Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [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 500000""", co
n)
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (400000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
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	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577

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

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

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

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

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

In [9]:

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

In [10]:

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

Out[10]:

```
(286837, 10)
```

In [11]:

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

Out[11]:

```
71.70925
```

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

In [12]:

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

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883

In [13]:

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

In [14]:

```
#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[14]:

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

segragating datapoints w.r.t calss labels

In [15]:

```
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 [16]:

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

```
(45234, 10)
```

In [17]:

```
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 [18]:

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

In [19]:

```
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
```

continued from 1.11

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [21]:

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

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

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

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

```
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 min
e introduced me to Wilton Ready To Use Icing, I was hooked! I couldn't believe that an icing so pe
rfect for decorating and so creamy could taste so good! In my opinion, it's not crisco-y at all co
mpared to most recipes made with shortening. It's sweet but not too sweet to over power 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 f
or a reasonably priced food which was better in quality than the cheap chain crap. I didn't want t
o pay $2 a pound for a high quality food and Canidae was just right for my budget and quality need
s. 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 men
tioned.) 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 hi
gh quality food at a reasonable price so if anyone knows of something high quality at around the s
ame price point please respond to this post and let me know.<br /><br />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 a
nd 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 avoi
d 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 s
ingle problem with this food new or old formula.
==================================================
This container is medium size but I like the size and it was convient for my to purchase this item
because I could only find small container at the local grocery store.  I love to cook with beef fl
avored bouillon so it was a great purchase for me.
==================================================
```

In [22]:

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

```
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
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.
```

In [23]:

```
# 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)
```

```
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 min
e introduced me to Wilton Ready To Use Icing, I was hooked! I couldn't believe that an icing so pe
rfect for decorating and so creamy could taste so good! In my opinion, it's not crisco-y at all co
mpared to most recipes made with shortening. It's sweet but not too sweet to over power 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 f
or a reasonably priced food which was better in quality than the cheap chain crap. I didn't want t
o pay $2 a pound for a high quality food and Canidae was just right for my budget and quality need
s. 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 men
tioned.) 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 hi
gh quality food at a reasonable price so if anyone knows of something high quality at around the s
ame price point please respond to this post and let me know.On a different note, was I the only pe
rson who actually DID hear about the formula change before it was too late? When I went to purchas
e 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 c
hange, and the proper way to transition your dog from the old to the new food to avoid digestive i
ssues. As I had 1 full bag of old formula left, I followed the instructions and guess what, no iss
ues whatsoever with the transition. My dog did NOT get sick at all and has not had a single proble
m with this food new or old formula.
==================================================
This container is medium size but I like the size and it was convient for my to purchase this item
because I could only find small container at the local grocery store.  I love to cook with beef fl
avored bouillon so it was a great purchase for me.
```

In [20]:

```
# 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', "y
ou'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', 'e
ach', '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', "d
esn'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"])


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

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

```
100%|████████████████████████████████████████████████████████████████████████| 90468/90468
[00:39<00:00, 2278.65it/s]
100%|████████████████████████████████████████████████████████████████████████| 90468/90468
[00:00<00:00, 1886376.51it/s]
```

```
90468
```

In [21]:

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

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

In [21]:

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-db217a501677> in <module>()
----> 1 sent_1500 = decontracted(sent_1500)
      2 print(sent_1500)
      3 print("="*50)

NameError: name 'sent_1500' is not defined
```

In [0]:

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

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victor  a
nd  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 bef
ore 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 ch
ocolate flavor and gives the cookie sort of a coconut type consistency Now let is also remember th
at 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 c
ookie 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 indivi
dually 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 Nabiso 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', "y
ou'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', 'e
ach', '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', "d
esn'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 stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

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

In [0]:

```
preprocessed_reviews[1500]
```

Out[0]:

```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry review
s 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 cook
ies 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 wa
nt something hard crisp suggest nabiso ginger snaps want cookie soft chewy tastes like combination
chocolate oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

In [0]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [0]:

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

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

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

[4.2] Bi-Grams and n-Grams.

In [0]:

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

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-
learn.org/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_counts.get_s
hape()[1])
```

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

[4.3] TF-IDF

In [0]:

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

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

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

[4.4] Word2Vec

In [0]:

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

In [0]:

```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful',
0.9946032166481018), ('excellent', 0.9944332838058472), ('especially', 0.9941144585609436),
('baked', 0.9940600395202637), ('salted', 0.994047224521637), ('alternative', 0.9937226176261902),
('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn',
0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.9992451071739197), ('melitta',
0.999218761920929), ('choice', 0.9992102384567261), ('american', 0.9991837739944458), ('beef',
0.9991780519485474), ('finish', 0.9991567134857178)]
```

In [0]:

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

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

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

[4.4.1.1] Avg W2v

In [0]:

```
# average Word2Vec
```

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

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

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:

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

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

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

[5] Assignment 4: Apply Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)

2. **The hyper paramter tuning(find best Alpha)**

   - Find the best hyper parameter which will give the maximum AUC value
   - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

   - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB and print their corresponding feature names

4. **Feature engineering**

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

5. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

6. **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 Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

In [22]:

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

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

In [23]:

```
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 [24]:

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

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

In [25]:

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

```
After vectorizations
(60613, 48155) (60613,)
(29855, 48155) (29855,)
```

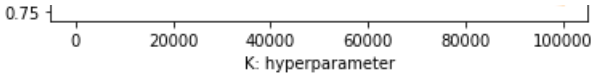
In [26]:

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

```
<class 'scipy.sparse.csr.csr_matrix'>
(60613, 48155)
```

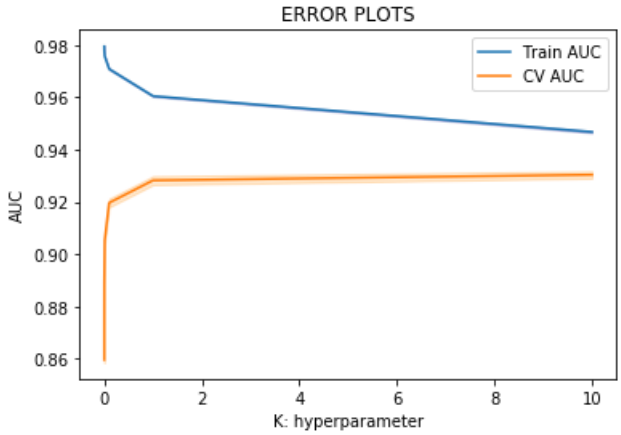
In [27]:

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

mnb = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000,100000]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000,100000]


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='darkb
lue')

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]:

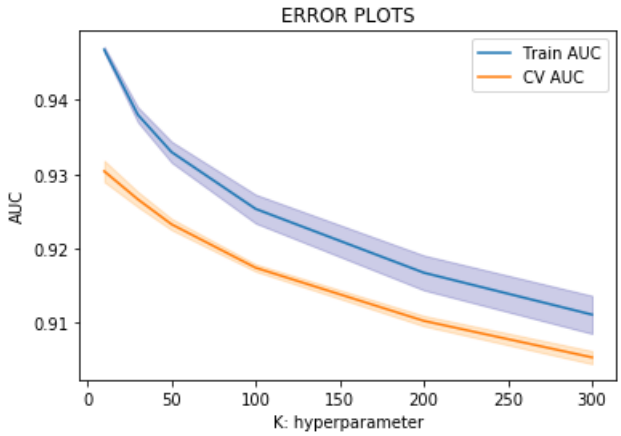
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [0.00001,0.0001, 0.001, 0.01, 0.1, 1, 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='darkb
lue')

