

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 will be set to "positive". Otherwise, it will be set to "negative".

In [2]:

```
%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 [3]:

```
# 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 5000
00""", 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 r
ating(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[3]:

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
----	-----------	--------	-------------	----------------------	------------------------	-------	------

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

In [4]:

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

In [5]:

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

(80668, 7)
```

Out[5]:

	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 [6]:

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

Out[6]:

	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 [7]:

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

Out[7]:

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 [8]:

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

Out[8]:

	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 delete 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 [10]:

```
#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 [11]:

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

Out[11]:

(286837, 10)

In [12]:

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

Out[12]:

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 calculations

In [13]:

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

display.head()
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	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 [14]:

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

In [15]:

```
#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[15]:
```

```
1      241601
```

```
0      45234
```

```
Name: Score, dtype: int64
```

segragating datapoints w.r.t calss labels

```
In [16]:
```

```
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 [17]:
```

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

```
(45234, 10)
```

```
In [18]:
```

```
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 [19]:
```

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

```
In [20]:
```

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

```
<class 'pandas.core.frame.DataFrame'>
```

```
(90468, 10)
```

```
1      45234
```

```
0      45234
```

```
Name: Score, dtype: int64
```

[3] Preprocessing

```
In [21]:
```

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

    preprocessed_reviews.append(sentence.strip())

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

```

90468

In [22]:

```
print(len(preprocessed_reviews))
print(type(final_new_frame))
print(final_new_frame.shape)
```

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

[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 observed to be better than Porter Stemming)

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

In [21]:

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

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

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

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

The same author wrote "Where the Wild Things Are." Carol King wrote a great song that matches all the lyrics. The illustrations are fabulous. I wish I could buy it hardbound and larger. It is a tiny book and easily misplaced. My 3 year-old carries it everywhere.

I've made a million different icing recipes because I am such a picky person when it comes to frosting and I am against buying canned icing because it tastes too generic. After a friend of mine introduced me to Wilton Ready To Use Icing, I was hooked! I couldn't believe that an icing so perfect for decorating and so creamy could taste so good! In my opinion, it's not crisco-y at all compared to most recipes made with shortening. It's sweet but not too sweet to overpower the flavor of the cake. Wilton hit a home run on this one!

I have been using this food for 4 years since I read raving reviews on many sites. I was looking for a reasonably priced food which was better in quality than the cheap chain cra

p. I didn't want to pay \$2 a pound for a high quality food and Canidae was just right for my budget and quality needs. My dog loves it (but you can't really judge it on that alone as he is a lab and will eat anything) however his coat is nice, his stool is the right consistency, and he is hardly ever ill. He also NEVER EVER has gas (but he does burp from time to time but not smelly like others have mentioned.) I wasn't happy when the price increased and the quality supposedly dropped a bit but I am still happy with this food overall and have continued to use it. I am always on the lookout for high quality food at a reasonable price so if anyone knows of something high quality at around the same price point please respond to this post and let me know.

On a different note, was I the only person who actually DID hear about the formula change before it was too late? When I went to purchase more food and saw that the 40lb bag was discontinued I went looking for a reason why and to make sure there weren't any other issues. Right on their website it mentioned the formula change, bag change, and the proper way to transition your dog from the old to the new food to avoid digestive issues. As I had 1 full bag of old formula left, I followed the instructions and guess what, no issues whatsoever with the transition. My dog did NOT get sick at all and has not had a single problem with this food new or old formula.

=====

This container is medium size but I like the size and it was convenient for my to purchase this item because I could only find small container at the local grocery store. I love to cook with beef flavored bouillon so it was a great purchase for me.

=====

In [0]:

```
# 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?
 />
The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

In [0]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

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

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

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

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

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. The best thing was that there were a lot of "brown" chips in the bag (my favorite), so I bought some more through amazon and shared with family and friends. I am a little disappointed 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 [0]:

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

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

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

In [0]:

```
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. 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. 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. 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 [0]:

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

Why is this\$[...] when the same product is available for \$[...] here? The

Victor and traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

In [0]:

```
#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 other wants crispy cookies Hey I am sorry but these reviews do nobody any good beyond 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 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 is also remember that tastes differ so I have 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 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 together Soft cookies tend to do that They are not individually wrapped which would add 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 Nabisco is Ginger Snaps If you want a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal give these a try I am here to place my second order

In [0]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
, 'you', "you're", "you've", \
    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', \
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', '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', 'over', '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', 'mightn', "mightn't", 'mustn', \
    'mustn't', 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
    'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:

```
# Combining all the above students
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence= BeautifulSoup(sentence, 'lxml').get_text()
```

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

```
preprocessed_reviews[1500]
```

'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry reviews nobody good beyond reminding us look ordering chocolate oatmeal cookies not like 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 differ 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 soft however confusion yes stick together soft cookies tend not individually wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet want something hard crisp suggest nabisco ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal give try place second order'

```
#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 (90468, 5000)
the number of unique words including both unigrams and bigrams 5000
```

[4.3] TF-IDF

In [25]:

```
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) ['aa', 'aafco', 'aback', 'abandoned', 'a
bc', 'abdomen', 'abdominal', 'abdominal pain', 'ability', 'able']
=====
```

```
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer (90468, 54016)
the number of unique words including both unigrams and bigrams 54016
```

[4.4] Word2Vec

In [31]:

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

In [33]:

```
# 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/#.Wl7SRFAzZPY
# you can comment this whole cell
# or change these variable 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 occurred at least 5 times
w2v_model=Word2Vec(list_of_sentence,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 ")
```

```
[('fantastic', 0.8749005794525146), ('wonderful', 0.8430687189102173), ('awesome', 0.8413
587808609009), ('good', 0.8320281505584717), ('terrific', 0.8142518997192383), ('excellen
t', 0.8011894822120667), ('amazing', 0.7942032814025879), ('perfect', 0.7901023030281067)
, ('incredible', 0.7160885334014893), ('fabulous', 0.706480860710144)]
=====
```

```
[('nastiest', 0.8269538283348083), ('best', 0.7308410406112671), ('greatest', 0.730632424
3545532), ('foulest', 0.6892600059509277), ('disgusting', 0.676751971244812), ('vile', 0.
6687464118003845), ('weakest', 0.6443219184875488), ('saltiest', 0.6227551698684692), ('g
rossest', 0.6104087829589844), ('horrid', 0.6095949411392212)]
```

In [34]:

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

```
number of words that occurred minimum 5 times 18587
sample words ['even', 'though', 'statement', 'right', 'front', 'packages', 'gum', 'claim
s', 'gluten', 'free', 'not', 'believe', 'really', 'bought', 'pack', 'peppermint', 'health
', 'food', 'store', 'daughter', 'chewed', 'gf', 'celiac', 'reacted', 'tried', 'days', 'la
ter', 'thinking', 'perhaps', 'something', 'else', 'eaten', 'considering', 'says', 'reacti
on', 'worse', 'second', 'time', 'around', 'company', 'sources', 'xylitol', 'china', 'almo
st', 'companies', 'looked', 'online', 'purchase', 'also', 'make']
```

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

[4.4.1.1] Avg W2v

In [35]:

```
# 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_sentence): # 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%|████████████████████████████████████████████████████████████████████████████████| 90468/9
0468 [03:05<00:00, 487.67it/s]
```

[4.4.1.2] TFIDF weighted W2v

In [36]:

```
# 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 [37]:

```
# 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_sentence): # 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 corpus
            # sent.count(word) = tf value 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%|██████████████████████████████████████████████████████████████| 90468/  
90468 [42:42<00:00, 35.30it/s]
```

[5] Assignment 5: Apply Logistic Regression

1. Apply Logistic Regression on these feature sets

- **SET 1:** Review text, preprocessed one converted into vectors using (BOW)
- **SET 2:** Review text, preprocessed one converted into vectors using (TFIDF)
- **SET 3:** Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:** Review text, preprocessed one converted into vectors using (TFIDF W2v)

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

- 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

3. Perturbation Test

- Get the weights W after fit your model with the data X i.e Train data.
- Add a noise to the X ($X' = X + e$) and get the new data set X' (if X is a sparse matrix, $X.data+=e$)
- Fit the model again on data X' and get the weights W'
- Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e $W=W+10^{-6}$ and $W' = W'+10^{-6}$
- Now find the % change between W and W' ($| (W-W') / (W) | * 100$)
- Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage change vector

percentage_change_vector

- Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
- Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. Sparsity

- Calculate sparsity on weight vector obtained after using L1 regularization

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

5. Feature importance

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

6. Feature engineering

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

7. Representation of results

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#).

8. [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 Logistic Regression

[5.1] Logistic Regression on BOW, **SET 1**

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

In [38]:

```
print(len(preprocessed_reviews))
print(type(final_new_frame))
print(final_new_frame.shape)
```

```
90468
<class 'pandas.core.frame.DataFrame'>
```


(90468, 10)

In [183]:

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

In [241]:

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

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

In [41]:

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

```
After vectorizations
(60613, 48209) (60613,)
(29855, 48209) (29855,)
```

In [42]:

```
print(type(X_train_bow))
print(X_train_bow.get_shape())
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(60613, 48209)
```

In [47]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1')
parameters = [{ 'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_train_bow, y_train)

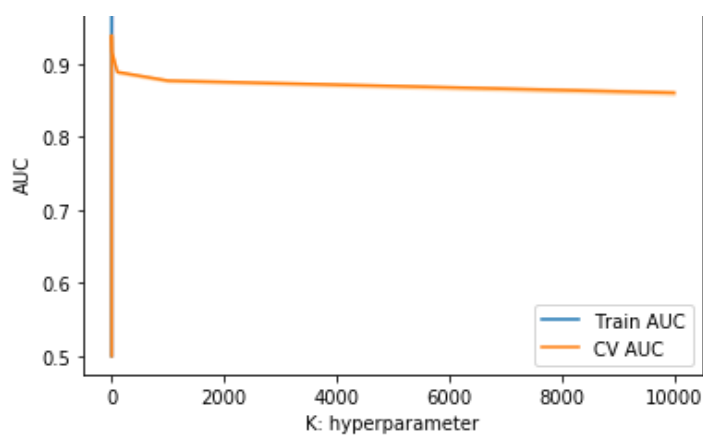
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

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

plt.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, color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

ERROR PLOTS



In [49]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

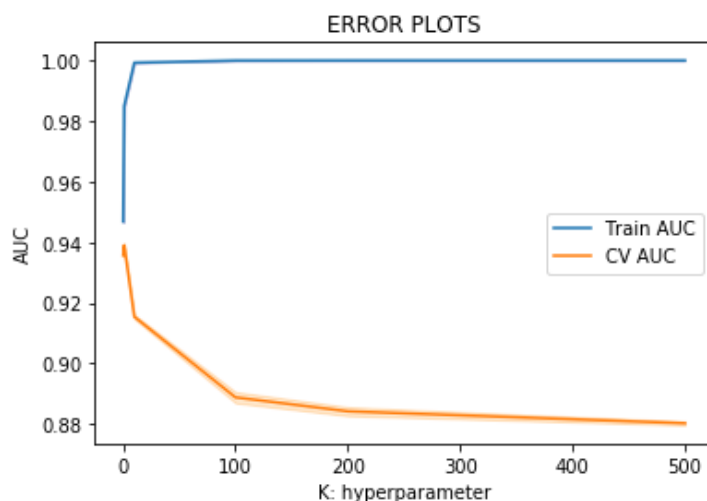
lr = LogisticRegression(penalty='l1')
parameters = [{'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

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, 1, 10, 100,200,500]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [50]:

