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 [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
```

```python
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 [26]:

```python
# using SQLite Table to read data.
```

```python
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 500
000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 11500
0""", 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 (115000, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summa
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality D Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertise
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [27]:

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

In [28]:

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

```
(80668, 7)
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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 [29]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

	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 [30]:
```
display['COUNT(*)'].sum()
```

```
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 [31]:
```
display= pd.read_sql_query("""
SELECT *
```

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

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

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 [32]:
```
#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 [33]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='f
irst', inplace=False)
final.shape
```

```
(99724, 10)
```

In [34]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
86.71652173913044
```

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 [35]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure co taste wi crunchy almonds inside

In [36]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [37]:
```
#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()
```

```
(99722, 10)


1    83711
0    16011
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [38]:

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont take any chances till they know what is going on with the china imports.
==================================================
IF YOU LIKE SALMON YOU WILL LOVE THESE OMAHA STEAKS SALMON VERY VERY GOOD
==================================================
OK....I thought I'd put a bit of punch to hubby's sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!<br />One bite and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!<br />I guess it was that-bad!<br />I'll see if my neighbor will be able to use it w/her family.<br />If you  are a BEST FOODS lover---walk away---do NOT purchase this product!
==================================================
These people from Bavaria really know how to make this stuff. The Landjagers are super (you have to let them dry for them to develop their full, intended flavor), and it is worth any sausage fan's time to check out their complete offering of German style sausages and hams.  Due to the perishability of some of their products their S&H charges appear outrageous but I guess that sending frozen or refrigerated foods costs money.  Personally, I recommend their coarse grind liverwurst but that is a matter of personal taste.
==================================================
```

In [39]:

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

```
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
```

In [40]:

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
```

```
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
==================================================
IF YOU LIKE SALMON YOU WILL LOVE THESE OMAHA STEAKS SALMON VERY VERY GOOD
==================================================
OK....I thought I'd put a bit of punch to hubby's sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!One bi
te and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!I guess it was that-bad!I'll see if my neighbor will be able to
use it w/her family.If you  are a BEST FOODS lover---walk away---do NOT purchase this product!
==================================================
These people from Bavaria really know how to make this stuff. The Landjagers are super (you have to let them dry for them to
develop their full, intended flavor), and it is worth any sausage fan's time to check out their complete offering of German
style sausages and hams.  Due to the perishability of some of their products their S&H charges appear outrageous but I guess
that sending frozen or refrigerated foods costs money.  Personally, I recommend their coarse grind liverwurst but that is a
matter of personal taste.
```

In [41]:

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

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

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

In [42]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
OK....I thought I would put a bit of punch to hubby is sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!<
br />One bite and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!<br />I guess it was that-bad!<br />I will see if my
neighbor will be able to use it w/her family.<br />If you  are a BEST FOODS lover---walk away---do NOT purchase this produc
t!
==================================================
```

In [43]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
```

In [44]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
OK I thought I would put a bit of punch to hubby is sandwich instead of the ho hum Best Foods Mayo ohOoooOh FAILURE br One b
ite and he said Please DO NOT EVER SERVE THIS TO ME AGAIN br I guess it was that bad br I will see if my neighbor will be ab
le to use it w her family br If you are a BEST FOODS lover walk away do NOT purchase this product
```

In [45]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step
```

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'th
ey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
t'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'might
n', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wa
sn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

```
In [46]:
```

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

```
100%|██████████| 99722/99722 [00:54<00:00, 1843.94it/s]
```

In [47]:

```
preprocessed_reviews[1500]
```

```
'ok thought would put bit punch hubby sandwich instead ho hum best foods mayo ohoooooh failure one bite said please not ever
serve guess bad see neighbor able use w family best foods lover walk away not purchase product'
```

[3.2] Preprocessing Review Summary

In [48]:

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

In [129]:

```
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KDTree
from sklearn.metrics import accuracy_score
```

```
from sklearn.cross_validation import cross_val_score
from collections import Counter
import collections
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve,confusion_matrix,auc
from sklearn import cross_validation
```

In [50]:

```
# Splitting our Data into Train , validation and test
X_1, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed_reviews,final[
'Score'], test_size=0.2, random_state=0)
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.25)
```

In [51]:

```
print(np.asarray(X_1).shape,np.asarray(X_test).shape,np.asarray(X_tr).shape,np.asarray(X
_test).shape,np.asarray(X_cv).shape)
```

```
(79777,) (19945,) (59832,) (19945,) (19945,)
```

[4] Featurization

[4.1] BAG OF WORDS

In [52]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
BOW_Train = count_vect.fit_transform(X_tr)
BOW_test = count_vect.transform(X_test)
BOW_CV = count_vect.transform(X_cv)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)
print("the type of count vectorizer ",type(BOW_Train))
```

```python
print("the shape of out text BOW vectorizer ",BOW_Train.get_shape())
print("the number of unique words ", BOW_Train.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaah', 'aaaah', 'aaaand', 'aaah', 'aachen', 'aadp', 'aaf']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (59832, 45723)
the number of unique words  45723
```

[4.2] Bi-Grams and n-Grams.

In [58]:

```python
#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/module
s/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_co
unts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (103930, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [53]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
TFIDF_Train = tf_idf_vect.fit_transform(X_tr)
TFIDF_Test = tf_idf_vect.transform(X_test)
TFIDF_Validation = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(TFIDF_Train))
print("the shape of out text TFIDF vectorizer ",TFIDF_Train.get_shape())
print("the number of unique words including both unigrams and bigrams ", TFIDF_Train.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (59832, 35189)
the number of unique words including both unigrams and bigrams  35189
```

[4.4] Word2Vec

In [58]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
list_of_sentance_cv=[]
list_of_sentance_test=[]
for sentance in X_tr:
    list_of_sentance.append(sentance.split())
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [59]:
```
# Using Google News Word2Vectors
```

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin'
, binary=True)
        print(w2v_model.wv.most_similar('great'))
```

```
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")
```

```
[('awesome', 0.8341949582099915), ('good', 0.8306764960289001), ('wonderful', 0.8152521848678589), ('fantastic', 0.8107672333717346), ('terrific', 0.7866730690002441), ('excellent', 0.7623412609100342), ('perfect', 0.7575859427452087), ('fabulous', 0.7238317131996155), ('nice', 0.6980041861534119), ('amazing', 0.6876950860023499)]
==================================================
[('greatest', 0.7991034984588623), ('tastiest', 0.737838864326477), ('best', 0.728508710861206), ('disgusting', 0.6677118539810181), ('nastiest', 0.6519845724105835), ('coolest', 0.6407157182693481), ('horrible', 0.6200476884841919), ('awful', 0.6000495553016663), ('smoothest', 0.5976978540420532), ('closest', 0.588812530040741)]
```

In [60]:

```
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  14567
sample words  ['wisdom', 'pouches', 'unseasoned', 'er', 'gag', 'mornings', 'benefited', 'happybellies', 'instance', 'depot', 'theatre', 'pollux', 'undoubtedly', 'positives', 'inhibit', 'chewey', 'sniffing', 'crunching', 'spiciness', 'experts', 'beneath', 'buttons', 'props', 'hearing', 'charging', 'unpalatable', 'abomination', 'backed', 'clueless', 'peaberry', 'oriented', 'sinensis', 'sparse', 'yorkies', 'concentrations', 'brighter', 'farther', 'forbid', 'two', 'feedings', 'distract', 'unit', 'identified', 'licensing', 'fishing', 'dunked', 'subsides', 'healthy', 'sayin', 'woofed']
```

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

[4.4.1.1] Avg W2v

In [61]:

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

```
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors_test.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 59832/59832 [17:48<00:00, 55.99it/s]
100%|██████████| 19945/19945 [07:05<00:00, 46.91it/s]
100%|██████████| 19945/19945 [07:01<00:00, 47.29it/s]


59832
50
```

[4.4.1.2] TFIDF weighted W2v

In [152]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_tr)
tf_idf_matrix_cv = model.transform(X_cv)
tf_idf_matrix_test = model.transform(X_test)
# 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 [153]:

```
# 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
tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in this l
ist
row=0;
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
```

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

```
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 59832/59832 [1:13:36<00:00, 13.55it/s]
100%|██████████| 19945/19945 [20:40<00:00, 16.08it/s]
100%|██████████| 19945/19945 [23:53<00:00, 13.91it/s]
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) 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. **Apply Knn(kd tree version) on these feature sets**
    NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this [link]()
    - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

    ```
                count_vect = CountVectorizer(min_df=10, max_features
    =500)
                count_vect.fit(preprocessed_reviews)
    ```

    - SET 6:Review text, preprocessed one converted into vectors using (TFIDF) but with restriction on maximum features generated.

```
                tf_idf_vect = TfidfVectorizer(min_df=10, max_fea
tures=500)
                tf_idf_vect.fit(preprocessed_reviews)
```

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

3. **The hyper paramter tuning(find best K)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

4. **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

5. **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.