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 [29]:

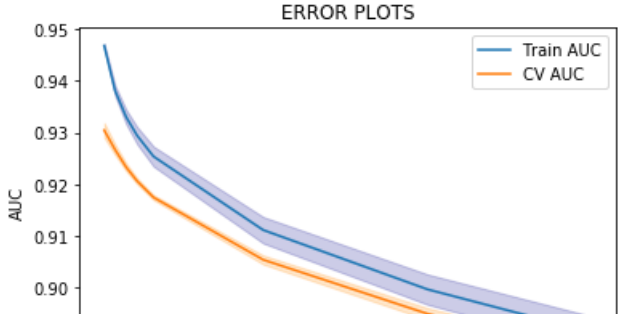
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[10,30,50,100,200,300]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [10,30,50,100,200,300]


plt.plot(K, train_auc, label='Train AUC')
```

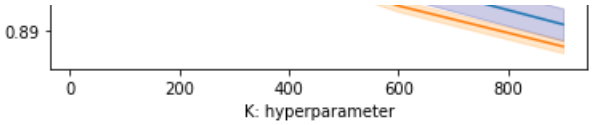
```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkb
lue')

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 [30]:

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

mnb = MultinomialNB()
parameters = {'alpha':[10,30,50,70,100,300,600,900]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [10,30,50,70,100,300,600,900]


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='darkb
lue')

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()
```

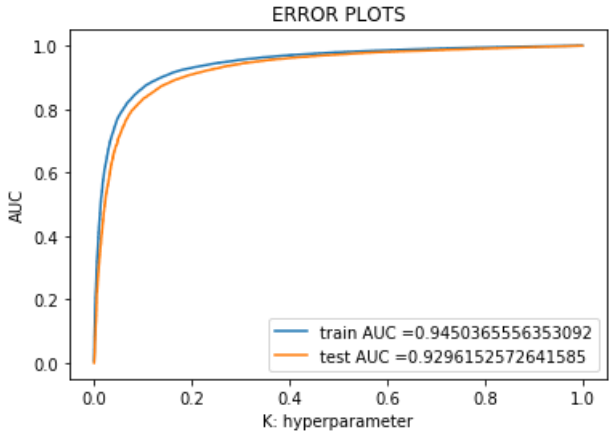




Choosing Hyper Parameter 10

In [31]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_bow, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,mnb.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,mnb.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 [32]:

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

```
Train confusion matrix
[[27184  3112]
 [ 3907 26410]]
Test confusion matrix
[[13159  1779]
 [ 2200 12717]]
```

In [33]:

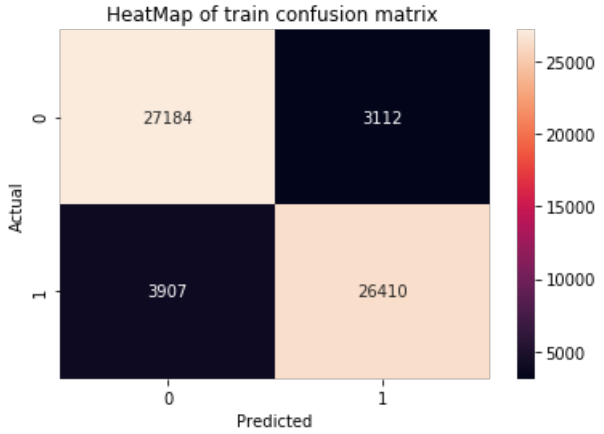
```
print(x.shape)
print(type(x))
print(y.shape)
print(type(y))
```

```
(2, 2)
<class 'numpy.ndarray'>
(2, 2)
```

<class 'numpy.ndarray'>

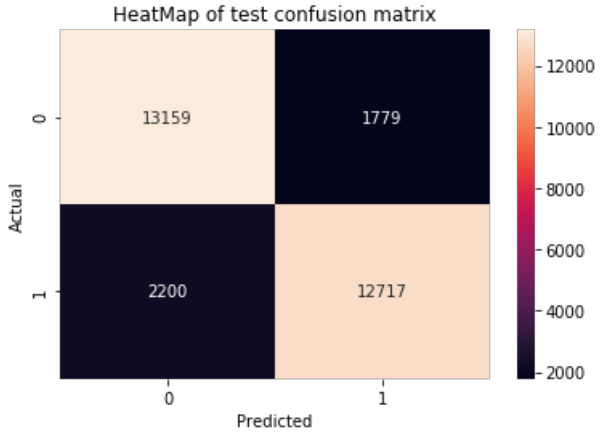
In [34]:

```
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 [35]:

```
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()
```



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

In [1]:

```
# Code refrence from https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in
-naive-bayes
```

In [61]:

```
from sklearn.naive_bayes import MultinomialNB
vectorizer = CountVectorizer()
vectorizer.fit(X_train)

X_train_bow = vectorizer.transform(X_train)
```

```
X_test_bow = vectorizer.transform(X_test)

mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_bow, y_train)



neg_class_prob_sorted=mnb.feature_log_prob_[0, :].argsort()
pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()


print("=========== Positive Most Important Features ====================")
#print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[:10]))
print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[-10:]))
```

```
=========== Positive Most Important Features ====================
['would' 'love' 'coffee' 'product' 'taste' 'one' 'great' 'good' 'like'
 'not']
```

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

In [62]:

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

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

mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_bow, y_train)


neg_class_prob_sorted=mnb.feature_log_prob_[0, :].argsort()
pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()



print("===========negative most important features====================")

print(np.take(vectorizer.get_feature_names(), neg_class_prob_sorted[-10:]))
```

```
===========negative most important features====================
['flavor' 'coffee' 'no' 'good' 'one' 'taste' 'would' 'product' 'like'
 'not']
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [0]:

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

In [39]:

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

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

In [40]:

```
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 [41]:

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

Out[41]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=10,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [42]:

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

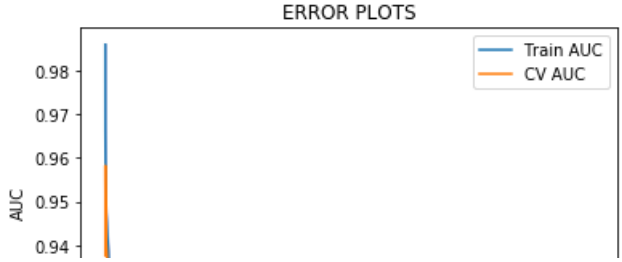
In [43]:

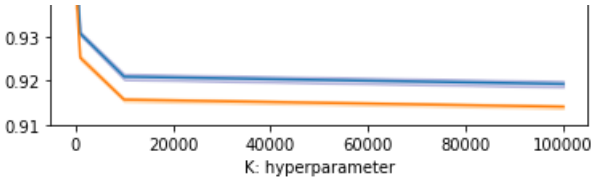
```
print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(60613, 36270)
(29855, 36270)
```

In [44]:

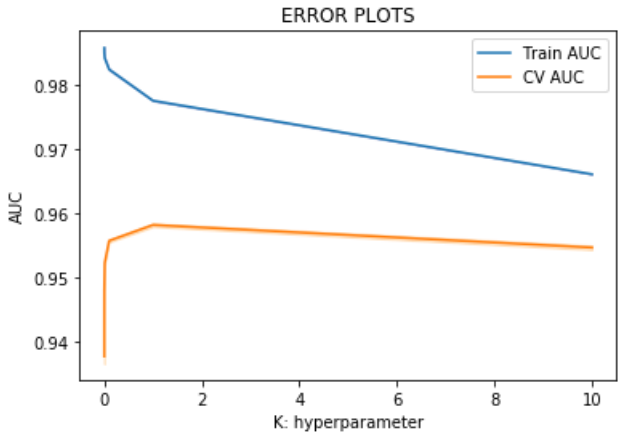
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000,100000]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000,100000]


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='darkb
lue')

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 [45]:

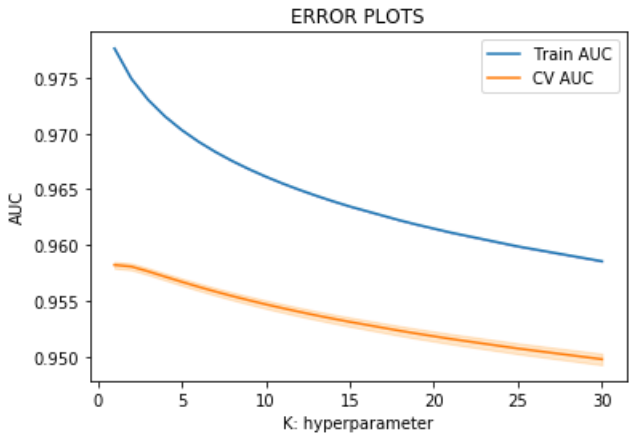
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [0.00001,0.0001, 0.001, 0.01, 0.1, 1, 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='darkb
lue')

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 [46]:

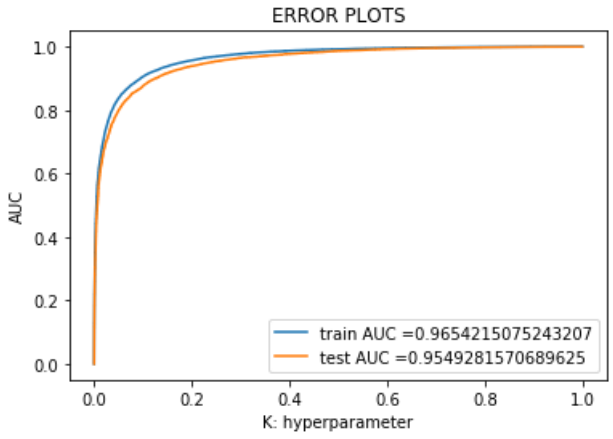
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,19,21,25,30]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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, 11, 12, 13, 14, 15, 18, 19, 21, 25, 30]
```

```
K = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,19,21,25,30]


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='darkb
lue')

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 [47]:

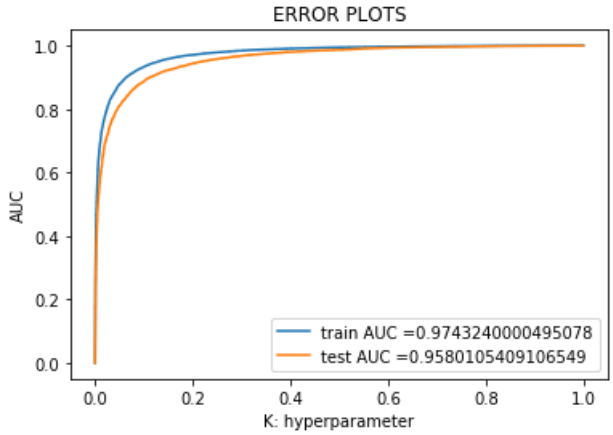
```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_tfidf, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,mnb.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,mnb.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 [48]:

```
from sklearn.metrics import roc_curve, auc
```

```
mnb = MultinomialNB()
mnb.fit(X_train_tfidf, y_train)


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



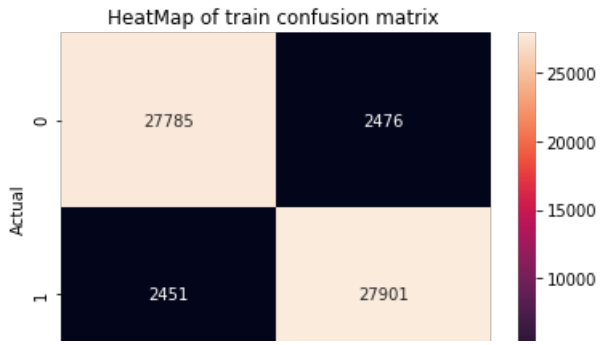
In [50]:

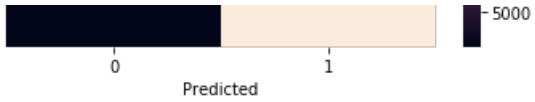
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
x1=confusion_matrix(y_train, mnb.predict(X_train_tfidf))
print(x1)
print("Test confusion matrix")
y1=confusion_matrix(y_test, mnb.predict(X_test_tfidf))
print(y1)
```

```
Train confusion matrix
[[27785  2476]
 [ 2451 27901]]
Test confusion matrix
[[13436  1537]
 [ 1605 13277]]
```

In [51]:

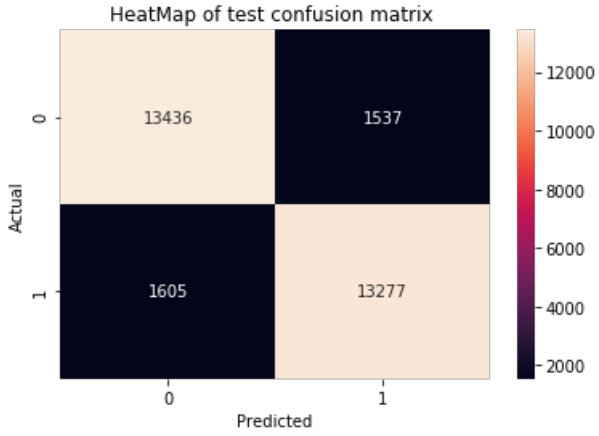
```
ax = plt.axes()
sns.heatmap(x1, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of train confusion matrix ")
plt.show()
```





In [52]:

```
ax = plt.axes()
sns.heatmap(y1, ax = ax,annot=True, fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
ax.set_title("HeatMap of test confusion matrix ")
plt.show()
```



In [53]:

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

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

In [78]:

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

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 [79]:

```
from sklearn.naive_bayes import MultinomialNB
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)

mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_tfidf, y_train)



neg_class_prob_sorted=mnb.feature_log_prob_[0, :].argsort()
pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()
```

```
print("========== Positive Most Important Features ====================")
#print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[:10]))
print(np.take(tf_idf_vect.get_feature_names(), pos_class_prob_sorted[-10:]))
```

```
========== Positive Most Important Features ====================
['taste' 'product' 'one' 'tea' 'coffee' 'like' 'love' 'good' 'great' 'not']
```

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

In [0]:

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

In [80]:

```
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)

mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_tfidf, y_train)


neg_class_prob_sorted=mnb.feature_log_prob_[0, :].argsort()
pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()



print("==========negative most important features====================")

print(np.take(tf_idf_vect.get_feature_names(), neg_class_prob_sorted[-10:]))
```

```
==========negative most important features====================
['good' 'no' 'flavor' 'one' 'coffee' 'would' 'taste' 'product' 'like'
 'not']
```

[6] Conclusions

In [0]:

```
#https://stackoverflow.com/questions/41937786/add-column-to-a-sparse-matrix
```

[6]Feature Engineering Adding review Length

Calculating The length of each review

In [54]:

```
print(type(preprocessed_reviews))
review_length_train=[]

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

print(review_length_train[0])
```

```
<class 'list'>
235
```

In [55]:

```
print(len(review_length_train))
print(len(X_train))
```

```
60613
60613
```

In [56]:

```
review_length_test=[]

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

print(review_length_test[0])
```

```
416
```

In [57]:

```
print(len(review_length_test))
print(len(X_test))
```

```
29855
29855
```

In [58]:

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

```
<class 'scipy.sparse.csr.csr_matrix'>
(60613, 48155)
```

In [59]:

```
review_length_train=np.asarray(review_length_train)
```

Adding length of reviews to the data

In [60]:

```
#Code refrenced from below URL
#https://stackoverflow.com/questions/41937786/add-column-to-a-sparse-matrix

from scipy import sparse

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

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

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

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

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

```
<class 'scipy.sparse.coo.coo_matrix'>
(60613, 48156)
<class 'scipy.sparse.coo.coo_matrix'>
(29855, 48156)
<class 'scipy.sparse.csr.csr_matrix'>
(29855, 48155)
```

In [61]:

```
z=X_test_bow_f1.toarray()
```

In [62]:

```
print(z[0])
```

```
[  0   0   0 ...   0   0 416]
```

In [63]:

```
X_train_bow
```

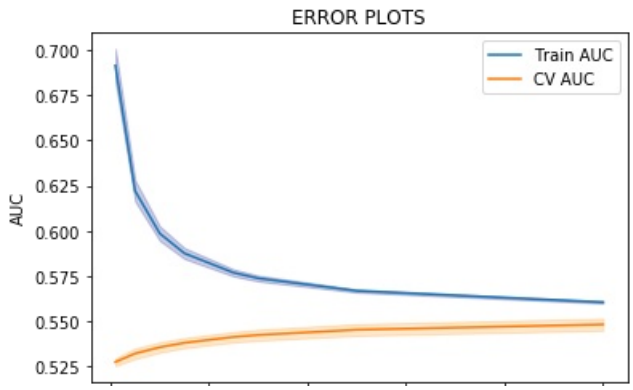
Out[63]:

```
<60613x48155 sparse matrix of type '<class 'numpy.int64'>'
 with 2048076 stored elements in Compressed Sparse Row format>
```

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

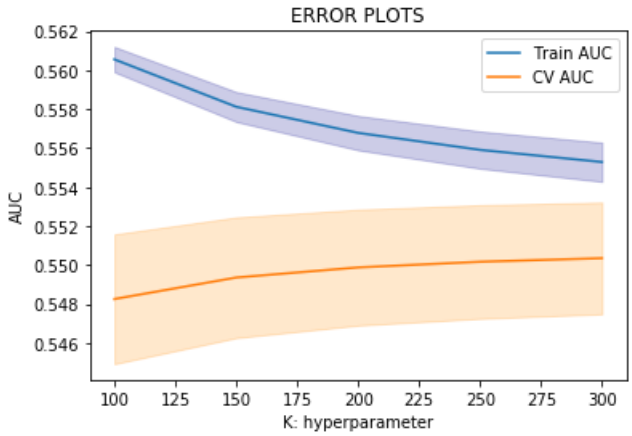
In [64]:

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

mnb = MultinomialNB()
parameters = {'alpha':[1,5,10,15,25,30,50,100]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [1,5,10,15,25,30,50,100]


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='darkb
lue')

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()
```



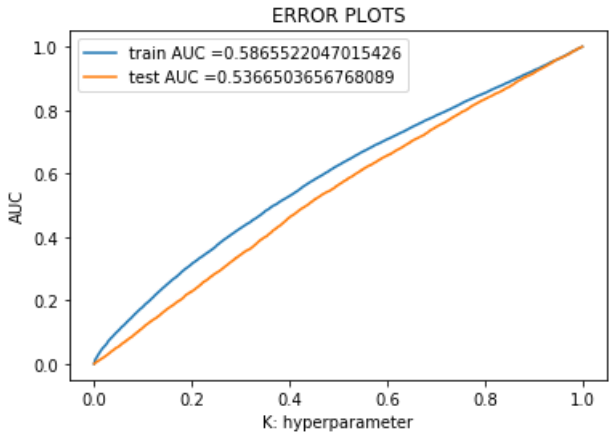
In [65]:

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

mnb = MultinomialNB()
parameters = {'alpha':[100,150,200,250,300]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
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 = [100,150,200,250,300]


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='darkb
lue')

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 [66]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_bow_f1, y_train)


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



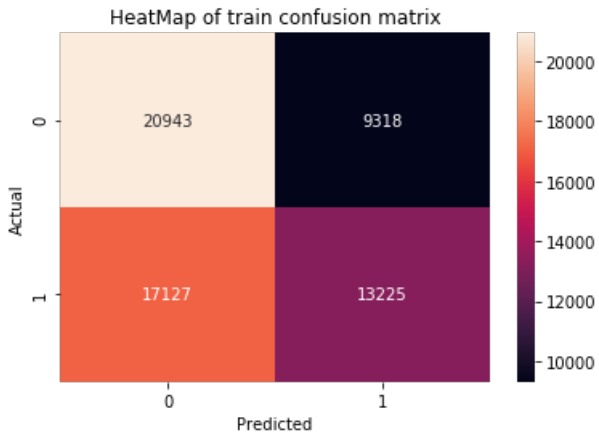
In [67]:

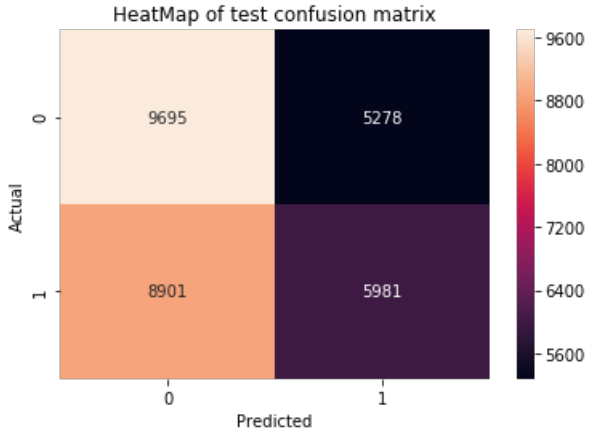
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

```
Train confusion matrix
[[20943  9318]
 [17127 13225]]
Test confusion matrix
[[9695 5278]
 [8901 5981]]
```

