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

In [0]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

```
#from nltk.stem.wordnet import WordaNetLemmatizer

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

from tqdm import tqdm
import os
```

[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 [8]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [15]:

```
con = sqlite3.connect('/content/drive/My Drive/Colab Notebooks/databas
e.sqlite')
filtered_data_50k = pd.read_sql_query("SELECT * FROM Reviews WHERE Scor
e != 3 LIMIT 50000", con)
def partition(x):
    if x < 3:
        return 0
    return 1

actualScore = filtered_data_50k['Score']
positiveNegative = actualScore.map(partition)
filtered_data_50k['Score'] = positiveNegative
print("Number of data points in our data", filtered_data_50k.shape)
filtered_data_50k.head(1)
```

```
Number of data points in our data (50000, 10)
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

In [16]:

```
filtered_data_20k = pd.read_sql_query(""" SELECT * FROM Reviews WHERE S
core != 3 LIMIT 20000""", con)
actualScore = filtered_data_20k['Score']
positiveNegative = actualScore.map(partition)
filtered_data_20k['Score'] = positiveNegative
print("Number of data points in our data", filtered_data_20k.shape)
filtered_data_20k.head(1)
```

```
Number of data points in our data (20000, 10)
```

Out[16]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

In [0]:

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

In [18]:

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

```
(80668, 7)
```

Out[18]:

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

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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 [19]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[19]:

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

In [20]:
```
display['COUNT(*)'].sum()
```
Out[20]: 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 [21]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```
Out[21]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

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

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

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

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

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

In [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data_50k=filtered_data_50k.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data_20k=filtered_data_20k.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [24]:
```
#Deduplication of entries
final_50k=sorted_data_50k.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final_50k.shape
```

Out[24]:
```
(46072, 10)
```

In [25]:
```
#Deduplication of entries
final_20k=sorted_data_20k.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final_20k.shape
```

Out[25]:
```
(19354, 10)
```

In [26]:
```
#Checking to see how much % of data still remains
```

```
(final_50k['Id'].size*1.0)/(filtered_data_50k['Id'].size*1.0)*100
```

Out[26]: 92.144

In [27]:

```
#Checking to see how much % of data still remains
(final_20k['Id'].size*1.0)/(filtered_data_20k['Id'].size*1.0)*100
```

Out[27]: 96.77

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 [28]:

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

display.head()
```

Out[28]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [0]:
```
final_50k=final_50k[final_50k.HelpfulnessNumerator<=final_50k.HelpfulnessDenominator]
```

In [0]:
```
final_20k=final_20k[final_20k.HelpfulnessNumerator<=final_20k.HelpfulnessDenominator]
```

In [31]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final_50k.shape)

#How many positive and negative reviews are present in our dataset?
final_50k['Score'].value_counts()
```

```
(46071, 10)
```

Out[31]:
```
1    38479
0     7592
Name: Score, dtype: int64
```

In [32]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final_20k.shape)

#How many positive and negative reviews are present in our dataset?
final_20k['Score'].value_counts()
```

```
(19354, 10)
```

Out[32]:
```
1    16339
0     3015
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 [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

    # general
```

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

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
```

```
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n'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 [35]:
```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_reviews_50k = []
# tqdm is for printing the status bar
for sentance in tqdm(final_50k['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_50k.append(sentance.strip())
```

```
100%|██████████| 46071/46071 [00:18<00:00, 2461.14it/s]
```

In [36]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews_20k = []
# tqdm is for printing the status bar
for sentance in tqdm(final_20k['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_20k.append(sentance.strip())
```

```
100%|██████████| 19354/19354 [00:07<00:00, 2501.19it/s]
```

In [37]:
```
len(preprocessed_reviews_50k)
```

Out[37]:
```
46071
```

In [38]:
```
len(preprocessed_reviews_20k)
```

Out[38]:
```
19354
```

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

Started working on knn assignment from here :

working with brute force so here using 50k data

In [40]:
```
#refering sample assignment solution to solve this assignment
#here preprocessed_review is my X and final['Score'] is my Y
print(len(preprocessed_reviews_50k))
print(len(final_50k['Score']))
X=preprocessed_reviews_50k
Y=final_50k['Score']
#if both are of same lenght then proceed....
```

```
46071
46071
```

In [0]:
```
#here i am performing splittig operation as train test and cv...
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting
```

In [42]:
```
#checking the types of test and train X,y
print(type(X_train))
print(type(X_test))
print(type(X_cv))
print(type(y_train))
print(type(y_test))
print(type(y_cv))
#now i have xtrain ,xtest,tcv and ytrain,ytest ,ycv....
```

```
<class 'list'>
<class 'list'>
<class 'list'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

In [0]:
```
#now you are ready with xtrain ,xtest xcv and ytrain ,ytest ,ycv
#now there is no problem to proceed with featurization
#first we will do Bow AND LATER OTHER...
```

[4] Featurization

[4.1] BAG OF WORDS

In [44]:
```
#BoW
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fitting on train data ,we cant perform fit on
 test or cv

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
#you can also check X_train_bow is of sparse matrix type or not
#below is code for that
print(type(X_train_bow))
#displaying number of unique words in each of splitted dataset
print("the number of unique words in train: ", X_train_bow.get_shape()[
1])
print("the number of unique words in cv: ", X_cv_bow.get_shape()[1])
print("the number of unique words in test: ", X_test_bow.get_shape()[1
])
```

```
After vectorizations
(20680, 26986) (20680,)
(10187, 26986) (10187,)
(15204, 26986) (15204,)
========================================================================
============================
<class 'scipy.sparse.csr.csr_matrix'>
the number of unique words in train:  26986
the number of unique words in cv:  26986
the number of unique words in test:  26986
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#we directly jump to tfidf since we are not working on bigrams
#bi-gram, tri-gram and n-gram

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

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

[4.3] TF-IDF

In [45]:
```
#below code for converting to tfidf
#i refered sample solution to write this code
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(X_train)
X_test_tf_idf = tf_idf_vect.transform(X_test)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(X_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_shap
e())
```

```
print("the number of unique words including both unigrams and bigrams "
, X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble buy', 'able drink', 'able eat', 'able enjoy', 'able find', 'able ge
t', 'able give', 'able make']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (20680, 12308)
the number of unique words including both unigrams and bigrams  12308
```

[4.4] average W2V:

In [46]:

```
#in average w2v the output is of list form and here we write same code
 of all train ,test and cv
#this code is for train data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
```

```
sent_vectors_train = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u 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/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
  1%|          | 129/20680 [00:00<00:16, 1270.64it/s]
```

```
number of words that occured minimum 5 times  8675
sample words  ['using', 'freeze', 'dried', 'liver', 'treats', 'train',
'reward', 'dogs', 'years', 'like', 'dog', 'nothing', 'not', 'makes', 't
raining', 'breeze', 'current', 'samson', 'sit', 'front', 'pantry', 'dro
ol', 'give', 'treat', 'expensive', 'cut', 'quarters', 'last', 'longer',
'meat', 'no', 'filler', 'rottweiler', 'go', 'wrong', 'bought', 'boyfrie
nd', 'funny', 'christmas', 'present', 'favorite', 'cereal', 'absolutel
y', 'loved', 'kids', 'eat', 'ordered', 'product', 'sure', 'would']
```

```
100%|██████████| 20680/20680 [00:28<00:00, 719.39it/s]
```

```
(20680, 50)
[ 0.79489957 -0.23318568  0.07177058  0.4096118   0.43662737  0.2278465
3
  0.52908794  0.18701127 -0.74666204 -0.28506993  0.574211    0.2654398
1
 -0.33241371 -0.34210452 -0.20294909  0.7873406  -0.46751948 -0.9874292
3
 -0.34789148  0.12603242  0.0504184   0.27773902  0.23824785  0.1473882
3
```

