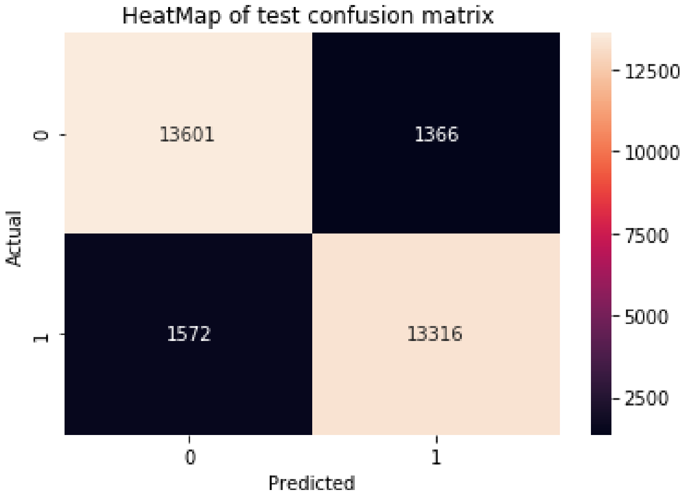
In [171]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(X_train_tfidf))
y=confusion_matrix(y_test, SGDclas.predict(X_test_tfidf))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()
```

```
ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[28022  2245]
 [ 2619 27727]]
Test confusion matrix
[[13601  1366]
 [ 1572 13316]]
```





In [174]:

```
W1=SGDclas.coef_
l1=tf_idf_vect.get_feature_names()
```

Top 10 positive and Negative features

In [175]:

```
cp5=np.argsort(W1[0])
topnve=cp5[:10]
toppve=cp5[-10:]
print("top 10 positive")

pve=[]
for i in toppve:
    pve.append(l1[i])
print(pve)

print("\n")
print("top 10 negative")
print("\n")
nve=[]
for i in topnve:
    nve.append(l1[i])
print(nve)
```

```
top 10 positive
['nice', 'favorite', 'loves', 'love', 'excellent', 'perfect', 'good', 'delicious', 'best'
, 'great']


top 10 negative


['not', 'disappointed', 'not good', 'worst', 'disappointing', 'terrible', 'not worth', 'a
wful', 'unfortunately', 'thought']
```

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

In [176]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)
```

In [177]:

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

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
```

In [178]:

```
w2v_words = list(w2v_model.wv.vocab)
```

In [179]:

```
sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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])



list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

print(type(list_of_sentance_test[0]))
sent_vectors_test = [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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%|████████████████████████████████████████████████████████████████████████████| 60613/6
0613 [01:57<00:00, 516.23it/s]
```

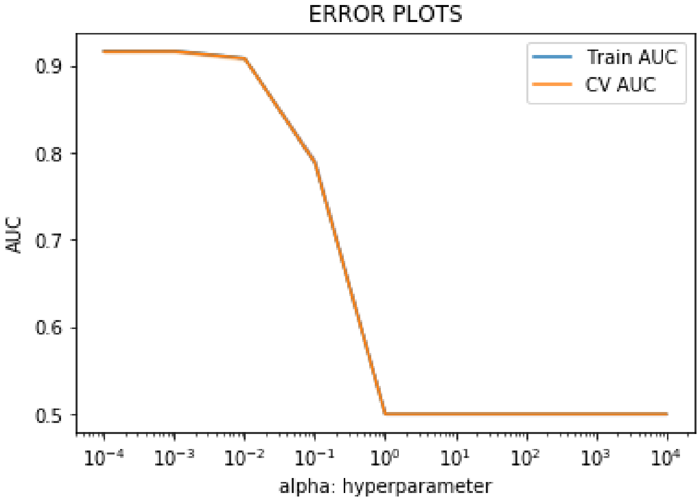
```
(60613, 50)
[-0.09804119  0.37421013 -0.14607843  0.09532788 -0.06557527  0.38974356
 -0.27933954  0.20111394 -0.00698583 -0.00730734  0.33420094  0.11953833
  0.1011827   0.57800431 -0.21254056  0.38111491 -0.08411088 -0.11940992
  0.256993    0.12807402  0.17914339  0.19586476 -0.35497102  0.11715417
 -0.35619367  0.79169725 -0.29410133  0.4603536   0.12747794  0.63280566
  0.22003534  0.05604896  0.03352399 -0.76776944  0.49276192 -0.18530212
  0.03107816  0.40579155 -0.26556654  0.21462101 -0.02496316 -0.3304775
  0.49041638  0.023297   -0.3913093   0.01901032 -0.2084887  -0.4967471
 -0.11018946  0.53206158]
<class 'list'>
```

```
100%|████████████████████████████████████████████████████████████████████████████| 29855/2
9855 [00:59<00:00, 501.42it/s]
```

```
(29855, 50)
[ 4.06159966e-01  5.80696413e-01 -2.13111852e-01  2.73840589e-02
  4.04459285e-01  2.88901717e-01 -1.53841402e-02 -1.11040703e-01
 -4.67601023e-01 -3.32367760e-01  3.10955263e-01 -2.07886669e-01
  8.99715751e-02  4.82112029e-01 -4.83541784e-02  2.47161302e-01
  3.20288421e-01 -3.19328247e-01  3.95265526e-01  1.97816207e-01
  5.49975935e-01  7.06280689e-01 -5.44816607e-01  5.34759774e-05
 -1.78329930e-01  5.85383980e-01  3.00981534e-01 -7.26424966e-02
  4.58518739e-01  7.40677274e-01 -5.27323406e-01  6.53464251e-01
  1.18966565e-01 -3.69698718e-01  4.76642530e-01  2.00650279e-01
  9.51867080e-02  3.45298324e-01 -6.07869771e-01 -1.96767746e-01
 -7.58328760e-02 -3.99876820e-01  7.41966608e-03  2.67261055e-01
  4.00867526e-02  1.05887173e-01 -2.29143947e-01 -5.42005496e-01
 -6.16412563e-01  2.50225034e-01]
```

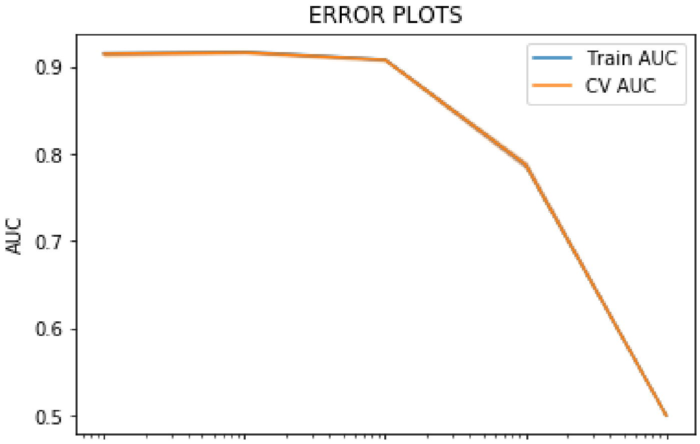
In [180]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



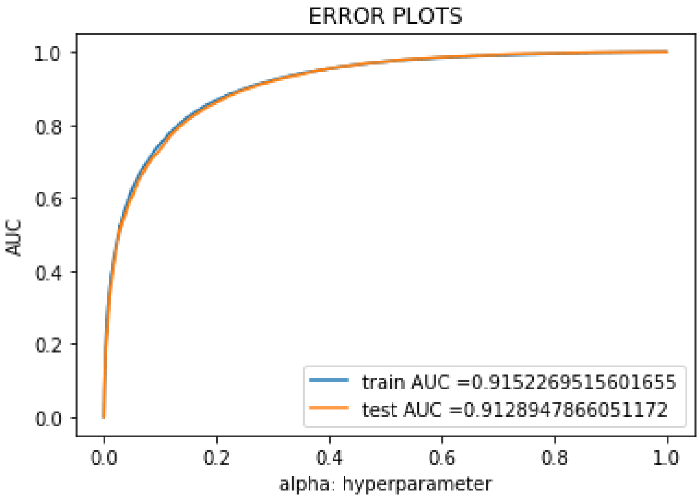
In [181]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [182]:

```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l1',alpha=0.0001,class_weight='balanced')
SGDclas.fit(sent_vectors_train, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(sent_vectors_tra
in)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(sent_vectors_test)[
:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