```

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1')
parameters = [{'C':[0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]]}
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow, y_train)

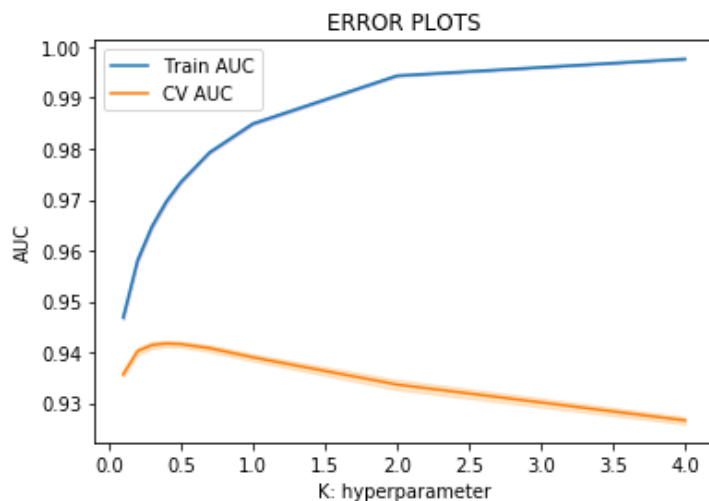
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



In [51]:

```

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1')
parameters = [{'C':[0.1,0.2,0.3,0.4,0.5]]}
clf = GridSearchCV(lr, 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']

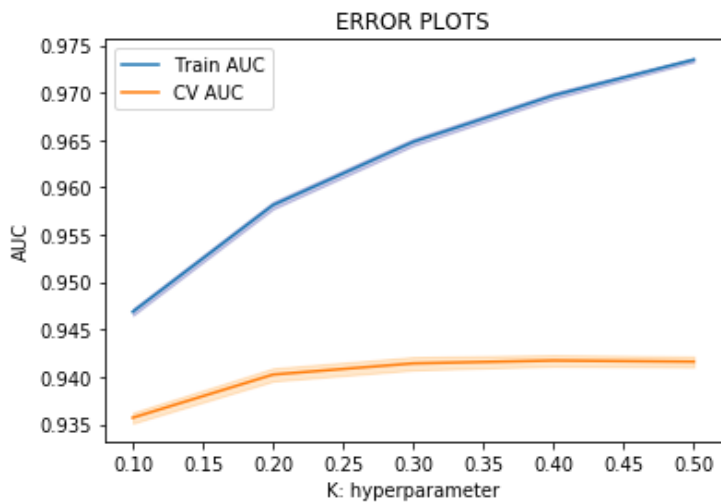
```

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

```
K = [0.1,0.2,0.3,0.4,0.5]
```

```
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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



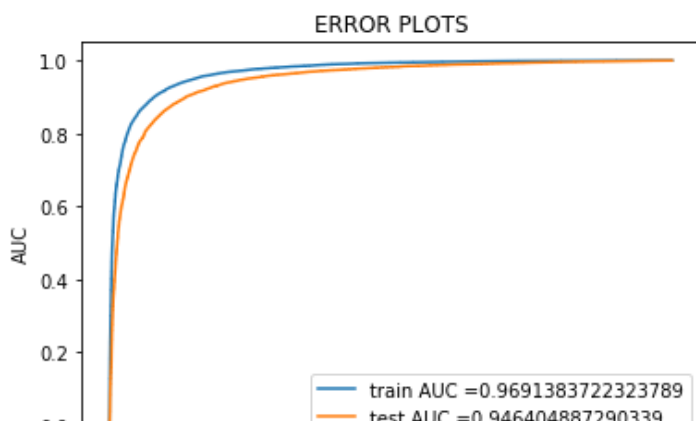
In [54]:

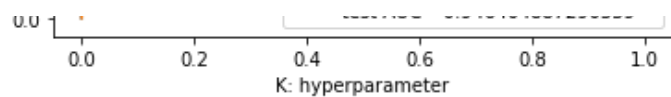
```
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(penalty='l1',C=0.4)
lr.fit(X_train_bow, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(X_train_bow)[:,-1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```





In [55]:

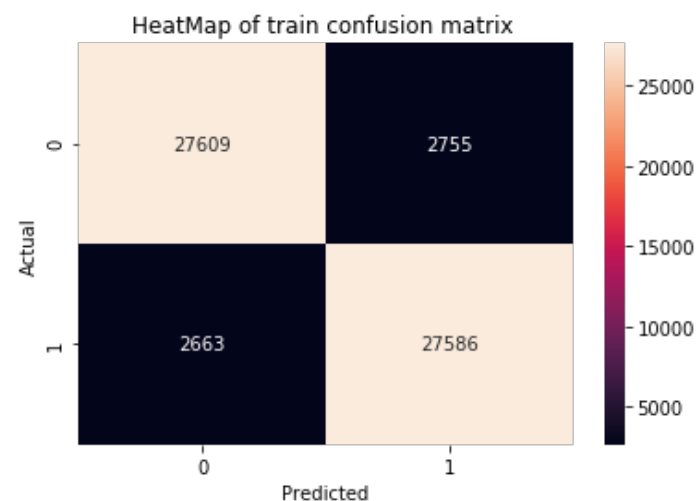
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(X_train_bow))
y=confusion_matrix(y_test, lr.predict(X_test_bow))
print(x)
print("Test confusion matrix")
print(y)
```

```
Train confusion matrix
[[27609  2755]
 [ 2663 27586]]
Test confusion matrix
[[13084  1786]
 [ 1718 13267]]
```

In [56]:

```
import seaborn as sn

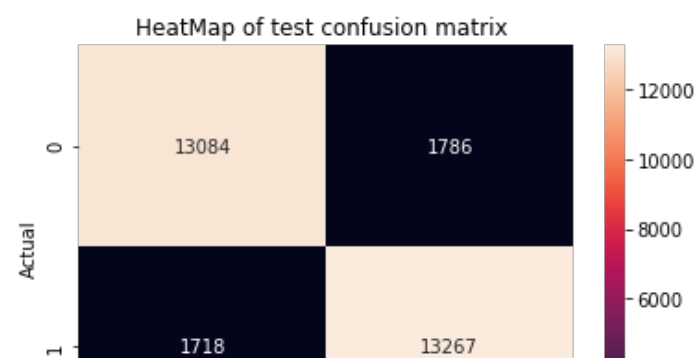
ax = plt.axes()
sns.heatmap(x, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()
```

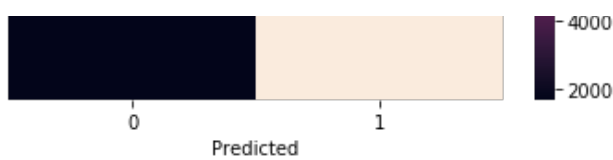


In [57]:

```
import seaborn as sn

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()
```





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

In [58]:

```
w=lr.coef_
```

In [59]:

```
print(w.shape)
print(type(w))
```

```
(1, 48209)
<class 'numpy.ndarray'>
```

In [60]:

```
print(np.count_nonzero(w))
```

```
3369
```

Hyper Parameter Tunning Now Using the BOW vectorized data which contains Review length feature added

In [61]:

```
from scipy import sparse
review_length_train=[]

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

In [62]:

```
review_length_test=[]

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

print(review_length_test[0])
```

```
96
```

In [75]:

```
print(type(X_train_bow))
print(X_train_bow.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(60613, 48209)
```

In [76]:

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

In [77]:

```
print(type(X_train_bow_f1))
print(X_train_bow_f1.shape)
```

```
<class 'scipy.sparse.coo.coo_matrix'>
(60613, 48210)
```

In [78]:

```
print(type(X_test_bow))
print(X_test_bow.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(29855, 48209)
```

In [79]:

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

In [80]:

```
print(type(X_test_bow_f1))
print(X_test_bow_f1.shape)
```

```
<class 'scipy.sparse.coo.coo_matrix'>
(29855, 48210)
```

In [81]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
```

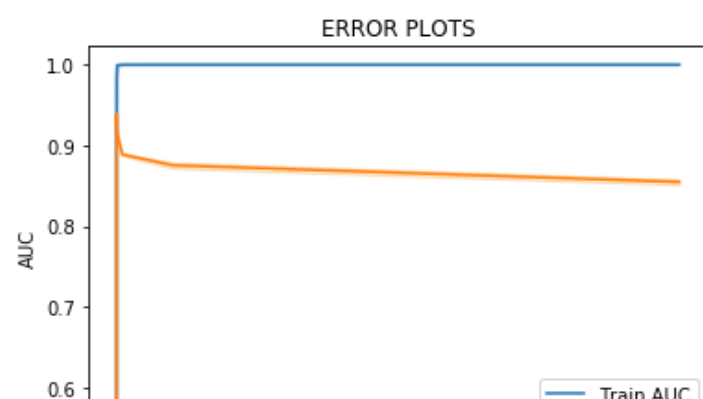
```
lr = LogisticRegression(penalty='l1')
parameters = [{ 'C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_f1, y_train)
```

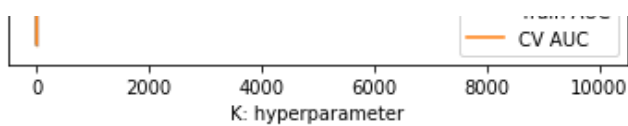
```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

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

```
plt.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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```





In [83]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

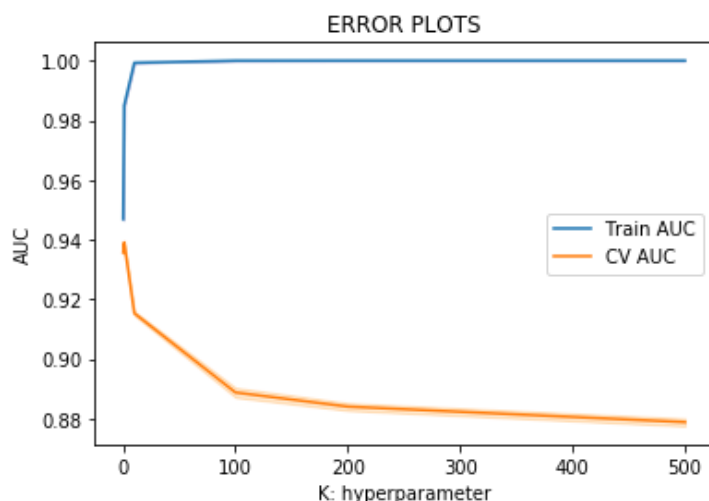
lr = LogisticRegression(penalty='l1')
parameters = [{'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_fl, y_train)

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, 1, 10, 100,200,500]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [84]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
```

```
lr = LogisticRegression(penalty='l1')
```



```

parameters = [{ 'C':[0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_fl, y_train)

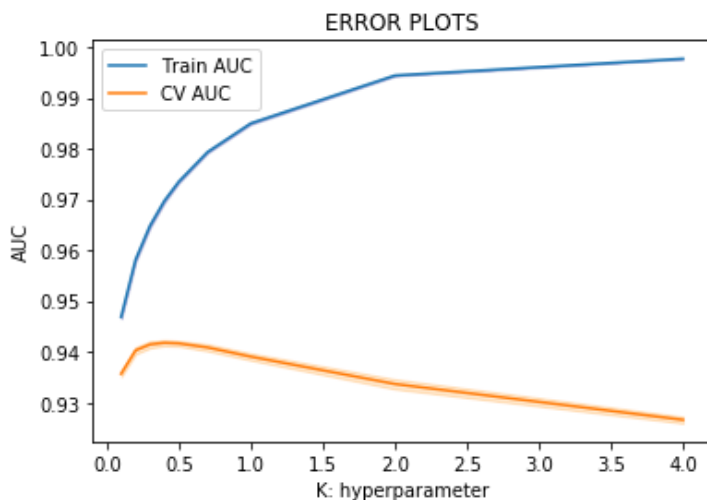
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



In [85]:

```

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1')
parameters = [{ 'C':[0.1,0.2,0.3,0.4,0.5,0.6,0.7]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_bow_fl, y_train)

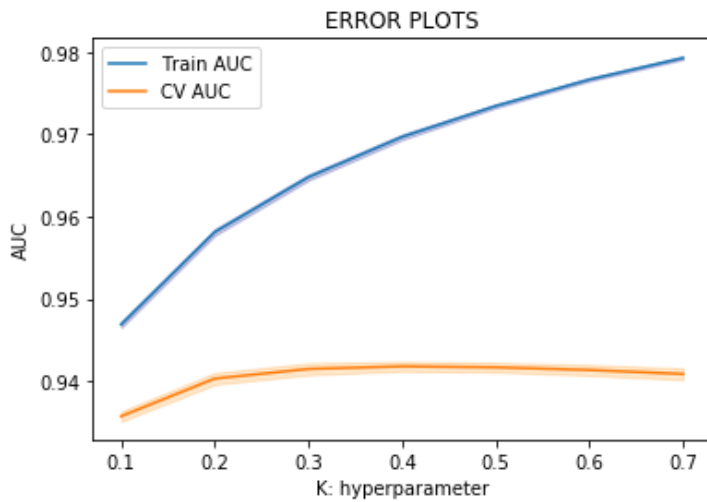
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1,0.2,0.3,0.4,0.5,0.6,0.7]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



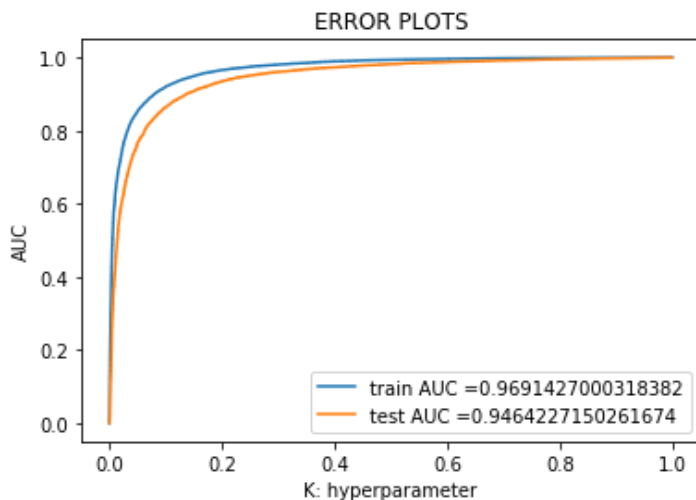
In [88]:

```
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(penalty='l1',C=0.4)
lr.fit(X_train_bow_f1, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(X_train_bow_f1)[: ,1
])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [90]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
```

```

x=confusion_matrix(y_train, lr.predict(X_train_bow_fl))
y=confusion_matrix(y_test, lr.predict(X_test_bow_fl))
print(x)
print("Test confusion matrix")
print(y)

```

```
import seaborn as sn
```

```

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()

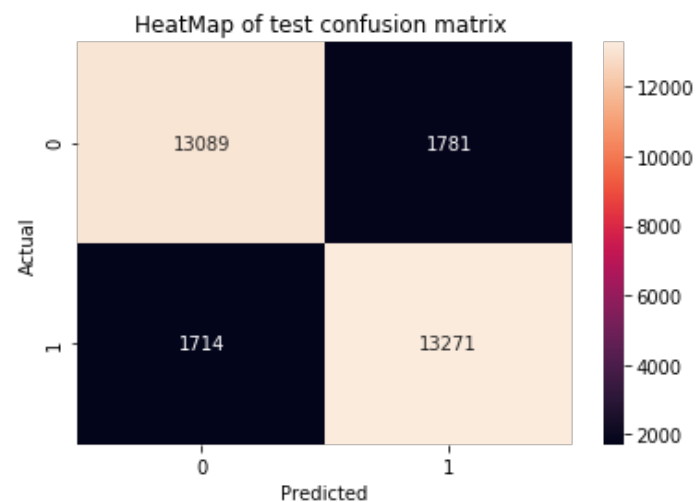
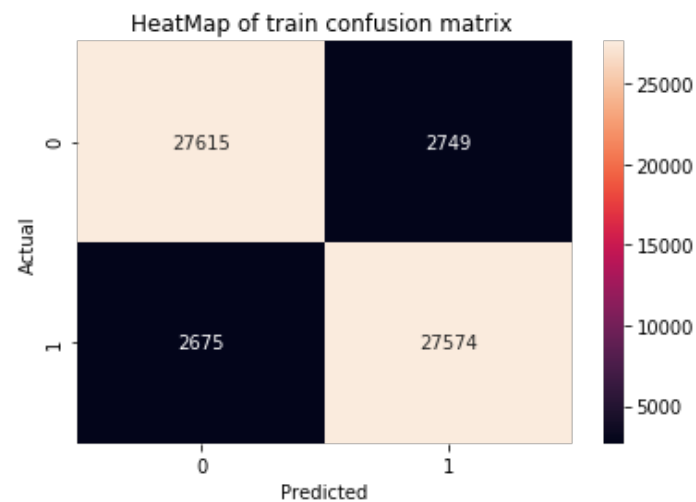
```

Train confusion matrix

```
[[27615  2749]
 [ 2675 27574]]
```

Test confusion matrix

```
[[13089  1781]
 [ 1714 13271]]
```



In []:

In []:

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

In [0]:

```
# Please write all the code with proper documentation
```

In [91]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

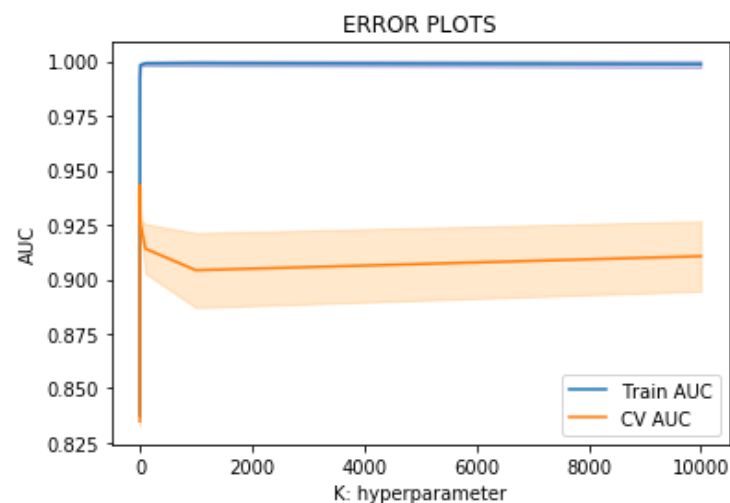
lr = LogisticRegression(penalty='l2')
parameters = [{ 'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_train_bow, y_train)

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.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, color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [93]:

```
from sklearn.metrics import roc_curve, auc

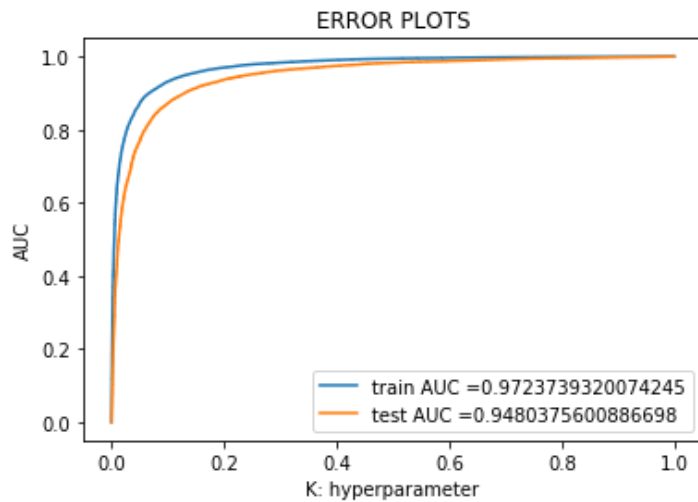
lr = LogisticRegression(penalty='l2', C=0.1)
lr.fit(X_train_bow, y_train)
```

```

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(X_train_bow)[: ,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



In [94]:

```

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(X_train_bow))
y=confusion_matrix(y_test, lr.predict(X_test_bow))
print(x)
print("Test confusion matrix")
print(y)

```

```

Train confusion matrix
[[27840  2524]
 [ 2514 27735]]
Test confusion matrix
[[13152  1718]
 [ 1677 13308]]

```

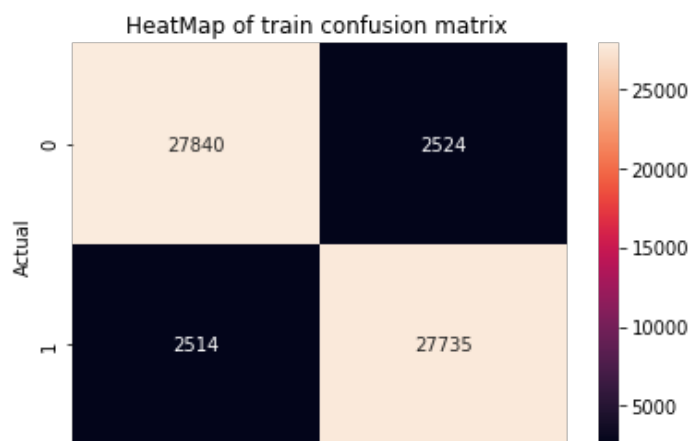
In [95]:

```

import seaborn as sn

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()

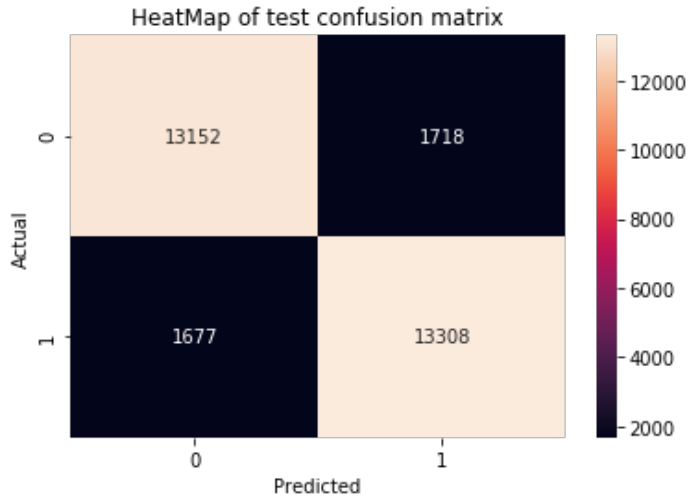
```



0 Predicted 1

In [96]:

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



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

In [0]:

```
# Please write all the code with proper documentation
```

In [185]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l2',C=0.1)
lr.fit(X_train_bow, y_train)
W1=lr.coef_
```

In [186]:

```
print (X_train_bow[0].shape)
```

(1, 48313)

In [187]:

```
print (W1.shape)
print (W1[0].shape)
```

(1, 48313)

(48313,)

In [129]:

```
print(np.random.normal(0, 0.01))
print(np.random.uniform(0.001,0.002))
```