In [68]:

```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```

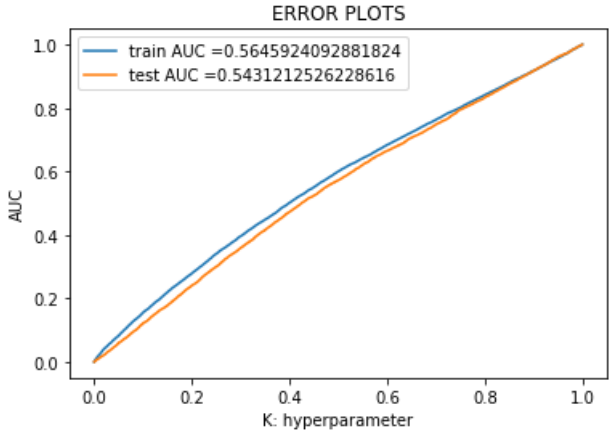




In [69]:

In [69]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=40)
mnb.fit(X_train_bow_f1, y_train)


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



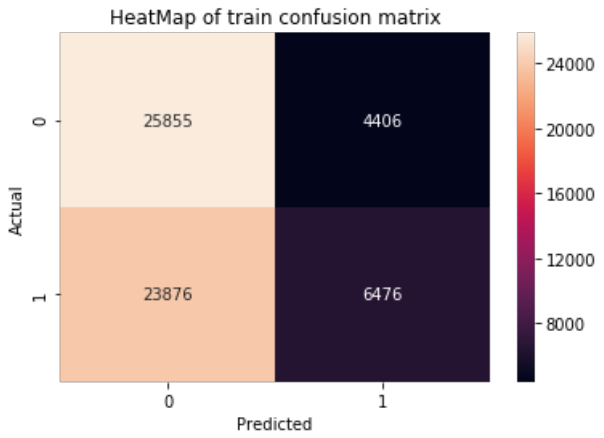
In [70]:

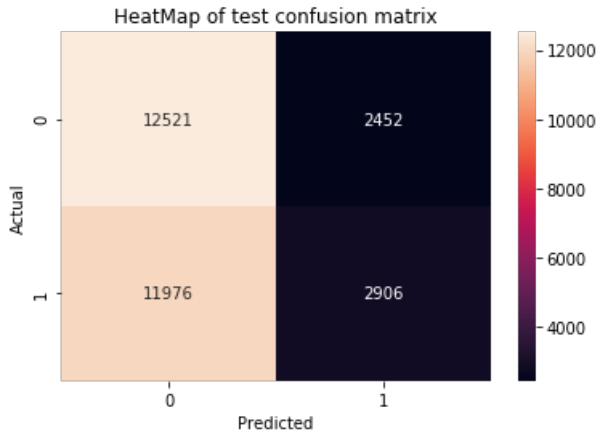
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

```
Train confusion matrix
[[25855  4406]
 [23876  6476]]
Test confusion matrix
[[12521  2452]
 [11976  2906]]
```

In [71]:

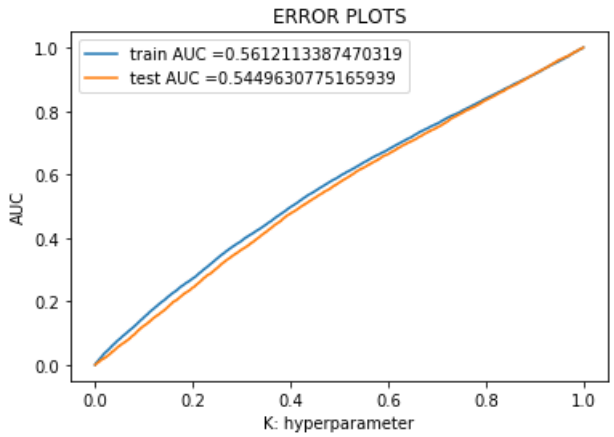
```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```





In [72]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=60)
mnb.fit(X_train_bow_f1, y_train)


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



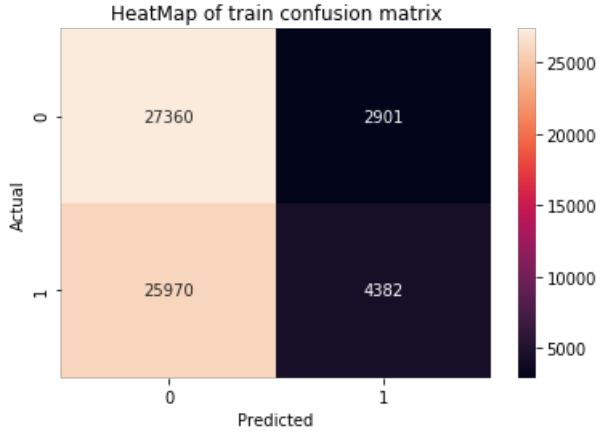
In [74]:

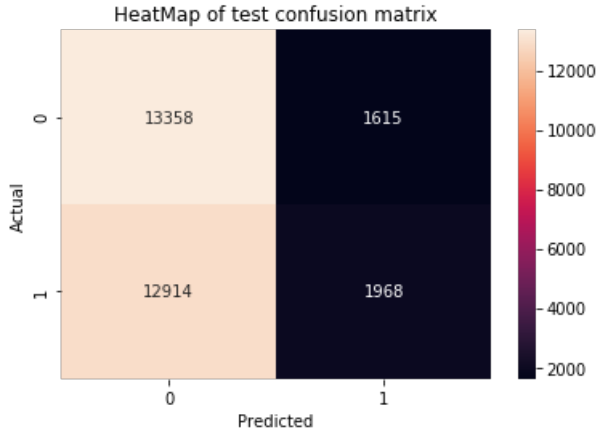
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

```
Train confusion matrix
[[27360  2901]
 [25970  4382]]
Test confusion matrix
[[13358  1615]
 [12914  1968]]
```

In [75]:

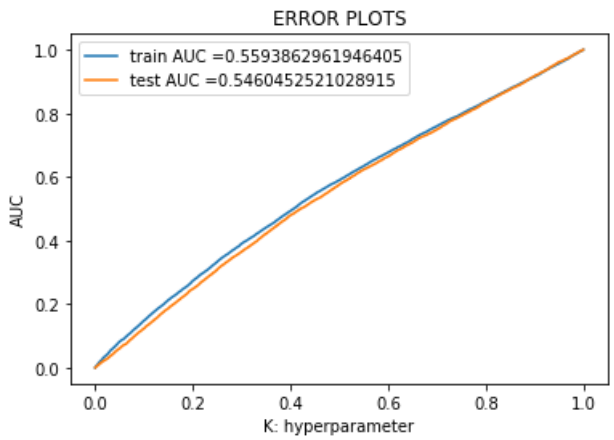
```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```





In [76]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=80)
mnb.fit(X_train_bow_f1, y_train)


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



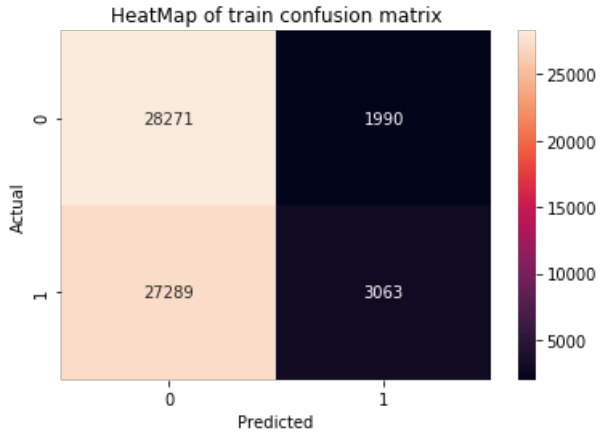
In [77]:

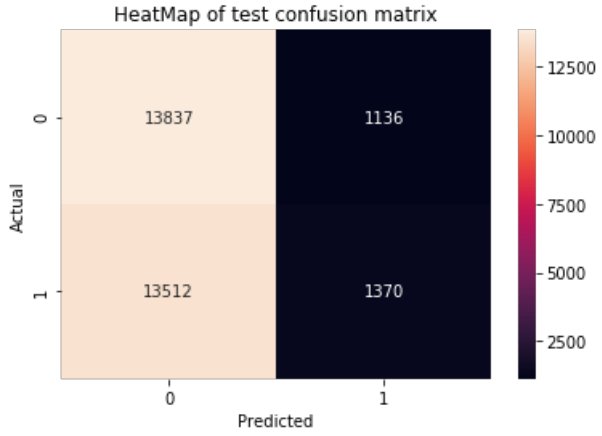
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