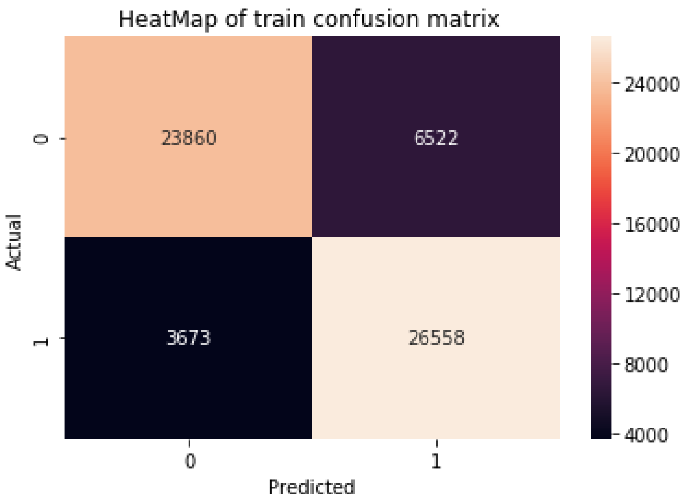


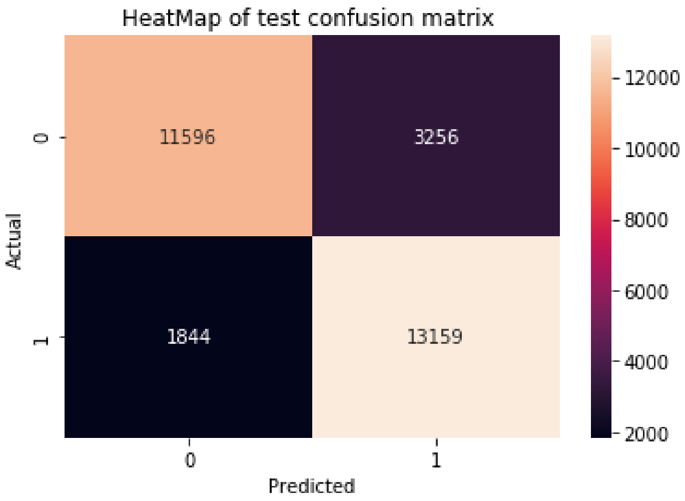
In [183]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(sent_vectors_train))
y=confusion_matrix(y_test, SGDclas.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[23860  6522]
 [ 3673 26558]]
Test confusion matrix
```

```
[[11596  3256]
 [ 1844 13159]]
```





Applying Linear SVM on AVG W2V using SGD Classifier with L2 Regularizer

In [31]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)

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

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
```

In [32]:

```
sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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])



list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

print(type(list_of_sentance_test[0]))
sent_vectors_test = [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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%|██████████████████████████████████████████████████████████████| 60613/6
0613 [01:41<00:00, 594.83it/s]
```

```
(60613, 50)
[-0.00262086  0.359042   -0.63058528  1.54398394  0.37075932  0.45010761
  0.01027398 -0.34577484 -1.06801568 -0.01129758  0.27783755 -0.36220155
  0.08359718 -0.00235397  0.37212813  0.03736874 -0.27174074 -1.03019477
  0.49876743 -0.41577828  1.26036787 -0.62976678 -0.21600645 -0.97295624
  0.34506776 -0.25555594 -0.19910531  0.64709185  0.23434117  0.64512842
 -0.23789696  0.10845431  0.28711488 -0.13707246 -0.00757376 -0.02930307
 -0.11319924  0.01351548 -0.27845243  0.24969434  0.17038149  0.07408482
  0.33459728 -0.62937077 -0.05425043 -0.64893758  0.6471648  -0.0219595
  0.40869577  0.00894916]
<class 'list'>
```

```
100%|██████████████████████████████████████████████████████████████| 29855/2
9855 [00:50<00:00, 586.92it/s]
```

```
(29855, 50)
[-0.15003498  0.25297482 -0.66765696  0.46424813 -0.01817711 -0.13538689
 -0.08982068  0.3811414  -0.16522329  0.28238909 -0.01263968 -0.04667559
  0.52086725  0.46069723  0.41226213  0.09257206 -0.51672832 -0.54146985
  0.19092167 -0.28230929  0.4172036  -0.00287608  0.61793233 -0.23167457
  0.12343246 -0.52897665  0.50382647  0.31970577 -0.17553031  0.28891472
 -0.07419128  0.03594353  0.14880423  0.09690938  0.38128618  0.0459354
  0.0291858   0.16491352 -0.44958265 -0.10948481 -0.09260431 -0.69183513
  0.16019227  0.05952085 -0.12809879 -0.2230675  -0.07294726  0.06193724
 -0.15773173 -0.02922601]
```

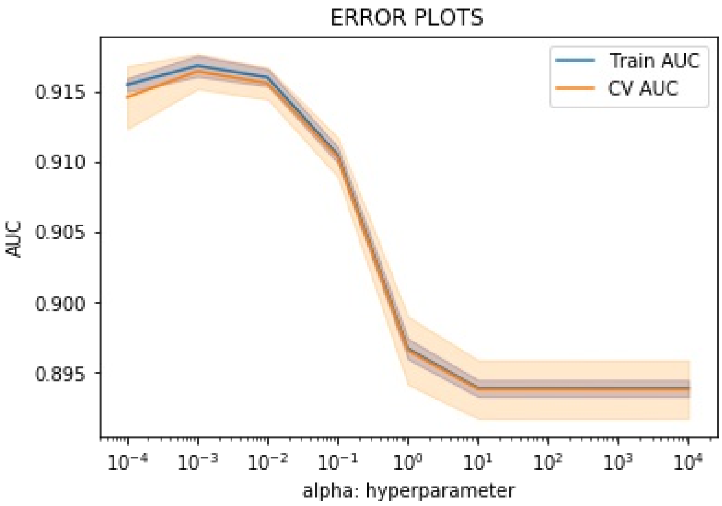
In [184]:

```
SGDclas = SGDClassifier(loss='hinge',class_weight='balanced')
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],'penalty':['l1',
'l2']}
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)
print(clf.best_estimator_)
```

```
SGDClassifier(alpha=0.001, 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=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

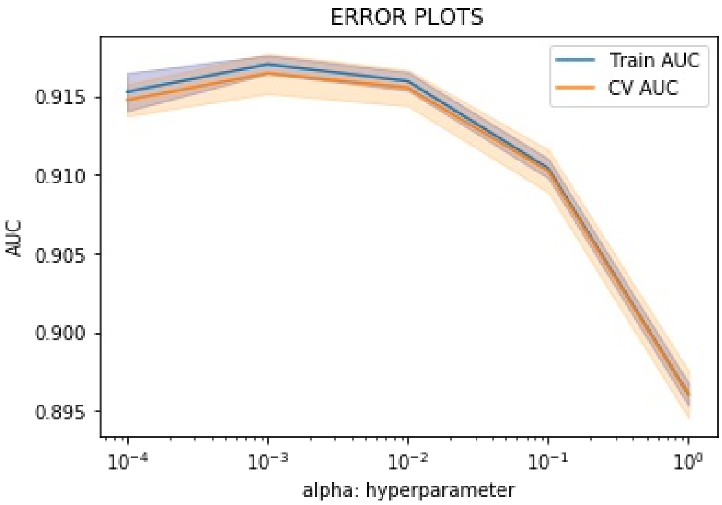
In [185]:

```
SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



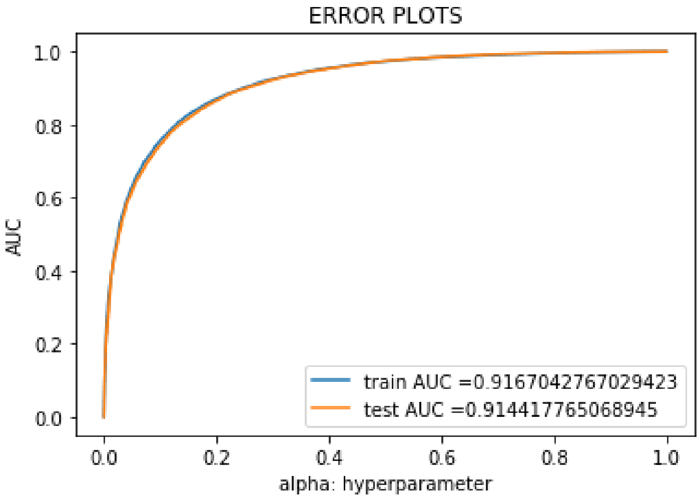
In [186]:

```
SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [187]:

```
SGDclas = SGDClassifier(loss='hinge',penalty='l2',alpha=0.001,class_weight='balanced')
SGDclas.fit(sent_vectors_train, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(sent_vectors_tra
in)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(sent_vectors_test)[
:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