[5.1] Applying KNN brute force

[5.1.1] Applying KNN brute force on BOW, SET 1

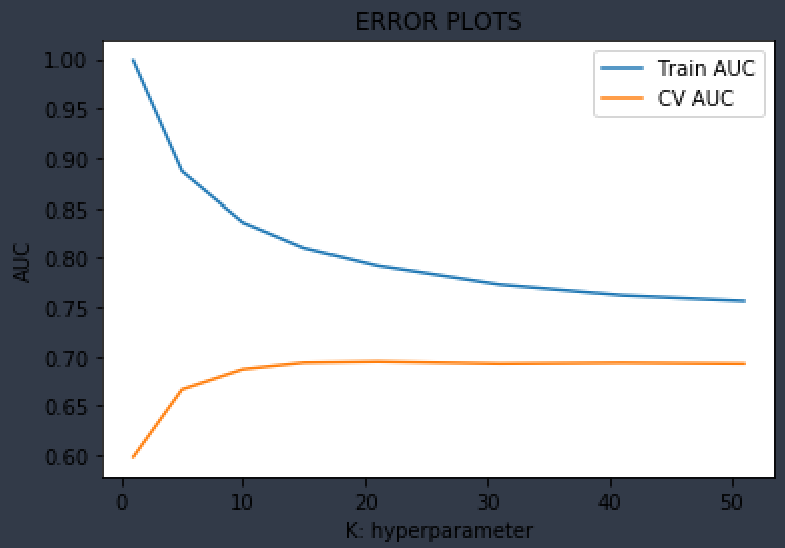
Hyper parameter tuning using AUC as accuracy and applying simple cross validation

In [126]:

```
BOW_train_accuracy = []
BOW_cv_accuracy = []
k=[1, 5, 10, 15, 21, 31, 41, 51]
for i in k:
    Knn = KNeighborsClassifier(n_neighbors=i)
    Knn.fit(BOW_Train,y_tr)
    train_pred = []
    valid_pred = []
    for i in range(0, BOW_Train.shape[0], 1000):
        train_pred.extend(Knn.predict_proba(BOW_Train[i:i+1000])[:,1])
    for i in range(0, BOW_CV.shape[0], 1000):
        valid_pred.extend(Knn.predict_proba(BOW_CV[i:i+1000])[:,1])
```

```
    BOW_train_accuracy.append(roc_auc_score(y_tr,train_pred))
    BOW_cv_accuracy.append(roc_auc_score(y_cv,valid_pred))
```

In [127]:
```
plt.plot(k, BOW_train_accuracy, label='Train AUC')
plt.plot(k, BOW_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [128]:
```
# from observing we can say that
bestk = 21
```

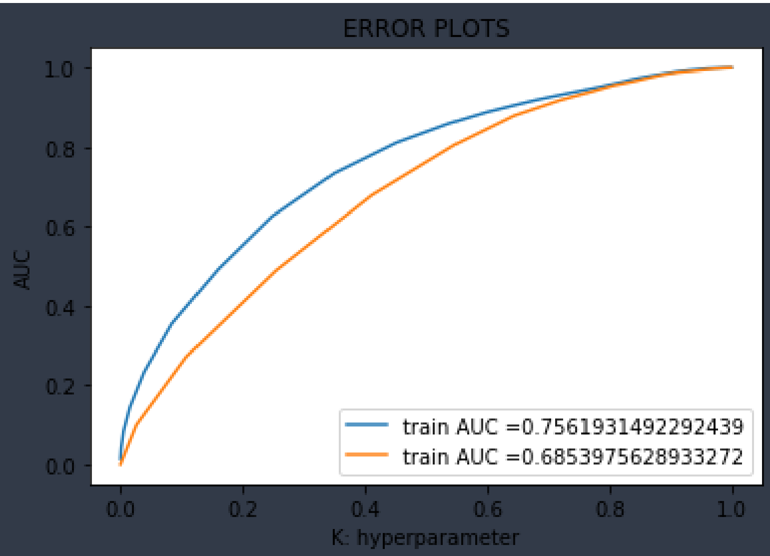
Observation

Here we can say that k=21 is the best hyperparameter value for evaluation

Testin with test data

In [131]:

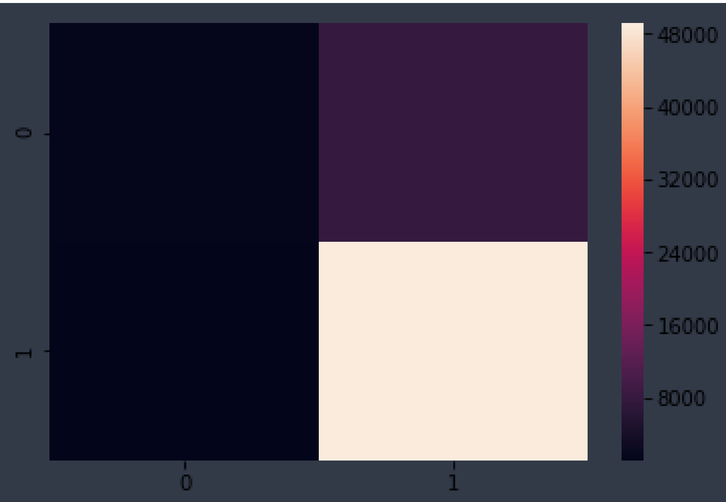
```
Knn = KNeighborsClassifier(n_neighbors=bestk)
Knn.fit(BOW_Train,y_tr)
test_pred = []
for i in range(0, BOW_test.shape[0], 1000):
        test_pred.extend(Knn.predict_proba(BOW_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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



Confusion matrix

In [136]:

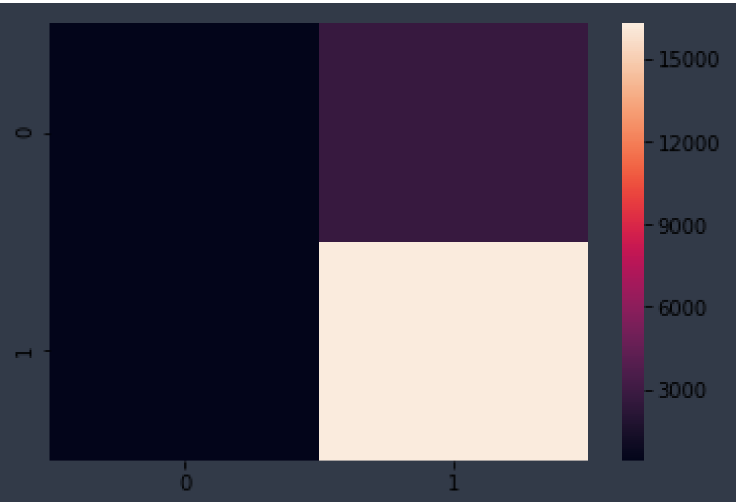
```
ytrain=[]
ytest=[]
for i in range(0,BOW_Train.shape[0],1000):
    ytrain.extend(Knn.predict(BOW_Train[i:i+1000]))
for i in range(0,BOW_test.shape[0],1000):
    ytest.extend(Knn.predict(BOW_test[i:i+1000]))
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
plt.show()
```



In [139]:

```
sns.heatmap(ctest)
print(ctest)
print(ctrain)
plt.show()
```

```
[[  432  2785]
 [  431 16297]]
[[ 1587  8004]
 [ 1165 49076]]
```



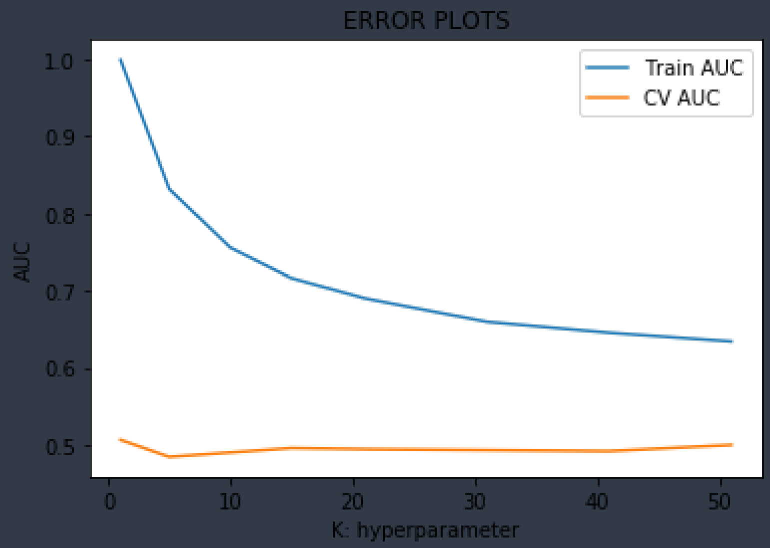
[5.1.2] Applying KNN brute force on TFIDF, SET 2