0.008439429427564737
0.0010559861792919201

In []:

```
In [188]:
```

```
epsilon = np.random.uniform(0.001,0.002)
X_train_bow.data = X_train_bow.data + epsilon
```

```
In [189]:
```

```
lr1 = LogisticRegression(penalty='l2',C=0.1)
lr1.fit(X_train_bow, y_train)
W2=lr1.coef_
```

```
In [190]:
```

```
print(W1.shape)
print(W2.shape)
```

```
(1, 48313)
(1, 48313)
```

```
In [191]:
```

```
print(type(W1))
print(type(W1[0]))
print(W1[0].shape)
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(48313,)
```

```
In [192]:
```

```
W11=W1+0.000001
W22=W2+0.000001
```

```
In [193]:
```

```
change1=abs((W11-W22)/(W11))
```

```
In [194]:
```

```
percchange1=change1*100
```

```
In [195]:
```

```
percentile1c=np.percentile(percchange1,[0,10,20,30,40,50,60,70,80,90,100])
```

```
In [197]:
```

```
k2=[0,10,20,30,40,50,60,70,80,90,100]
print(type(k2))
#percentile2c=percentile1c.reshape(-1,11)
```

```
<class 'list'>
```

```
In [200]:
```

```
percentile2c=percentile1c.reshape(11,-1)
```

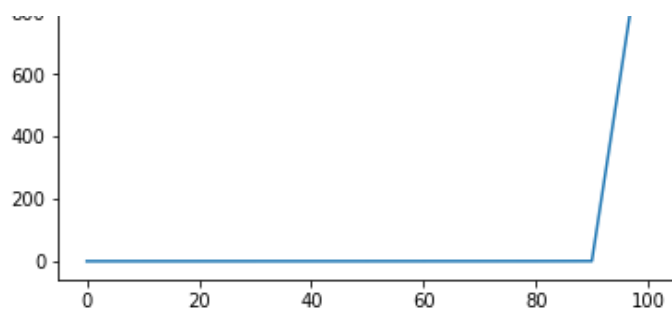
```
In [201]:
```

```
k1=[0,10,20,30,40,50,60,70,80,90,100]
plt.plot(k1, percentile2c, label='CV AUC')
```

```
Out[201]:
```

```
[<matplotlib.lines.Line2D at 0x2143db1a160>]
```

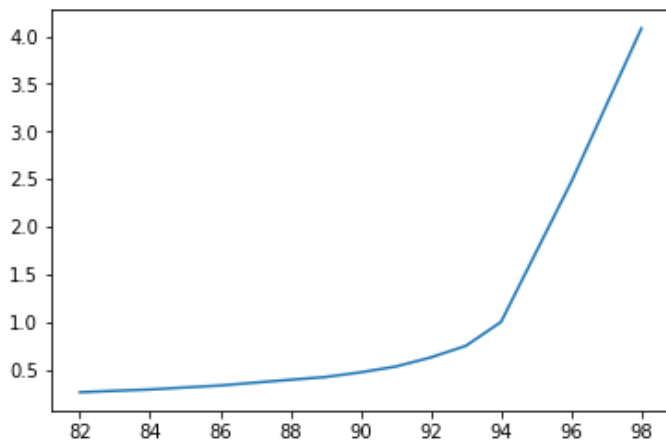




In [204]:

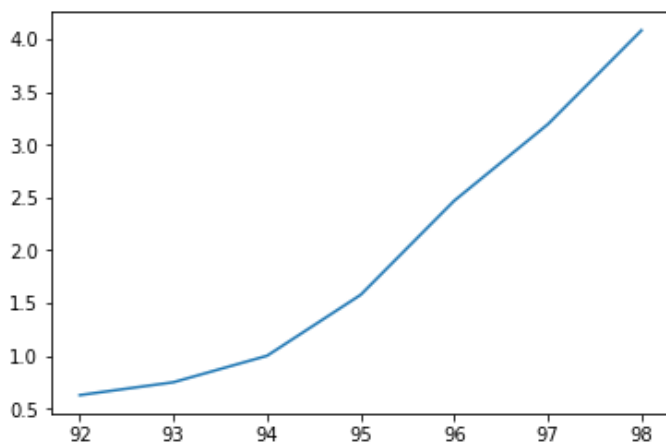
```
percentile3c=np.percentile(percchange1,[82,84,86,88,89,90,91,92,93,94,96,98])
k3=[82,84,86,88,89,90,91,92,93,94,96,98]

plt.plot(k3,percentile3c, label='CV AUC')
plt.show()
```



In [216]:

```
percentile4c=np.percentile(percchange1,[92,93,94,95,96,97,98])
k4=[92,93,94,95,96,97,98]
plt.plot(k4,percentile4c, label='CV AUC')
plt.show()
```



In []:

```
for i in
np.percentile(percchange1,[i])
```

In [221]:

```
for i in range (93,101):
    print(np.percentile(percchange1,[i]))
    print(i)
```

```
[0.74938355]
93
[1.00002326]
```



```
94
[1.5771436]
95
[2.46844759]
96
[3.19295162]
97
[4.08234122]
98
[7.29015969]
99
[1166.11293474]
100
```

In [229]:

```
lst1=[99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100]
```

In [230]:

```
for i in lst1:
    print(i)
    print(np.percentile(percchange1,[i]))
```

```
99.1
[8.12494463]
99.2
[10.13092255]
99.3
[11.895127]
99.4
[12.91343129]
99.5
[14.70549994]
99.6
[15.40886852]
99.7
[17.9507764]
99.8
[25.09326495]
99.9
[40.94506954]
100
[1166.11293474]
```

In [224]:

```
#99.9 40
```

In [236]:

```
features1=vectorizer.get_feature_names()
```

Features names of the features whose % change is higher than that of the treshold

In [239]:

```
fl=[]
for i in np.where(percchange1 > 40)[1]:
    fl.append(features1[i])
print(fl)
```

```
['accompanied', 'astra', 'axful', 'bikesomewhere', 'bonk', 'cheapie', 'cherishes', 'chewe
r', 'chihuahuas', 'cuisinart', 'deceitful', 'dips', 'disolve', 'disruption', 'donckels',
'dozens', 'draft', 'dredged', 'effendi', 'feeding', 'fizzes', 'freekeh', 'friction', 'fro
thier', 'headquarters', 'imitators', 'jujubes', 'kenzi', 'lagavulin', 'logical', 'mazar',
'moreover', 'napkins', 'nigh', 'obesei', 'pad', 'palouse', 'passion', 'pebbley', 'perishi
ble', 'prepaid', 'receptions', 'rican', 'ricochet', 'shops', 'smartbalance', 'sourcing',
'stongly', 'streusel', 'teance', 'underactive']
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [262]:

```
cp5=np.argsort(W1[0])
```

In [266]:

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

In [263]:

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

In [267]:

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

```
['best', 'wonderful', 'smooth', 'complaint', 'loves', 'highly', 'perfect', 'pleased', 'excellent', 'delicious']
```

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

In [265]:

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

```
['disappointing', 'worst', 'terrible', 'disappointment', 'disappointed', 'awful', 'horrible', 'stale', 'yuck', 'tasteless']
```

[5.2] Logistic Regression on TFIDF, SET 2

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

In [268]:

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

In [269]:

```
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 [270]:

```
from sklearn.model_selection import GridSearchCV  
from sklearn.naive_bayes import MultinomialNB  
from sklearn.metrics import roc_auc_score
```

```

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1')
parameters = [{'C':[0.0001, 0.001, 0.01, 0.1, 1,2,5,8, 10,20,50,75,100,1000,10000]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

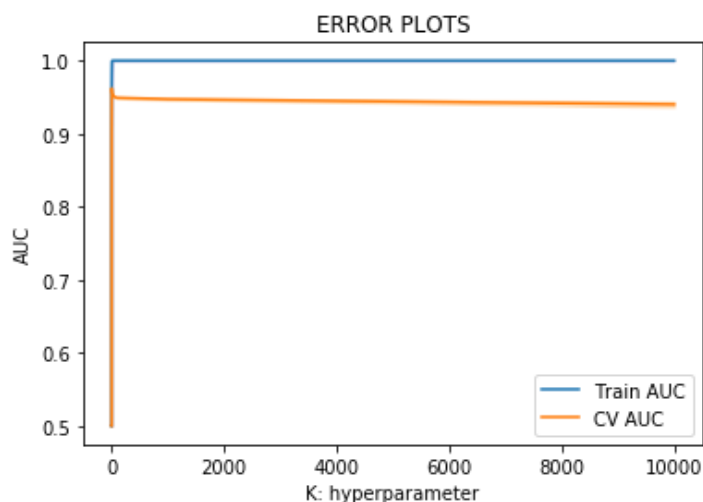
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1,2,5,8, 10,20,50,75,100,1000,10000]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



In [271]:

```

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1')
parameters = [{'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

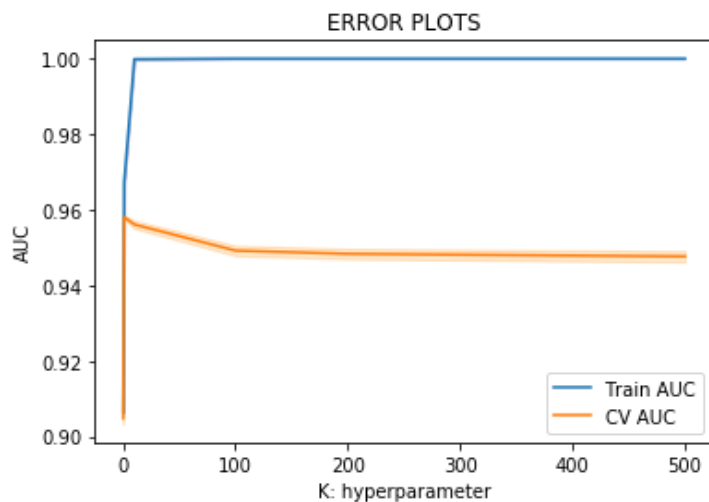
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1, 1, 10, 100,200,500]

```

```
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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [273]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

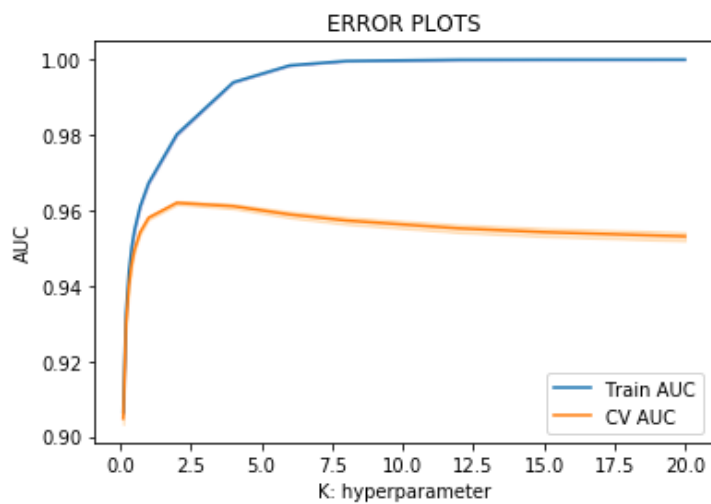
lr = LogisticRegression(penalty='l1',n_jobs=4)
parameters = [{'C':[0.1,0.2,0.3,0.4,0.5,0.7,1,2,4,6,8,12,15,20]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

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 = [0.1,0.2,0.3,0.4,0.5,0.7,1,2,4,6,8,12,15,20]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [28]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

