Amazon-Reviews-on-KNN_100K

March 5, 2019

1 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.

2 [1]. Reading Data

2.1 [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 [236]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import confusion_matrix,precision_score,recall_score,f1_score,r

from tqdm import tqdm
import os
```

In [237]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data poi
```

```
# 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
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 2

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negat
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 (200000, 10)
```

Out[237]:

	Id	ProductId	UserId	ProfileName	\
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\
0	1	1	1	1303862400	
1	0	0	0	1346976000	
2	1	1	1	1219017600	

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...

In [238]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [239]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[239]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [240]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[240]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [241]:
```
display['COUNT(*)'].sum()
```

Out[241]: 393063

3 [2] Exploratory Data Analysis

3.1 [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 [242]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[242]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1399507200	LOVE IT	I bought this for my husband who plays the pia...
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1399507200	LOVE IT	I bought this for my husband who plays the pia...
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1399507200	LOVE IT	I bought this for my husband who plays the pia...
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1399507200	LOVE IT	I bought this for my husband who plays the pia...
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1399507200	LOVE IT	I bought this for my husband who plays the pia...

```
   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                            Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

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 [243]: #Sorting data according to ProductId in ascending order
          sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fa

In [244]: #Deduplication of entries
          final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep
          final.shape

Out[244]: (160178, 10)

In [245]: #Checking to see how much % of data still remains
          (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

Out[245]: 80.089
```

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 [246]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[246]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

In [247]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [248]:
```
#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()
```

```
(160176, 10)
```

Out[248]:
```
1    134799
0     25377
Name: Score, dtype: int64
```

In [249]:
```
##Sorting data for Time Based Splitting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, inplace=False, l


final = time_sorted_data.take(np.random.permutation(len(final))[:100000])
print(final.shape)
final.head()
```

```
(100000, 10)
```

Out[249]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
59213	64324	B001EO5YAC	ACJFVBXMC3RZ9	Timothy J. Giordano	0	0	1	1315958400	Excellent coffee	This coffee is NOT for espresso. It is made f...
41118	44688	B001EQ55RW	AU4H7OG5UFDWO	S. Gould "gouldpjaks"	0	0	1	1211328000	sinfully great taste	WOW! I can't believe that these chocolaty almo...
62930	68370	B003EM7J9Q	A19Y4ONOPK7N5V	Mac	7	9	0	1300752000	WARNING! CONTAINS MSG and MALTODEXTRIN	Being a gluten sensitive individual, I was lur...
99621	108212	B007TJGY46	A1GAR9XCQ195B2	Yuan-tai Lee "Anthony Lee"	0	0	1	1289088000	very bold	I never knew that the Jet Fuel flavor is so bo...
137422	149146	B002SQIRDG	A137GUM4X3D8J6	Laverne R. Funderburg "Classy Sassy Lady"	0	0	1	1328659200	Trident Layer GreenApple+GoldenPineapple Sugar...	We haven't been able to find this product loca...

4 [3] Preprocessing

4.1 [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 [250]: # printing some random reviews
          sent_0 = final['Text'].values[0]
          print(sent_0)
          print("="*50)

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

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

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

```
This coffee is NOT for espresso.  It is made for either a drip machine, stove top, or percolat
==================================================
I am British and personally I cannot stand Lipton tea for hot tea. Ice tea sure hot no way. I a
==================================================
This stuff is terrible! It tastes like a homemade fortified wine. We served it at a dinner part
==================================================
I love these bars they taste great and keep you feeling full! Try them warmed in the microwave
==================================================
```

```
In [251]: # 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)
```

```
This coffee is NOT for espresso.  It is made for either a drip machine, stove top, or percolat
```

```
In [252]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-al
          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)
```

```
This coffee is NOT for espresso.  It is made for either a drip machine, stove top, or percolato
==================================================
I am British and personally I cannot stand Lipton tea for hot tea. Ice tea sure hot no way. I a
==================================================
This stuff is terrible! It tastes like a homemade fortified wine. We served it at a dinner part
==================================================
I love these bars they taste great and keep you feeling full! Try them warmed in the microwave
```

In [253]:
```
# 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 [254]:
```
sent_1500 = decontracted(sent_1500)
```

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

```
This stuff is terrible! It tastes like a homemade fortified wine. We served it at a dinner part
==================================================
```

In [255]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
This coffee is NOT for espresso.  It is made for either a drip machine, stove top, or percolato
```

In [256]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
This stuff is terrible It tastes like a homemade fortified wine We served it at a dinner party
```

In [257]:
```
# 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', 'oursel
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [258]:
```
# Combining all the above stundents
from tqdm import tqdm
prepr_rev = []
# 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 stopw
            prepr_rev.append(sentance.strip())
```

```
100%|| 100000/100000 [00:54<00:00, 1833.72it/s]
```

```
In [259]: prepr_rev[1500]
```

```
Out[259]: 'stuff terrible tastes like homemade fortified wine served dinner party ten nobody li
```

```
In [260]: print(len(prepr_rev))
          final.shape
```

```
100000
```

```
Out[260]: (100000, 10)
```

```
In [261]: final ['prepr_rev']= prepr_rev
          final.head(5)
```

```
Out[261]:             Id   ProductId          UserId  \
          59213    64324  B001EO5YAC   ACJFVBXMC3RZ9
          41118    44688  B001EQ55RW   AU4H7OG5UFDWO
          62930    68370  B003EM7J9Q  A19Y4ONOPK7N5V
          99621   108212  B007TJGY46  A1GAR9XCQ195B2
          137422  149146  B002SQIRDG  A137GUM4X3D8J6

                                              ProfileName  HelpfulnessNumerator  \
          59213                          Timothy J. Giordano                     0
          41118                       S. Gould "gouldpjaks"                      0
          62930                                         Mac                      7
          99621                  Yuan-tai Lee "Anthony Lee"                      0
          137422  Laverne R. Funderburg "Classy Sassy Lady"                      0

                  HelpfulnessDenominator  Score        Time  \
          59213                        0      1  1315958400
          41118                        0      1  1211328000
          62930                        9      0  1300752000
          99621                        0      1  1289088000
          137422                       0      1  1328659200

                                                      Summary  \
          59213                                 Excellent coffee
```

```
41118                                 sinfully great taste
62930             WARNING! CONTAINS MSG and MALTODEXTRIN
99621                                            very bold
137422  Trident Layer GreenApple+GoldenPineapple Sugar...

                                                    Text  \
59213   This coffee is NOT for espresso.  It is made f...
41118   WOW! I can't believe that these chocolaty almo...
62930   Being a gluten sensitive individual, I was lur...
99621   I never knew that the Jet Fuel flavor is so bo...
137422  We haven't been able to find this product loca...

                                                 prepr_rev
59213   coffee not espresso made either drip machine s...
41118   wow not believe chocolaty almonds contain no c...
62930   gluten sensitive individual lured false sense ...
99621        never knew jet fuel flavor bold good morning
137422  not able find product locally several months f...
```

In [262]:
```
# store final table into an SQlLite table for future.
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn,  schema=None, if_exists='replace',index=True, index_lab
conn.close()
#Loading data
conn = sqlite3.connect('final.sqlite')
data=pd.read_sql_query("""select * from Reviews""",conn)
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [263]:
```
from sklearn.model_selection import train_test_split
#splitting data into Train, C.V and Test
X_train, X_test, y_train, y_test = train_test_split(final ['prepr_rev'], final['Score
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33)
print("Train:",X_train.shape,y_train.shape)
print("CV:",X_cv.shape,y_cv.shape)
print("Test:",X_test.shape,y_test.shape)
```

```
Train: (44890,) (44890,)
CV: (22110,) (22110,)
Test: (33000,) (33000,)
```

In [264]:
```
vectorizer = CountVectorizer(min_df=10, max_features=500)
vectorizer.fit(X_train)
```

```
#vectorizer.fit(X_train) # fit has to happen only on train data
# 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)
```

```
After vectorizations
(44890, 500) (44890,)
(22110, 500) (22110,)
(33000, 500) (33000,)
```

5.2 [4.3] TF-IDF