Hyper parameter tuning using AUC as accuracy and applying simple cross validation

In [140]:

```
TFIDF_train_accuracy = []
TFIDF_cv_accuracy = []
k=[1, 5, 10, 15, 21, 31, 41, 51]
for i in k:
    Knn = KNeighborsClassifier(n_neighbors=i)
    Knn.fit(TFIDF_Train,y_tr)
    train_pred = []
    valid_pred = []
    for i in range(0, TFIDF_Train.shape[0], 1000):
        train_pred.extend(Knn.predict_proba(TFIDF_Train[i:i+1000])[:,1])
```

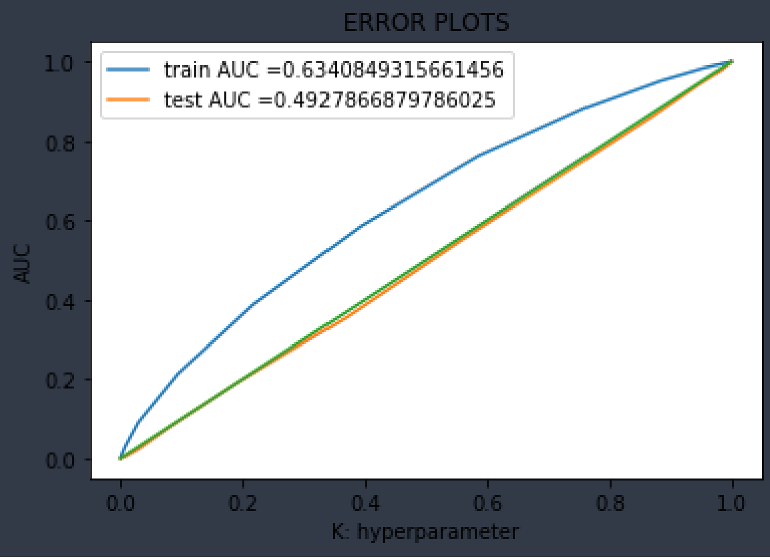
```
    for i in range(0, TFIDF_Validation.shape[0], 1000):
        valid_pred.extend(Knn.predict_proba(TFIDF_Validation[i:i+1000])[:,1])

    TFIDF_train_accuracy.append(roc_auc_score(y_tr,train_pred))
    TFIDF_cv_accuracy.append(roc_auc_score(y_cv,valid_pred))
plt.plot(k, TFIDF_train_accuracy, label='Train AUC')
plt.plot(k, TFIDF_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [141]:
```
bestk = 51
```

Testing on test data

```
In [142]: Knn = KNeighborsClassifier(n_neighbors=bestk)
Knn.fit(TFIDF_Train,y_tr)
test_pred = []
for i in range(0, TFIDF_Test.shape[0], 1000):
        test_pred.extend(Knn.predict_proba(TFIDF_Test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

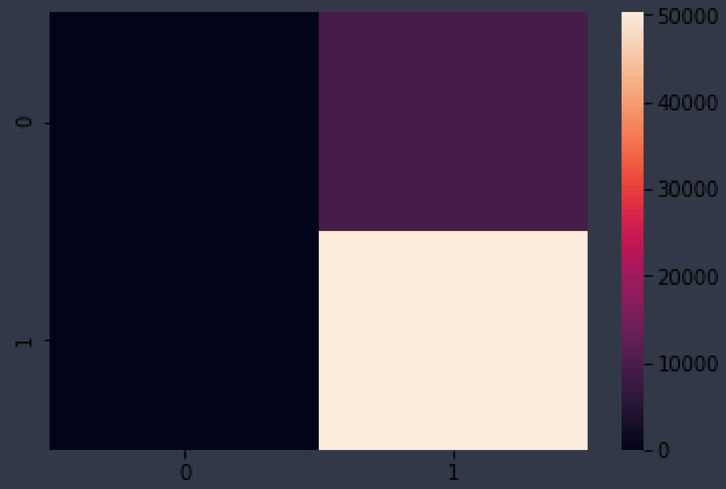


Confusion matrix

In [144]:

```
ytrain=[]
ytest=[]
for i in range(0,TFIDF_Train.shape[0],1000):
    ytrain.extend(Knn.predict(TFIDF_Train[i:i+1000]))
for i in range(0,BOW_test.shape[0],1000):
    ytest.extend(Knn.predict(TFIDF_Test[i:i+1000]))
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
print(ctrain)
plt.show()
```

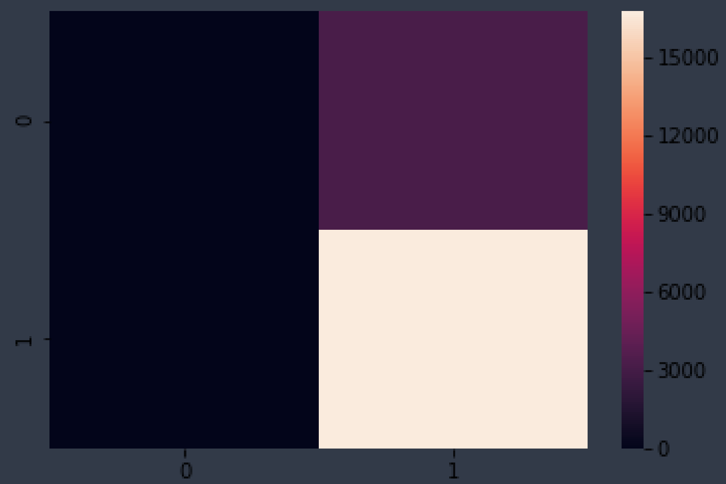
```
[[    0  9591]
 [    0 50241]]
```



In [145]:

```
print(ctest)
sns.heatmap(ctest)
plt.show()
```

```
[[    0  3217]
 [    0 16728]]
```



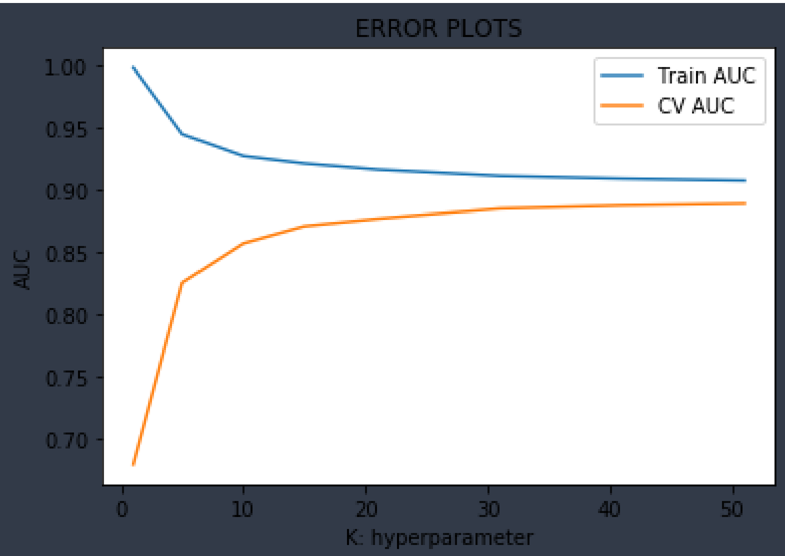
[5.1.3] Applying KNN brute force on AVG W2V, SET 3

Hyper parameter tuning using AUC as accuracy and applying simple cross validation

