03 Amazon Fine Food Reviews Analysis_KNN

April 23, 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 [2]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

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

from tqdm import tqdm
import os
```

In [6]:
```
# using SQLite Table to read data.
con = sqlite3.connect(r'D:\Sayantan\Personal\MLnAI\Assignments\K-NN\database.sqlite\da

# 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 point
# 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 5
```

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
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 (50000, 10)
```

Out[6]:

```
   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 [7]:

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

In [8]:

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

```
(80668, 7)
```

Out[8]:

```
                UserId   ProductId             ProfileName        Time  Score  \
0  #oc-R115TNMSPFT9I7  B005ZBZLT4                 Breyton  1331510400      2
```

```
1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4        Kim Cieszykowski  1348531200      1
3  #oc-R11O5J5ZVQE25C  B005HG9ESG           Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBEV0   Christopher P. Presta  1348617600      1

                                                Text  COUNT(*)
0  Overall its just OK when considering the price...         2
1  My wife has recurring extreme muscle spasms, u...         3
2  This coffee is horrible and unfortunately not ...         2
3  This will be the bottle that you grab from the...         3
4  I didnt like this coffee. Instead of telling y...         2
```

In [9]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[9]:
```
              UserId   ProductId                     ProfileName        Time  \
80638  AZY10LLTJ71NX  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

       Score                                               Text  COUNT(*)
80638      5  I bought this 6 pack because for the price tha...         5
```

In [10]:
```
display['COUNT(*)'].sum()
```

Out[10]:
```
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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[11]:
```
       Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

   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 [12]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fal
```

In [39]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[39]: (46072, 10)

In [40]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[40]: 92.144

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

         display.head()
```

```
Out[15]:       Id   ProductId          UserId              ProfileName  \
         0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
         1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

            HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
         0                     3                       1      5  1224892800
         1                     3                       2      4  1212883200

                                                Summary  \
         0            Bought This for My Son at College
         1  Pure cocoa taste with crunchy almonds inside

                                                         Text
         0  My son loves spaghetti so I didn't hesitate or...
         1  It was almost a 'love at first bite' - the per...
```

```
In [41]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [42]: #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()
```

```
(46071, 10)
```

```
Out[42]: 1    38479
         0     7592
         Name: Score, dtype: int64
```

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 [18]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
==================================================
```

```
In [19]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
         sent_150 = re.sub(r"http\S+", "", sent_1500)
         sent_4900 = re.sub(r"http\S+", "", sent_4900)

         print(sent_0)
```

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

```
In [20]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
         from bs4 import BeautifulSoup
```

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
```

In [21]:
```
# 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 [22]:
```
sent_1500 = decontracted(sent_1500)
```

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

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
```

```
In [23]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

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

```
In [24]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually protein powders are hig
```

```
In [25]: # 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', 'ourselve
                     "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 [43]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
```

```
        sentance = BeautifulSoup(sentance, 'lxml').get_text()
        sentance = decontracted(sentance)
        sentance = re.sub("\S*\d\S*", "", sentance).strip()
        sentance = re.sub('[^A-Za-z]+', ' ', sentance)
        # https://gist.github.com/sebleier/554280
        sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
        preprocessed_reviews.append(sentance.strip())
```

```
100%|| 46071/46071 [00:23<00:00, 1948.79it/s]
```

In [27]:
```
preprocessed_reviews[1500]
```

Out[27]:
```
'great flavor low calories high nutrients high protein usually protein powders high p
```

[3.2] Preprocessing Review Summary

In [37]:
```
## Similartly you can do preprocessing for review summary also.
#Deduplication of entries
final1=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Summary"}, 
print(final1.shape)

#Checking to see how much % of data still remains
print((final1['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100)

#value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not pr
#hence these  rows too are removed from calcualtions
final1=final1[final1.HelpfulnessNumerator<=final1.HelpfulnessDenominator]

#Before starting the next phase of preprocessing lets see the number of entries left
print(final1.shape)

#How many positive and negative reviews are present in our dataset?
print(final1['Score'].value_counts())

from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final1['Summary'].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
    preprocessed_summary.append(sentance.strip())
```

```
(46035, 10)
92.07
```

```
(46034, 10)
1    38455
0     7579
Name: Score, dtype: int64


100%|| 46034/46034 [00:22<00:00, 2016.48it/s]
```

In [38]:
```
preprocessed_summary[1000]
```

Out[38]:
```
'demand local grocer carry product'
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [46]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
#print("some feature names ", count_vect.get_feature_names()[:])
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaa
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 39364)
the number of unique words  39364
```

5.2 [4.2] Bi-Grams and n-Grams.

In [47]:
```
#bi-gram, tri-gram and n-gram

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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 5000)
the number of unique words including both unigrams and bigrams  5000
```

5.3 [4.3] TF-IDF

```
In [49]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
         tf_idf_vect.fit(preprocessed_reviews)
         print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
         print('='*50)

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able chew',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (46071, 27311)
the number of unique words including both unigrams and bigrams  27311
```

5.4 [4.4] Word2Vec

```
In [50]: # Train your own Word2Vec model using your own text corpus
         i=0
         list_of_sentance=[]
         for sentance in preprocessed_reviews:
             list_of_sentance.append(sentance.split())
```

```
In [51]: # Using Google News Word2Vectors

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


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