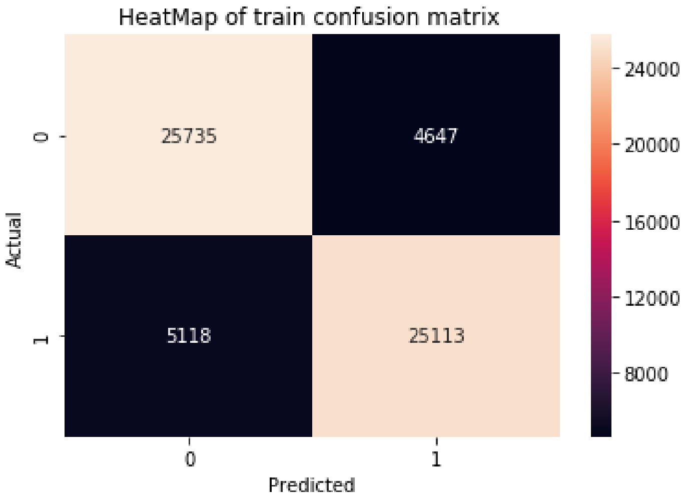


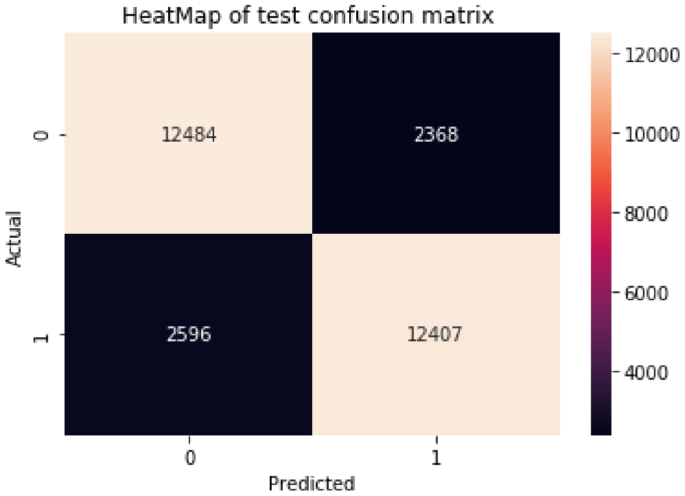
In [188]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(sent_vectors_train))
y=confusion_matrix(y_test, SGDclas.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)
```

```
ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[25735  4647]
 [ 5118 25113]]
Test confusion matrix
[[12484  2368]
 [ 2596 12407]]
```





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

In [189]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)
```

In [190]:

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

list_of_sentance_train=[]
```

```
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
```

In [191]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [192]:

```
w2v_words = list(w2v_model.wv.vocab)
```

In [193]:

```
tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 60613/
60613 [24:53<00:00, 40.58it/s]
```

In [194]:

```
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [195]:

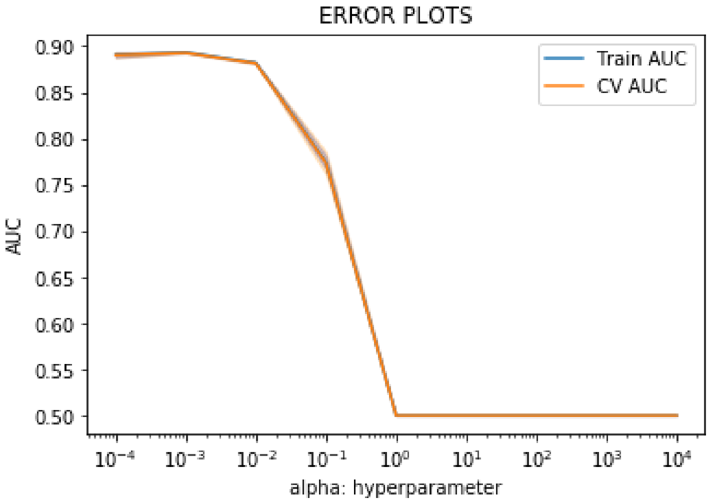
```
tfidf_sent_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 29855/
29855 [09:32<00:00, 52.11it/s]
```

In [196]:

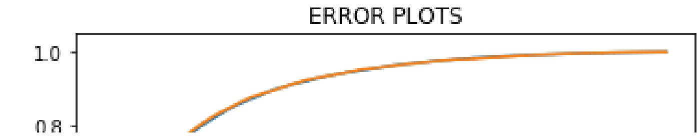
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
```

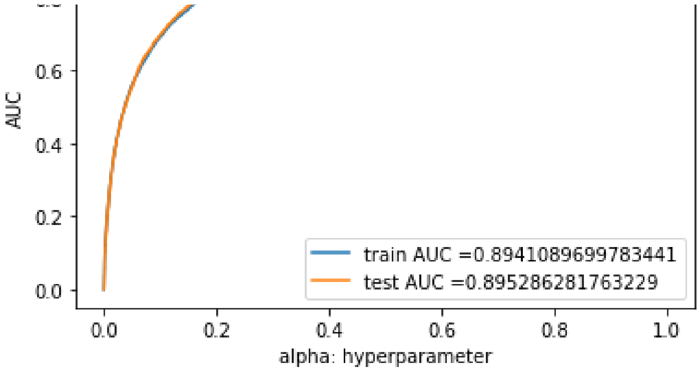
```
SGDclas = SGDClassifier(loss='hinge',penalty='l1',class_weight='balanced')
parameters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(tfidf_sent_vectors, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [199]:

```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l1',alpha=0.001,class_weight='balanced')
SGDclas.fit(tfidf_sent_vectors, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(tfidf_sent_vectors, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(tfidf_sent_vecto
rs)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(tfidf_sent_vectors_
test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

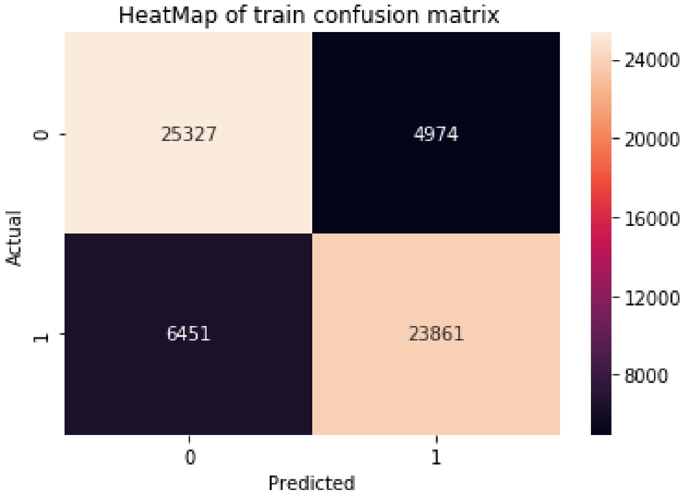


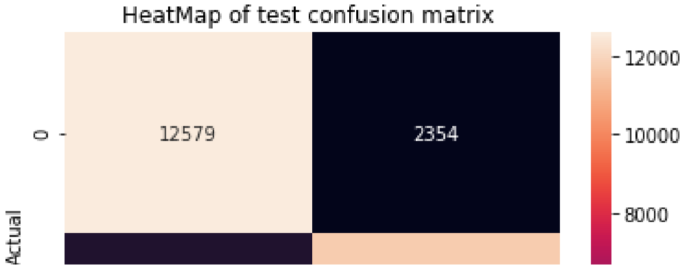


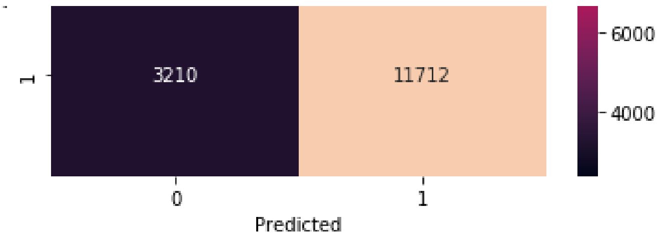
In [200]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(tfidf_sent_vectors))
y=confusion_matrix(y_test, SGDclas.predict(tfidf_sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[25327  4974]
 [ 6451 23861]]
Test confusion matrix
[[12579  2354]
 [ 3210 11712]]
```







Applying Linear SVM on TFIDF W2V using SGD Classifier with L2 Regularizer

In [40]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_90, final_new_f
rame_90['Score'], test_size=0.33)

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

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

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

w2v_words = list(w2v_model.wv.vocab)
```

In [41]:

```
tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████| 60613/
60613 [22:51<00:00, 44.19it/s]
```

In [42]:

```
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [43]:

```
tfidf_sent_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_test.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████| 29855/
29855 [10:44<00:00, 46.32it/s]
```

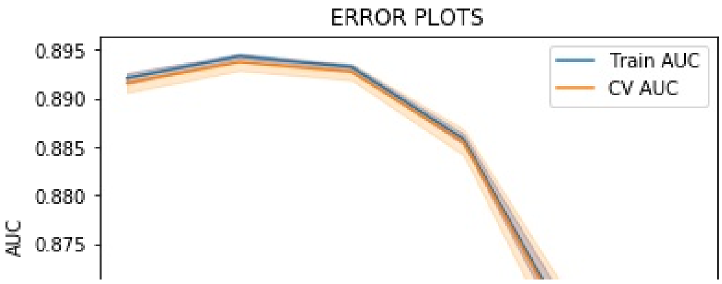
In [201]:

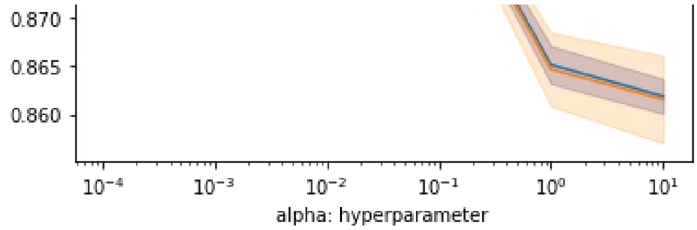
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

SGDclas = SGDClassifier(loss='hinge',class_weight='balanced')
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],'penalty':['l1',
'l2']}
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(tfidf_sent_vectors, y_train)
print(clf.best_estimator_)
```

```
SGDClassifier(alpha=0.001, 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=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [203]:

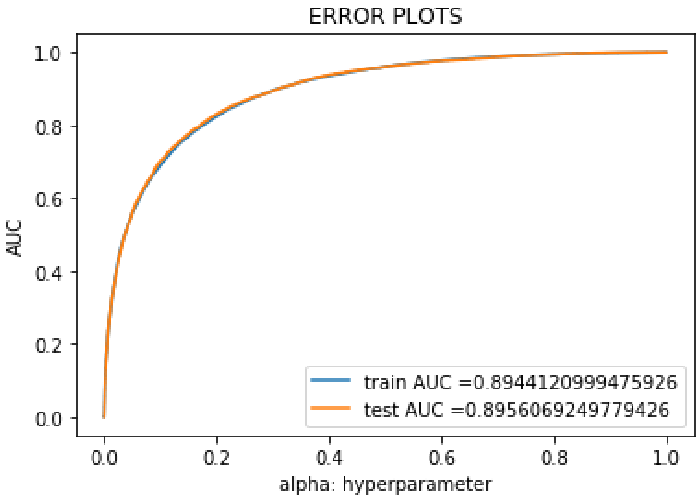
```
SGDclas = SGDClassifier(loss='hinge',penalty='l2',class_weight='balanced')
parameters = [{'alpha':[0.0001,0.001,0.01, 0.1, 1,10]}]
clf = GridSearchCV(SGDclas, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(tfidf_sent_vectors, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001,0.001,0.01, 0.1, 1,10]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```





In [204]:

```
from sklearn.calibration import CalibratedClassifierCV
SGDclas = SGDClassifier(loss='hinge',penalty='l2',alpha=0.001,class_weight='balanced')
SGDclas.fit(tfidf_sent_vectors, y_train)
model = CalibratedClassifierCV(SGDclas,cv='prefit')
model.fit(tfidf_sent_vectors, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,model.predict_proba(tfidf_sent_vecto
rs)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,model.predict_proba(tfidf_sent_vectors_
test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

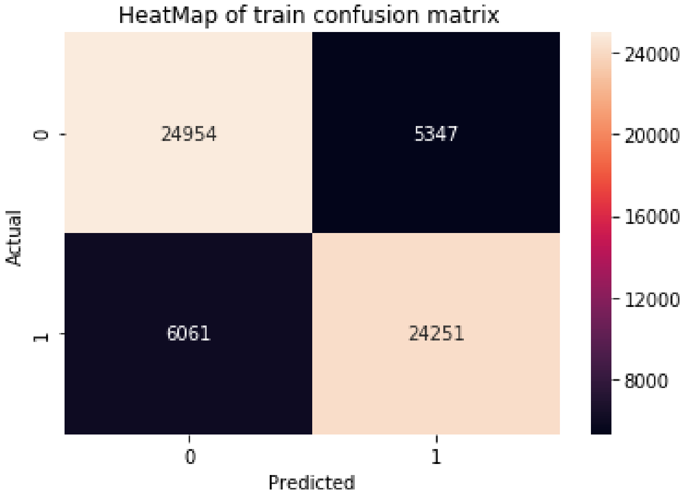


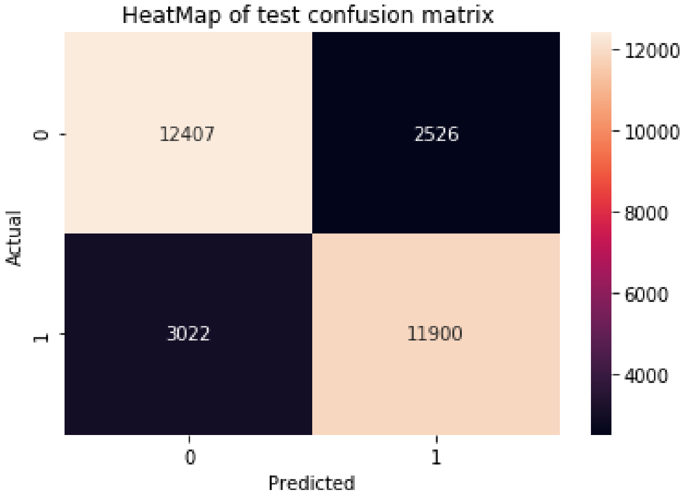
In [205]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, SGDclas.predict(tfidf_sent_vectors))
y=confusion_matrix(y_test, SGDclas.predict(tfidf_sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
```

```
plt.show()
```

```
Train confusion matrix
[[24954  5347]
 [ 6061 24251]]
Test confusion matrix
[[12407  2526]
 [ 3022 11900]]
```





[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW,SET 1

In [206]:

```
print(len(preprocessed_reviews_90))
print(len(preprocessed_reviews_20))
print(type(final_new_frame_90))
print(type(final_new_frame_20))
print(final_new_frame_90.shape)
print(final_new_frame_20.shape)
```

```
90468
20000
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
(90468, 10)
(20000, 10)
```

In [ ]:

```
##min_df = 10, max_features = 500
```

In [207]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_20, final_new_f
rame_20['Score'], test_size=0.33)
```

In [208]:

```
vectorizer = CountVectorizer(min_df=10,max_features=500)
vectorizer.fit(X_train)

X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)
```

In [209]:

```
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
```

```
After vectorizations
(13400, 500) (13400,)
(6600, 500) (6600,)
```

In [211]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf')
parameters ={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],
             'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)
clf.best_estimator_
```

Out[211]:

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [212]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf',class_weight='balanced')
parameters ={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],
             'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)
clf.best_estimator_
```

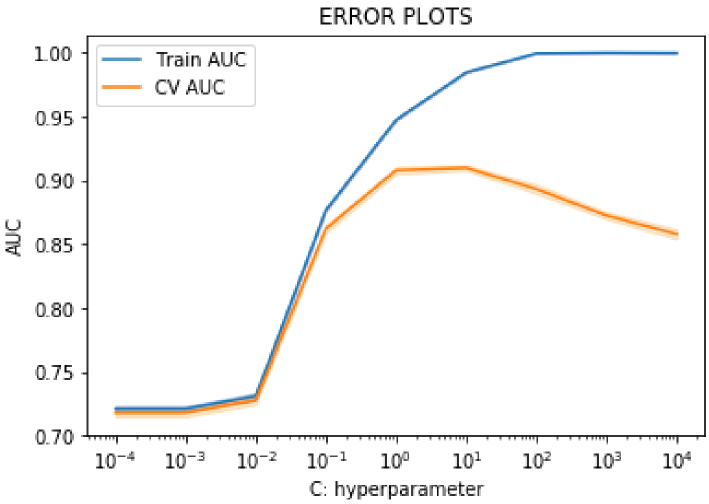
Out[212]:

```
SVC(C=10, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [213]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf',gamma=0.01,class_weight='balanced')
```

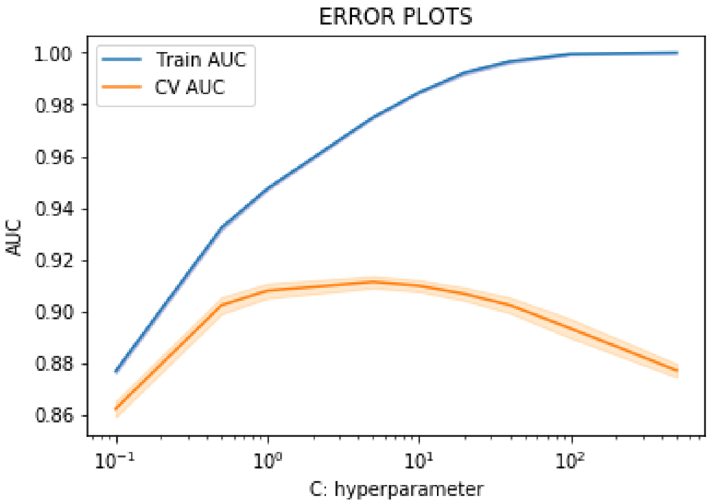
```
parameters ={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [214]:

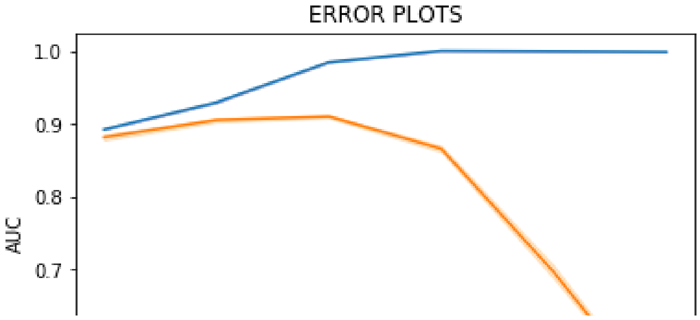
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf',gamma=0.01,class_weight='balanced')
parameters ={'C': [0.1,0.5,1,5,10,20,40,100,500]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1,0.5,1,5,10,20,40,100,500]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
```