```
In [265]: vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=500)
          tf_idf_vect = vect.fit(X_train)
          # we use the fitted CountVectorizer to convert the text to vector
          X_train_tfidf = tf_idf_vect.transform(X_train)
          X_cv_tfidf = tf_idf_vect.transform(X_cv)
          X_test_tfidf = tf_idf_vect.transform(X_test)
          print("After vectorizations")
          print(X_train_tfidf.shape, y_train.shape)
          print(X_cv_tfidf.shape, y_cv.shape)
          print(X_test_tfidf.shape, y_test.shape)
```

```
After vectorizations
(44890, 500) (44890,)
(22110, 500) (22110,)
(33000, 500) (33000,)
```

5.3 [4.4] Word2Vec

```
In [266]: # List of sentence in X_train text
          sent_of_train=[]
          for sent in X_train:
              sent_of_train.append(sent.split())

          # List of sentence in X_test text
          sent_of_test=[]
          for sent in X_test:
              sent_of_test.append(sent.split())
```

```
        # Train your own Word2Vec model using your own train text corpus
        # min_count = 5 considers only words that occured atleast 5 times
        w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)
        print(w2v_model.wv.most_similar('great'))
        print('='*50)
        print(w2v_model.wv.most_similar('worst'))

        w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
[('good', 0.8355696201324463), ('fantastic', 0.81317538022995), ('awesome', 0.8107229471206665)
==================================================
[('best', 0.7373487949371338), ('tastiest', 0.7226552963256836), ('greatest', 0.721556782722473
number of words that occured minimum 5 times  13052
```

```
In [267]: 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  13052
sample words  ['chef', 'michael', 'grilled', 'sirloin', 'flavor', 'went', 'well', 'dogs', 'olde
```

5.4 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

```
In [268]: i=0
          list_of_sentance_cv=[]
          for sentance in X_cv:
              list_of_sentance_cv.append(sentance.split())
```

```
In [269]: # 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, you might need
              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_cv = np.array(sent_vectors_cv)
          print(sent_vectors_cv.shape)
          print(sent_vectors_cv[0])
```

```
100%|| 22110/22110 [01:20<00:00, 275.45it/s]


(22110, 50)
[ 6.40141896e-01  1.65565873e-01  5.45156041e-04 -4.83197974e-02
  8.14865203e-01  1.66074187e-01  3.83481219e-01  1.08467172e+00
  2.99989388e-01  9.37161609e-02 -7.11664122e-01 -2.46524717e-01
  1.46751736e-01 -8.21764619e-01 -1.01640202e+00 -6.57342891e-01
  4.76908823e-02  4.98940908e-01 -4.61014963e-01 -8.12570147e-01
 -1.01199670e+00 -1.64849413e-02  4.77687016e-01 -3.65739985e-01
 -3.77690720e-01 -2.45962588e-01 -2.17871111e-01 -1.83557018e-01
  2.41751056e-01  1.16346208e-01 -1.40221407e-01  1.73522961e-01
 -2.01370255e-01 -4.04055710e-01  5.72291207e-02 -2.40862023e-01
  8.32034365e-02 -2.37122482e-01 -1.91606928e-01 -1.09323112e+00
  6.58239331e-01  4.10074698e-01 -6.40845558e-01 -1.05233904e+00
  6.39282192e-01  1.55575553e+00 -7.68250938e-01  7.76720946e-02
 -4.39380927e-01 -1.16122336e-01]
```

```
In [270]: # compute average word2vec for X_test .
          test_vectors = [];
          for sent in tqdm(sent_of_test):
              sent_vec = np.zeros(50)
              cnt_words =0;
              for word in sent: #
                  if word in w2v_words:
                      vec = w2v_model.wv[word]
                      sent_vec += vec
                      cnt_words += 1
              if cnt_words != 0:
                  sent_vec /= cnt_words
              test_vectors.append(sent_vec)

          test_vectors = np.array(test_vectors)

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

```
100%|| 33000/33000 [01:48<00:00, 304.52it/s]

(33000, 50)
[ 0.13149368  0.53507636  0.03349382  0.24257337  0.06364785 -0.09737388
  0.29055828  0.53663889 -0.42472929  0.06558944 -0.06801226 -0.05606223
 -0.07747965 -0.15261251 -0.12327582 -0.01578109 -0.00089555  0.28218343
 -0.17076624 -0.3808611  -0.16170004  0.10303379  0.2130523  -0.40600754
 -0.49868093 -0.42266457  0.09850173  0.07533056 -0.1556289   0.19961759
  0.69414762 -0.08483144 -0.50951038 -0.52698408  0.06863119  0.07362089
  0.03136746  0.02432328 -0.59137582 -0.76118302  0.69535185  0.14526926
 -0.01985027 -0.19174361  0.0057541   0.44852804 -0.44190205 -0.01999674
```

```
 -0.15912292  0.21023861]
```

```
In [271]: # compute average word2vec for X_train .
          train_vectors = [];
          for sent in tqdm(sent_of_train):
              sent_vec = np.zeros(50)
              cnt_words =0;
              for word in sent: #
                  if word in w2v_words:
                      vec = w2v_model.wv[word]
                      sent_vec += vec
                      cnt_words += 1
              if cnt_words != 0:
                  sent_vec /= cnt_words
              train_vectors.append(sent_vec)

          train_vectors = np.array(train_vectors)

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

```
100%|| 44890/44890 [02:06<00:00, 353.72it/s]

(44890, 50)
[ 0.55150505  0.79742902 -0.94709299  0.07702221 -0.44349033  0.22306524
  0.34625905  0.30404395  0.56758892  0.40178951  0.15039816 -0.05742265
  0.1685874  -0.38224099 -0.45617988  0.01628661 -0.20033693  0.07101953
  0.53640892  0.1770468  -0.18105118  0.11027315  0.92701126 -0.27100497
  0.09039791 -0.78234372 -0.09383892  0.58044558  0.40189414  0.08147066
 -0.13695744 -0.32579878 -0.04375842 -0.08581809 -1.004104    0.07182197
  0.25475633 -0.10741304 -0.3534188  -0.59304814  1.38036202  0.81544861
  0.35184077 -0.39782124 -0.73262468  1.20879144  0.26702491 -0.30910442
 -0.51106485  0.48434056]
```

[4.4.1.2] TFIDF weighted W2v

```
In [272]: tf_idf_vect = TfidfVectorizer()

          # final_tf_idf1 is the sparse matrix with row= sentence, col=word and cell_val = tfi
          final_tf_idf1 = tf_idf_vect.fit_transform(X_train)
          dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))
```

```
        # tfidf words/col-names
        tfidf_feat = tf_idf_vect.get_feature_names()

        # compute TFIDF Weighted Word2Vec for X_test .
        tfidf_test_vectors = [];
        row=0;
        for sent in tqdm(sent_of_test):
            sent_vec = np.zeros(50)
            weight_sum =0;
            for word in sent:
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_test_vectors.append(sent_vec)

        tfidf_test_vectors = np.array(tfidf_test_vectors)
        print(tfidf_test_vectors.shape)
        print(tfidf_test_vectors[0])
```

```
100%|| 33000/33000 [19:17<00:00, 28.51it/s]
```

```
(33000, 50)
[ 0.05674556  0.45534664  0.00478524  0.10021247  0.14966017 -0.15160245
  0.14770405  0.354371   -0.22304123  0.01087953 -0.07946353 -0.04112905
  0.0169637  -0.08088503 -0.04408359  0.03733224 -0.16256878  0.21873269
 -0.0885564  -0.11193214 -0.11707955  0.06813254  0.18394441 -0.25571486
 -0.34545013 -0.2815951   0.11274173  0.11160742 -0.10503162  0.09896749
  0.4604654  -0.0864644  -0.29628719 -0.41639876 -0.08949415  0.03268891
  0.05832278  0.02870465 -0.4545801  -0.49367995  0.52847396  0.1305022
 -0.02626145 -0.26193574  0.10635008  0.37774505 -0.32122394  0.07186619
 -0.17350636  0.25186179]
```

```
In [273]: # compute TFIDF Weighted Word2Vec for X_train .
          tfidf_train_vectors = [];
          row=0;
          for sent in tqdm(sent_of_train):
              sent_vec = np.zeros(50)
              weight_sum =0;
              for word in sent:
                  if word in w2v_words and word in tfidf_feat:
                      vec = w2v_model.wv[word]
```