```
Train confusion matrix
[[28271  1990]
 [27289  3063]]
Test confusion matrix
[[13837  1136]
 [13512  1370]]
```

In [78]:

```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```

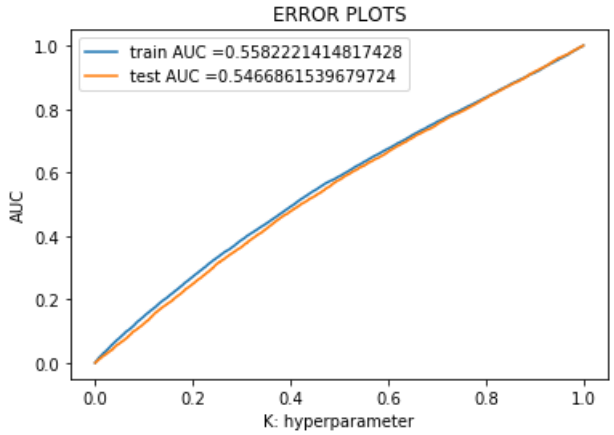




In [79]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=100)
mnb.fit(X_train_bow_f1, y_train)


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



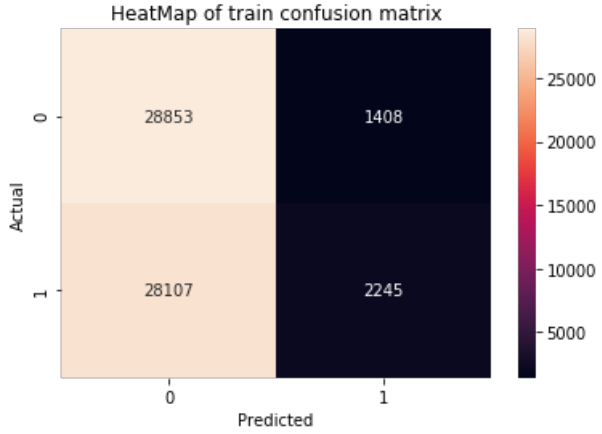
In [80]:

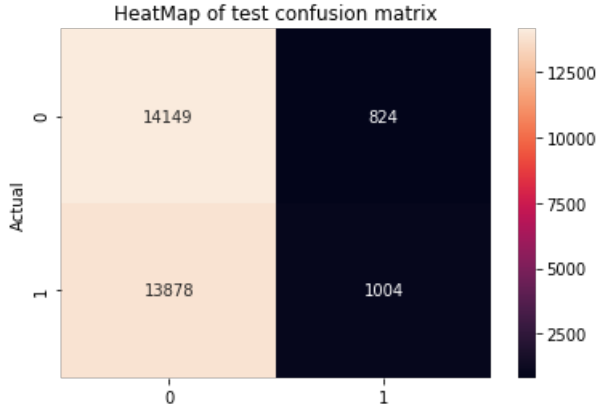
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

```
Train confusion matrix
[[28853  1408]
 [28107  2245]]
Test confusion matrix
[[14149   824]
 [13878  1004]]
```

In [81]:

```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```

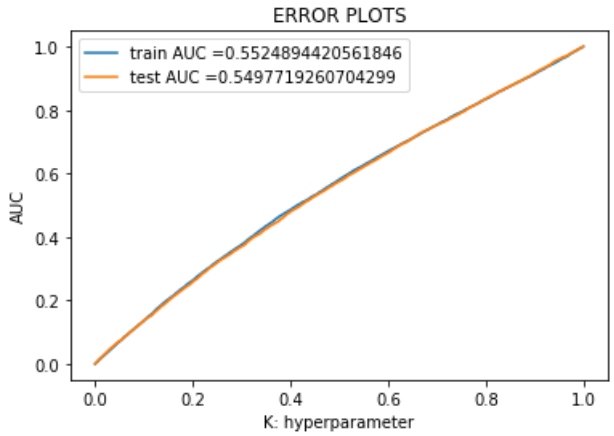




Predicted

In [82]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=1000)
mnb.fit(X_train_bow_f1, y_train)


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



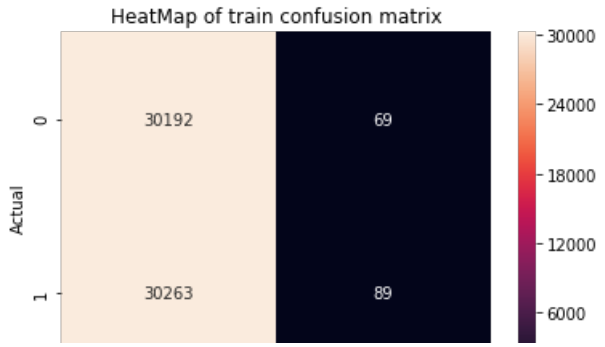
In [83]:

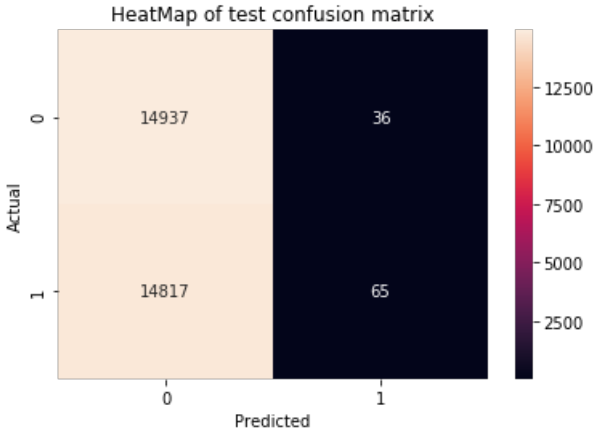
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

```
Train confusion matrix
[[30192    69]
 [30263    89]]
Test confusion matrix
[[14937    36]
 [14817    65]]
```

In [84]:

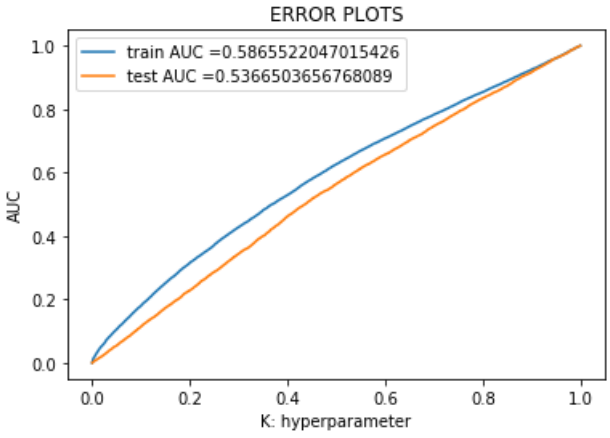
```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```