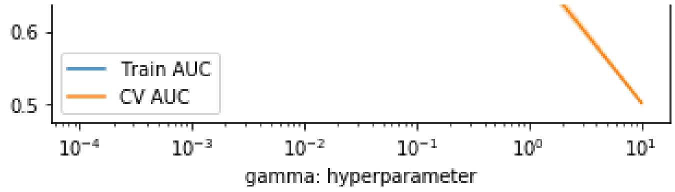
```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [215]:

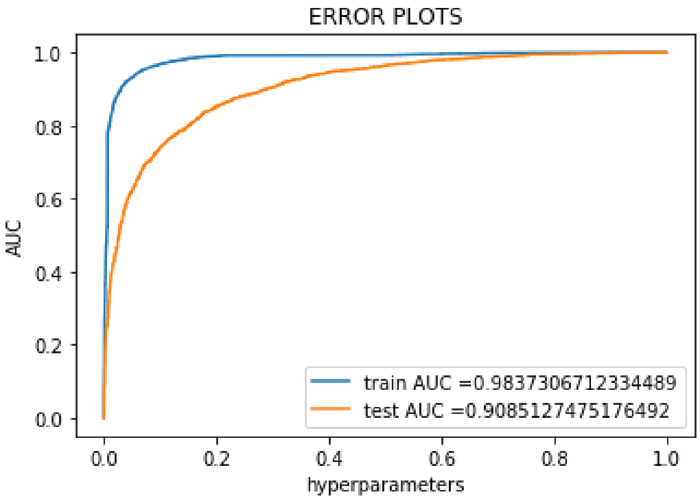
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf',C=10,class_weight='balanced')
parameters ={'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("gamma: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```





In [216]:

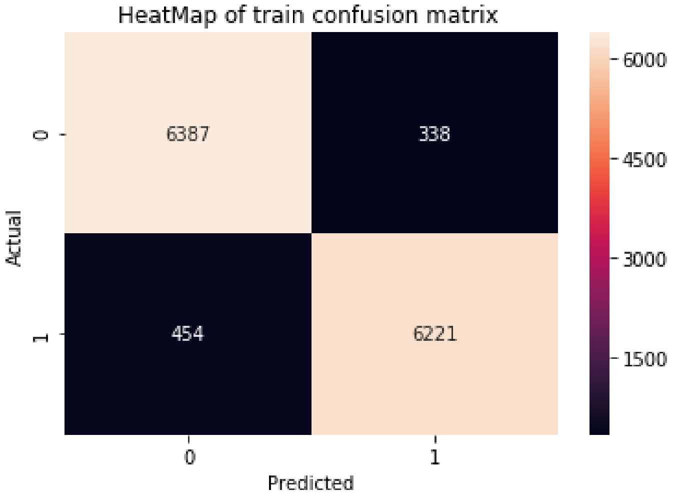
```
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(C=10,kernel='rbf',gamma=0.01,probability=True,class_weight='balanced')

svc.fit(X_train_bow, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,svc.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,svc.predict_proba(X_test_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("hyperparameters")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

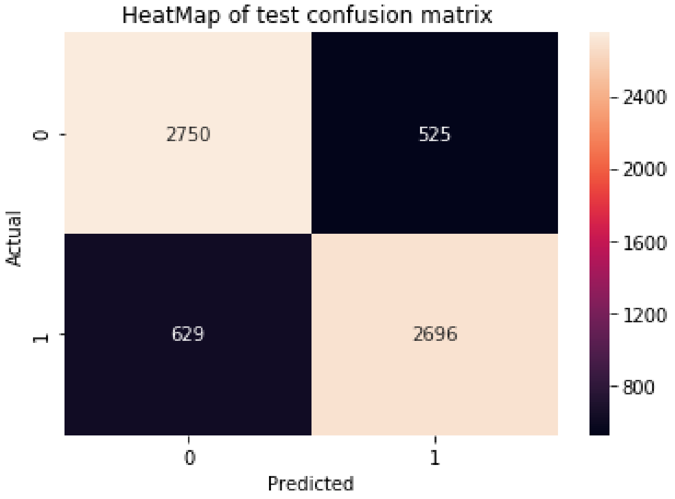


In [217]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

print("Train confusion matrix")
x=confusion_matrix(y_train, svc.predict(X_train_bow))
y=confusion_matrix(y_test, svc.predict(X_test_bow))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

```
Train confusion matrix
[[6387  338]
 [ 454 6221]]
Test confusion matrix
[[2750  525]
 [ 629 2696]]
```





[5.2.2] Applying RBF SVM on TFIDF,SET 2

In [218]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
tf_idf_vect.fit(X_train)


X_train_tfidf = tf_idf_vect.transform(X_train)
X_test_tfidf = tf_idf_vect.transform(X_test)
```

In [48]:

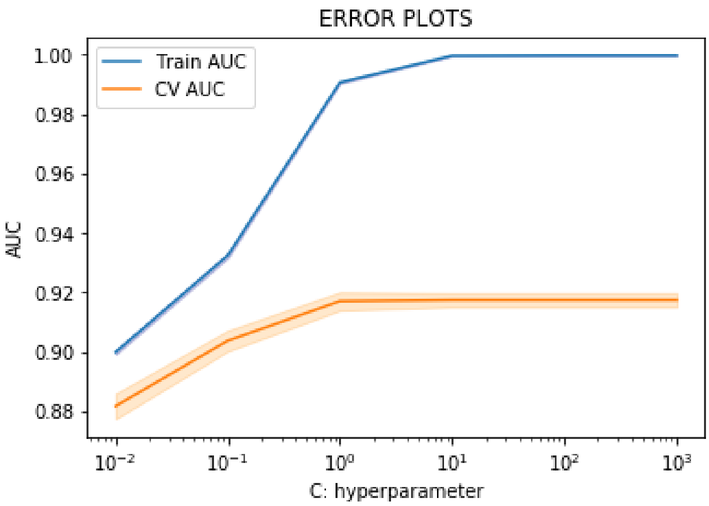
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf')
parameters ={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],
             'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)
print(clf.best_estimator_)
```

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
    probability=False, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

In [219]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(kernel='rbf',gamma=1,class_weight='balanced')
parameters ={'C': [0.01, 0.1, 1, 10, 100,1000]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(X_train_tfidf, y_train)
print(clf1.best_estimator_)


train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.01, 0.1, 1, 10, 100,1000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

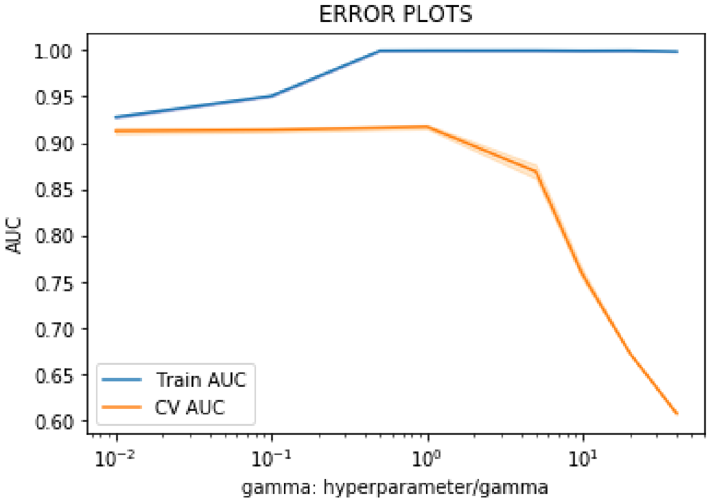
```
SVC(C=10, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
    probability=False, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```



In [220]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(C=10,kernel='rbf',class_weight='balanced')
parameters ={'gamma': [0.01,0.1, 0.5, 1,5,10,20,40]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(X_train_tfidf, y_train)
```