```
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_train_vectors.append(sent_vec)

        tfidf_train_vectors = np.array(tfidf_train_vectors)
        print(tfidf_train_vectors.shape)
        print(tfidf_train_vectors[0])
```

```
100%|| 44890/44890 [24:48<00:00, 30.15it/s]


(44890, 50)
[ 0.6604019   1.05134164 -1.23869904  0.07978345 -0.6585853   0.15262296
  0.46197408  0.28721333  0.71947858  0.34444516  0.52122447 -0.27308447
  0.1730235  -0.48959467 -0.57064343 -0.10663995 -0.12631126  0.03022428
  0.7270707   0.56086075 -0.09364828  0.17929872  1.13248566 -0.20724076
  0.28880089 -0.80982647 -0.19054757  0.76324156  0.61436021 -0.04084342
 -0.33269857 -0.45597689  0.23606132  0.07524744 -1.2631875   0.08809292
  0.12740885 -0.09895227 -0.2356558  -0.47455338  1.77025447  0.91280937
  0.31991652 -0.49194256 -1.00246888  1.41514742  0.53722848 -0.54572239
 -0.86944496  0.5310506 ]
```

6 [5] Assignment 3: KNN

```
<li><strong>Apply Knn(brute force version) on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Apply Knn(kd tree version) on these feature sets</strong>
    <br><font color='red'>NOTE: </font>sklearn implementation of kd-tree accepts only dense mat
    <ul>
        <li><font color='red'>SET 5:</font>Review text, preprocessed one converted into vectors
        <pre>
        count_vect = CountVectorizer(min_df=10, max_features=500)
        count_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 6:</font>Review text, preprocessed one converted into vectors
        <pre>
```

```
            tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
            tf_idf_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best K)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

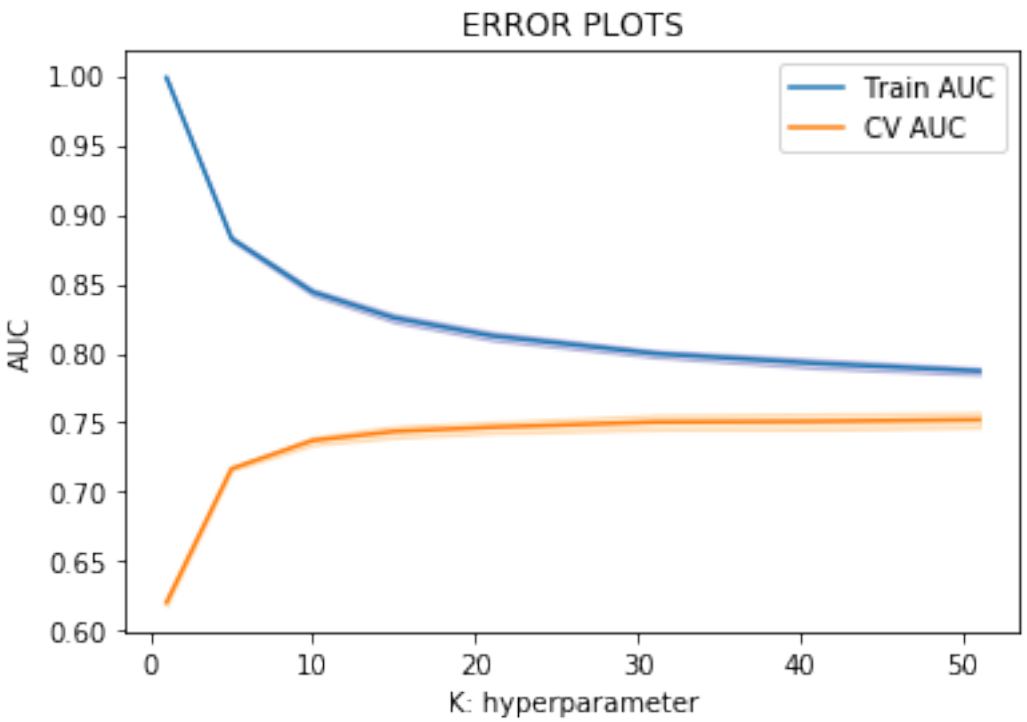
Note: Data Leakage

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.

6.1 [5.1] Applying KNN brute force

6.1.1 [5.1.1] Applying KNN brute force on BOW, SET 1

In [274]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSear
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier(algorithm='brute')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_bow, y_train)

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

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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

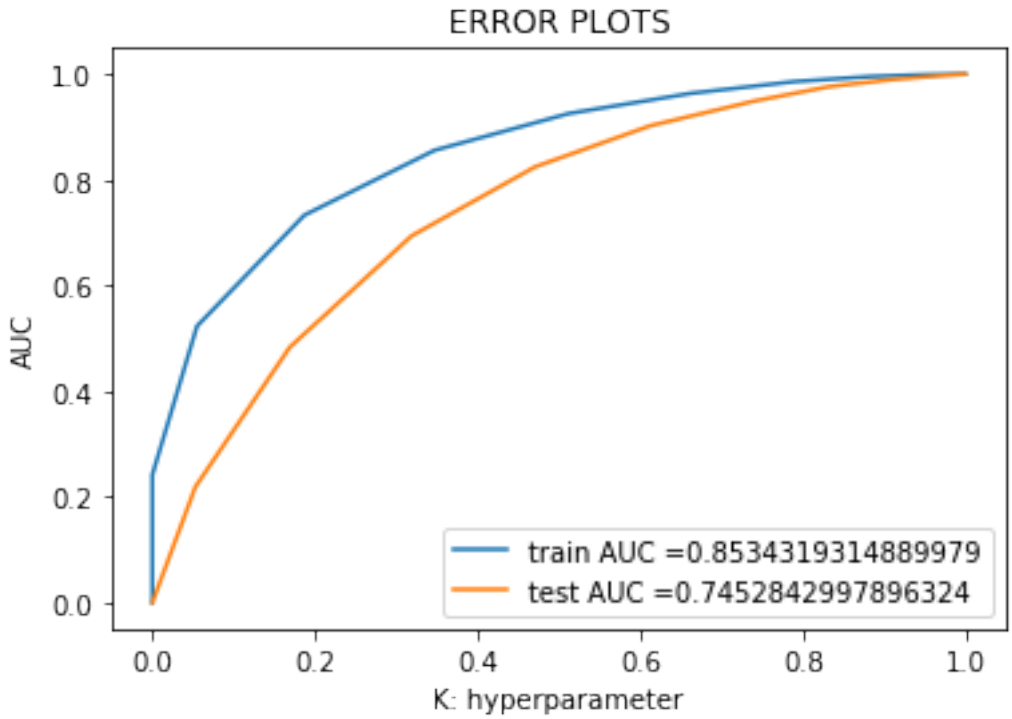


In [277]:
```
best_k=10
```

In [278]:
```
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 o
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bo
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:

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)

from sklearn.metrics import confusion_matrix
```

```
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))


# Variables for table
bow_brute_K = best_k
bow_brute_train= train_auc
bow_brute_test = cv_auc
```