In [147]:

```
w2v_train_accuracy = []
w2v_cv_accuracy = []
```

```
k=[1, 5, 10, 15, 21, 31, 41, 51]
for i in k:
    Knn = KNeighborsClassifier(n_neighbors=i)
    Knn.fit(sent_vectors,y_tr)
    train_pred = []
    valid_pred = []
    for i in range(0, np.asarray(sent_vectors).shape[0], 1000):
        train_pred.extend(Knn.predict_proba(sent_vectors[i:i+1000])[:,1])
    for i in range(0, np.asarray(sent_vectors_cv).shape[0], 1000):
        valid_pred.extend(Knn.predict_proba(sent_vectors_cv[i:i+1000])[:,1])

    w2v_train_accuracy.append(roc_auc_score(y_tr,train_pred))
    w2v_cv_accuracy.append(roc_auc_score(y_cv,valid_pred))
plt.plot(k, w2v_train_accuracy, label='Train AUC')
plt.plot(k, w2v_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [148]:

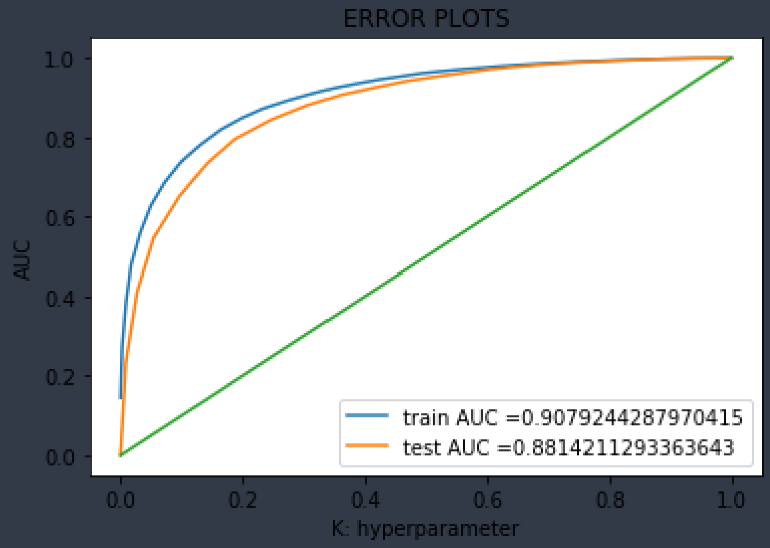
```
bestk = 31
```

Testing on test data

In [149]:

```
Knn = KNeighborsClassifier(n_neighbors=bestk)
Knn.fit(sent_vectors,y_tr)
test_pred = []
for i in range(0, np.asarray(sent_vectors_test).shape[0], 1000):
        test_pred.extend(Knn.predict_proba(sent_vectors_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
```

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



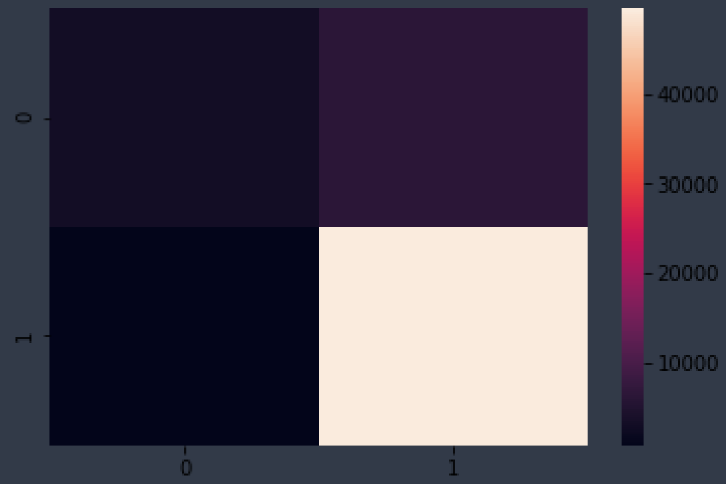
Confusion Matrix

In [150]:

```
ytrain=[]
ytest=[]
for i in range(0,np.asarray(sent_vectors).shape[0],1000):
    ytrain.extend(Knn.predict(sent_vectors[i:i+1000]))
for i in range(0,np.asarray(sent_vectors_test).shape[0],1000):
    ytest.extend(Knn.predict(sent_vectors_test[i:i+1000]))
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
```

```
print(ctrain)
plt.show()
```

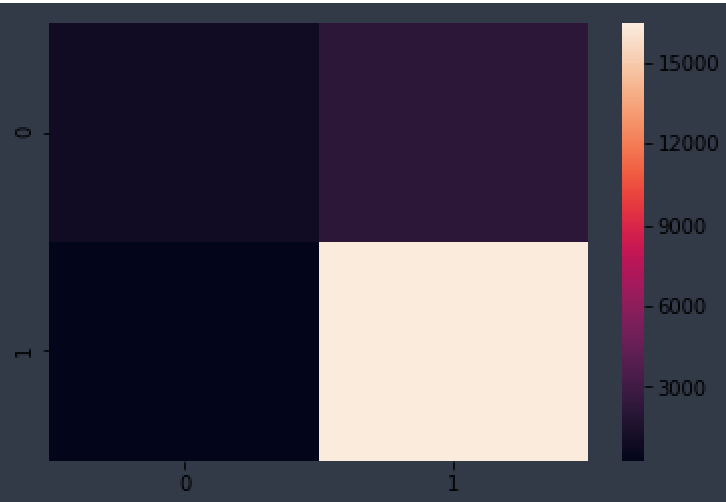
```
[[ 3173  6418]
 [  752 49489]]
```



In [151]:

```
print(ctest)
sns.heatmap(ctest)
plt.show()
```

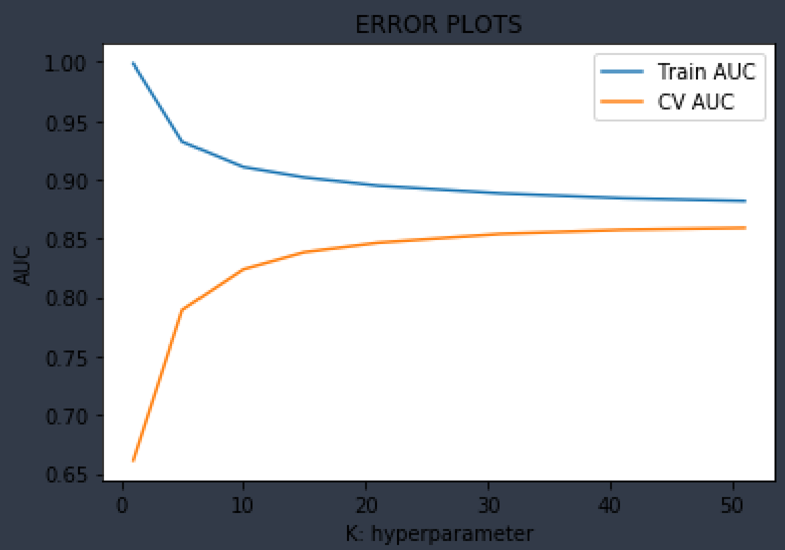
```
[[ 1024  2193]
 [  300 16428]]
```



[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

In [154]:
```
tfw2v_train_accuracy = []
tfw2v_cv_accuracy = []
k=[1, 5, 10, 15, 21, 31, 41, 51]
for i in k:
    Knn = KNeighborsClassifier(n_neighbors=i)
    Knn.fit(tfidf_sent_vectors,y_tr)
    train_pred = []
    valid_pred = []
    for i in range(0, np.asarray(tfidf_sent_vectors).shape[0], 1000):
        train_pred.extend(Knn.predict_proba(tfidf_sent_vectors[i:i+1000])[:,1])
    for i in range(0, np.asarray(tfidf_sent_vectors_cv).shape[0], 1000):
        valid_pred.extend(Knn.predict_proba(tfidf_sent_vectors_cv[i:i+1000])[:,1])

    tfw2v_train_accuracy.append(roc_auc_score(y_tr,train_pred))
    tfw2v_cv_accuracy.append(roc_auc_score(y_cv,valid_pred))
```

```
plt.plot(k, tfw2v_train_accuracy, label='Train AUC')
plt.plot(k, tfw2v_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