```
 -0.75772438 -1.02486684  0.13835567 -0.4029482   0.10740734  0.5540317
1
  0.28268819 -0.65580413  0.06936001 -0.32759205 -0.14610172 -0.5963281
5
  0.03508124 -0.45310848 -0.33366664 -0.35721502  0.98454429  0.5788181
2
 -0.18053636 -0.35666596  0.37805662 -0.12188227  0.34918146 -0.7184232
6
  0.60520551  0.85449652]
```

In [47]:

```
#this code is for test data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4
)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is store
d 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, yo
u 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/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
  1%|          | 88/15204 [00:00<00:17, 867.03it/s]
```

```
number of words that occured minimum 5 times  7429
sample words  ['wonderful', 'wake', 'morning', 'soothing', 'late', 'aft
ernoon', 'strong', 'enough', 'not', 'bitter', 'bought', 'walmart', 'las
t', 'week', 'figured', 'heck', 'made', 'night', 'sooo', 'easy', 'thinki
ng', 'probably', 'even', 'taste', 'good', 'man', 'wrong', 'absolutely',
'best', 'fudge', 'ever', 'tasted', 'homemade', 'stores', 'delectable',
'went', 'buy', 'left', 'bummed', 'hopefully', 'find', 'target', 'x', 'e
ve', 'ya', 'gotta', 'try', 'stuff', 'received', 'sickly']
```

```
100%|██████████| 15204/15204 [00:18<00:00, 800.22it/s]
```

```
(15204, 50)
[ 0.02433675 -1.17931224  0.22981873  0.5696413  -0.00235228 -0.4212039
6
 -0.11091579  0.33099244 -0.69069316  0.19081398  0.65773099  0.2644106
2
  0.00320039  0.63483009  0.43883441 -0.51288705 -0.94980766  0.0336295
8
 -0.74862584  0.07856928  0.41631702  1.00812794  0.47226014  0.2952585
1
 -0.26016624  0.57364848 -0.36025037  0.49799217 -0.62405558 -0.1866370
9
  0.08505194 -0.71336278  0.05121423 -0.25253982 -1.39118552 -0.2119186
7
  0.12167646 -0.32869749  0.52655804  0.19565117  0.45876187 -0.0505087
```

```
4
 -0.62322717 -0.21936065 -0.09600944 -0.04849669  1.2870628   0.2678683
7
  0.25095776  1.41927707]
```

In [48]:

```
#this code is for cv data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actual code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u 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/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
```

```
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
sent_vectors_cv= np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
  1%|▏         | 131/10187 [00:00<00:07, 1279.72it/s]
```

```
number of words that occured minimum 5 times  6115
sample words  ['hazelnut', 'crunch', 'chocolate', 'big', 'chunks', 'sal
t', 'seems', 'makes', 'crunchy', 'not', 'hazelnuts', 'package', 'no',
'happy', 'quick', 'delivery', 'given', 'tracking', 'method', 'determin
e', 'greenies', 'telling', 'friends', 'relatives', 'good', 'experienc
e', 'love', 'cereal', 'bars', 'delicious', 'however', 'loved', 'greate
r', 'variety', 'choose', 'really', 'missed', 'mark', 'cranberry', 'almo
nd', 'favorite', 'cinnamon', 'raisin', 'peanut', 'butter', 'like', 'eve
ryone', 'else', 'amazon', 'com']
```

```
100%|██████████| 10187/10187 [00:11<00:00, 901.07it/s]
```

```
(10187, 50)
[ 0.46096856 -0.50493715 -0.1394162  -0.1347917  -0.19076426  0.0707335
2
  0.01388143  0.06538044 -0.50158978  0.29981282  0.92718315  0.0467756
9
 -0.08845421  0.19662397  0.02062016 -0.6672414  -0.33046045 -0.4223260
4
 -0.8966547  -0.41356894 -0.07329099  0.31893181  0.35929342 -0.4971865
7
 -0.65396955 -0.7549034  -0.03925625  0.27827    -0.33295671 -0.3238680
3
 -0.13829398 -0.78820792  0.75784881 -0.30862713 -0.68104622  0.2089941
3
  0.64585553 -0.3447644  -0.13046265  0.03584346 -0.07324038  0.0851524
6
 -0.5071412  -0.43829338  0.18659075  0.23361826  0.68139241 -0.1736486
1
  0.12717913  0.82283625]
```

In [0]:
```
#now after going through all three big lines of code we have sent_vecto
rs_train,sent_vectors_test,sent_vectors_cv
#and also we have y_train,y_test,Y_cv
#now with these six parameters lets apply it to knn and lets find first
 best k and then auc values
```

TFIDF weighted W2v:

In [0]:
```
#do it for all: train,test,cv
```

In [51]:
```
#this is for train data
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())


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



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

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
```

```
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    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_train.append(sent_vec)
    row += 1
tfidf_sent_vectors_train= np.array(sent_vectors_train)
print(tfidf_sent_vectors_train.shape)
print(tfidf_sent_vectors_train[0])
```

```
100%|██████████| 20680/20680 [04:27<00:00, 77.24it/s]
```

```
(20680, 50)
[ 0.79489957 -0.23318568  0.07177058  0.4096118   0.43662737  0.2278465
3
  0.52908794  0.18701127 -0.74666204 -0.28506993  0.574211    0.2654398
1
 -0.33241371 -0.34210452 -0.20294909  0.7873406  -0.46751948 -0.9874292
3
 -0.34789148  0.12603242  0.0504184   0.27773902  0.23824785  0.1473882
3
 -0.75772438 -1.02486684  0.13835567 -0.4029482   0.10740734  0.5540317
1
  0.28268819 -0.65580413  0.06936001 -0.32759205 -0.14610172 -0.5963281
5
  0.03508124 -0.45310848 -0.33366664 -0.35721502  0.98454429  0.5788181
2
 -0.18053636 -0.35666596  0.37805662 -0.12188227  0.34918146 -0.7184232
6
  0.60520551  0.85449652]
```

In [52]:

```
#this is for test data
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


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



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

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/r
eview
    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
tfidf_sent_vectors_test= np.array(sent_vectors_test)
print(tfidf_sent_vectors_test.shape)
print(tfidf_sent_vectors_test[0])
```

```
100%|██████████| 15204/15204 [02:55<00:00, 86.39it/s]
```

```
(15204, 50)
[ 0.02433675 -1.17931224  0.22981873  0.5696413  -0.00235228 -0.4212039
6
 -0.11091579  0.33099244 -0.69069316  0.19081398  0.65773099  0.2644106
2
  0.00320039  0.63483009  0.43883441 -0.51288705 -0.94980766  0.0336295
8
 -0.74862584  0.07856928  0.41631702  1.00812794  0.47226014  0.2952585
1
 -0.26016624  0.57364848 -0.36025037  0.49799217 -0.62405558 -0.1866370
9
  0.08505194 -0.71336278  0.05121423 -0.25253982 -1.39118552 -0.2119186
7
  0.12167646 -0.32869749  0.52655804  0.19565117  0.45876187 -0.0505087
4
 -0.62322717 -0.21936065 -0.09600944 -0.04849669  1.2870628   0.2678683
7
  0.25095776  1.41927707]
```

In [53]:

```
#this is for cv data
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())


# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_cv)
```

```
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



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

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is
 stored in this list
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/r
eview
    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_cv= np.array(sent_vectors_cv)
print(tfidf_sent_vectors_cv.shape)
print(tfidf_sent_vectors_cv[0])
```

```
100%|██████████| 10187/10187 [01:50<00:00, 106.23it/s]
```

```
(10187, 50)
[ 0.46096856 -0.50493715 -0.1394162  -0.1347917  -0.19076426  0.0707335
```