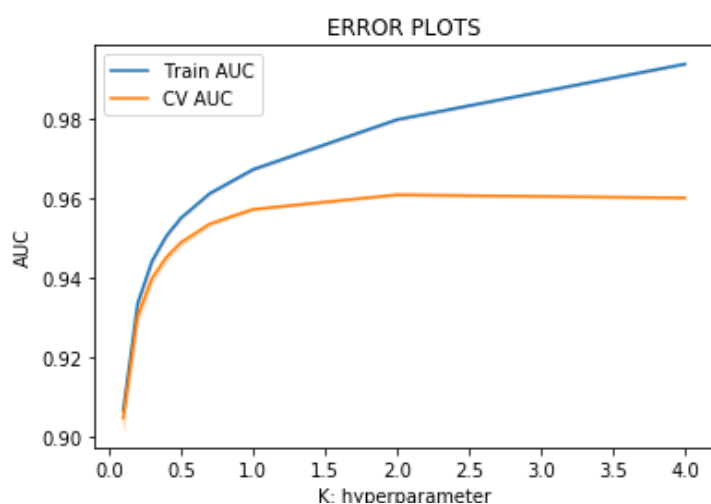
lr = LogisticRegression(penalty='l1',n_jobs=4)
parameters = [{ 'C':[0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

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 = [0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



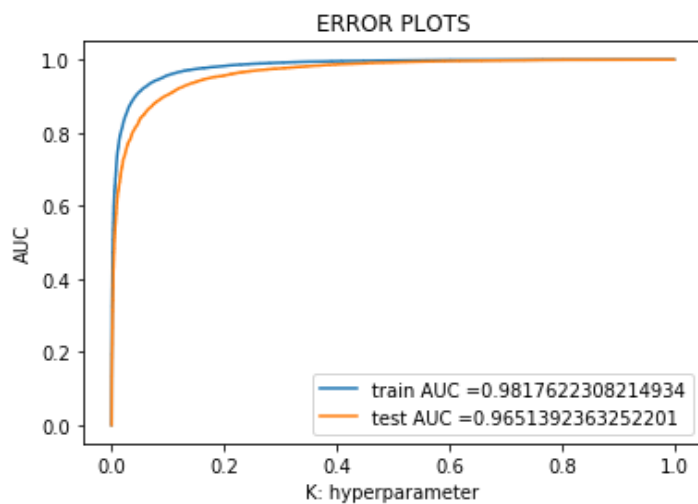
In [276]:

```
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(penalty='l1',C=2)
lr.fit(X_train_tfidf, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(X_train_tfidf)[:,1]
)
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [277]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(X_train_tfidf))
y=confusion_matrix(y_test, lr.predict(X_test_tfidf))
print(x)
print("Test confusion matrix")
print(y)
```

```
Train confusion matrix
[[28213  2006]
 [ 2047 28347]]
Test confusion matrix
[[13571  1444]
 [ 1467 13373]]
```

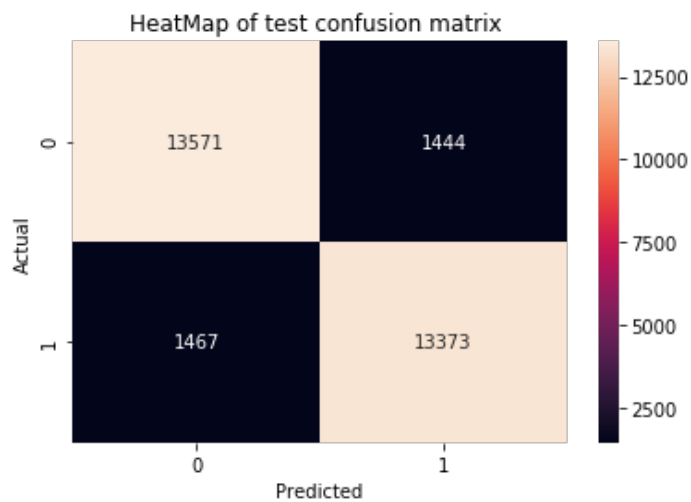
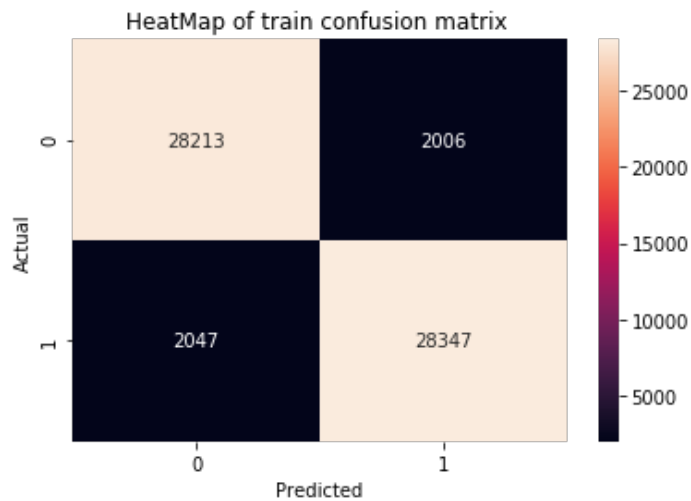
In [278]:

```
import seaborn as sn

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()
```



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

In [279]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression# Please write all the code with proper documentation
```

In [281]:

```
lr = LogisticRegression(penalty='l2')
parameters = [{'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

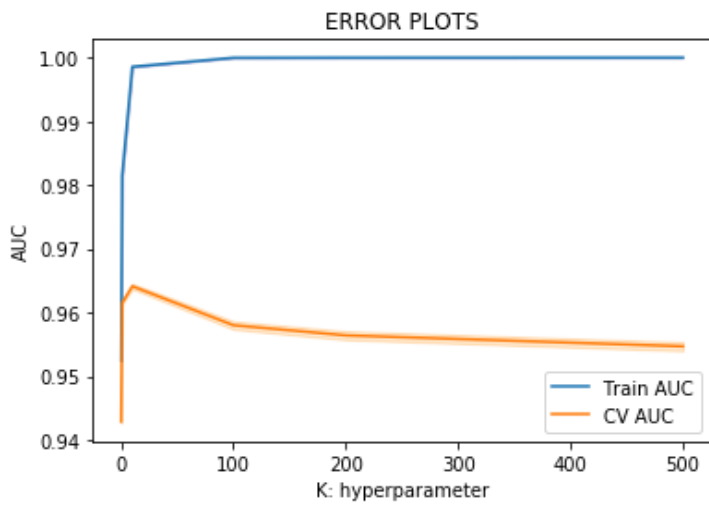
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1, 1, 10, 100,200,500]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [283]:

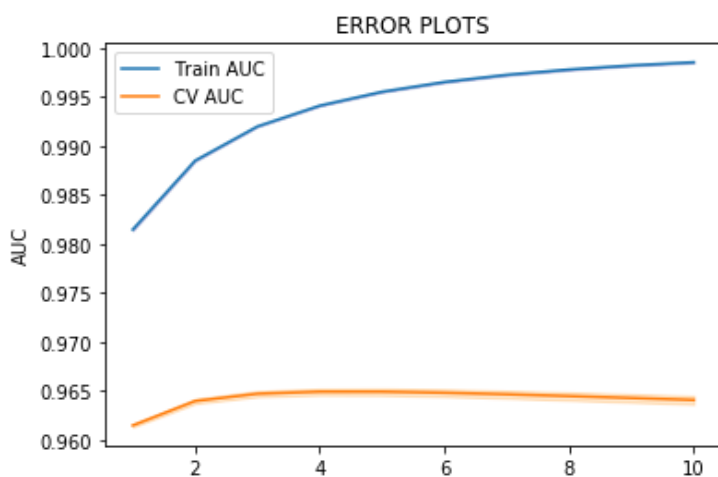
```
lr = LogisticRegression(penalty='l2')
parameters = [{ 'C': [1,2,3,4,5,6,7,8,9,10] }]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_train_tfidf, y_train)

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 = [1,2,3,4,5,6,7,8,9,10]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [285]:

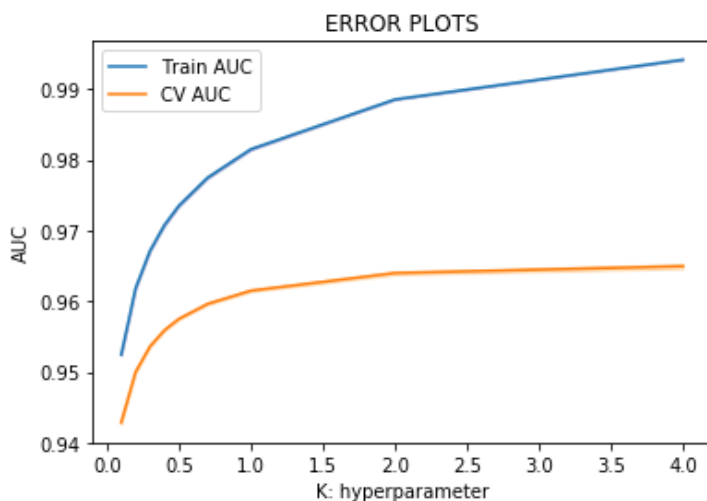
```
lr = LogisticRegression(penalty='l2',n_jobs=4)
parameters = [{ 'C':[0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_train_tfidf, y_train)

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 = [0.1,0.2,0.3,0.4,0.5,0.7,1,2,4]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



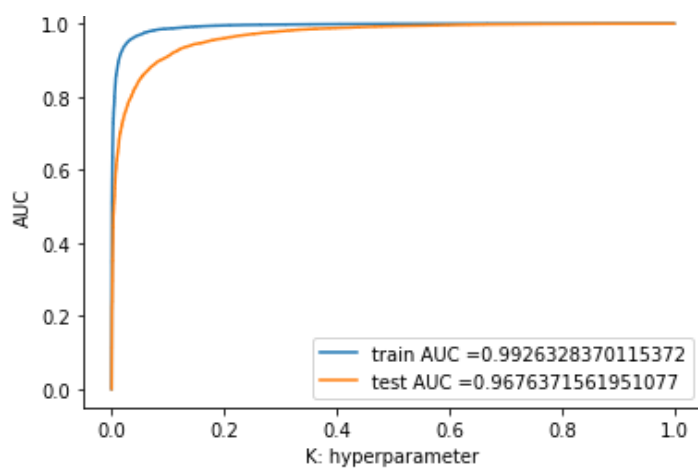
In [288]:

```
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(penalty='l2',C=4)
lr.fit(X_train_tfidf, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(X_train_tfidf)[:,-1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [289]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(X_train_tfidf))
y=confusion_matrix(y_test, lr.predict(X_test_tfidf))
print(x)
print("Test confusion matrix")
print(y)
```