In [85]:

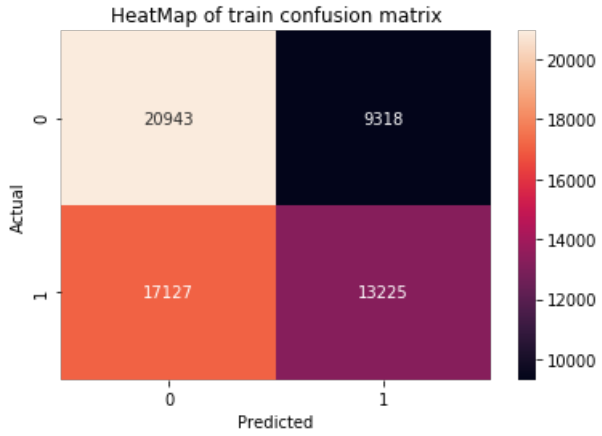
```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_bow_f1, y_train)


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



from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)))
```

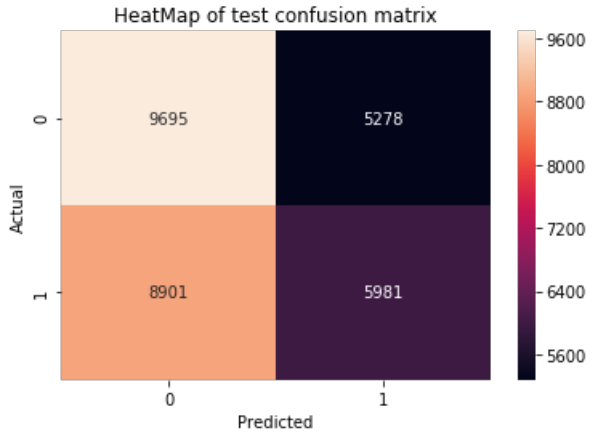



```
Train confusion matrix
[[20943  9318]
 [17127 13225]]
Test confusion matrix
[[9695 5278]
 [8901 5981]]
```

In [86]:

```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_bow_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_bow_f1)),"test")
```





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

In [87]:

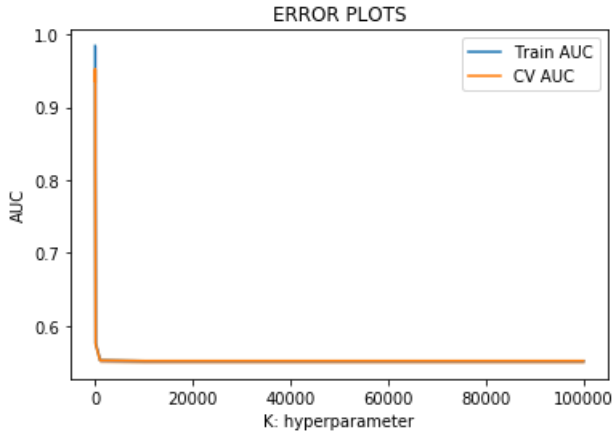
```
from scipy import sparse

X_train_tfidf_f1=sparse.hstack((X_train_tfidf,np.array(review_length_train)[:,None]))

print(type(X_train_tfidf_f1))
print(X_train_tfidf_f1.shape)

X_test_tfidf_f1=sparse.hstack((X_test_tfidf,np.array(review_length_test)[:,None]))

print(type(X_test_tfidf_f1))
print(X_test_tfidf_f1.shape)
```

```
<class 'scipy.sparse.coo.coo_matrix'>
(60613, 36271)
<class 'scipy.sparse.coo.coo_matrix'>
(29855, 36271)
```

In [88]:

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

mnb = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000,100000]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
```

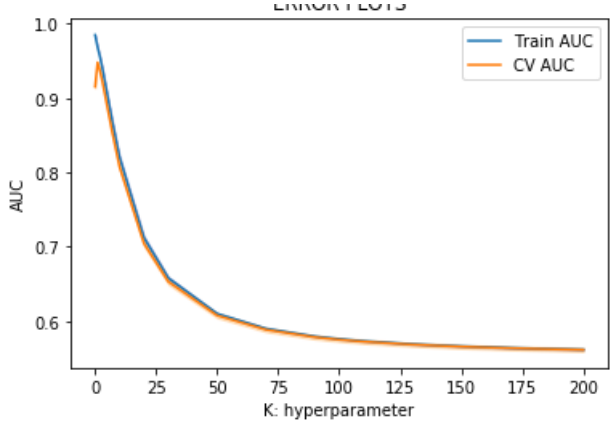
```
clf.fit(X_train_tfidf_f1,y_train)

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

K = [0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100,1000,10000,100000]


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='darkb
lue')

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 [89]:

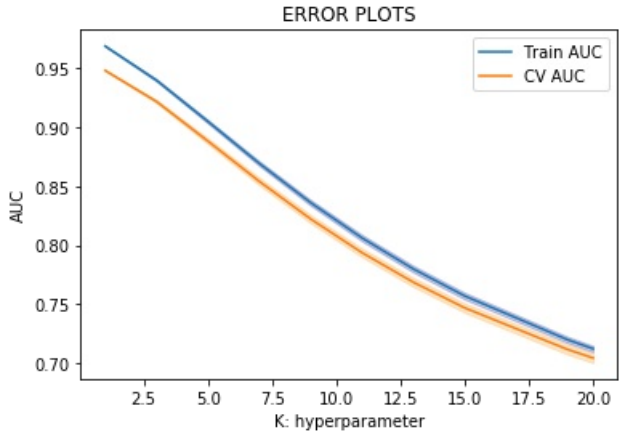
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[0, 1, 2,3,7,10,20,30,50,70,90,100,110,130,150,170,200]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf_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 = [0.0, 1, 2,3,7,10,20,30,50,70,90,100,110,130,150,170,200]


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='darkb
lue')

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 [90]:

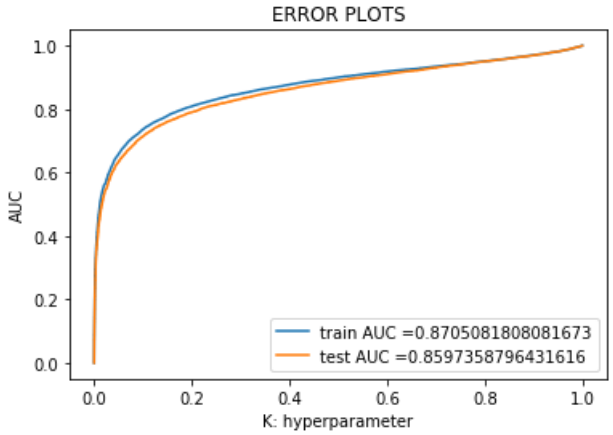
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

mnb = MultinomialNB()
parameters = {'alpha':[1,3,5,7,9,11,13,15,19,20]}
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf_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 = [1,3,5,7,9,11,13,15,19,20]


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='darkb
lue')

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 [91]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=10)
mnb.fit(X_train_tfidf_f1, y_train)
```