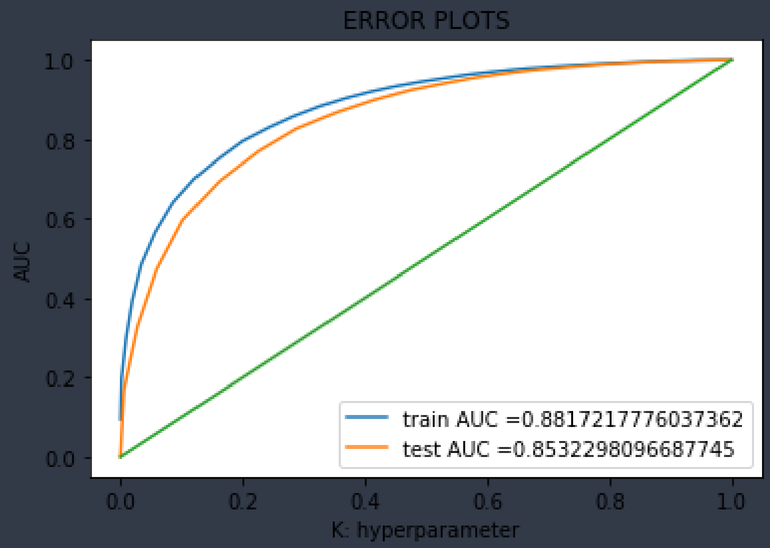


In [165]:
```
bestk = 31
```

Testing on test data

In [156]:
```
Knn = KNeighborsClassifier(n_neighbors=bestk)
Knn.fit(tfidf_sent_vectors,y_tr)
test_pred = []
for i in range(0, np.asarray(tfidf_sent_vectors_test).shape[0], 1000):
```

```
        test_pred.extend(Knn.predict_proba(tfidf_sent_vectors_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

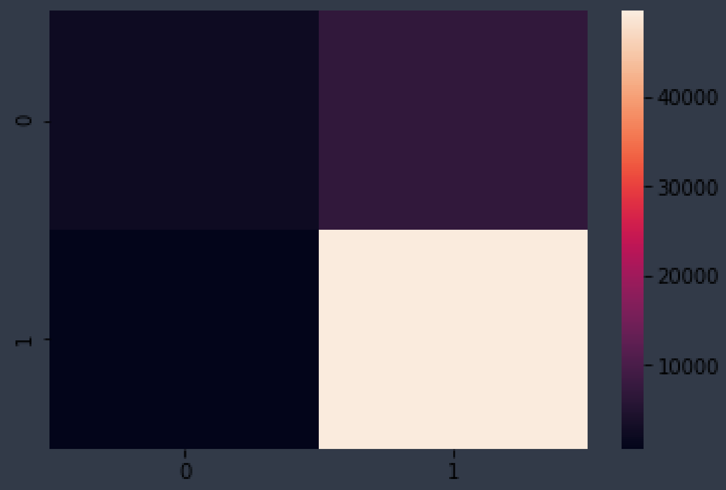


Confusion Matrix

In [158]:
```
ytrain=[]
```

```
ytest=[]
for i in range(0,np.asarray(tfidf_sent_vectors).shape[0],1000):
    ytrain.extend(Knn.predict(tfidf_sent_vectors[i:i+1000]))
for i in range(0,np.asarray(tfidf_sent_vectors_test).shape[0],1000):
    ytest.extend(Knn.predict(tfidf_sent_vectors_test[i:i+1000]))
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
sns.heatmap(ctrain)
print(ctrain)
plt.show()
```

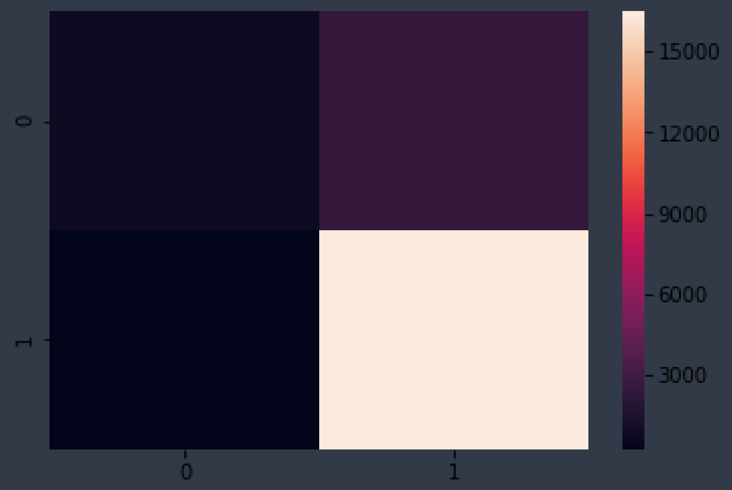
```
[[ 2564  7027]
 [  668 49573]]
```



In [159]:

```
print(ctest)
sns.heatmap(ctest)
plt.show()
```

```
[[  797  2420]
 [  265 16463]]
```



[5.2] Applying KNN kd-tree

In [160]:

```
_count_vect = CountVectorizer(min_df=10, max_features=500) #in scikit-learn
_BOW_Train = _count_vect.fit_transform(X_tr[0:12000])
_BOW_test = _count_vect.transform(X_test[0:4000])
_BOW_CV = _count_vect.transform(X_cv[0:4000])
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)
print("the type of count vectorizer ",type(_BOW_Train))
print("the shape of out text BOW vectorizer ",_BOW_Train.get_shape())
print("the number of unique words ", _BOW_Train.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaah', 'aaaah', 'aaaand', 'aaah', 'aachen', 'aadp', 'aaf']
==================================================
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (12000, 500)
the number of unique words  500
```

In [161]:
```
_BOW_Train = _BOW_Train.toarray()
_BOW_test = _BOW_test.toarray()
_BOW_CV = _BOW_CV.toarray()
```

In [162]:
```
_tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
_TFIDF_Train = _tf_idf_vect.fit_transform(X_tr[0:12000])
_TFIDF_Test = _tf_idf_vect.transform(X_test[0:4000])
_TFIDF_Validation = _tf_idf_vect.transform(X_cv[0:4000])
print("the type of count vectorizer ",type(TFIDF_Train))
print("the shape of out text TFIDF vectorizer ",TFIDF_Train.get_shape())
print("the number of unique words including both unigrams and bigrams ", TFIDF_Train.get
_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (59832, 35189)
the number of unique words including both unigrams and bigrams  35189
```

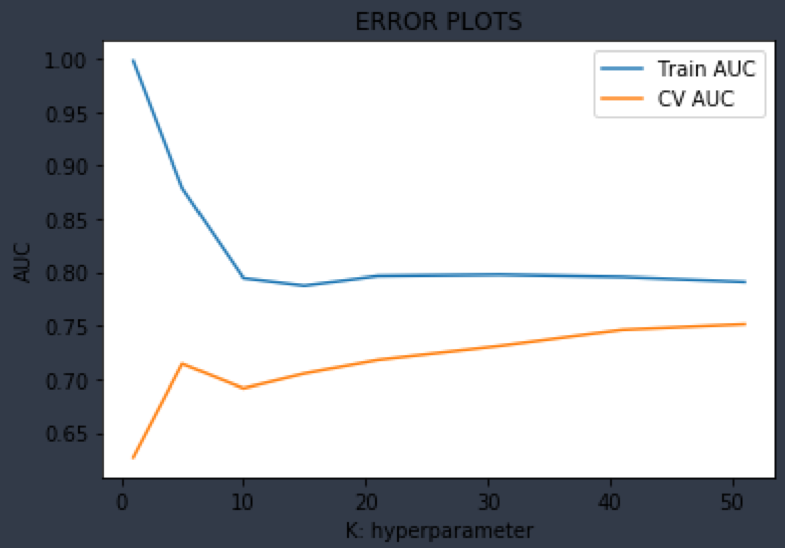
In [163]:
```
_TFIDF_Train = _TFIDF_Train.toarray()
_TFIDF_Test = _TFIDF_Test.toarray()
_TFIDF_Validation = _TFIDF_Validation.toarray()
```

Hyper parameter tuning using AUC as accuracy and applying simple cross validation

In [191]:
```
def accuracy(indi,Y2,K):
    prob = []
```

```
    for i in range(indi.shape[0]):
        count=0
        a=indi[i]
        for y in range(K):
            if Y2[a[y]]==1:
                count = count + 1
        p=float(count/K)
        prob.append(p)
    return prob