```
2
  0.01388143  0.06538044 -0.50158978  0.29981282  0.92718315  0.0467756
9
 -0.08845421  0.19662397  0.02062016 -0.6672414  -0.33046045 -0.4223260
4
 -0.8966547  -0.41356894 -0.07329099  0.31893181  0.35929342 -0.4971865
7
 -0.65396955 -0.7549034  -0.03925625  0.27827    -0.33295671 -0.3238680
3
 -0.13829398 -0.78820792  0.75784881 -0.30862713 -0.68104622  0.2089941
3
  0.64585553 -0.3447644  -0.13046265  0.03584346 -0.07324038  0.0851524
6
 -0.5071412  -0.43829338  0.18659075  0.23361826  0.68139241 -0.1736486
1
  0.12717913  0.82283625]
```

In [0]:
```
#now after going through all three big lines of code we have tfidf_sent_vectors_train,tfidf_sent_vectors_test,tfidf_sent_vectors_cv
#and also we have y_train,y_test,Y_cv
#now with these six parameters lets apply it to knn and lets find first best k and then auc values
```

[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_features=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

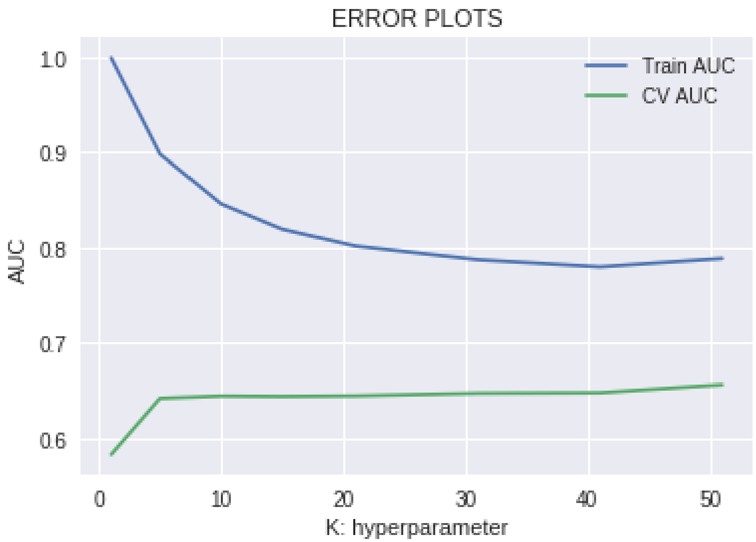
[5.1.1] Applying KNN brute force on BOW, SET 1

In [55]:
```
#here i am applying knn brute force method for bow vectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
```

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]

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

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



In [0]:
```
best_k=41
```

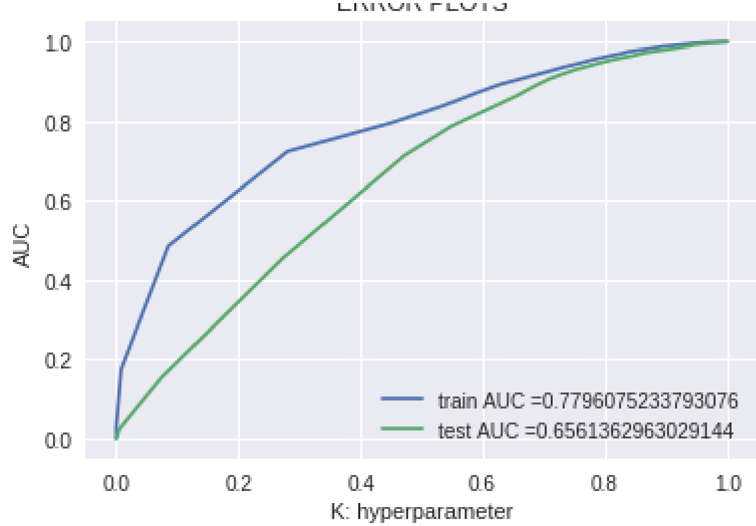
In [57]:
```
#this the code after choosing best k and that we are applying to brute force knn
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

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

print("="*100)
```

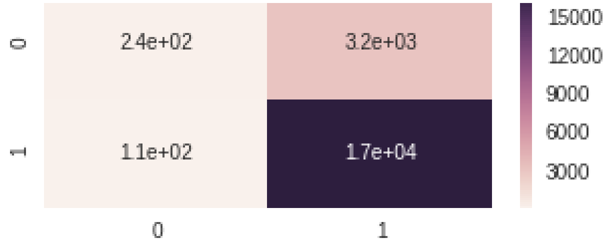



```
==========================================================================================================
```

In [58]:
```
#for seaborn confusion matrix :https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(X_train_bow))
df_1= pd.DataFrame(arr1, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

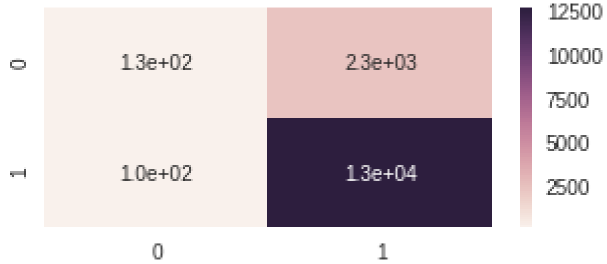
Out[58]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc6fce9f60>
```



In [59]:
```
print("Test confusion matrix")
arr2=confusion_matrix(y_test, neigh.predict(X_test_bow))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

```
Test confusion matrix
```

Out[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc63d39780>
```



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

In [60]:
```
# here i am applying brute force knn on tfidf vectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
```

```
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_tf_idf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_tf_idf)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tf_idf)[:,1]

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

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

In [0]:
```
best_k=31
```

In [76]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
```

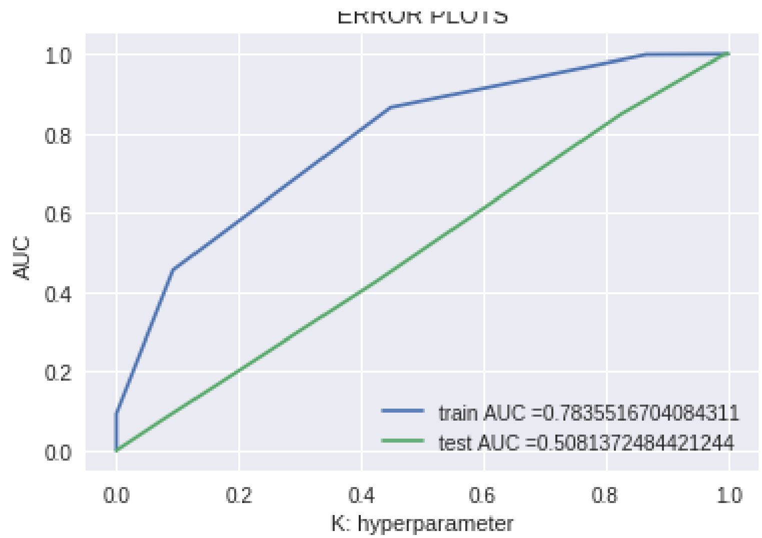
```
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_tf_idf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_pro
ba(X_train_tf_idf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(
X_test_tf_idf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_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()

print("="*100)
```

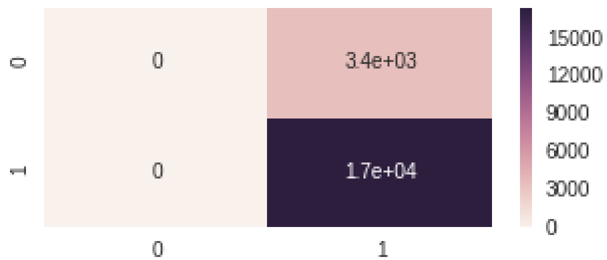


==
==============================