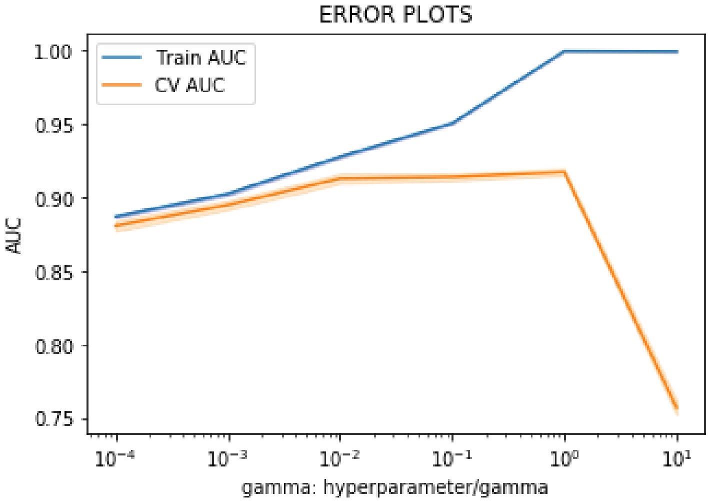
```
train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K =[0.01,0.1, 0.5, 1,5,10,20,40]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("gamma: hyperparameter/gamma")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [223]:

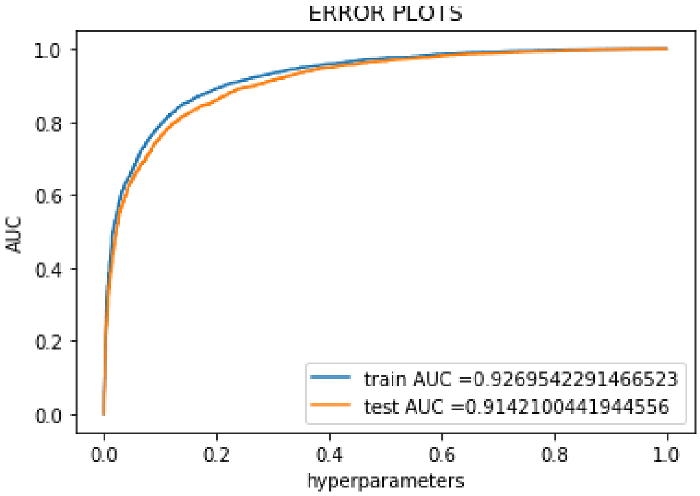
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(C=10,kernel='rbf',class_weight='balanced')
parameters ={'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(X_train_tfidf, y_train)



train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1, 10]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
```

```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("gamma: hyperparameter/gamma")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

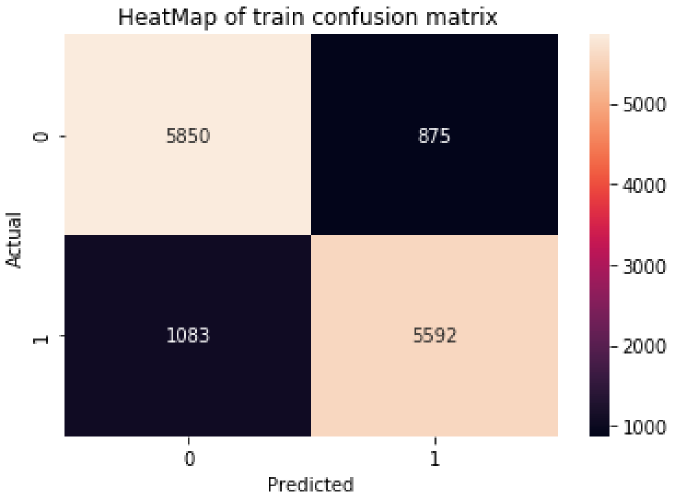


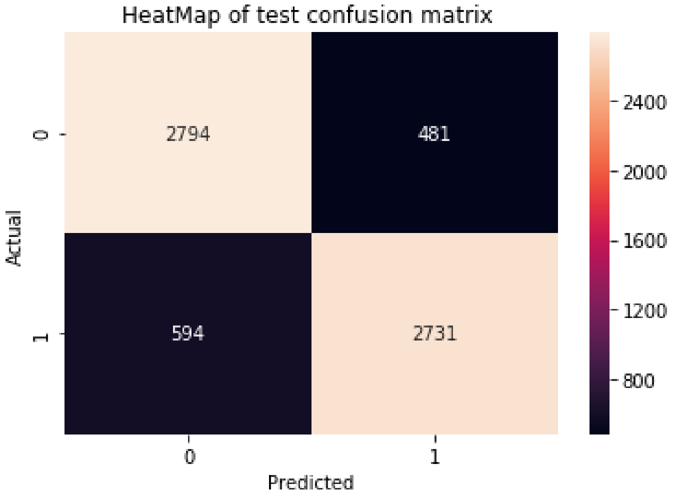
In [225]:

```
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import seaborn as sn

svc = SVC(C=10,kernel='rbf',gamma=0.01,probability=True,class_weight='balanced')

svc.fit(X_train_tfidf, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,svc.predict_proba(X_train_tfidf)[:,1
])
test_fpr, test_tpr, thresholds = roc_curve(y_test,svc.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("hyperparameters")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("Train confusion matrix")
x=confusion_matrix(y_train, svc.predict(X_train_tfidf))
y=confusion_matrix(y_test, svc.predict(X_test_tfidf))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```



```
Train confusion matrix
[[5850  875]
 [1083 5592]]
Test confusion matrix
[[2794  481]
 [ 594 2731]]
```





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

In [226]:

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

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
```

In [227]:

```
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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])



list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

print(type(list_of_sentance_test[0]))
sent_vectors_test = [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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%|████████████████████████████████████████████████████████████████████████| 13400/1
3400 [00:15<00:00, 869.56it/s]
```

```
(13400, 50)
[ 0.46012911  0.19255838  0.04813114  0.22064492 -0.32963636 -0.4048187
 -0.23335441 -0.26880824  0.12150427  0.52073591 -0.13284586 -0.06519413
  0.54288935  0.30599423  0.41188939 -0.01076978 -0.37208066  0.00528922
  0.33973396  0.3220866   0.66009748 -0.04701975  0.44161352 -0.14755402
 -0.64723817  0.55797487  0.0178396   0.33487631  0.04252902  0.65301995
  0.05347333  0.32503007  0.42857987 -0.40195519  0.56236081 -0.03325459
  0.32757081  0.71509955 -0.40117225  0.16830315 -0.18767803 -0.32371938
  0.34101899  0.07868763 -0.43244138  0.28385369 -0.17657967 -0.61594394
 -0.62812399  0.37568817]
<class 'list'>
```

```
100%|██████████████████████████████████████████████████████████████████████████| 6600/
6600 [00:08<00:00, 792.28it/s]
```

```
(6600, 50)
[-0.03372158  0.07793589 -0.00603694  0.46922456 -0.4047623  -0.37914939
 -0.27175687 -0.16005872  0.20559565  0.11687505 -0.3384141   0.09616493
  0.50650826  0.43748719  0.29179555 -0.09766582 -0.38132629 -0.28959625
  0.45494629 -0.01760771  0.69570625  0.0166733   0.55072168 -0.21910343
 -0.42281424  0.36001055 -0.01801671  0.18612201 -0.02980637  0.87564855
 -0.01418455 -0.16663717  0.435072   -0.18032367  0.55105593  0.19680477
  0.18952497  0.52723686 -0.2274026   0.11417634 -0.26982032 -0.49343797
 -0.13781327  0.10590316 -0.89419442  0.21816756 -0.43635595 -0.16011797
 -0.41403612  0.45672026]
```

In [228]:

```
print(type(sent_vectors_train))
print(type(sent_vectors_train))
print(sent_vectors_train.shape)
print(type(sent_vectors_test))
print(sent_vectors_test.shape)
```

```
<class 'numpy.ndarray'>
(13400, 50)
<class 'numpy.ndarray'>
(6600, 50)
```

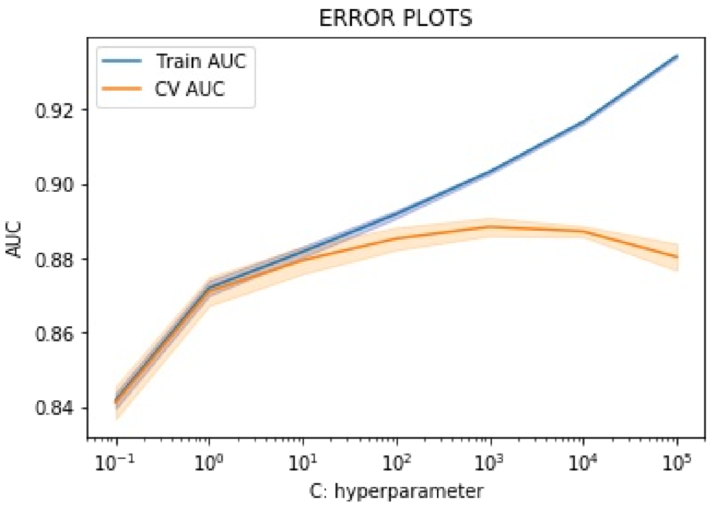
In [230]:

```
svc = SVC(kernel='rbf',class_weight='balanced')
parameters ={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],
             'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)
print(clf.best_estimator_)
```