BOW_train_accuracy = []
BOW_cv_accuracy = []
Tree = KDTree(np.asarray(_BOW_Train),leaf_size=2)
lst=[1, 5, 10, 15, 21, 31, 41, 51]
for i in lst:
    indices=[]
    train_indices=[]
    for j in range(0,np.asarray(_BOW_CV).shape[0],1000):
        indices.extend(Tree.query(np.asarray(_BOW_CV[j:j+1000]),k=i,return_distance = False))
    for l in range(0,np.asarray(_BOW_Train).shape[0],1000):
        train_indices.extend(Tree.query(np.asarray(_BOW_Train[l:l+1000]),k=i,return_distance = False))
    vali_pred = accuracy(np.asarray(indices),np.asarray(y_tr),i)
    train_pred = accuracy(np.asarray(train_indices),np.asarray(y_tr),i)
    BOW_train_accuracy.append(roc_auc_score(y_tr[0:12000],train_pred))
    BOW_cv_accuracy.append(roc_auc_score(y_cv[0:4000],vali_pred))
plt.plot(k, BOW_train_accuracy, label='Train AUC')
plt.plot(k, BOW_cv_accuracy, label='CV AUC')
plt.legend()
```

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

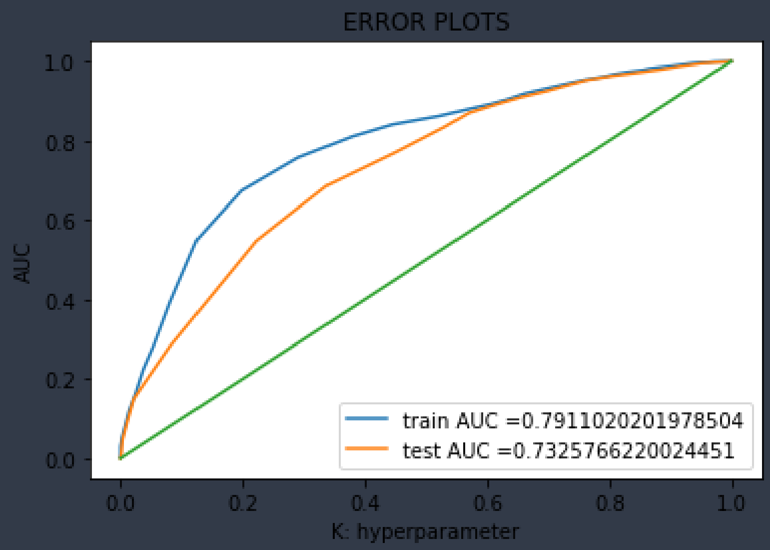


In [193]:
```
bestk= 31
```

Testing on test data

In [198]:
```
test_indices=[]
for j in range(0,np.asarray(_BOW_test).shape[0],1000):
        test_indices.extend(Tree.query(np.asarray(_BOW_test[j:j+1000]),k=bestk,return_di
stance = False))
test_pred = accuracy(np.asarray(test_indices),np.asarray(y_tr),bestk)
train_fpr, train_tpr, thresholds = roc_curve(y_tr[0:12000], train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test[0:4000], test_pred)
```

```
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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Confusion matrix

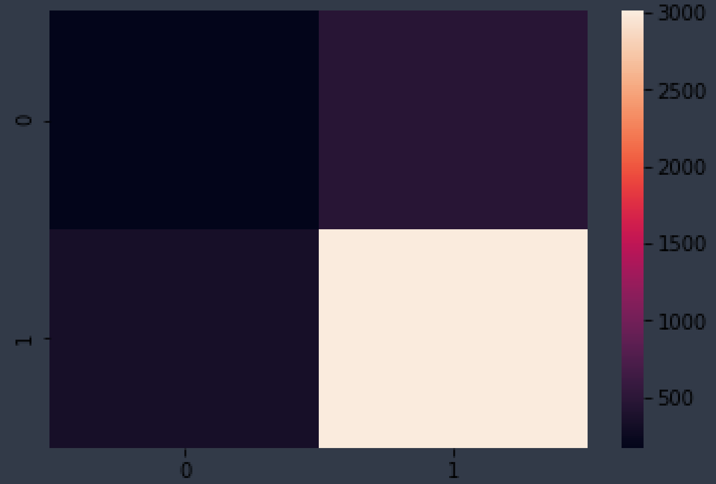
In [200]:

```
def accuracy2(indi,Y1):
    predicted=[]
    for i in range(4000):
        a = collections.Counter(indi[i]).most_common()[0][0]# <---https://stackoverflow.
```

```
com/questions/6252280/find-the-most-frequent-number-in-a-numpy-vector
        predicted.append(Y1[a])
    return predicted
final = accuracy2(np.asarray(test_indices),np.asarray(y_tr))
cf = confusion_matrix(y_test[0:4000],final)
print(cf)
sns.heatmap(cf)
plt.plot()
```

```
[[ 171  472]
 [ 348 3009]]
```

```
[]
```



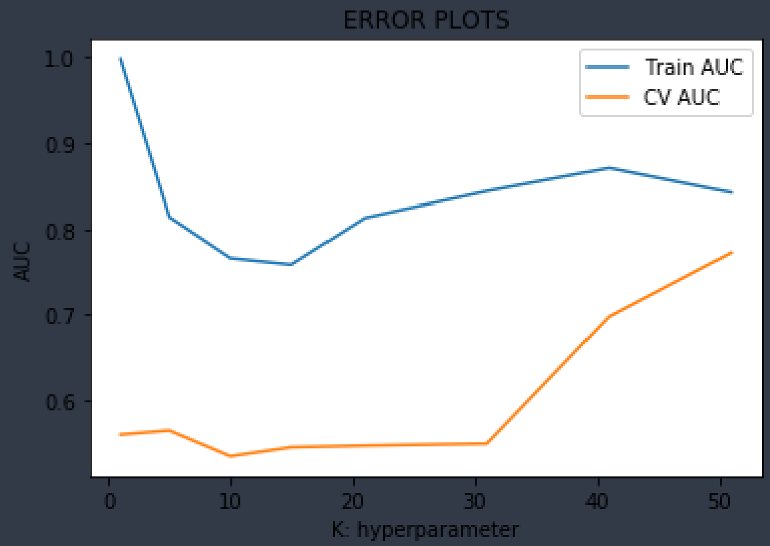
[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

Hyper parameter tuning using AUC as accuracy and applying simple cross validation

In [201]:

```
def accuracy(indi,Y2,K):
    prob = []
    for i in range(indi.shape[0]):
        count=0
        a=indi[i]
        for y in range(K):
            if Y2[a[y]]==1:
                count = count + 1
        p=float(count/K)
        prob.append(p)
    return prob

TFIDF_train_accuracy = []
TFIDF_cv_accuracy = []
Tree = KDTree(np.asarray(_TFIDF_Train),leaf_size=2)
lst=[1, 5, 10, 15, 21, 31, 41, 51]
for i in lst:
    indices=[]
    train_indices=[]
    for j in range(0,np.asarray(_TFIDF_Validation).shape[0],1000):
        indices.extend(Tree.query(np.asarray(_TFIDF_Validation[j:j+1000]),k=i,return_dis
tance = False))
    for l in range(0,np.asarray(_TFIDF_Train).shape[0],1000):
        train_indices.extend(Tree.query(np.asarray(_TFIDF_Train[l:l+1000]),k=i,return_di
stance = False))
    vali_pred = accuracy(np.asarray(indices),np.asarray(y_tr),i)
```

```
    train_pred = accuracy(np.asarray(train_indices),np.asarray(y_tr),i)
    TFIDF_train_accuracy.append(roc_auc_score(y_tr[0:12000],train_pred))
    TFIDF_cv_accuracy.append(roc_auc_score(y_cv[0:4000],vali_pred))
plt.plot(k, TFIDF_train_accuracy, label='Train AUC')
plt.plot(k, TFIDF_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

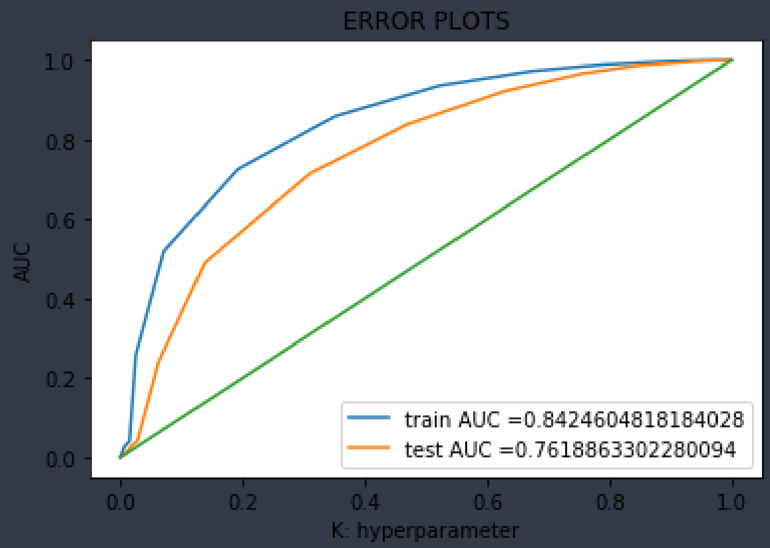


In [203]:
```
bestk = 51
```

Testing on test data

In [204]:
```
test_indices=[]
```

```
for j in range(0,np.asarray(_TFIDF_Test).shape[0],1000):
        test_indices.extend(Tree.query(np.asarray(_TFIDF_Test[j:j+1000]),k=bestk,return_
distance = False))
test_pred = accuracy(np.asarray(test_indices),np.asarray(y_tr),bestk)
train_fpr, train_tpr, thresholds = roc_curve(y_tr[0:12000], train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test[0:4000], test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