```
In [63]: print("Train confusion matrix")
         arr1=confusion_matrix(y_train, neigh.predict(X_train_tf_idf))
         df_1= pd.DataFrame(arr1, range(2),range(2))
         plt.figure(figsize = (5,2))
         sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

```
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc6244e668>
```

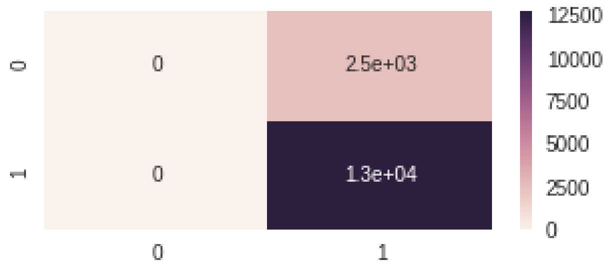



```
In [64]: print("Test confusion matrix")
```

```
arr2=confusion_matrix(y_test, neigh.predict(X_test_tf_idf))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

Test confusion matrix

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc62267588>



[5.1.3] Applying KNN brute force on AVG W2V

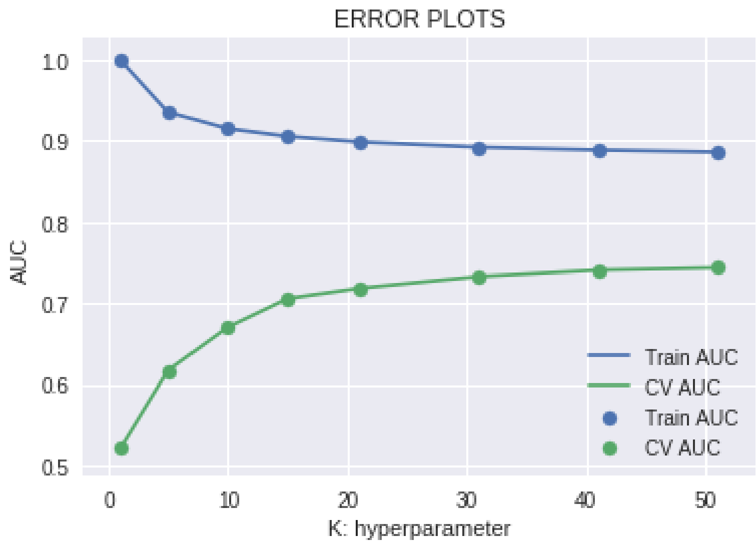
In [65]:

```
# here i am applying brute force knn to avg w2v vectorizer
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(sent_vectors_cv)[:,1]

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

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

```
plt.plot(K, cv_auc, label='CV AUC')
plt.scatter(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

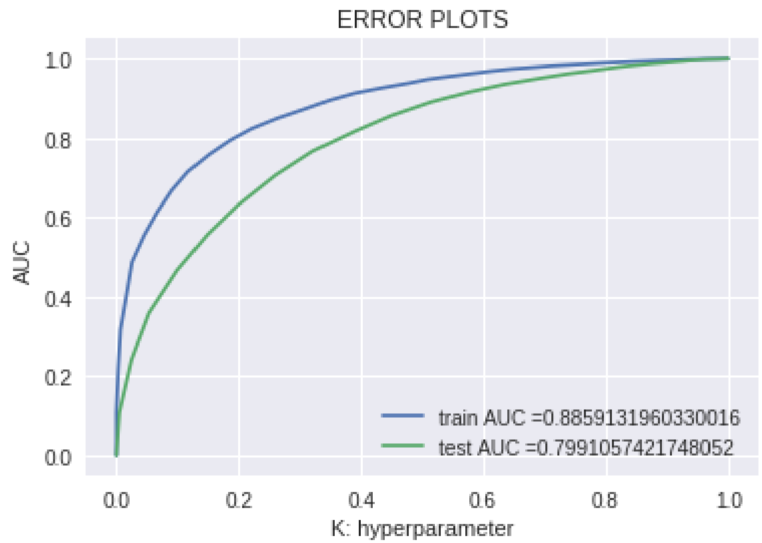


In [0]:
```
best_k=51
```

In [67]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(sent_vectors_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

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

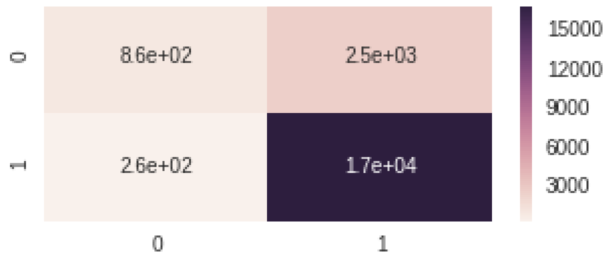
```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [68]:
```
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(sent_vectors_train))
df_1= pd.DataFrame(arr1, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc61b20a20>



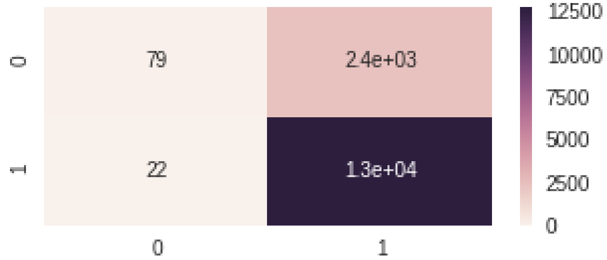
In [69]:

```
print("Test confusion matrix")
arr2=confusion_matrix(y_test, neigh.predict(sent_vectors_test))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

```
Test confusion matrix
```

Out[69]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc5f7644a8>
```



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

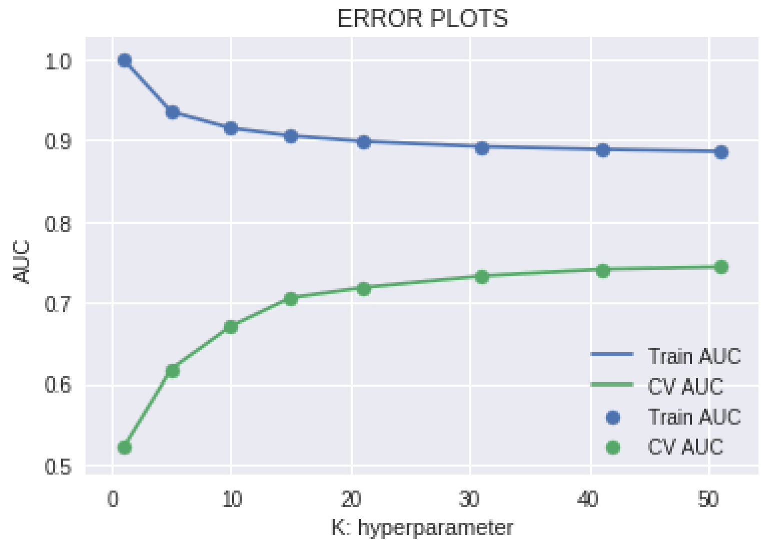
In [70]:

```
# here i am applying rce knn tobrute fo tfidf w2v vectorizer
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(tfidf_sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
```

```
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(tfidf_sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(tfidf_sent_vectors_cv)[:,1]