```
SVC(C=10, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

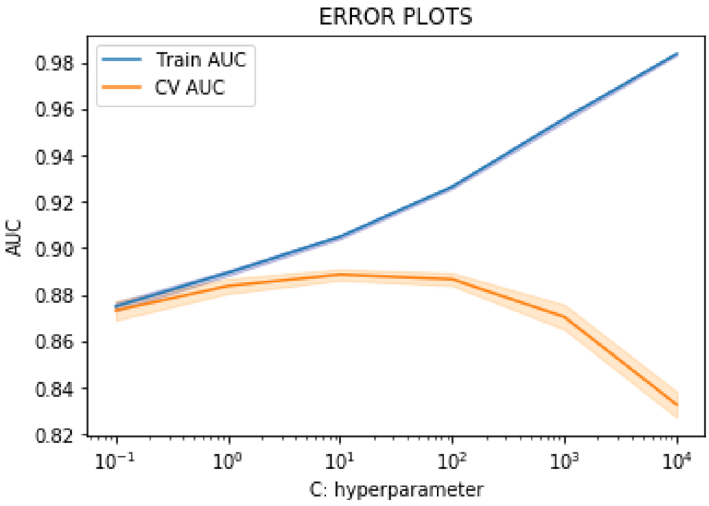
In [229]:

```
svc = SVC(kernel='rbf',gamma=0.01,class_weight='balanced')
parameters ={'C': [0.1, 1, 10, 100,1000,10000,100000]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(sent_vectors_train, y_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.1, 1, 10, 100,1000,10000,100000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [231]:

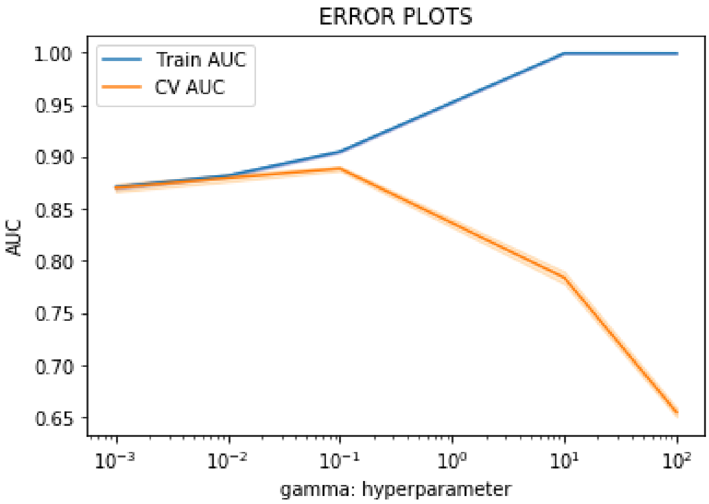
```
svc = SVC(kernel='rbf',gamma=0.1,class_weight='balanced')
parameters ={'C': [0.1,1,10,100,1000,10000]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(sent_vectors_train, y_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.1,1,10,100,1000,10000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [232]:

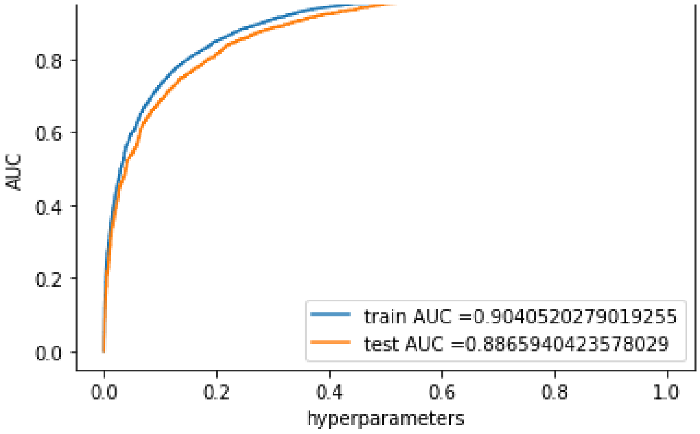
```
svc = SVC(C=10,kernel='rbf',class_weight='balanced')
parameters ={'gamma': [0.001,0.01,0.1,10,100]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(sent_vectors_train, y_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.001,0.01,0.1,10,100]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
```

```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("gamma: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

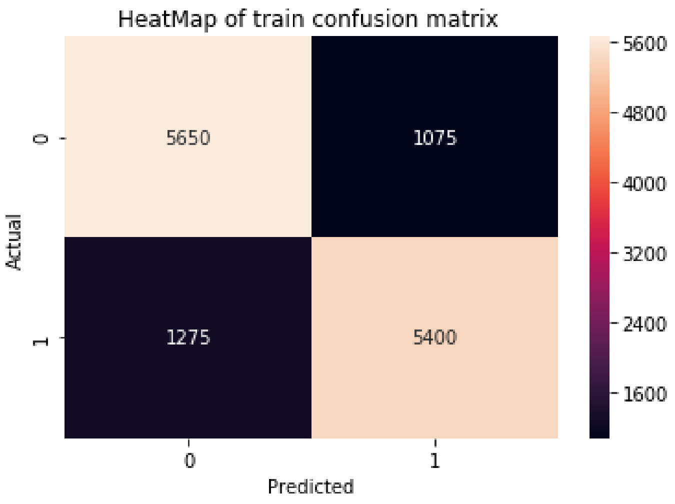


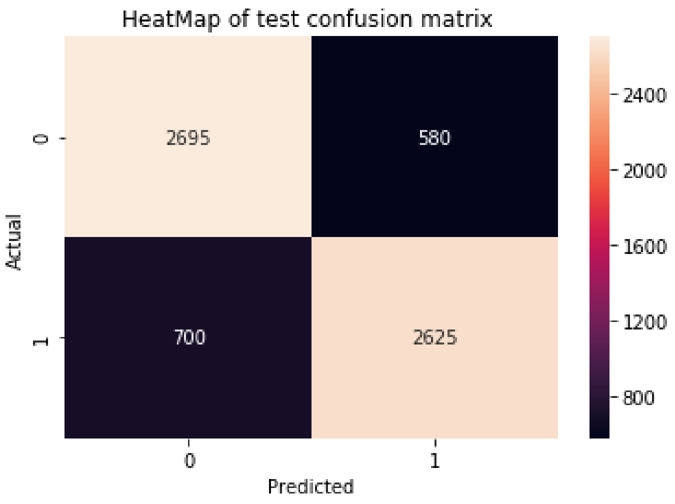
In [233]:

```
svc = SVC(C=10,kernel='rbf',gamma=0.1,probability=True,class_weight='balanced')

svc.fit(sent_vectors_train, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,svc.predict_proba(sent_vectors_train
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,svc.predict_proba(sent_vectors_test)[:,
1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("hyperparameters")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()



print("Train confusion matrix")
x=confusion_matrix(y_train, svc.predict(sent_vectors_train))
y=confusion_matrix(y_test, svc.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```



```
Train confusion matrix
[[5650 1075]
 [1275 5400]]
Test confusion matrix
[[2695  580]
 [ 700 2625]]
```





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

In [235]:

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

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

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
```

In [236]:

In [236]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [237]:

```
w2v_words = list(w2v_model.wv.vocab)

tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|█████████████████████████████████████████████| 13400/1
3400 [02:06<00:00, 106.20it/s]
```

In [238]:

```
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [239]:

```
tfidf_sent_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_test.append(sent_vec)
    row += 1
```

```
100%|███████████████████████████████████████████████| 6600
/6600 [01:19<00:00, 82.60it/s]
```

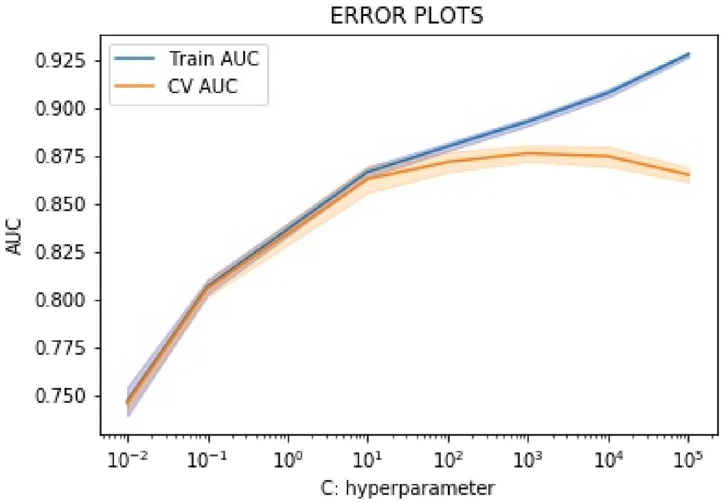
In [240]:

```
svc = SVC(kernel='rbf',class_weight='balanced')
parameters ={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000],
            'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}

clf = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(tfidf_sent_vectors, y_train)
print(clf.best_estimator_)
```

```
SVC(C=1000, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [241]:

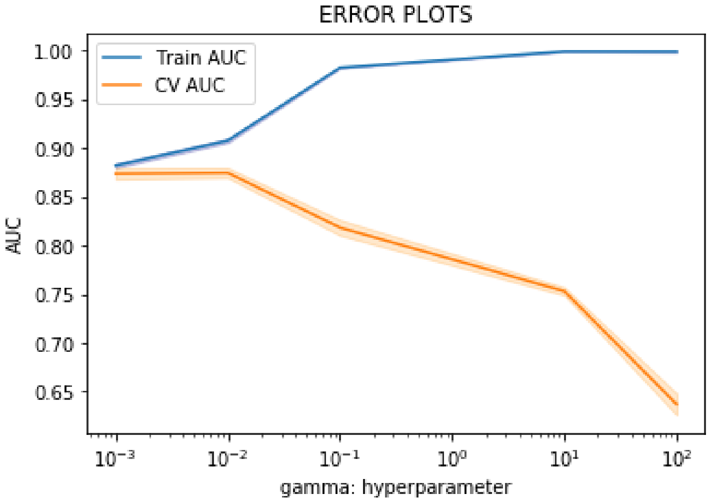
```
svc = SVC(kernel='rbf',gamma=0.01,class_weight='balanced')
parameters ={'C': [0.01,0.1,10,100,1000,10000,100000]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(tfidf_sent_vectors, y_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.01,0.1,10,100,1000,10000,100000]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [243]:

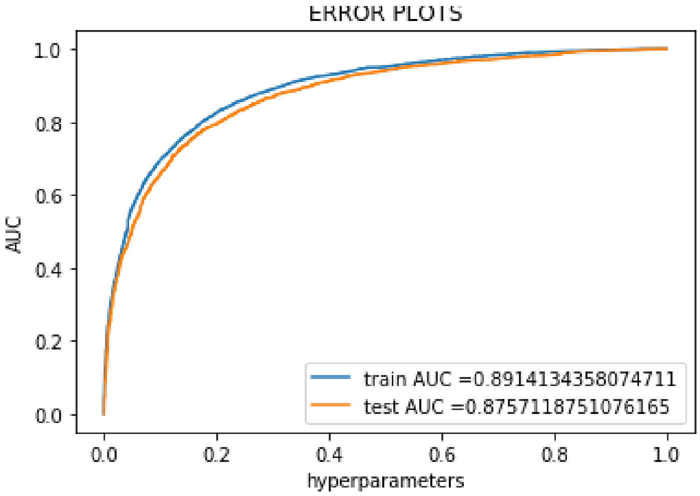
```
svc = SVC(C=10000,kernel='rbf',class_weight='balanced')
parameters ={'gamma': [0.001,0.01,0.1,10,100]}

clf1 = GridSearchCV(svc, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1.fit(tfidf_sent_vectors, y_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

K = [0.001,0.01,0.1,10,100]


plt.semilogx(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

plt.semilogx(K, cv_auc, label='CV AUC')
```

```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkor
ange')
plt.legend()
plt.xlabel("gamma: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

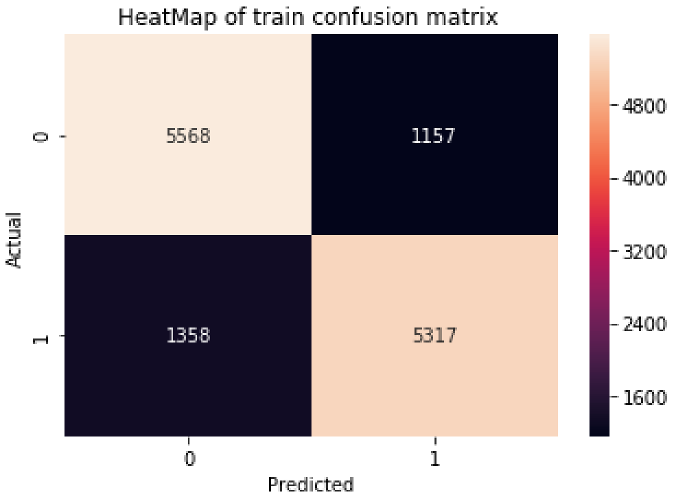


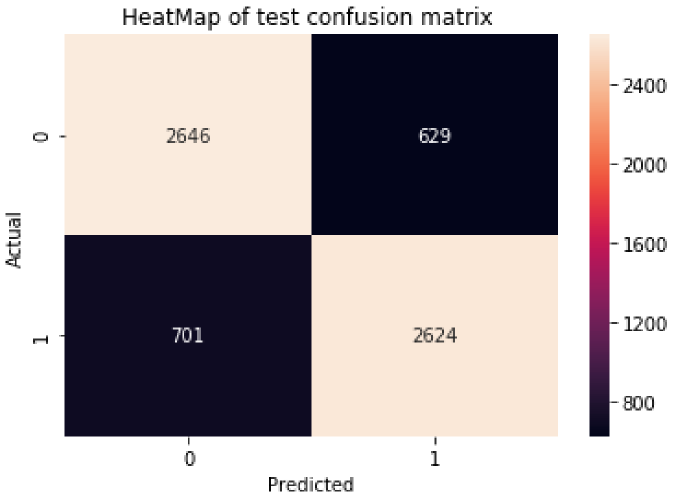
In [244]:

```
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

svc = SVC(C=1000,kernel='rbf',gamma=0.01,probability=True,class_weight='balanced')

svc.fit(tfidf_sent_vectors, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,svc.predict_proba(tfidf_sent_vectors
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,svc.predict_proba(tfidf_sent_vectors_te
st)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("hyperparameters")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()


print("Train confusion matrix")
x=confusion_matrix(y_train, svc.predict(tfidf_sent_vectors))
y=confusion_matrix(y_test, svc.predict(tfidf_sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)

ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()

ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```



```
Train confusion matrix
[[5568 1157]
 [1358 5317]]
Test confusion matrix
[[2646  629]
 [ 701 2624]]
```





[6] Conclusions

In [246]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Vectorizer","Hyper Parameter(alpha-α)","Regularizer","AUC"]
x.add_row(["Linear-kernel SVM","BOW","0.0001","L-1","0.9249"])
x.add_row(["Linear-kernel SVM","BOW","0.001","L-2","0.9445"])
x.add_row(["Linear-kernel SVM","TF-IDF","0.0001","L-1","0.9489"])
x.add_row(["Linear-kernel SVM","TF-IDF","0.0001","L-2","0.9635"])
```

```
x.add_row(["Linear-kernel SVM","AVG W2V","0.0001","L-1","0.9128"])
x.add_row(["Linear-kernel SVM","AVG W2V","0.001","L-2","0.9144"])
x.add_row(["Linear-kernel SVM","TFIDF W2V","0.001","L-1","0.8944"])
x.add_row(["Linear-kernel SVM","TFIDF W2V","0.001","L-2","0.8956"])
print("\n")
print(x)
print("\n")
print("Feature engineered output after adding review length to BOW vectorized data:=")
print("\n")
y = PrettyTable()
y.field_names = ["Model","Vectorizer","Hyper Parameter(alpha-α)","Regularizer","AUC"]
y.add_row(["Linear SVM","BOW","0.0001","L-1","0.9038"])
y.add_row(["Linear SVM","BOW","0.0001","L-2","0.9182"])
print(y)
print("\n")
print("RBF Kernel SVM")
print("\n")
z=PrettyTable()
z.field_names = ["Model","Vectorizer","Hyper Parameter(C)","Hyper Parameter(gamma-γ)","A
UC"]
z.add_row(["RBF-Kernel SVM","BOW","10","0.01","0.9085"])
z.add_row(["RBF-Kernel SVM","TF-IDF","10","0.01","0.9142"])
z.add_row(["RBF-Kernel SVM","AVG W2V","10","0.1","0.8865"])
z.add_row(["RBF-Kernel SVM","TFIDF W2V","1000","0.01","0.8757"])
print(z)
```

```
+-------------------+------------+--------------------------+--------------+--------+
|       Model       | Vectorizer | Hyper Parameter(alpha-α) | Regularizer  |  AUC   |
+-------------------+------------+--------------------------+--------------+--------+
| Linear-kernel SVM |    BOW     |          0.0001          |     L-1      | 0.9249 |
| Linear-kernel SVM |    BOW     |          0.001           |     L-2      | 0.9445 |
| Linear-kernel SVM |   TF-IDF   |          0.0001          |     L-1      | 0.9489 |
| Linear-kernel SVM |   TF-IDF   |          0.0001          |     L-2      | 0.9635 |
| Linear-kernel SVM |  AVG W2V   |          0.0001          |     L-1      | 0.9128 |
| Linear-kernel SVM |  AVG W2V   |          0.001           |     L-2      | 0.9144 |
| Linear-kernel SVM | TFIDF W2V  |          0.001           |     L-1      | 0.8944 |
| Linear-kernel SVM | TFIDF W2V  |          0.001           |     L-2      | 0.8956 |
+-------------------+------------+--------------------------+--------------+--------+


Feature engineered output after adding review length to BOW vectorized data:=


+------------+------------+--------------------------+--------------+--------+
|   Model    | Vectorizer | Hyper Parameter(alpha-α) | Regularizer  |  AUC   |
+------------+------------+--------------------------+--------------+--------+
| Linear SVM |    BOW     |          0.0001          |     L-1      | 0.9038 |
| Linear SVM |    BOW     |          0.0001          |     L-2      | 0.9182 |
+------------+------------+--------------------------+--------------+--------+


RBF Kernel SVM


+----------------+------------+--------------------+--------------------------+--------
+
|     Model      | Vectorizer | Hyper Parameter(C) | Hyper Parameter(gamma-γ) |  AUC   |
+----------------+------------+--------------------+--------------------------+--------
+
| RBF-Kernel SVM |    BOW     |         10         |           0.01           | 0.9085 |
| RBF-Kernel SVM |   TF-IDF   |         10         |           0.01           | 0.9142 |
| RBF-Kernel SVM |  AVG W2V   |         10         |           0.1            | 0.8865 |
| RBF-Kernel SVM | TFIDF W2V  |        1000        |           0.01           | 0.8757 |
+----------------+------------+--------------------+--------------------------+--------
+
```

Observations

Adding additional feature review length does not seem to affect the classifier much the AUC score of the model remained similar wheras we could observe slight degrade in precision and AUC score upon adding the feature

Both L-1 Regularization and L-2 Regularization performed similarly

Feature importance could be observed using the weights assigned to the features

The behaviour of the models corresponding to all the featurizations was ideal where the best performance was observed from Lenier kernel SVM implemented using SGDClassifier with TF-IDF vectorizer and l2 regularizer