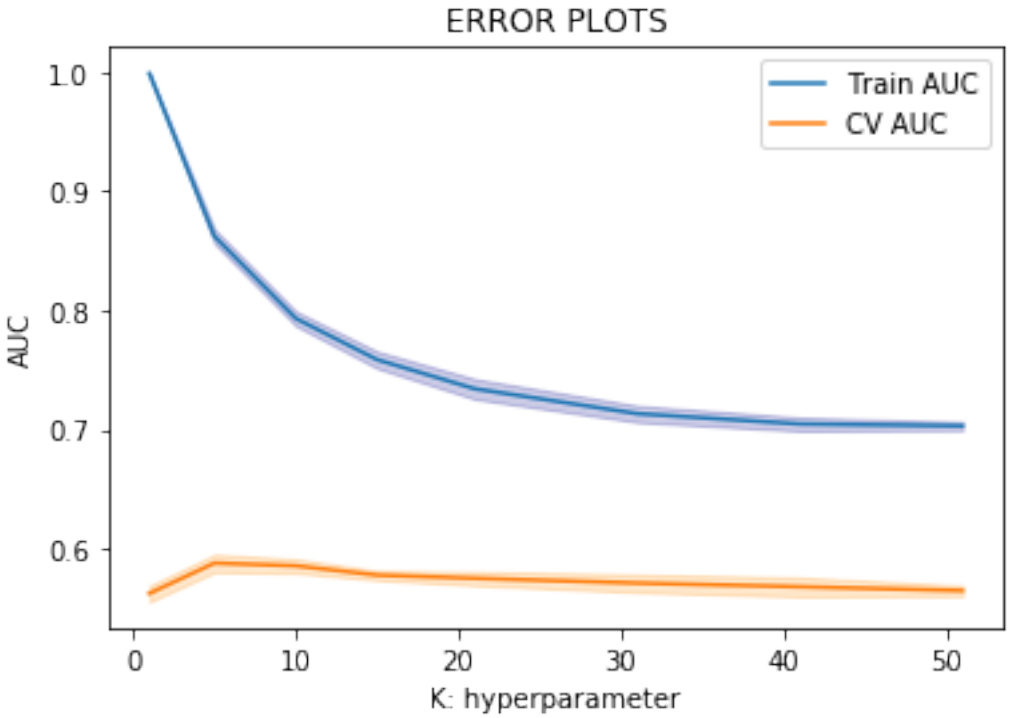



```
==========================================================================================
Train confusion matrix
[[ 3507  3684]
 [ 2808 34891]]
Test confusion matrix
[[ 1976  3126]
 [ 2720 25178]]
```

6.1.2 [5.1.2] Applying KNN brute force on TFIDF, SET 2

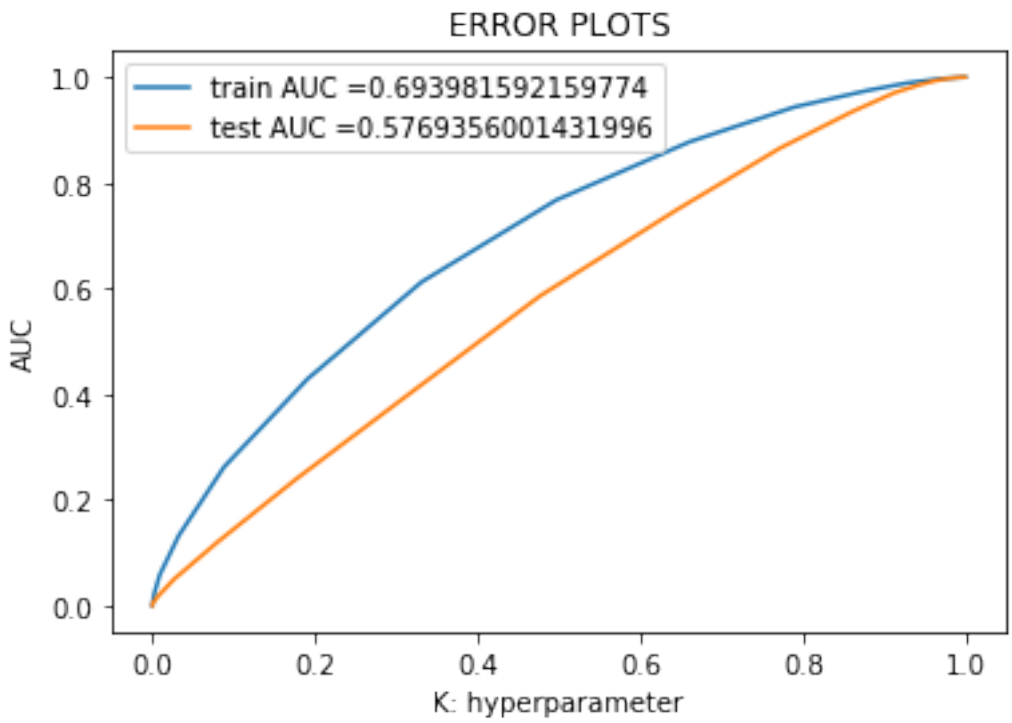
```
In [279]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSear
          from sklearn.model_selection import GridSearchCV
```

```
neigh = KNeighborsClassifier(algorithm='brute')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf, y_train)

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

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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



In [282]:
```
best_k = 51
```

In [283]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_tfidf, y_train)


# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tf
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tfidf)

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)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf)))



# Variables for table


tfidf_brute_K = best_k
tfidf_brute_train = train_auc
tfidf_brute_test = cv_auc
```

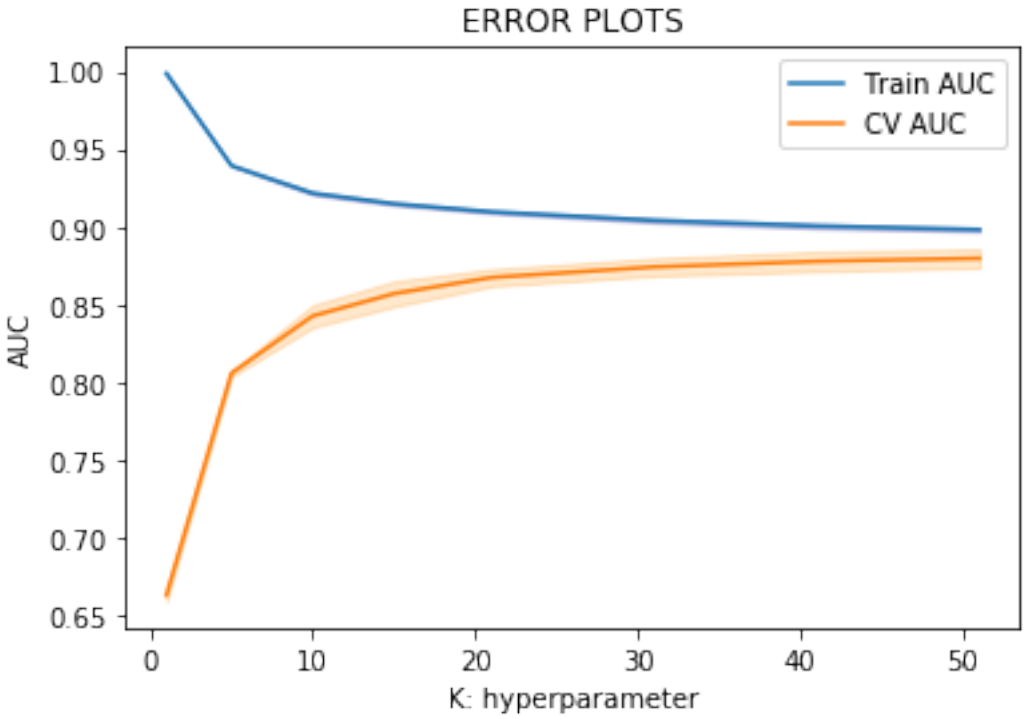



```
====================================================================================================
Train confusion matrix
[[   10  7181]
 [    2 37697]]
Test confusion matrix
[[    9  5093]
 [    1 27897]]
```

6.1.3 [5.1.3] Applying KNN brute force on AVG W2V, SET 3

```
In [284]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSear
          from sklearn.model_selection import GridSearchCV

          neigh = KNeighborsClassifier(algorithm='brute')
          parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
          clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
          clf.fit(train_vectors, y_train)

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

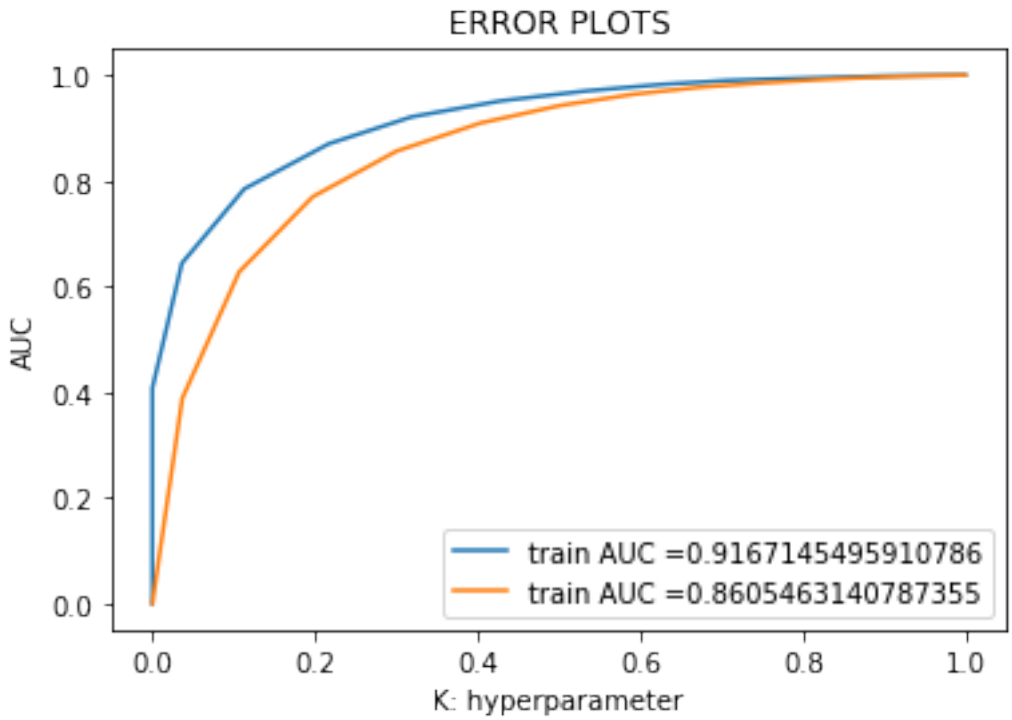
```
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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