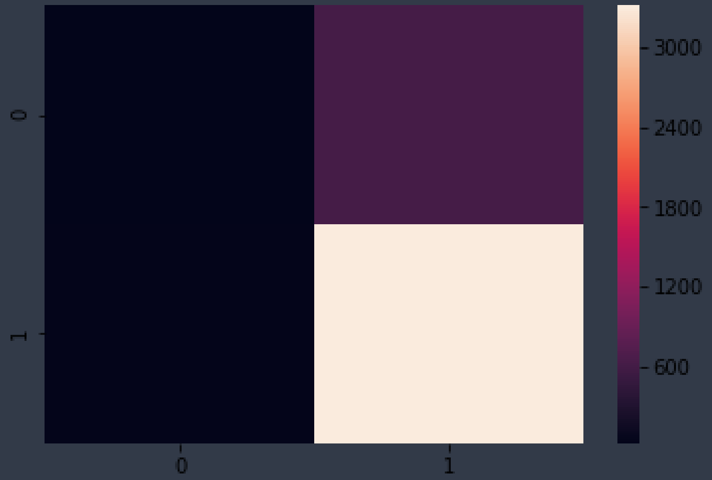


Confusion matrix

In [205]:
```
def accuracy2(indi,Y1):
    predicted=[]
    for i in range(4000):
        a = collections.Counter(indi[i]).most_common()[0][0]# <---https://stackoverflow.com/questions/6252280/find-the-most-frequent-number-in-a-numpy-vector
        predicted.append(Y1[a])
    return predicted
final = accuracy2(np.asarray(test_indices),np.asarray(y_tr))
cf = confusion_matrix(y_test[0:4000],final)
print(cf)
sns.heatmap(cf)
plt.plot()
```

```
[[  25  618]
 [  45 3312]]
```

```
[]
```



[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

Hyper parameter tuning using AUC as accuracy and applying simple cross validation

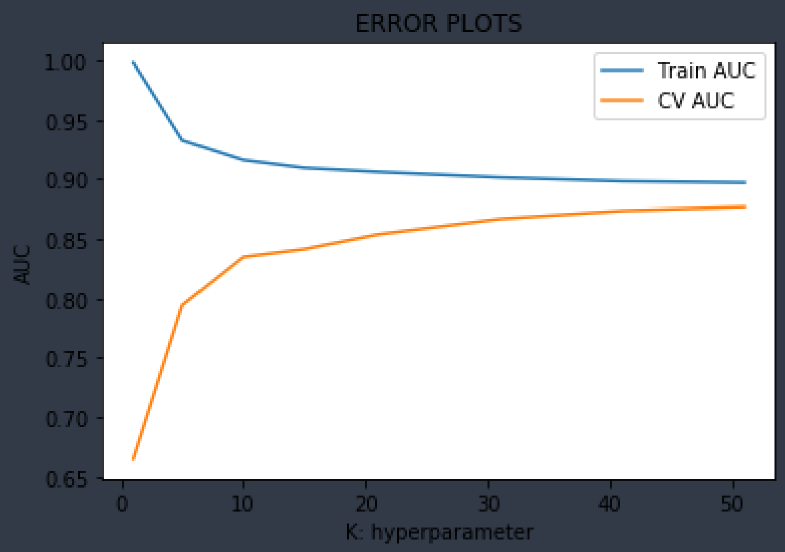
In [206]:

```
_sent_vectors = sent_vectors[0:12000]
_sent_vectors_cv = sent_vectors_cv[0:4000]
_sent_vectors_test = sent_vectors_test[0:4000]
```

In [209]:

```
def accuracy(indi,Y2,K):
    prob = []
    for i in range(indi.shape[0]):
        count=0
        a=indi[i]
        for y in range(K):
```

```
            if Y2[a[y]]==1:
                count = count + 1
        p=float(count/K)
        prob.append(p)
    return prob

w2v_train_accuracy = []
w2v_cv_accuracy = []
Tree = KDTree(np.asarray(_sent_vectors),leaf_size=2)
lst=[1, 5, 10, 15, 21, 31, 41, 51]
for i in lst:
    indices=[]
    train_indices=[]
    for j in range(0,np.asarray(_sent_vectors_cv).shape[0],1000):
        indices.extend(Tree.query(np.asarray(_sent_vectors_cv[j:j+1000]),k=i,return_dist
ance = False))
    for l in range(0,np.asarray(_sent_vectors).shape[0],1000):
        train_indices.extend(Tree.query(np.asarray(_sent_vectors[l:l+1000]),k=i,return_d
istance = False))
    vali_pred = accuracy(np.asarray(indices),np.asarray(y_tr),i)
    train_pred = accuracy(np.asarray(train_indices),np.asarray(y_tr),i)
    w2v_train_accuracy.append(roc_auc_score(y_tr[0:12000],train_pred))
    w2v_cv_accuracy.append(roc_auc_score(y_cv[0:4000],vali_pred))
plt.plot(k, w2v_train_accuracy, label='Train AUC')
plt.plot(k, w2v_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

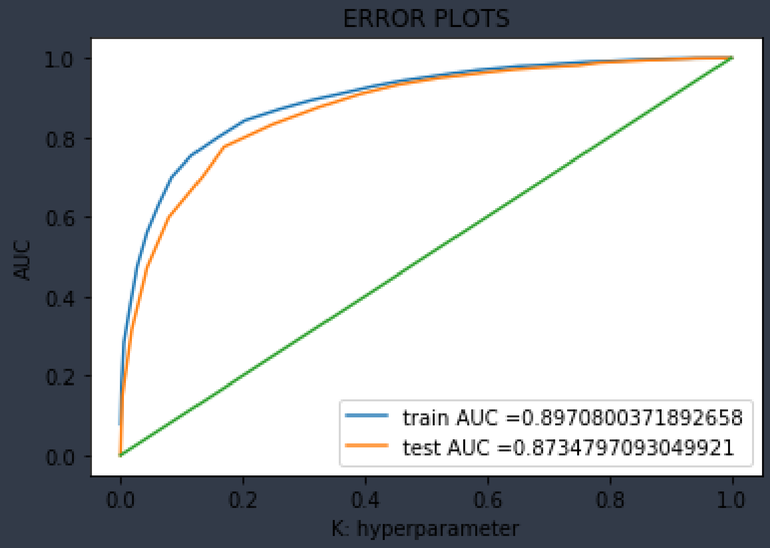



```
In [210]: bestk = 31
```

Testing on test data

```
In [211]: test_indices=[]
for j in range(0,np.asarray(_sent_vectors_test).shape[0],1000):
        test_indices.extend(Tree.query(np.asarray(_sent_vectors_test[j:j+1000]),k=bestk,
return_distance = False))
test_pred = accuracy(np.asarray(test_indices),np.asarray(y_tr),bestk)
train_fpr, train_tpr, thresholds = roc_curve(y_tr[0:12000], train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test[0:4000], test_pred)

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.plot([0.0,1.0],[0.0,1.0])
```

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



Confusion Matrix

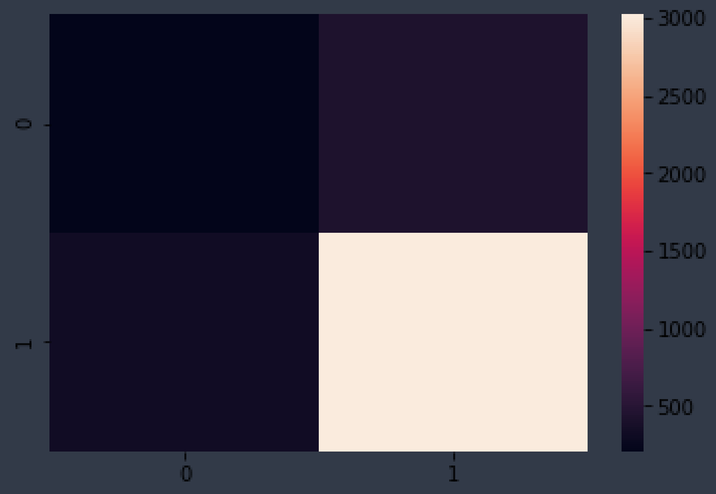
In [212]:

```
def accuracy2(indi,Y1):
    predicted=[]
    for i in range(4000):
        a = collections.Counter(indi[i]).most_common()[0][0]# <---https://stackoverflow.com/questions/6252280/find-the-most-frequent-number-in-a-numpy-vector
        predicted.append(Y1[a])
    return predicted
final = accuracy2(np.asarray(test_indices),np.asarray(y_tr))
```

```
cf = confusion_matrix(y_test[0:4000],final)
print(cf)
sns.heatmap(cf)
plt.plot()
```