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

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



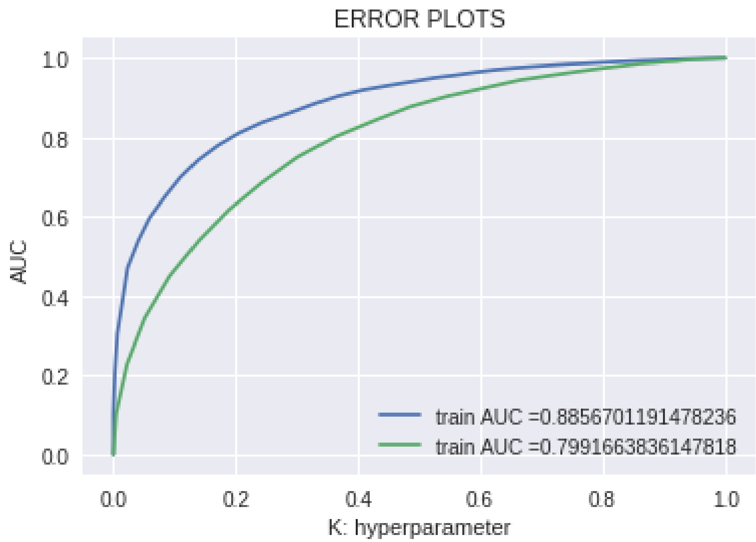
In [0]:
```
best_k=53
```

In [72]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(tfidf_sent_vectors_train, y_train)
```

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_pro
ba(tfidf_sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(
tfidf_sent_vectors_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_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()
```

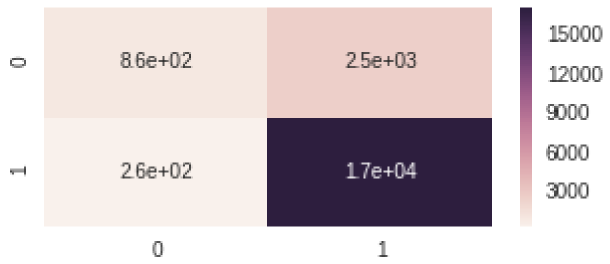


In [73]:
```
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(tfidf_sent_vectors_train))
df_1= pd.DataFrame(arr1, range(2),range(2))
```

```
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

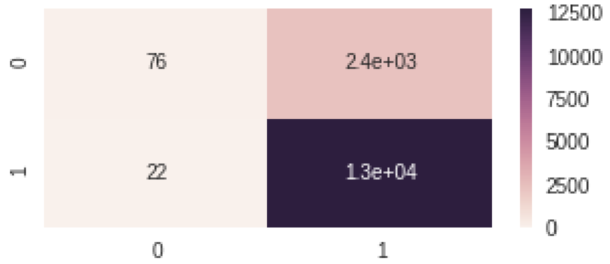
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc6244dba8>



In [74]:
```
print("Test confusion matrix")
arr2=confusion_matrix(y_test, neigh.predict(tfidf_sent_vectors_test))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

```
Test confusion matrix
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc5f86eac8>



[5.2] Applying KNN kd-tree so ussing only 20k of data here:

In [77]:
```
#refering sample assignment solution to solve this assignment
#here preprocessed_review is my X and final['Score'] is my Y
print(len(preprocessed_reviews_20k))
print(len(final_20k['Score']))
X=preprocessed_reviews_20k
Y=final_20k['Score']
#if both are of same lenght then proceed....
```

```
19354
19354
```

In [0]:
```
#here i am performing splittig operation as train test and cv...
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

bow

In [79]:
```
#BoW
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fitting on train data ,we cant perform fit on
 test or cv

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
print("="*100)
#you can also check X_train_bow is of sparse matrix type or not
#below is code for that
print(type(X_train_bow))
#displaying number of unique words in each of splitted dataset
print("the number of unique words in train: ", X_train_bow.get_shape()[
1])
print("the number of unique words in cv: ", X_cv_bow.get_shape()[1])
print("the number of unique words in test: ", X_test_bow.get_shape()[1
])
```

```
After vectorizations
(8687, 18024) (8687,)
(4280, 18024) (4280,)
(6387, 18024) (6387,)
======================================================================
=============================
<class 'scipy.sparse.csr.csr_matrix'>
the number of unique words in train:  18024
the number of unique words in cv:  18024
the number of unique words in test:  18024
```

tfidf

In [80]:

```
#below code for converting to tfidf
#i refered sample solution to write this code
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(X_train)
X_test_tf_idf = tf_idf_vect.transform(X_test)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(X_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_shap
```

```
e())
print("the number of unique words including both unigrams and bigrams "
, X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble buy', 'able find', 'able get', 'able order', 'absolute', 'absolute
favorite', 'absolutely', 'absolutely best']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (8687, 5513)
the number of unique words including both unigrams and bigrams  5513
```

avg w2v

In [81]:

```
#avg w2v
#in average w2v the output is of list form and here we write same code
 of all train ,test and cv
#this code is for train data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
```

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



#this code is for test data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

```
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is store
d 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, yo
u 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/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
#this code is for cv data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
```

```
#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u 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/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
sent_vectors_cv= np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
  1%||          | 123/8687 [00:00<00:07, 1219.60it/s]
```

```
number of words that occured minimum 5 times  5591
sample words  ['item', 'arrived', 'horrible', 'mess', 'broken', 'stick
y', 'want', 'money', 'back', 'expected', 'better', 'amazon', 'even', 't
hough', 'listed', 'gluten', 'free', 'category', 'not', 'ingredients',
'state', 'contains', 'wheat', 'protein', 'order', 'need', 'gf', 'food',
```

```
'bad', 'reviewing', 'good', 'product', 'becomes', 'difficult', 'find',
'paste', 'store', 'convenient', 'way', 'shop', 'g', 'r', 'e', 'bought',
'getting', 'frustrated', 'able', 'grow', 'plant', 'house']
```

```
100%|██████████| 8687/8687 [00:09<00:00, 893.79it/s]
```

```
(8687, 50)
[ 0.424955   -0.47383716 -0.2357066   0.15721139  0.11870111  0.0069023
6
  0.10480667 -0.29871408 -0.79075245  0.29588864  0.65820314 -0.0267759
1
 -0.0812725  -0.30829108  0.22448453 -0.27420957 -0.60425005 -0.3079123
4
 -0.54447321 -0.91239444 -0.00434475  0.64699678  0.29881632 -0.1583756
 -0.48342951 -0.73056618 -0.10592381 -0.08068542 -0.21352984 -0.1619982
8
  0.21991885 -0.38265449 -0.13816791 -0.29885154 -0.1186995   0.1656225
2
 -0.1378229  -0.2370468  -0.83159481 -0.25114585  0.77791787  0.6638960
8
 -0.95094952 -1.06049211  0.06017698 -0.08070387  0.62733026  0.2658296
9
  0.27707071  0.31143757]
```

```
  2%|▏         | 149/6387 [00:00<00:04, 1488.40it/s]
```

```
number of words that occured minimum 5 times  4677
sample words  ['worked', 'claimed', 'no', 'leaking', 'brewed', 'reall
y', 'great', 'tasting', 'cup', 'coffee', 'happy', 'found', 'tastes', 'n
asty', 'not', 'even', 'drink', 'disgusting', 'believe', 'anything', 'bi
tter', 'acid', 'free', 'actually', 'threw', 'away', 'two', 'bags', 'bou
ght', 'pack', 'drank', 'almost', 'realizing', 'flavor', 'reminded', 'li
ke', 'water', 'got', 'local', 'grocery', 'store', 'tasted', 'first', 't
ime', 'taste', 'corn', 'nuts', 'quite', 'hard', 'would']
```

```
100%|██████████| 6387/6387 [00:06<00:00, 940.65it/s]
```

```
(6387, 50)
[ 0.24344472 -0.70808471  0.34283369  0.01496562  0.02342463 -0.3664771
7
 -0.22399061  0.40342275 -0.50156064  0.20655961  0.80811763  0.2104350
5
```

```
  0.0111706   0.22057268  0.31608491 -0.50692293 -0.59316409 -0.5629194
9
 -1.01278746 -0.6463846  -0.24519936  0.75148114  0.29498797  0.0496100
5
 -0.48211787 -0.19455543 -0.22535177  0.33095015  0.01737486 -0.0402567
4
 -0.03465141 -0.55240346  0.25522221 -0.14062533 -0.5174084   0.2086232
1
  0.09980315 -0.08782732  0.06531491  0.07877619 -0.08086726  0.2201823
7
 -0.67607614 -0.65537713  0.17773063  0.18419596  0.83949325 -0.0077678
1
  0.20204645  0.58724389]
```

```
  3%|▍         | 148/4280 [00:00<00:02, 1473.18it/s]
```

```
number of words that occured minimum 5 times  3579
sample words  ['delicious', 'crackers', 'stay', 'nice', 'crispy', 'eve
n', 'weather', 'unfortunately', 'particular', 'packaging', 'seems', 'pr
one', 'breaking', 'transit', 'still', 'taste', 'good', 'think', 'orde
r', 'different', 'seem', 'better', 'suppose', 'close', 'get', 'mass',
'produced', 'real', 'egg', 'pretty', 'convenient', 'sister', 'bought',
'keurig', 'started', 'using', 'mom', 'cans', 'coffee', 'not', 'want',
'go', 'waste', 'could', 'make', 'k', 'cups', 'first', 'easily', 'gettin
g']
```

```
100%|██████████| 4280/4280 [00:03<00:00, 1264.10it/s]
```

```
(4280, 50)
[ 0.34059321 -0.48564376 -0.01252869 -0.08350658  0.06436349 -0.0009256
4
 -0.04048682 -0.10700869 -0.60164815  0.25326533  0.86961178 -0.2127112
2
 -0.09504026  0.24572447  0.12738632 -0.1868247  -0.54696074 -0.5067508
2
 -0.72593075 -0.47430102  0.06380745  0.57973101  0.35106747 -0.1122766
2
 -0.63582338 -0.69626547  0.1560665   0.01063126 -0.08558797 -0.0072253
4
  0.05683373 -0.58758803  0.2289229  -0.39261728 -0.11574087  0.1629338
6
```

```
 0.26212262 -0.2739659  -0.21759212  0.06113676  0.15670787  0.2809335
-0.36243556 -0.62635881  0.19648275  0.06826043  0.59058601 -0.215075
 0.21527216  0.53031745]
```