Train confusion matrix

```
[[29102  1117]
 [ 1233 29161]]
```

Test confusion matrix

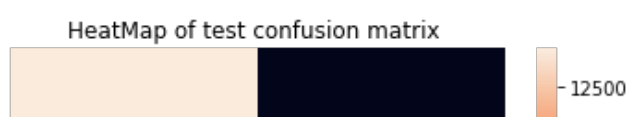
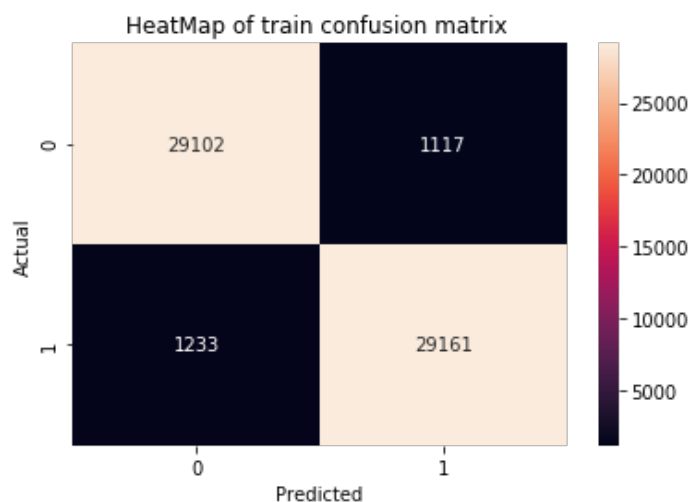
```
[[13661  1354]
 [ 1446 13394]]
```

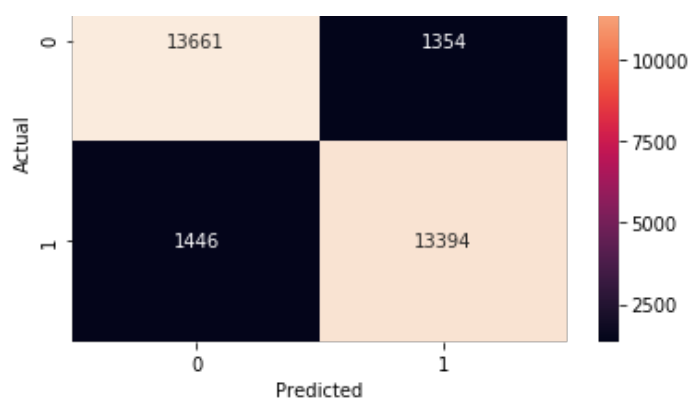
In [290]:

```
import seaborn as sn

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()
```





[5.2.3] Feature Importance on TFIDF, SET 2

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

In [291]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l2', C=4)
lr.fit(X_train_tfidf, y_train)
W1=lr.coef_
```

In [292]:

```
l1=tf_idf_vect.get_feature_names()
```

In [293]:

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

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

['pleased', 'loves', 'wonderful', 'excellent', 'good', 'perfect', 'not disappointed', 'best', 'delicious', 'great']
```

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

In [294]:

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

['disappointed', 'worst', 'terrible', 'not recommend', 'not good', 'not worth', 'not', 'disappointing', 'disappointment', 'awful']
```

[5.3] Logistic Regression on AVG W2V, SET 3

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

In [0]:

```
# Please write all the code with proper documentation
```

In [295]:

```
(60613, 50)
[ 0.29271276 -0.10869773 -0.08260114  0.07887674 -0.32727245 -0.24680514
  0.08787188  0.06232786  0.02791801 -0.32001237 -0.56387258  1.1441912
 -0.79080559 -0.09020881  0.24838257  0.11302868 -0.25123935  0.10457548
 -0.1644958  -0.0441842  -0.18127486 -0.32980947 -0.1719727  0.36814496
 -0.3094559   0.36767102 -0.25699203 -0.41332991 -0.01703794  0.21858921
  0.54585531 -0.01170962 -0.18001316  0.7472098   0.42844822 -0.0730633
 -0.10868804 -0.13914697 -0.05398418  0.91778549 -0.39289335 -0.04459822
 -0.76726731  0.37514215 -0.67414574  0.21356493 -0.57932565  0.05628924
 -0.14007385  0.05524279]
<class 'list'>
```

```
(29855, 50)
[ 0.77417616  0.04082119  0.57251132 -0.20811705 -0.01117748  0.13488478
  0.05489586 -0.09035491 -0.14901139 -0.09574768 -0.29611969  0.42962043
 -0.93457634 -0.61548697  0.67255716  0.63743302  0.44871169  0.09909279
 -0.04228177 -0.39132215  0.25571889 -0.57078466 -0.00689296  0.55197768
 -0.23107189 -0.09476398 -0.09365594 -0.07123326 -0.15300547  0.09370567
  0.04794243 -0.18711473  0.01247541 -0.20352477 -0.16808674 -0.07113717
  0.50753688 -0.23320846 -0.0610958  -0.12389537 -0.41503203 -0.51626623
 -0.23299044  0.16691488 -1.27364857 -0.03154187 -0.12358996 -0.54620094
 -0.02101265 -0.49089675]
```

In [300]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

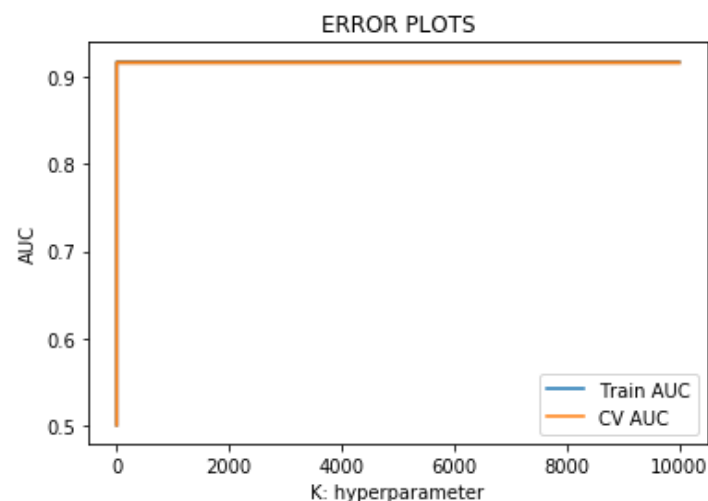
lr = LogisticRegression(penalty='l1')
parameters = [{'C':[0.0001, 0.001, 0.01, 0.1, 1,2,5,8, 10,20,50,75,100,1000,10000]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

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, 1,2,5,8, 10,20,50,75,100,1000,10000]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [302]:

```

lr = LogisticRegression(penalty='l1')
parameters = [{'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

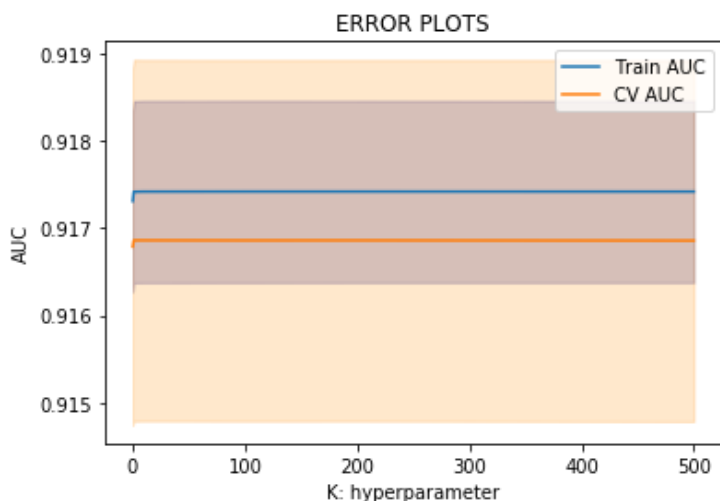
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1, 1, 10, 100,200,500]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



In [305]:

```

lr = LogisticRegression(penalty='l1')
parameters = [{'C':[ 0.1,0.2,0.5,0.7,1,2,3,5,7,9]}]
clf = GridSearchCV(lr, 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']

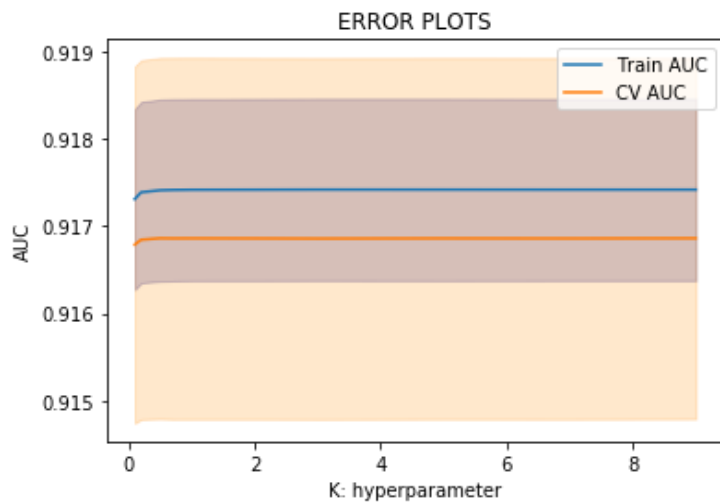
K = [ 0.1,0.2,0.5,0.7,1,2,3,5,7,9]

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,color='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='darkorange')
plt.legend()

```

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



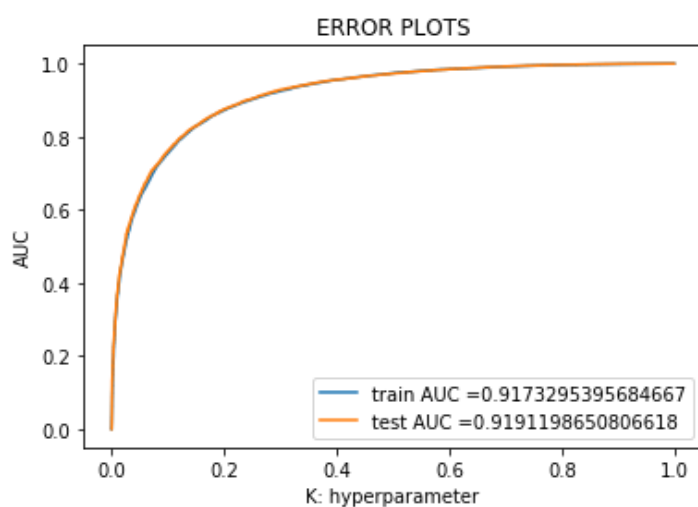
In [308]:

```
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(penalty='l1',C=1)
lr.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [309]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(sent_vectors_train))
y=confusion_matrix(y_test, lr.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)
```

```

Train confusion matrix
[[25725  4638]
 [ 5088 25162]]
Test confusion matrix
[[12681  2190]
 [ 2585 12399]]

```

In [310]:

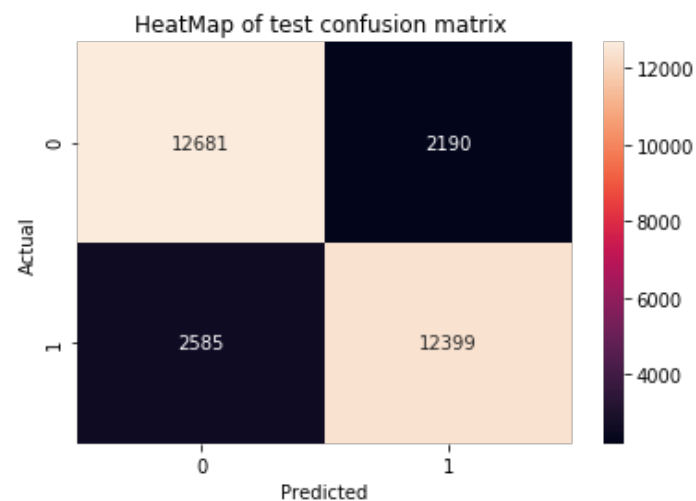
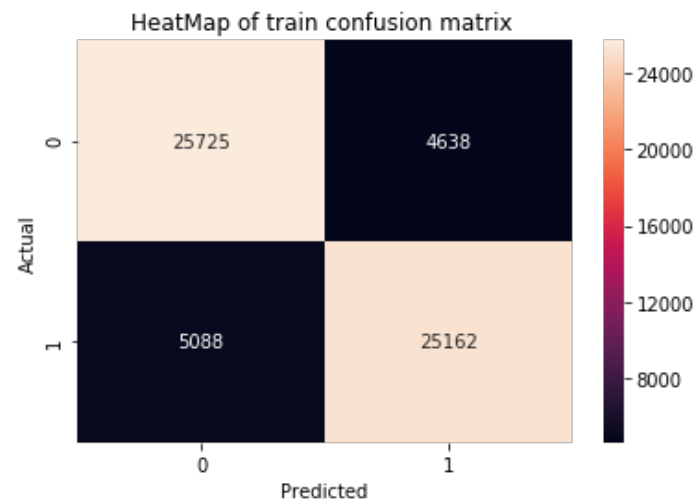
```

import seaborn as sn

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()