In [285]:
```
best_k = 15
```

In [286]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(train_vectors, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(train_vecto
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(test_vectors)
```

```
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()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(train_vectors)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(test_vectors)))


# Variables for table
Avg_Word2Vec_brute_K = best_k
Avg_Word2Vec_brute_train = train_auc
Avg_word2Vec_brute_test = cv_auc
```

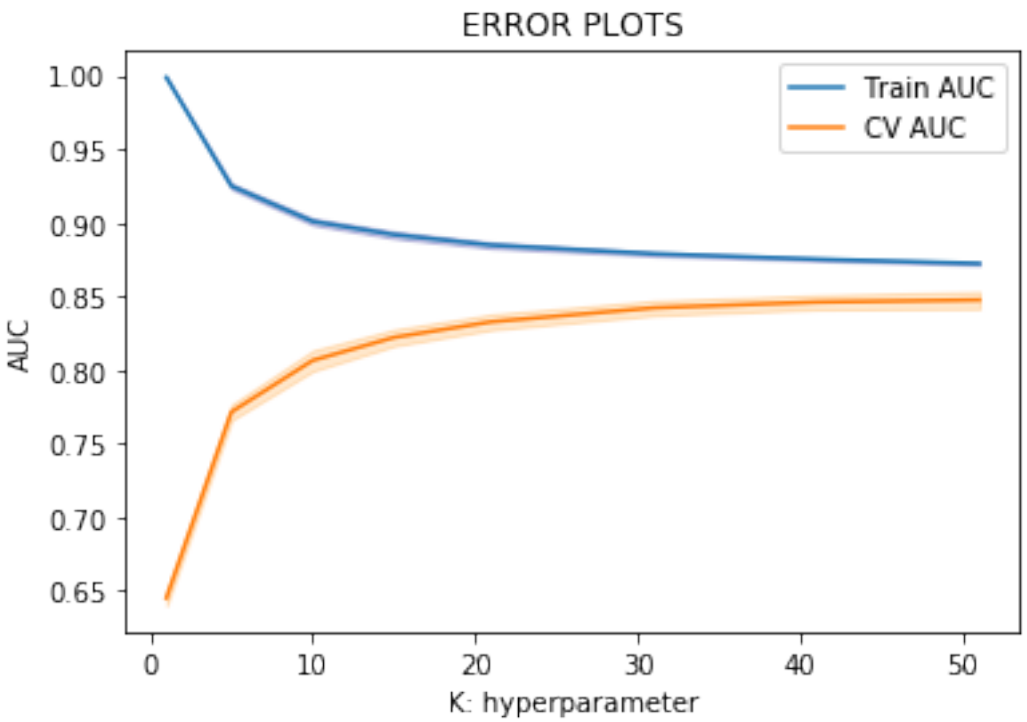


==========

```
Train confusion matrix
[[ 2683  4508]
 [  660 37039]]
Test confusion matrix
[[ 1666  3436]
 [  635 27263]]
```

6.1.4 [5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

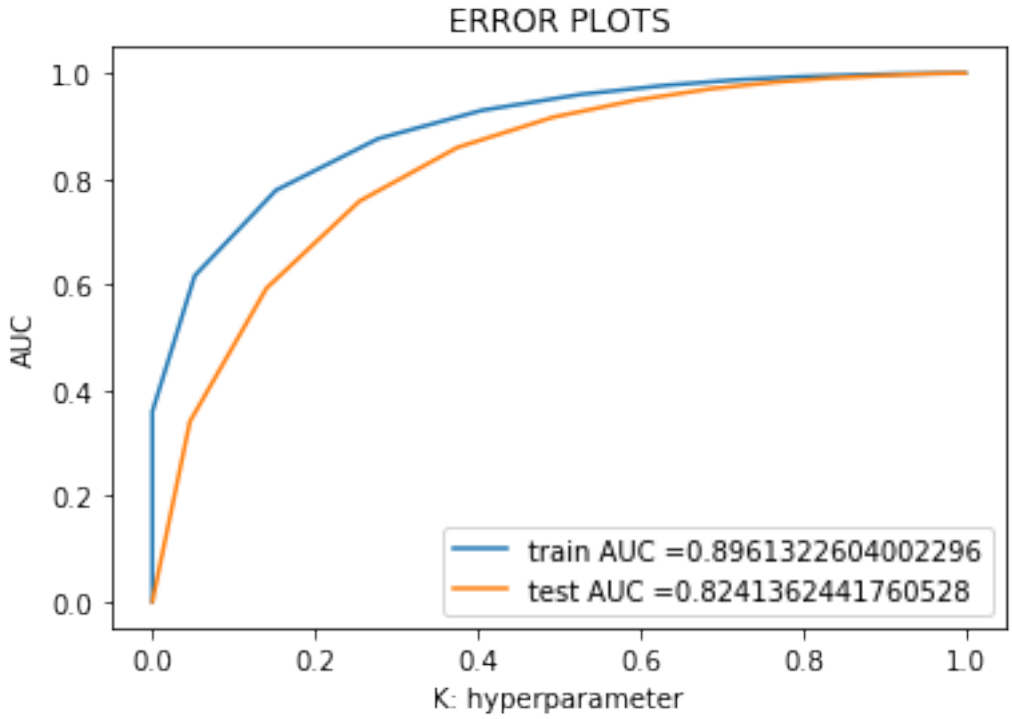
In [287]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSear
from sklearn.model_selection import GridSearchCV


neigh = KNeighborsClassifier(algorithm='brute')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(tfidf_train_vectors, y_train)

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

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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



In [288]:
```
best_k=14
```

In [289]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(tfidf_train_vectors, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_trai
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_test_ve

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)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(tfidf_train_vectors)))
```

```
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(tfidf_test_vectors)))

# Variables for table


TFIDF_Word2Vec_brute_K = best_k
TFIDF_Word2Vec_brute_train = train_auc
TFIDF_word2Vec_brute_test = cv_auc
```