tfidf w2v

In [82]:

```
#tfidf w2v
#this is for train data
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())


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



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

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    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_train.append(sent_vec)
    row += 1
tfidf_sent_vectors_train= np.array(sent_vectors_train)
print(tfidf_sent_vectors_train.shape)
print(tfidf_sent_vectors_train[0])
```

```
#this is for test data
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


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

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

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/r
eview
    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
tfidf_sent_vectors_test= np.array(sent_vectors_test)
print(tfidf_sent_vectors_test.shape)
print(tfidf_sent_vectors_test[0])
```

```
#this is for cv data
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())


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



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

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is
 stored in this list
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/r
eview
    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_cv= np.array(sent_vectors_cv)
print(tfidf_sent_vectors_cv.shape)
print(tfidf_sent_vectors_cv[0])
```

```
100%|██████████| 8687/8687 [01:13<00:00, 117.61it/s]
```

```
(8687, 50)
[ 0.424955   -0.47383716 -0.2357066   0.15721139  0.11870111  0.0069023
6
  0.10480667 -0.29871408 -0.79075245  0.29588864  0.65820314 -0.0267759
1
 -0.0812725  -0.30829108  0.22448453 -0.27420957 -0.60425005 -0.3079123
4
 -0.54447321 -0.91239444 -0.00434475  0.64699678  0.29881632 -0.1583756
 -0.48342951 -0.73056618 -0.10592381 -0.08068542 -0.21352984 -0.1619982
8
  0.21991885 -0.38265449 -0.13816791 -0.29885154 -0.1186995   0.1656225
2
 -0.1378229  -0.2370468  -0.83159481 -0.25114585  0.77791787  0.6638960
8
 -0.95094952 -1.06049211  0.06017698 -0.08070387  0.62733026  0.2658296
9
  0.27707071  0.31143757]
```

```
100%|██████████| 6387/6387 [00:46<00:00, 136.46it/s]
```

```
(6387, 50)
[ 0.24344472 -0.70808471  0.34283369  0.01496562  0.02342463 -0.3664771
7
 -0.22399061  0.40342275 -0.50156064  0.20655961  0.80811763  0.2104350

5
```

```
  0.0111706   0.22057268  0.31608491 -0.50692293 -0.59316409 -0.5629194
9
 -1.01278746 -0.6463846  -0.24519936  0.75148114  0.29498797  0.0496100
5
 -0.48211787 -0.19455543 -0.22535177  0.33095015  0.01737486 -0.0402567
4
 -0.03465141 -0.55240346  0.25522221 -0.14062533 -0.5174084   0.2086232
1
  0.09980315 -0.08782732  0.06531491  0.07877619 -0.08086726  0.2201823
7
 -0.67607614 -0.65537713  0.17773063  0.18419596  0.83949325 -0.0077678
1
  0.20204645  0.58724389]
```

```
100%|██████████| 4280/4280 [00:25<00:00, 165.00it/s]
```

```
(4280, 50)
[ 0.34059321 -0.48564376 -0.01252869 -0.08350658  0.06436349 -0.0009256
4
 -0.04048682 -0.10700869 -0.60164815  0.25326533  0.86961178 -0.2127112
2
 -0.09504026  0.24572447  0.12738632 -0.1868247  -0.54696074 -0.5067508
2
 -0.72593075 -0.47430102  0.06380745  0.57973101  0.35106747 -0.1122766
2
 -0.63582338 -0.69626547  0.1560665   0.01063126 -0.08558797 -0.0072253
4
  0.05683373 -0.58758803  0.2289229  -0.39261728 -0.11574087  0.1629338
6
  0.26212262 -0.2739659  -0.21759212  0.06113676  0.15670787  0.2809335
 -0.36243556 -0.62635881  0.19648275  0.06826043  0.59058601 -0.215075
  0.21527216  0.53031745]
```

[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [83]:

```
#As per the restriction given in the assignment we will find BoW for kd
-tree again and seperately
#BoW
```

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=500)
vectorizer.fit(X_train) # fitting on train data ,we cant perform fit on
 test or cv
# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
#you can also check X_train_bow is of sparse matrix type or not
#below is code for that
print(type(X_train_bow))
#displaying number of unique words in each of splitted dataset
print("the number of unique words in train: ", X_train_bow.get_shape()[
1])
print("the number of unique words in cv: ", X_cv_bow.get_shape()[1])
print("the number of unique words in test: ", X_test_bow.get_shape()[1
])
```

```
After vectorizations
(8687, 500) (8687,)
(4280, 500) (4280,)
(6387, 500) (6387,)
=======================================================================
=============================
<class 'scipy.sparse.csr.csr_matrix'>
the number of unique words in train:  500
the number of unique words in cv:  500
the number of unique words in test:  500
```

In [0]:

```
#x_train,x_test,x_cv all are sparse matrix lets covert it into dense ma
trix since kd-tree accepts only dense matrix
X_train_bow=X_train_bow.toarray()
X_test_bow=X_test_bow.toarray()
X_cv_bow=X_cv_bow.toarray()
```

In [85]:

```
#here i am applying kd tree knn to bow vectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]