```



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

In [311]:

```

lr = LogisticRegression(penalty='l2')
parameters = [{ 'C':[0.0001, 0.001, 0.01, 0.1, 1,2,5,8, 10,20,50,75,100,1000,10000]}]
clf = GridSearchCV(lr, 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']

```

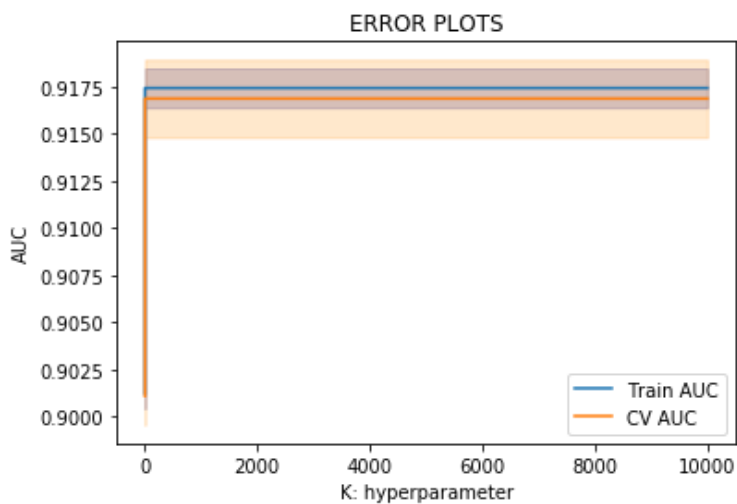


```
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.0001, 0.001, 0.01, 0.1, 1,2,5,8, 10,20,50,75,100,1000,10000]
```

```
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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [313]:

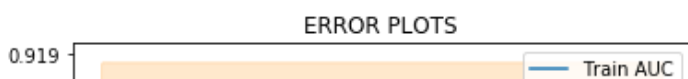
```
lr = LogisticRegression(penalty='l2')
parameters = [{ 'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(sent_vectors_train, y_train)

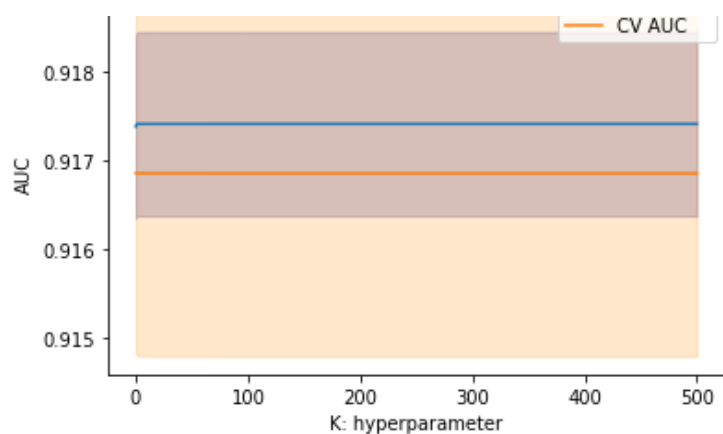
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

K = [0.1, 1, 10, 100,200,500]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



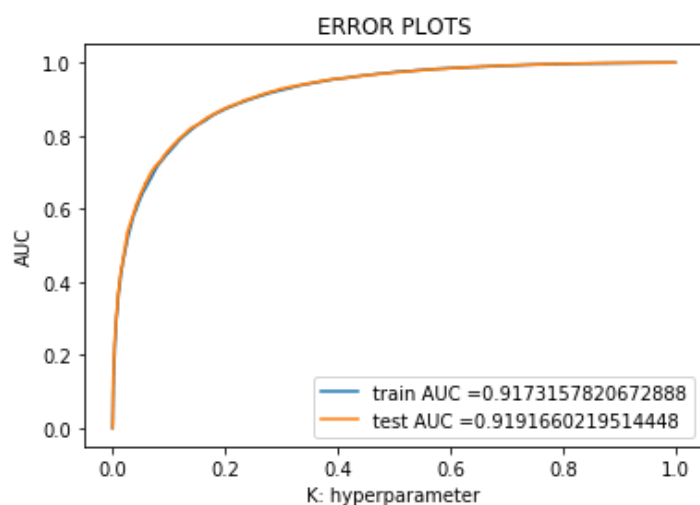


In [317]:

```
lr = LogisticRegression(penalty='l2',C=0.1)
lr.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

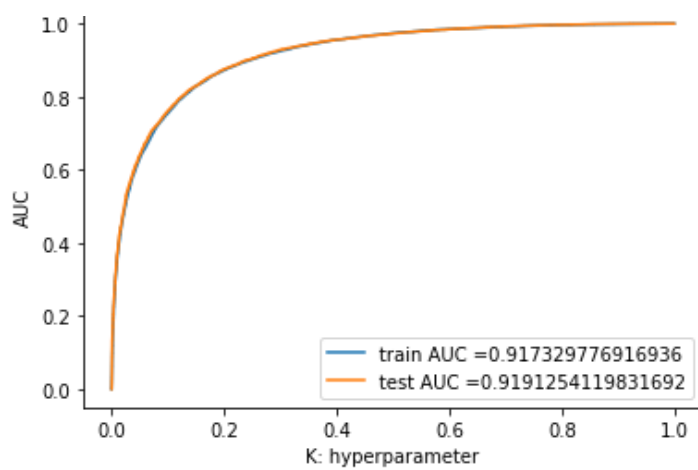


In [316]:

```
lr = LogisticRegression(penalty='l2',C=2)
lr.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [318]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(sent_vectors_train))
y=confusion_matrix(y_test, lr.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)
```

```
import seaborn as sn
```

```
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()
```

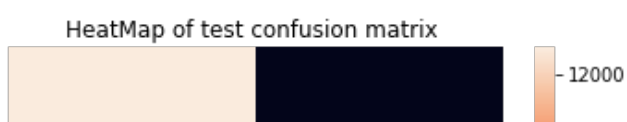
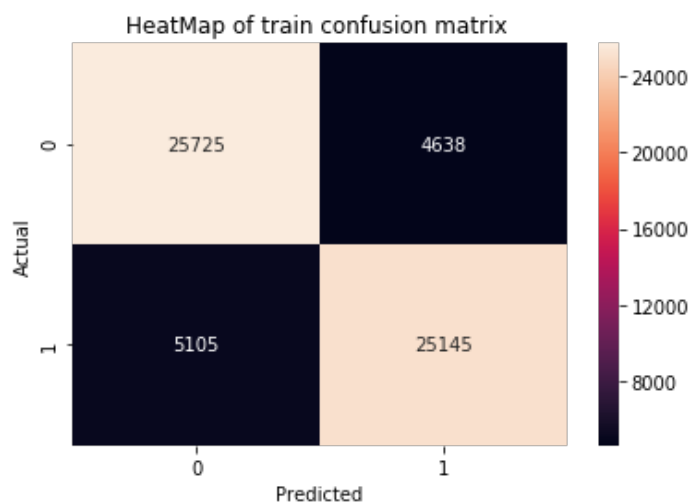
```
ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

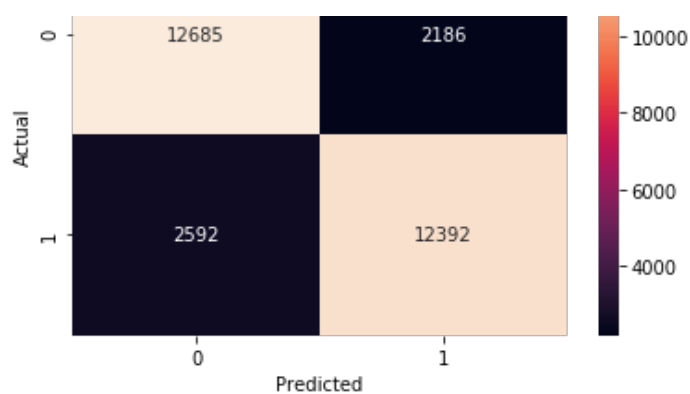
Train confusion matrix

```
[[25725  4638]
 [ 5105 25145]]
```

Test confusion matrix

```
[[12685  2186]
 [ 2592 12392]]
```





[5.4] Logistic Regression on TFIDF W2V, SET 4

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

In [319]:

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

list_of_sentence_train=[]
for sentence in X_train:
    list_of_sentence_train.append(sentence.split())

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

In [320]:

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

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

In [321]:

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

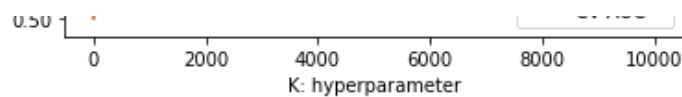
In [322]:

```
tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentence_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 [22:47<00:00, 44.32it/s]
```

In [323]:

```
list_of_sentence_test=[]
for sentence in X_test:
    list of sentence test.append(sentence.split())
```

In [328]:

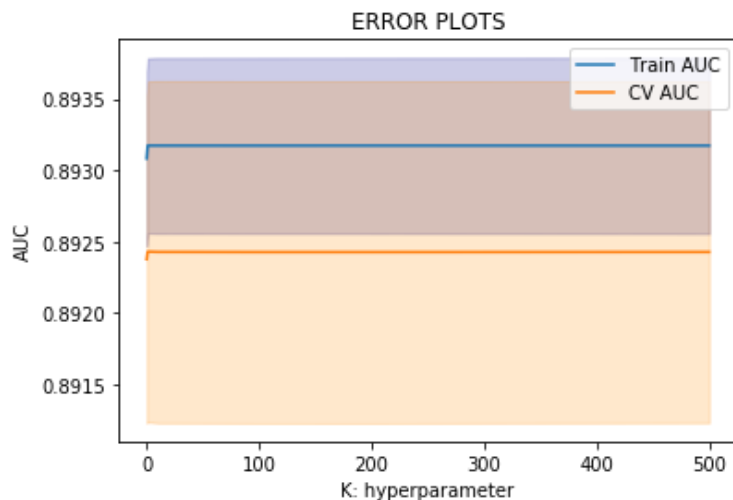
```
lr = LogisticRegression(penalty='l1')
parameters = [{ 'C':[ 0.1, 1, 10, 100,200,500]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(tfidf_sent_vectors, y_train)

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 = [0.1, 1, 10, 100,200,500]

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,color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



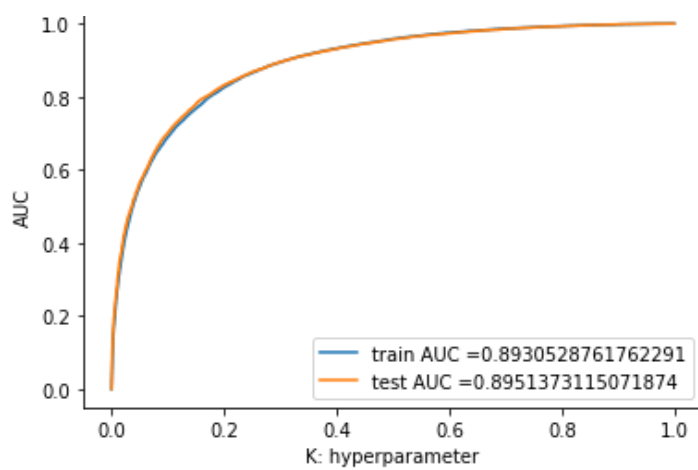
In [330]:

```
lr = LogisticRegression(penalty='l1',C=1)
lr.fit(tfidf_sent_vectors, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,lr.predict_proba(tfidf_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

ERROR PLOTS



In [331]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(sent_vectors_train))
y=confusion_matrix(y_test, lr.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)
```

```
import seaborn as sn
```

```
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()
```

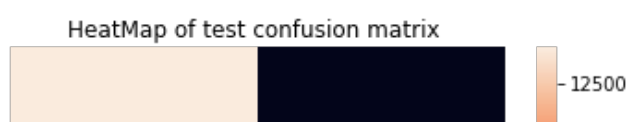
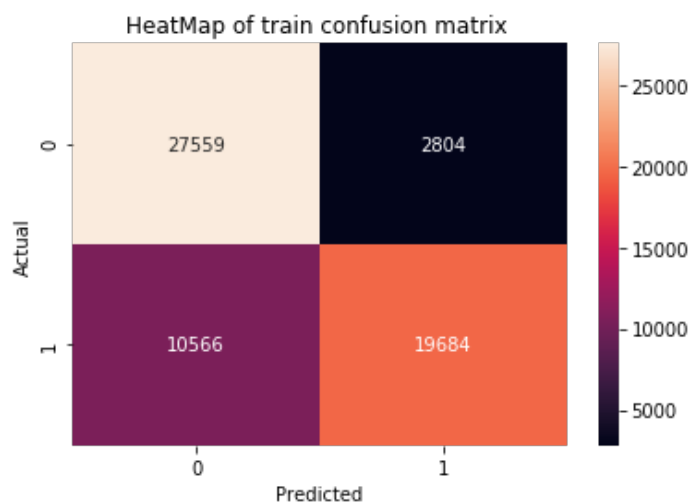
```
ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

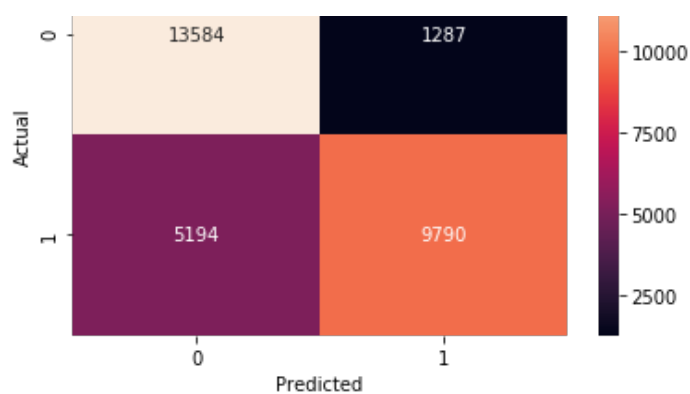
Train confusion matrix

```
[[27559  2804]
 [10566 19684]]
```

Test confusion matrix

```
[[13584  1287]
 [ 5194  9790]]
```





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

In [333]:

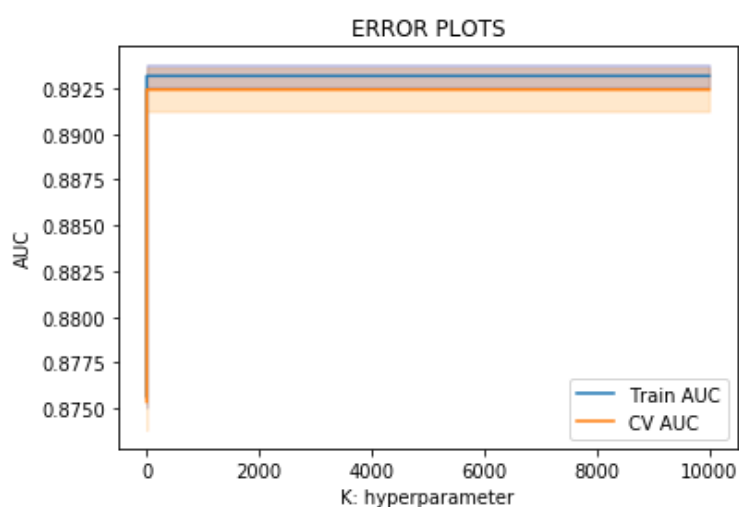
```
lr = LogisticRegression(penalty='l2')
parameters = [{ 'C': [0.0001, 0.001, 0.01, 0.1, 1, 2, 5, 8, 10, 20, 50, 75, 100, 1000, 10000]}]
clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(tfidf_sent_vectors, y_train)

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, 2, 5, 8, 10, 20, 50, 75, 100, 1000, 10000]

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, color='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='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [334]:

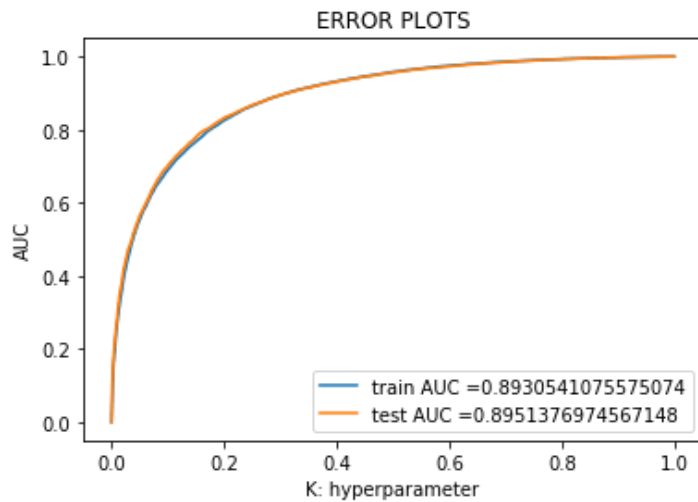
```
lr = LogisticRegression(penalty='l2', C=1)
lr.fit(tfidf_sent_vectors, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, lr.predict_proba(tfidf_sent_vectors))
```



```
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,lr.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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [335]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x=confusion_matrix(y_train, lr.predict(sent_vectors_train))
y=confusion_matrix(y_test, lr.predict(sent_vectors_test))
print(x)
print("Test confusion matrix")
print(y)
```

```
import seaborn as sn
```

```
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()
```

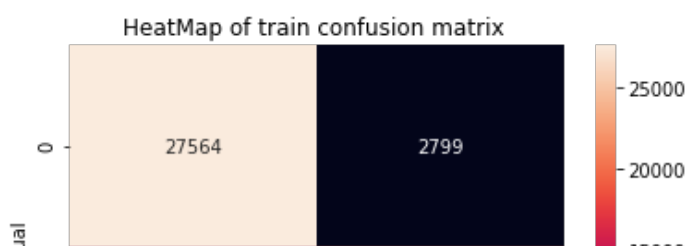
```
ax = plt.axes()
sns.heatmap(y, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```

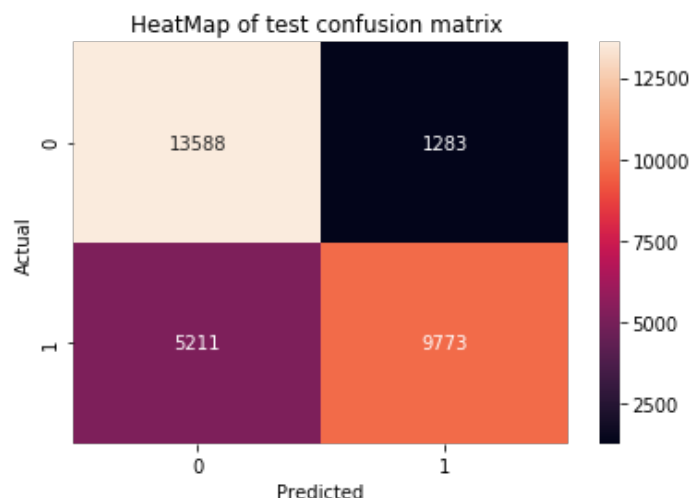
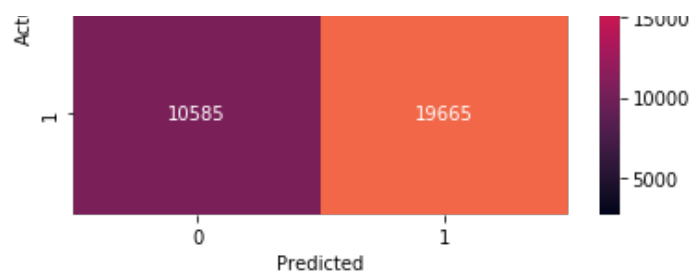
Train confusion matrix

```
[[27564  2799]
 [10585 19665]]
```

Test confusion matrix

```
[[13588  1283]
 [ 5211  9773]]
```





[6] Conclusions

In [339]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Vectorizer", "Hyper Parameter(C)", "Regularizer", "AUC"]
x.add_row(["Logistic Regression", "BOW", "0.4", "L-1", "0.9464"])
x.add_row(["Logistic Regression", "BOW", "0.1", "L-2", "0.9480"])
x.add_row(["Logistic Regression", "TF-IDF", "2", "L-1", "0.9651"])
x.add_row(["Logistic Regression", "TF-IDF", "4", "L-2", "0.9676"])
x.add_row(["Logistic Regression", "AVG W2V", "1", "L-1", "0.9191"])
x.add_row(["Logistic Regression", "AVG W2V", "0.1", "L-2", "0.9191"])
x.add_row(["Logistic Regression", "TFIDF W2V", "1", "L-1", "0.8951"])
x.add_row(["Logistic Regression", "TFIDF W2V", "1", "L-2", "0.8951"])

print(x)
print("\n")
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(C)", "Regularizer", "AUC"]
y.add_row(["Logistic Regression", "BOW", "0.4", "L-1", "0.9464"])
print(y)
```

Model	Vectorizer	Hyper Parameter(C)	Regularizer	AUC
Logistic Regression	BOW	0.4	L-1	0.9464
Logistic Regression	BOW	0.1	L-2	0.9480
Logistic Regression	TF-IDF	2	L-1	0.9651
Logistic Regression	TF-IDF	4	L-2	0.9676
Logistic Regression	AVG W2V	1	L-1	0.9191
Logistic Regression	AVG W2V	0.1	L-2	0.9191
Logistic Regression	TFIDF W2V	1	L-1	0.8951
Logistic Regression	TFIDF W2V	1	L-2	0.8951

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

Model	Vectorizer	Hyper Parameter(C)	Regularizer	AUC
Logistic Regression	BOW	0.4	L-1	0.9464

Observations

Adding additional feature review length does not seem to affect the classifier much the AUC score of the model was observed to remain the same upon adding the feature

L-1 Regularization Created Sparsity, BOW vectorized model with L1 Regularization resulted in a weight vector with 3369 non zero values out of 48209 values

Feature importance could be observed using the coefficients of weight vector

The behaviour of the models corresponding to all the featurizations was ideal wheras TFIDF W2V featurized models had a weaker Negative class performance having higher false negative and lower true negative values compared to other featurizations