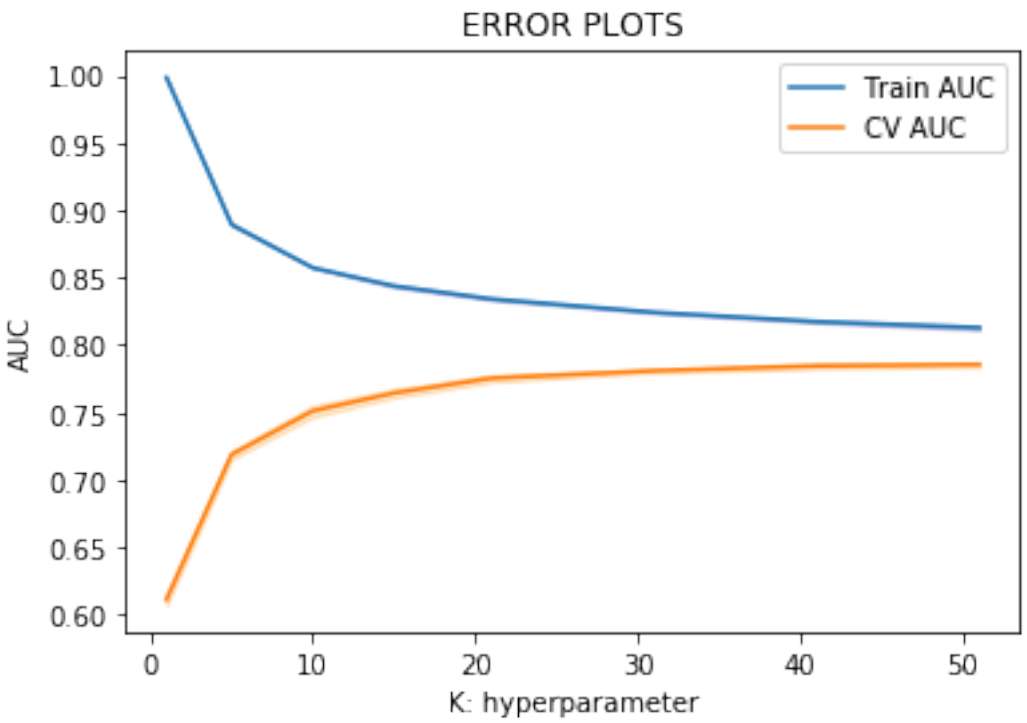



```
==========================================================================================
Train confusion matrix
[[ 2650  4541]
 [  869 36830]]
Test confusion matrix
[[ 1591  3511]
 [  820 27078]]
```

6.2 [5.2] Applying KNN kd-tree

6.2.1 [5.2.1] Applying KNN kd-tree on BOW, SET 5

In [290]:
```
# Please write all the code with proper documentation
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=100)
Xtrain_bow = svd.fit_transform(X_train_bow)
Xtest_bow = svd.transform(X_test_bow)


# Please write all the code with proper documentation
# Importing libraries
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier(algorithm='kd_tree')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(Xtrain_bow, y_train)

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

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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

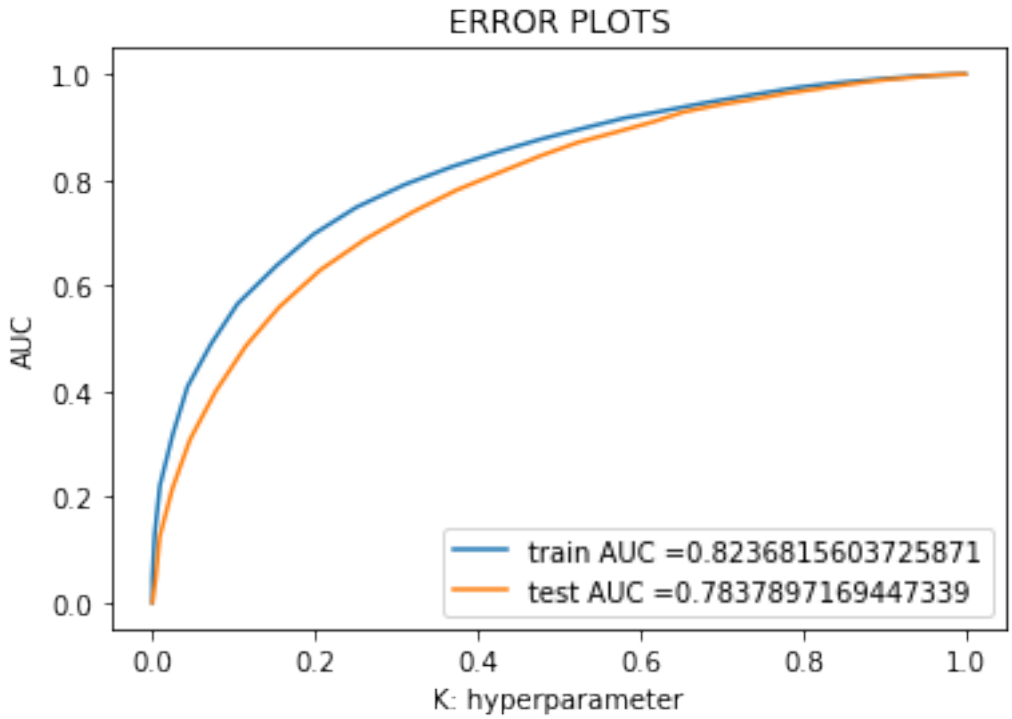


In [292]:
```
best_k=41
```

In [293]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(Xtrain_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(Xtrain_bow
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(Xtest_bow)[:,

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)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(Xtrain_bow)))
```

```
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(Xtest_bow)))
```

```
# Variables for table
bow_kdTree_K = best_k
bow_kdTree_train = train_auc
bow_kdTree_test = cv_auc
```

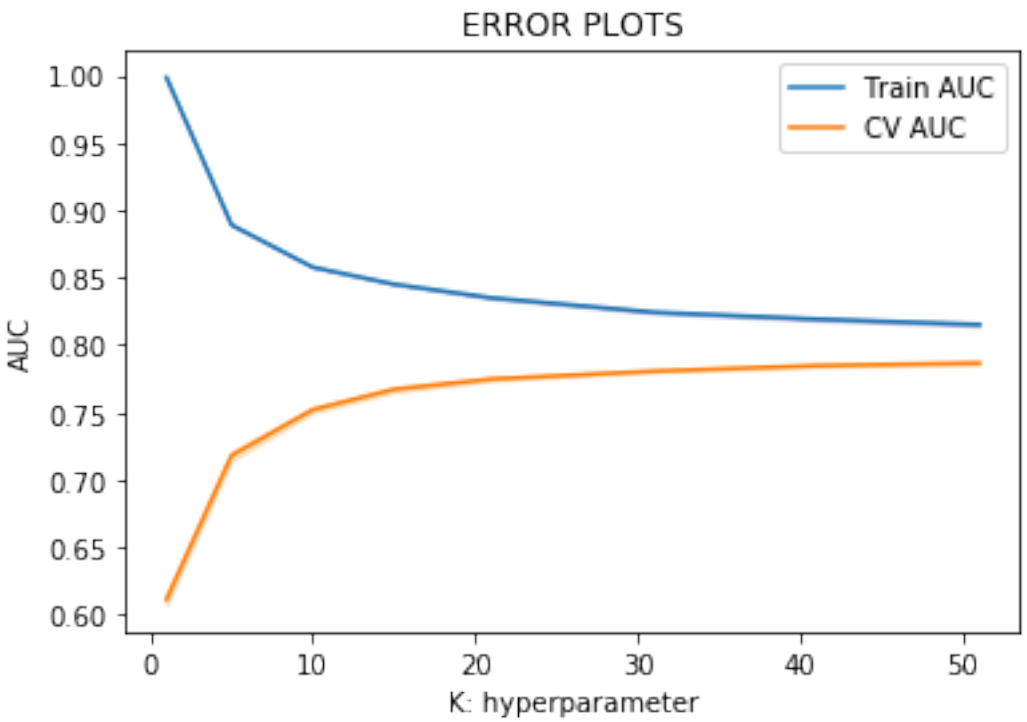



```
==========================================================================================
Train confusion matrix
[[ 1471  5720]
 [  939 36760]]
Test confusion matrix
[[  927  4175]
 [  778 27120]]
```