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

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

In [0]:
```
best_k=53
```

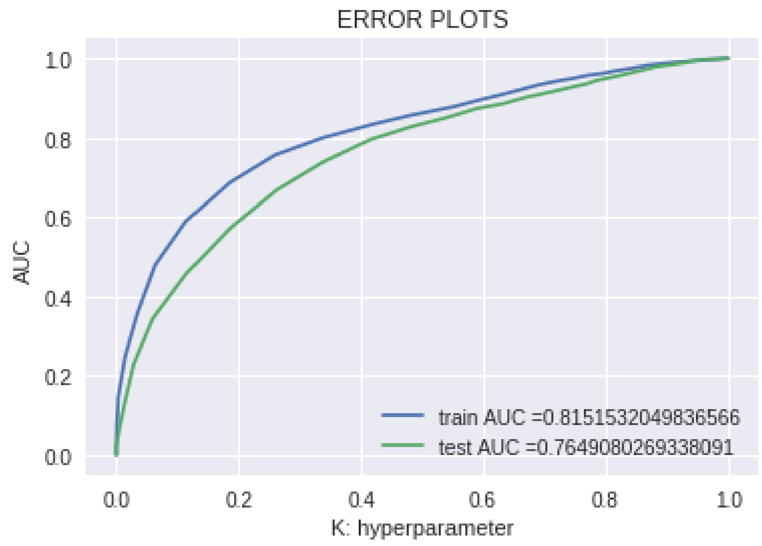
In [87]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

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

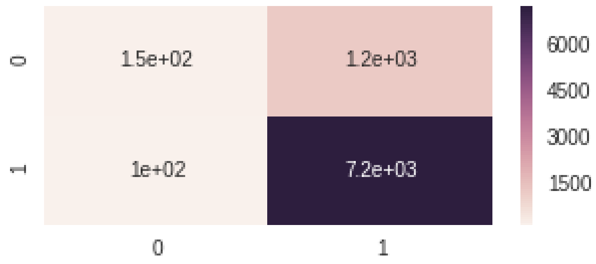
```
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [88]:
```
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(X_train_bow))
df_1= pd.DataFrame(arr1, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

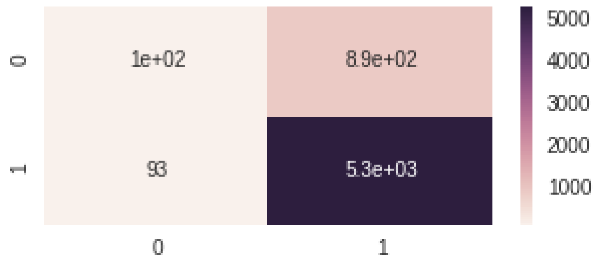
Out[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc6b549588>
```



In [89]:
```
print("Test confusion matrix")
arr2=confusion_matrix(y_test, neigh.predict(X_test_bow))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

```
Test confusion matrix
```

Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc6c9ed198>



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

In [90]:
```
#As per the restriction given in the assignment we will find BoW for kd
-tree again and seperately
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_feature
s=500)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
```

```
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(X_train)
X_test_tf_idf = tf_idf_vect.transform(X_test)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(X_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_shap
e())
print("the number of unique words including both unigrams and bigrams "
, X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['able', 'absolutely',
'actually', 'add', 'added', 'aftertaste', 'ago', 'almost', 'also', 'alt
ernative']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (8687, 500)
the number of unique words including both unigrams and bigrams  500
```

In [0]:

```
#x_train,x_test,x_cv all are sparse matrix lets covert it into dense ma
trix since kd-tree accepts only dense matrix
X_train_tf_idf=X_train_tf_idf.toarray()
X_test_tf_idf=X_test_tf_idf.toarray()
X_cv_tf_idf=X_cv_tf_idf.toarray()
```

In [92]:

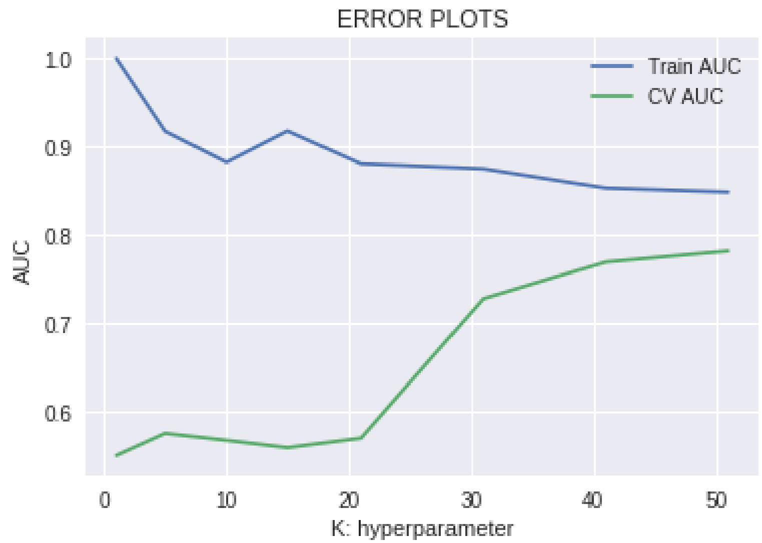
```
#here i am applying kd tree knn to tfidf vectorizer
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(X_train_tf_idf, y_train)
```

```
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_tf_idf)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tf_idf)[:,1]

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

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



In [0]:
```
best_k=51
```

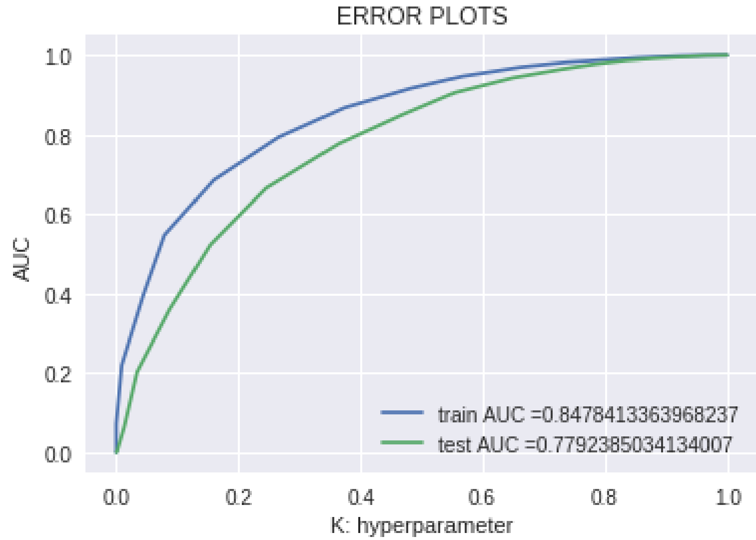
In [94]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
```

```
neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(X_train_tf_idf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_pro
ba(X_train_tf_idf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(
X_test_tf_idf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_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()

print("="*100)
```

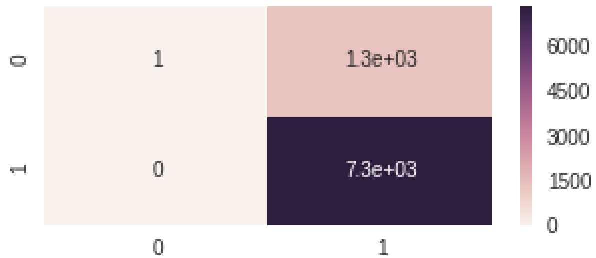



```
==========================================================================================================================
```

In [95]:
```
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(X_train_tf_idf))
df_1= pd.DataFrame(arr1, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

Out[95]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fcc6b549ac8>`

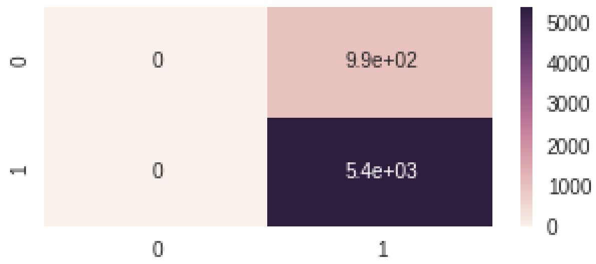


In [96]:
```
print("Test confusion matrix")
```

```
arr2=confusion_matrix(y_test, neigh.predict(X_test_tf_idf))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