```
train_fpr, train_tpr, thresholds = roc_curve(y_train,mnb.predict_proba(X_train_tfidf_f1)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,mnb.predict_proba(X_test_tfidf_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()
```



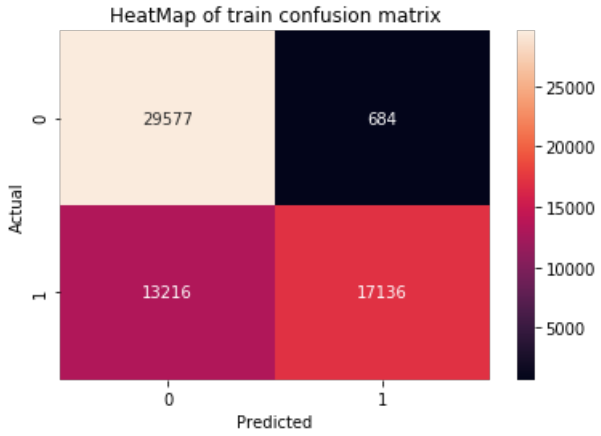
In [92]:

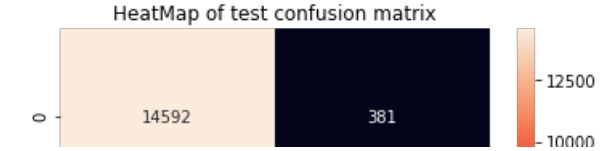
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_tfidf_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_tfidf_f1)))
```

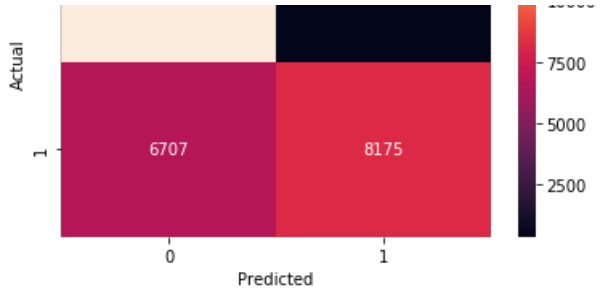
```
Train confusion matrix
[[29577   684]
 [13216 17136]]
Test confusion matrix
[[14592   381]
 [ 6707  8175]]
```

In [93]:

```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_tfidf_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_tfidf_f1)),"test")
```

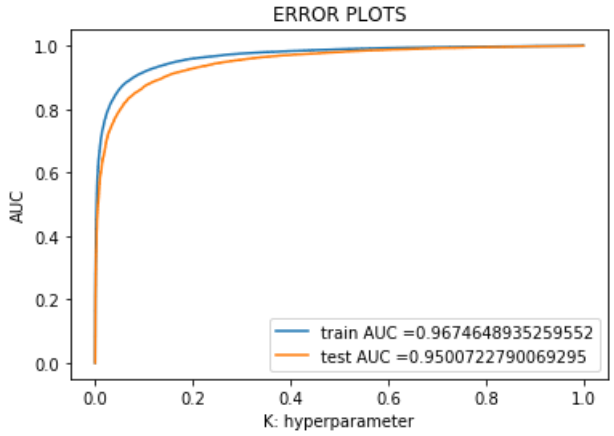






In [94]:

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB()
mnb.fit(X_train_tfidf_f1, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,mnb.predict_proba(X_train_tfidf_f1)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,mnb.predict_proba(X_test_tfidf_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()
```



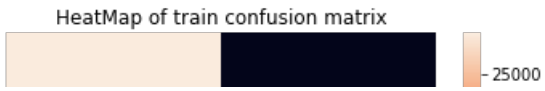
In [95]:

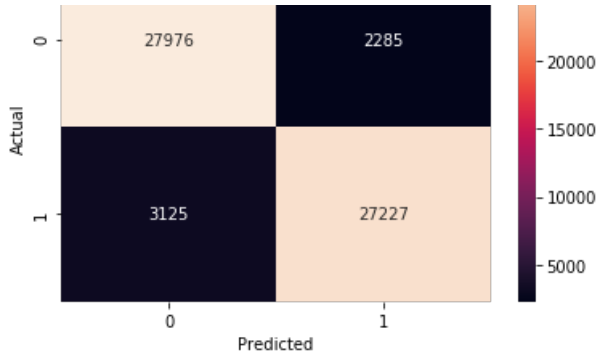
```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_tfidf_f1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_tfidf_f1)))
```

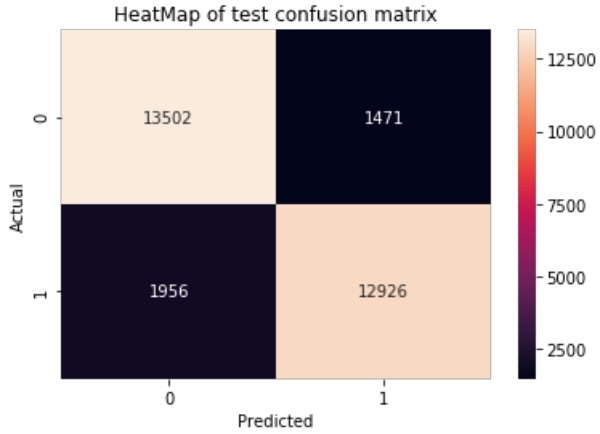
```
Train confusion matrix
[[27976  2285]
 [ 3125 27227]]
Test confusion matrix
[[13502  1471]
 [ 1956 12926]]
```

In [96]:

```
plotheatmap(confusion_matrix(y_train, mnb.predict(X_train_tfidf_f1)),"train")
plotheatmap(confusion_matrix(y_test, mnb.predict(X_test_tfidf_f1)),"test")
```







[7] Conclusions

In [150]:

```
from prettytable import PrettyTable


x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter(α)", "AUC"]
x.add_row(["BOW","NaiveBayes","10","0.931"])
x.add_row(["TFIDF","NaiveBayes","1","0.958"])



print(x)


print("Feature engineered output after adding review length to vectorized data:=")

y = PrettyTable()
y.field_names = ["Vectorizer", "Model", "Hyper Parameter(α)", "AUC"]
y.add_row(["BOW with review length","NaiveBayes","10","0.541"])
y.add_row(["TFIDF with review length","NaiveBayes","1","0.951"])



print(y)
```

```
+------------+------------+--------------------+-------+
| Vectorizer |   Model    | Hyper Parameter(α) |  AUC  |
+------------+------------+--------------------+-------+
|    BOW     | NaiveBayes |         10         | 0.931 |
|   TFIDF    | NaiveBayes |         1          | 0.958 |
+------------+------------+--------------------+-------+
Feature engineered output after adding review length to vectorized data:=
+--------------------------+------------+--------------------+-------+
|        Vectorizer        |   Model    | Hyper Parameter(α) |  AUC  |
+--------------------------+------------+--------------------+-------+
|  BOW with review length  | NaiveBayes |         10         | 0.541 |
| TFIDF with review length | NaiveBayes |         1          | 0.951 |
+--------------------------+------------+--------------------+-------+
```

Observation

Adding additional feature review length to BOW vectorized data and applying naive bayes does not seem to help with AUC Score the AUC score reduces with the addition of the feature

Adding additional feature review length to TFIDF vextorized data does not seem to have affected the AUC scores as it remains similar to score for data without additional feature