```
[[ 210  433]
 [ 336 3021]]
```

```
[]
```



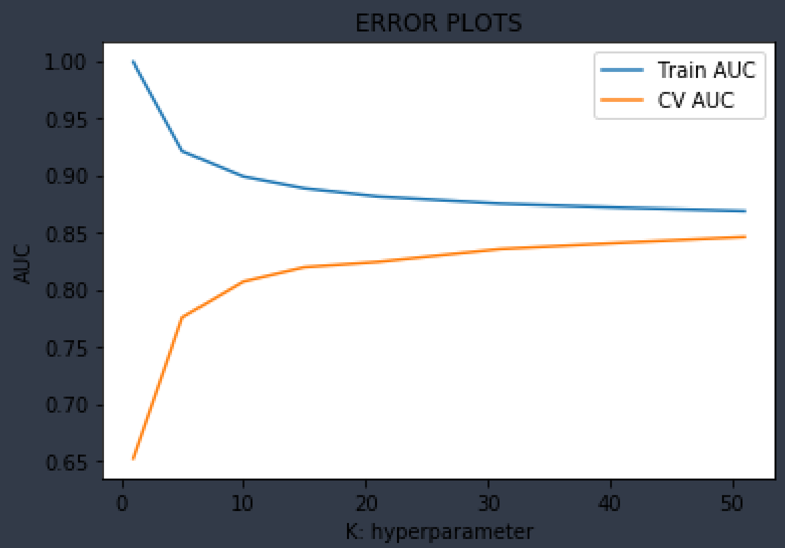
[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

Hyper parameter tuning using AUC as accuracy and applying simple cross validation

```
In [213]: _tfidf_sent_vectors = tfidf_sent_vectors[0:12000]
          _tfidf_sent_vectors_cv = tfidf_sent_vectors_cv[0:4000]
          _tfidf_sent_vectors_test = tfidf_sent_vectors_test[0:4000]
```

```
In [214]: def accuracy(indi,Y2,K):
              prob = []
              for i in range(indi.shape[0]):
                  count=0
                  a=indi[i]
                  for y in range(K):
                      if Y2[a[y]]==1:
                          count = count + 1
                  p=float(count/K)
                  prob.append(p)
              return prob

          tfw2v_train_accuracy = []
          tfw2v_cv_accuracy = []
          Tree = KDTree(np.asarray(_tfidf_sent_vectors),leaf_size=2)
          lst=[1, 5, 10, 15, 21, 31, 41, 51]
          for i in lst:
              indices=[]
              train_indices=[]
              for j in range(0,np.asarray(_tfidf_sent_vectors_cv).shape[0],1000):
                  indices.extend(Tree.query(np.asarray(_tfidf_sent_vectors_cv[j:j+1000]),k=i,retur
          n_distance = False))
              for l in range(0,np.asarray(_tfidf_sent_vectors).shape[0],1000):
                  train_indices.extend(Tree.query(np.asarray(_tfidf_sent_vectors[l:l+1000]),k=i,re
          turn_distance = False))
              vali_pred = accuracy(np.asarray(indices),np.asarray(y_tr),i)
```

```
    train_pred = accuracy(np.asarray(train_indices),np.asarray(y_tr),i)
    tfw2v_train_accuracy.append(roc_auc_score(y_tr[0:12000],train_pred))
    tfw2v_cv_accuracy.append(roc_auc_score(y_cv[0:4000],vali_pred))
plt.plot(k, tfw2v_train_accuracy, label='Train AUC')
plt.plot(k, tfw2v_cv_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

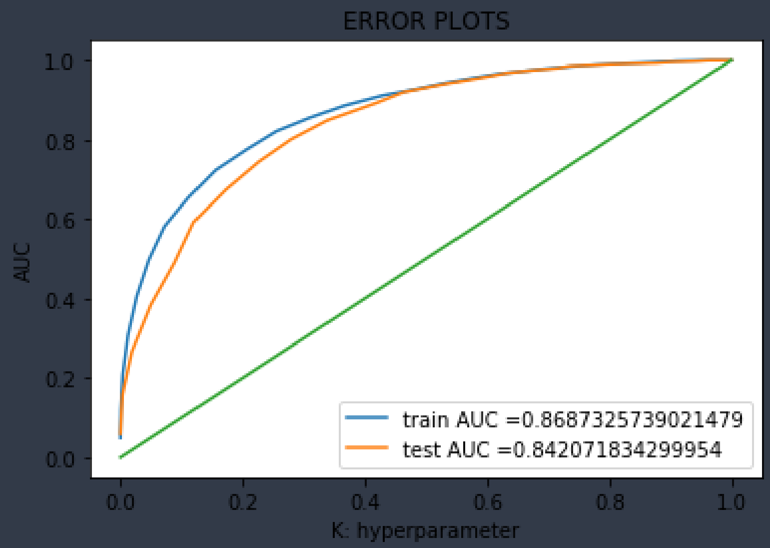


In [215]:
```
bestk = 41
```

Testing on our test data

In [216]:
```
test_indices=[]
```

```
for j in range(0,np.asarray(_tfidf_sent_vectors_test).shape[0],1000):
        test_indices.extend(Tree.query(np.asarray(_tfidf_sent_vectors_test[j:j+1000]),k=
bestk,return_distance = False))
test_pred = accuracy(np.asarray(test_indices),np.asarray(y_tr),bestk)
train_fpr, train_tpr, thresholds = roc_curve(y_tr[0:12000], train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test[0:4000], test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

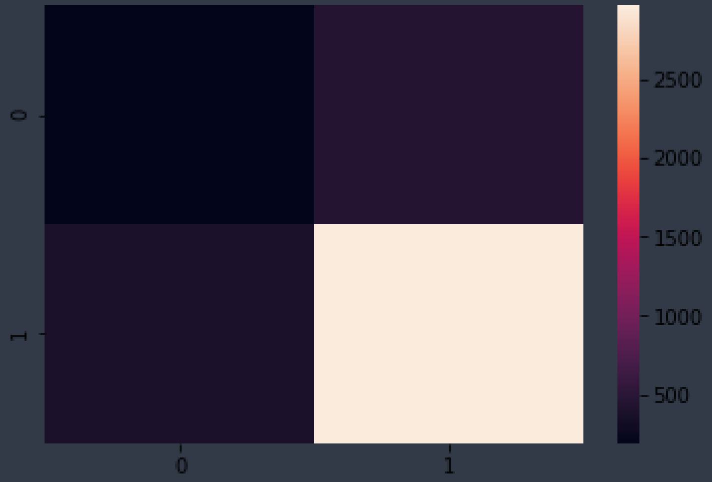


Confusion Matrix

In [217]:
```
def accuracy2(indi,Y1):
    predicted=[]
    for i in range(4000):
        a = collections.Counter(indi[i]).most_common()[0][0]# <---https://stackoverflow.com/questions/6252280/find-the-most-frequent-number-in-a-numpy-vector
        predicted.append(Y1[a])
    return predicted
final = accuracy2(np.asarray(test_indices),np.asarray(y_tr))
cf = confusion_matrix(y_test[0:4000],final)
print(cf)
sns.heatmap(cf)
plt.plot()
```

```
[[ 191  452]
 [ 391 2966]]
```

```
[]
```



[6] Conclusions

After running all our algorrithm on different models we can conclude that applying 31-NN on our avg w2v model gives us heighest AUC value of 0.88 and 0.87 which is largest and the worst model we have observed is TFIDF model

In [221]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","value of K","Train AUC","Test AUC"]

x.add_row(["BOW",21,0.75,0.68])
x.add_row(["TFIDF",51,0.63,0.49])
x.add_row(["Avg W2V",31,0.90,0.88])
x.add_row(["TFIDF_w2v",31,0.88,0.85])
x.add_row(["BOW_KD",31,0.79,0.73])
x.add_row(["TFIDF_KD",51,0.84,0.76])
```

```
x.add_row(["Avg W2V_KD",31,0.89,0.87])
x.add_row(["TFIDF_w2v_KD",41,0.86,0.84])

print(x)
```

```
+--------------+------------+-----------+----------+
|    Model     | value of K | Train AUC | Test AUC |
+--------------+------------+-----------+----------+
|     BOW      |     21     |    0.75   |   0.68   |
|    TFIDF     |     51     |    0.63   |   0.49   |
|   Avg W2V    |     31     |    0.9    |   0.88   |
|  TFIDF_w2v   |     31     |    0.88   |   0.85   |
|    BOW_KD    |     31     |    0.79   |   0.73   |
|   TFIDF_KD   |     51     |    0.84   |   0.76   |
|  Avg W2V_KD  |     31     |    0.89   |   0.87   |
| TFIDF_w2v_KD |     41     |    0.86   |   0.84   |
+--------------+------------+-----------+----------+
```

In [ ]:

```
# ----------------------END--------------------------------------------------------------
----
```