6.2.2 [5.2.2] Applying KNN kd-tree on TFIDF, SET 6

```
In [294]: # Please write all the code with proper documentation
          svd = TruncatedSVD(n_components=100)
          Xtrain_tfidf = svd.fit_transform(X_train_bow)
```

```
Xtest_tfidf = svd.transform(X_test_bow)

# Please write all the code with proper documentation
# Importing libraries
from sklearn.model_selection import GridSearchCV


neigh = KNeighborsClassifier(algorithm='kd_tree')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(Xtrain_tfidf, y_train)

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

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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

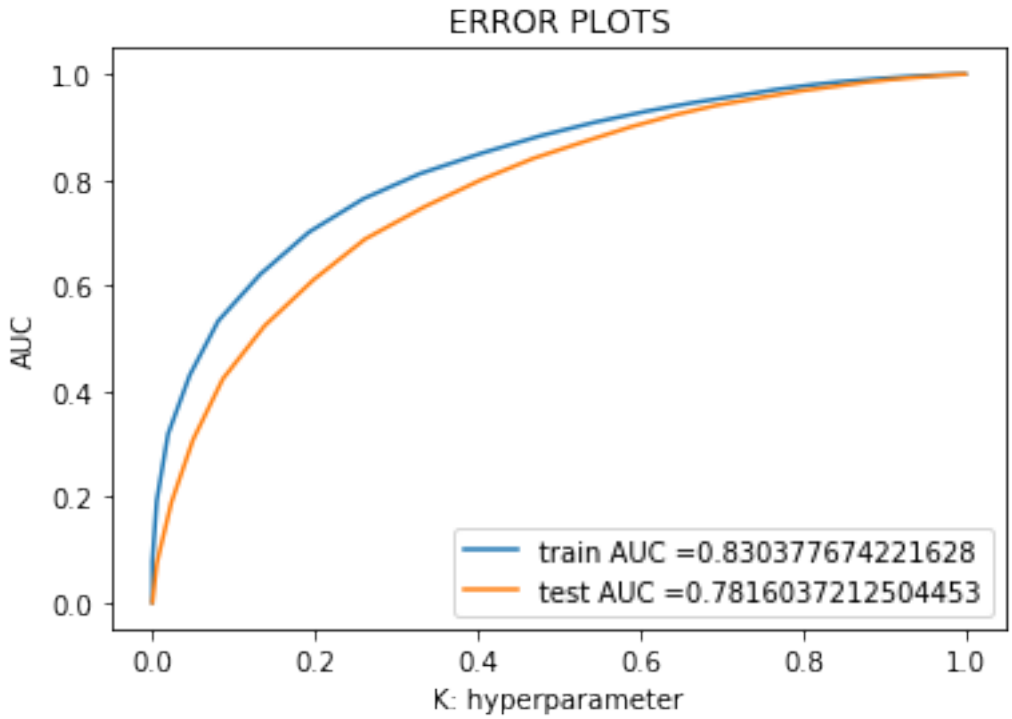


In [295]:
```
best_k=31
```

In [296]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(Xtrain_tfidf, y_train)
```

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(Xtrain_tfi
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(Xtest_tfidf)[

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)

from sklearn.metrics import confusion_matrix
```

```
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(Xtrain_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(Xtest_tfidf)))


# Variables for table
tfidf_kdTree_K = best_k
tfidf_kdTree_train = train_auc
tfidf_kdTree_test = cv_auc
```

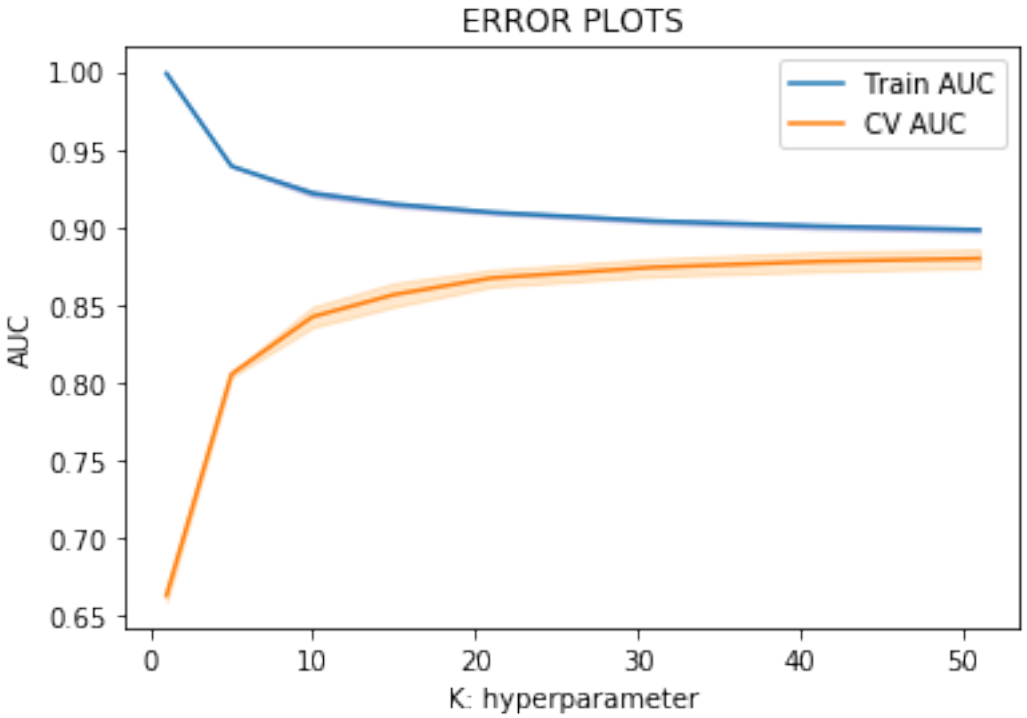



```
==========================================================================================
Train confusion matrix
[[ 1662  5529]
 [ 1082 36617]]
Test confusion matrix
[[ 1047  4055]
 [  898 27000]]
```

6.2.3 [5.2.3] Applying KNN kd-tree on AVG W2V, SET 7

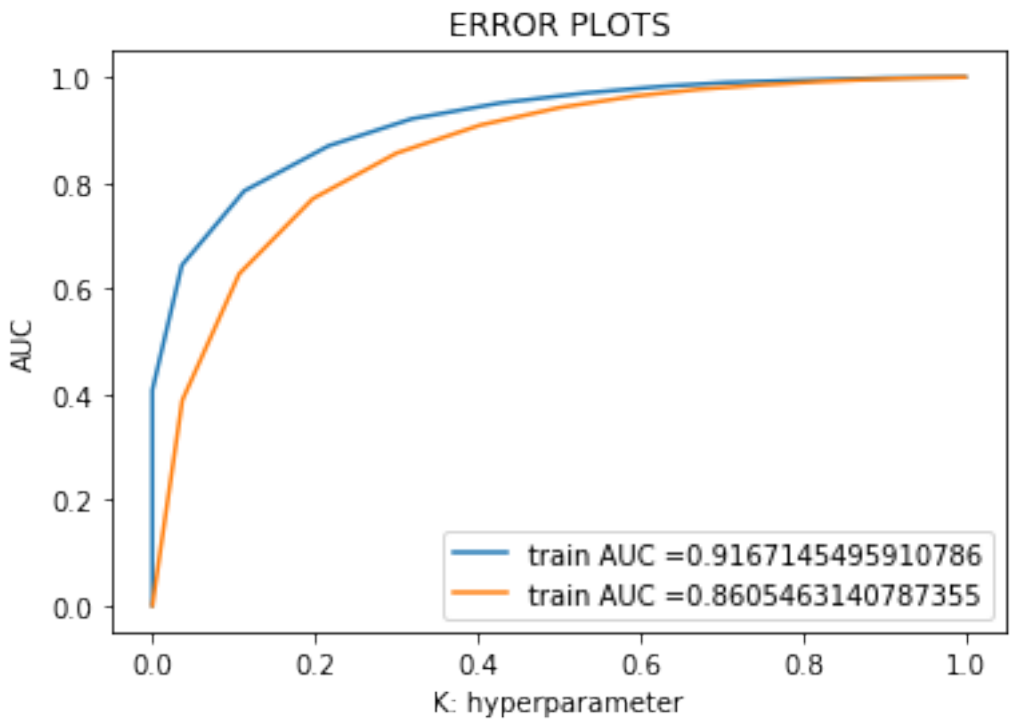
```
In [297]: from sklearn.model_selection import GridSearchCV
```

```
neigh = KNeighborsClassifier(algorithm='kd_tree')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(train_vectors, y_train)

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

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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



In [298]:
```
best_k=15
```

In [299]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(train_vectors, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(train_vecto
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(test_vectors)

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()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(train_vectors)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(test_vectors)))