Test confusion matrix

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc6baec9e8>



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

In [97]:
```
#here i am applying kd tree knn to avg w2v vectorizer
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(sent_vectors_cv)[:,1]

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

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

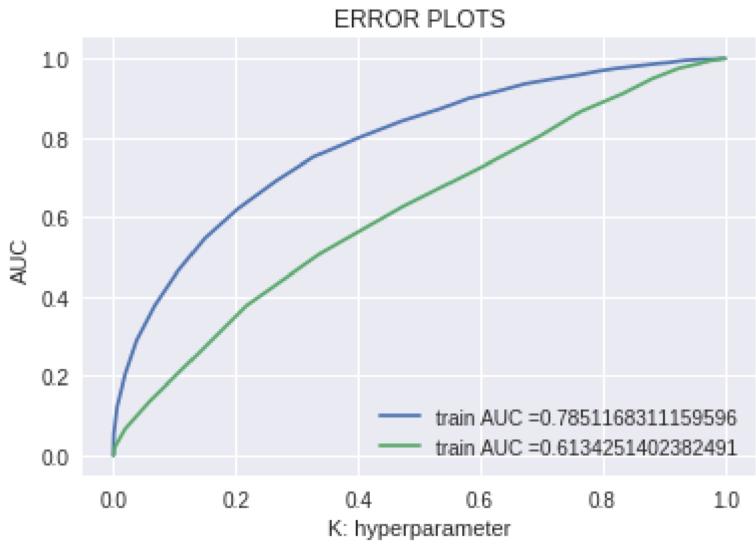
In [0]:
```
best_k=53
```

In [99]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(sent_vectors_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

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

```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_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()

print("="*100)
```



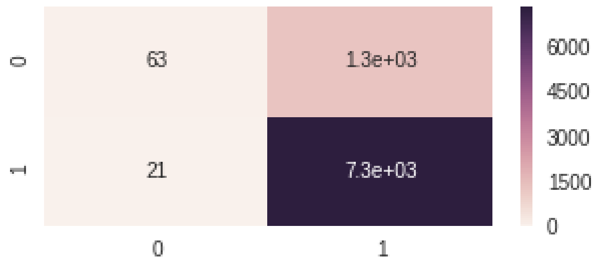

```
====================================================================================================
```

In [100]:

```
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(sent_vectors_train))
df_1= pd.DataFrame(arr1, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

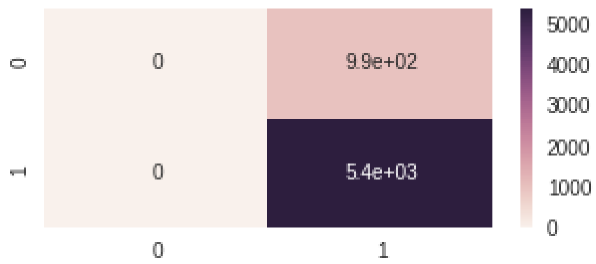
Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc6c2c5438>



In [101]:
```
print("Test confusion matrix")
arr2=confusion_matrix(y_test, neigh.predict(sent_vectors_test))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

Test confusion matrix

Out[101]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcc69f29048>



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

In [102]:
```
#here i am applying kd tree knn to tfidf w2v vectorizer
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
```

```
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(tfidf_sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(tfidf_sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(tfidf_sent_vectors_cv)[:,1]

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

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

In [0]:
```
best_k=53
```

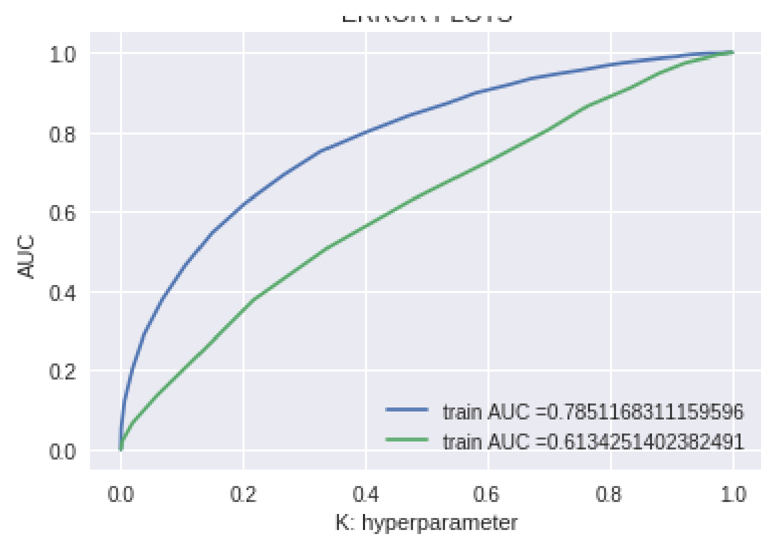
In [104]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(tfidf_sent_vectors_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_pro
ba(tfidf_sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(
tfidf_sent_vectors_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_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()

print("="*100)
```

ERROR PLOTS

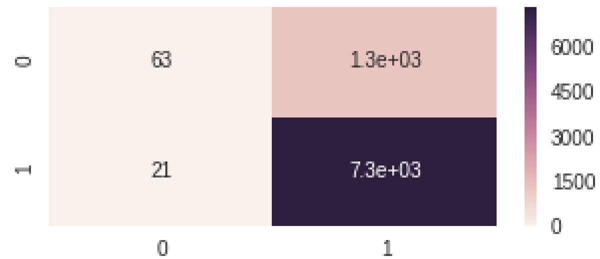



```
==========================================================================
==============================
```

In [105]:
```
print("Train confusion matrix")
arr1=confusion_matrix(y_train, neigh.predict(tfidf_sent_vectors_train))
df_1= pd.DataFrame(arr1, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_1, annot=True)
```

```
Train confusion matrix
```

Out[105]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc6c263208>
```



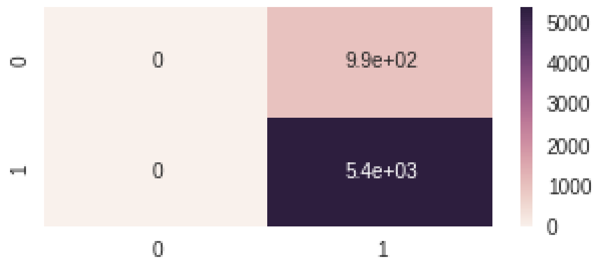
In [106]:
```
print("Test confusion matrix")
```

```
arr2=confusion_matrix(y_test, neigh.predict(tfidf_sent_vectors_test))
df_2= pd.DataFrame(arr2, range(2),range(2))
plt.figure(figsize = (5,2))
sn.heatmap(df_2, annot=True)
```

```
Test confusion matrix
```

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc6a23fb70>
```



[6] Conclusions

In [108]:

```
# creating
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]
x.add_row(["BoW", "Brute",41 , 0.65])
x.add_row(["tfidf", "Brute",31 , 0.50])
x.add_row(["avg w2v", "Brute",51 , 0.79])
x.add_row(["tfidfw2v", "Brute",53 , 0.79])
print(x)
print("-----------------------------------------------------------------
------------------")
y = PrettyTable()
y.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]
y.add_row(["BoW", "kd tree",53 , 0.76])
y.add_row(["tfidf", "kd tree",51 , 0.77])
y.add_row(["avg w2v", "kd tree",53 , 0.61])
y.add_row(["tfidf", "kd tree",53 , 0.61])
print(y)
```

Vectorizer	Model	Hyper parameter	AUC
BoW	Brute	41	0.65
tfidf	Brute	31	0.5
avg w2v	Brute	51	0.79
tfidfw2v	Brute	53	0.79

Vectorizer	Model	Hyper parameter	AUC
BoW	kd tree	53	0.76
tfidf	kd tree	51	0.77
avg w2v	kd tree	53	0.61
tfidf	kd tree	53	0.61