# Variables for table
Avg_Word2Vec_kdTree_K = best_k
Avg_Word2Vec_kdTree_train = train_auc
Avg_Word2Vec_kdTree_test = cv_auc
```

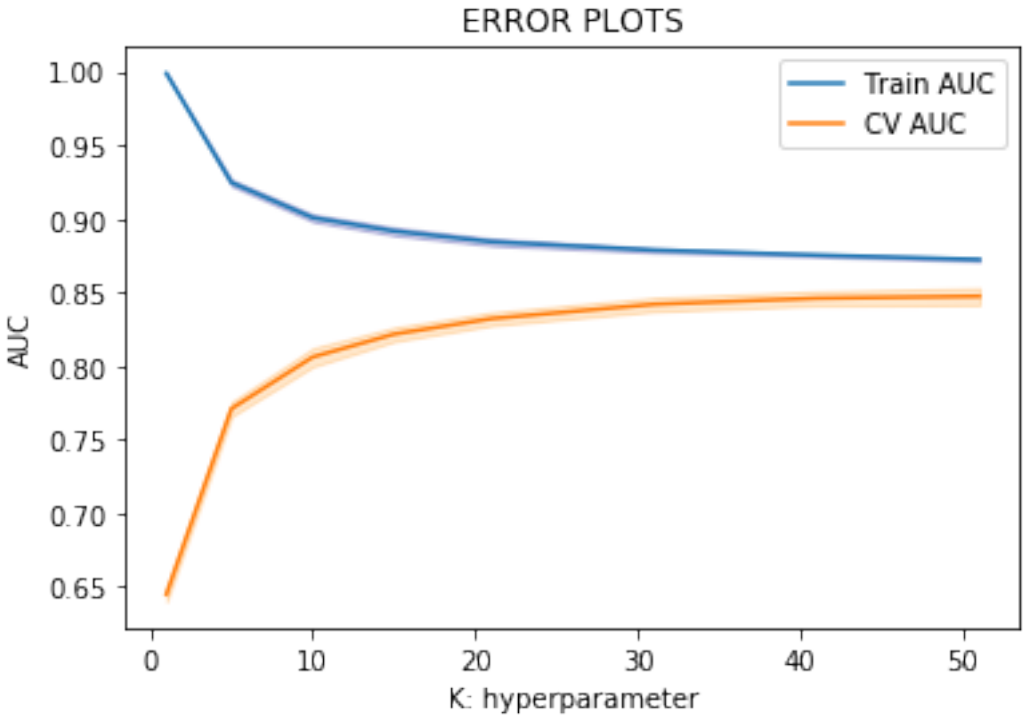



```
====================================================================================================
Train confusion matrix
[[ 2683  4508]
 [  660 37039]]
Test confusion matrix
[[ 1666  3436]
 [  635 27263]]
```

6.2.4 [5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 8

```
In [300]: neigh = KNeighborsClassifier(algorithm='kd_tree')
          parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
          clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
          clf.fit(tfidf_train_vectors, y_train)

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

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

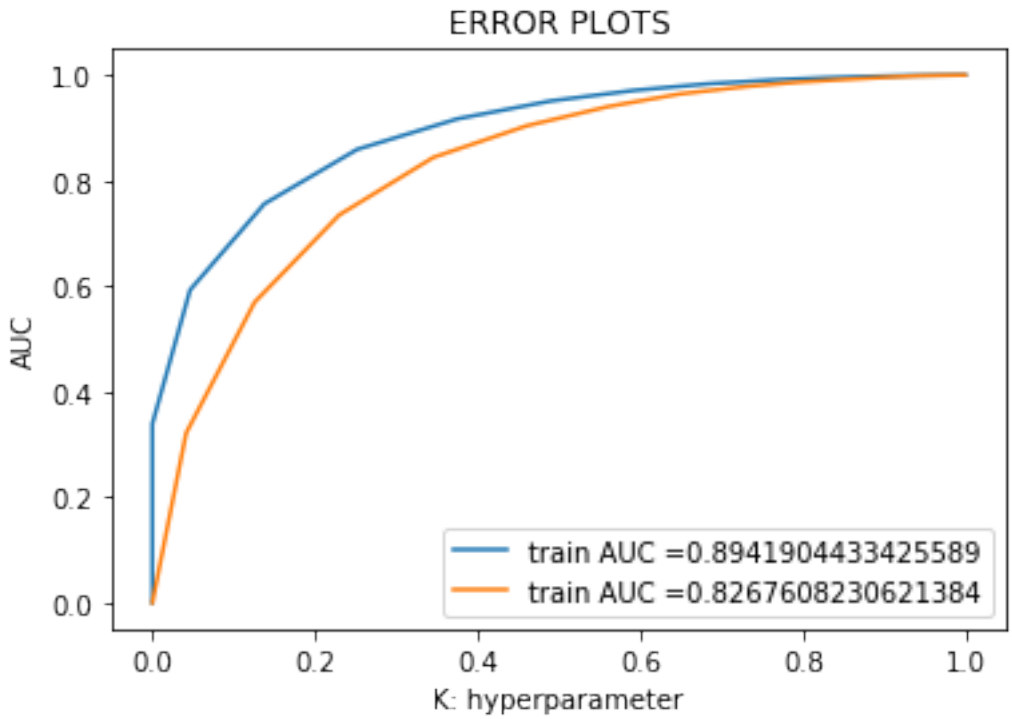
```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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



In [301]:
```
best_k=15
```

In [302]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(tfidf_train_vectors, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_trai
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_test_ve
```

```
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()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(tfidf_train_vectors)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(tfidf_test_vectors)))

# Variables for table
TFIDF_Word2Vec_kdTree_K = best_k
TFIDF_Word2Vec_kdTree_train = train_auc
TFIDF_Word2Vec_kdTree_test = cv_auc
```




```
====================================================================================================
Train confusion matrix
```

```
[[ 2271  4920]
 [  623 37076]]
Test confusion matrix
[[ 1363  3739]
 [  595 27303]]
```

7 [6] Conclusions

```
In [303]: from prettytable import PrettyTable

          # Initializing prettytable

          ptable = PrettyTable()

          names = ["brute for BoW", "kdTree for BoW", "brute for TFIDF", "kdTree' for TFIDF", "
                  "brute for TFIDF-Word2Vec", "kdTree' for TFIDF-Word2Vec"]

          k = ([bow_brute_K , bow_kdTree_K , tfidf_brute_K , tfidf_kdTree_K , Avg_Word2Vec_brut
                      TFIDF_Word2Vec_brute_K , TFIDF_Word2Vec_kdTree_K ])

          train = ([bow_brute_train, bow_kdTree_train, tfidf_brute_train, tfidf_kdTree_train, \
                       Avg_Word2Vec_brute_train, Avg_Word2Vec_kdTree_train, TFIDF_Word2Vec_brut
                       TFIDF_Word2Vec_kdTree_train])

          test = ([bow_brute_test, bow_kdTree_test , tfidf_brute_test , tfidf_kdTree_test , \
                      Avg_word2Vec_brute_test, Avg_Word2Vec_kdTree_test , TFIDF_word2Vec_brute_
                      TFIDF_Word2Vec_kdTree_test ])

          ptable.add_column("MODEL",names)
          ptable.add_column("K: hyperparameter",k)
          ptable.add_column("Train",train)
          ptable.add_column("Test",test)

          # Printing the Table
          print(ptable)
```

```
+--------------------------+--------------------+-----------------------------------------
|          MODEL           | K: hyperparameter |                                Train
+--------------------------+--------------------+-----------------------------------------
|       brute for BoW      |         10         | [0.99852681 0.88257485 0.84404492 0.8254343
|                          |                    |                         0.7926365  0.7871711 ]
|       kdTree for BoW     |         41         | [0.99852681 0.8895456  0.85743506 0.8439095
|                          |                    |                         0.81748903 0.81296166]
|     brute for TFIDF      |         51         | [0.99852681 0.86200396 0.79318869 0.7586135
|                          |                    |                         0.70465625 0.7031218 ]
|    kdTree' for TFIDF     |         31         | [0.99852681 0.8892246  0.85784474 0.8452606
```

		0.81935741 0.81532039]
brute for Avg-Word2Vec	15	[0.99874384 0.93933685 0.9215899 0.9148080
		0.90049257 0.8981785]
kdTree for Avg-Word2Vec	15	[0.99874844 0.93911958 0.9216491 0.9146285
		0.9004995 0.8981898]
brute for TFIDF-Word2Vec	14	[0.99873058 0.92486981 0.90084038 0.8918433
		0.87514275 0.87222066]
kdTree' for TFIDF-Word2Vec	15	[0.99874844 0.92462785 0.90075614 0.8916502
		0.87534625 0.87233934]