```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('fantastic', 0.8303105235099792), ('awesome', 0.8245354294776917), ('good', 0.82201284170150
==================================================
[('best', 0.7316393852233887), ('nastiest', 0.7310843467712402), ('greatest', 0.71936583518981
```

In [53]:
```
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  12798
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore', 'ha
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [54]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
```

```
            sent_vectors.append(sent_vec)
        print(len(sent_vectors))
        print(len(sent_vectors[0]))
```

```
100%|| 46071/46071 [02:43<00:00, 281.32it/s]


46071
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [39]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
         model = TfidfVectorizer()
         tf_idf_matrix = model.fit_transform(preprocessed_reviews)
         # we are converting a dictionary with word as a key, and the idf as a value
         dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [41]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

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

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

6 [5] Assignment 3: KNN

Apply Knn(brute force version) on these feature sets

```
    <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 ma
    <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.appliedaico
<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 t
    </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 f
<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.
<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 fo
    <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 Preprocessing on Data, SET 1

In [55]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

from gensim.models import Word2Vec
```

```
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [57]:
```
# using SQLite Table to read data.
con = sqlite3.connect(r'D:\Sayantan\Personal\MLnAI\Assignments\K-NN\database.sqlite\da
#copying code from above
# 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 poin
# 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
# Taking 50k data points as mentioned in the problem statement

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

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

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

In [58]:
```
#DeDuplication
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fal
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
```

In [59]:
```
#Removing entry where value of HelpfulnessNumerator is greater than HelpfulnessDenomi
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [60]:
```
# 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', 'ourselve
```

```
"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 [72]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
             preprocessed_reviews.append(sentance.strip())
         print(preprocessed_reviews[:3])
```

```
100%|| 46071/46071 [00:25<00:00, 1777.65it/s]


['dogs loves chicken product china wont buying anymore hard find chicken products made usa one
```

6.1.2 [5.1.2] Applying KNN brute force on BOW, SET 1

```
In [146]: Y = final['Score'].values
          X = np.asarray(preprocessed_reviews)
          print(X[:3])
          print(Y[:3])
          print(type(X))
          print(type(Y))
```

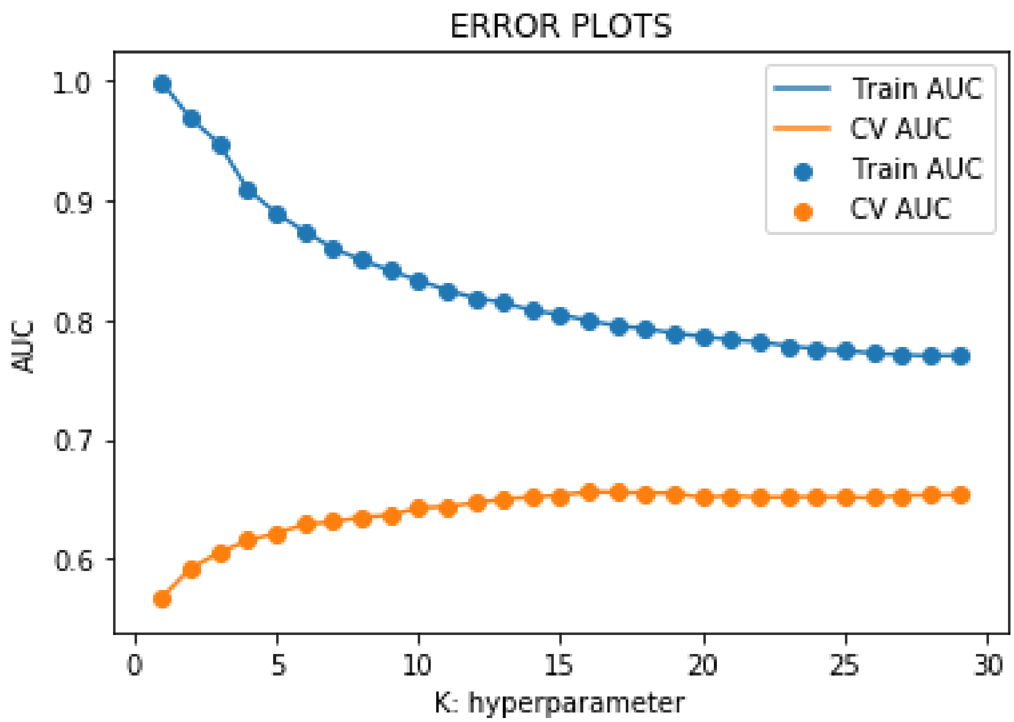
```
['dogs loves chicken product china wont buying anymore hard find chicken products made usa one
 'dogs love saw pet store tag attached regarding made china satisfied safe'
 'product available victor traps unreal course total fly genocide pretty stinky right nearby']
[0 1 1]
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

```
In [150]: #Splitting data in train, cv and test
          #error calc code taken from https://stackabuse.com/k-nearest-neighbors-algorithm-in-p
          from sklearn.model_selection import train_test_split
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import roc_auc_score

          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,shuffle=Fals
          X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,shu
          print(X_train.shape, y_train.shape)
          print(X_cv.shape, y_cv.shape)
          print(X_test.shape, y_test.shape)
          print("="*100)

          from sklearn.feature_extraction.text import CountVectorizer
          vectorizer = CountVectorizer()
          #applying the method fit_transform() on you train data, and apply the method transfo
          X_train_bow = vectorizer.fit_transform(X_train)
          X_cv_bow = vectorizer.transform(X_cv)
          X_test_bow = vectorizer.transform(X_test)

          train_auc = []
          cv_auc = []

          for i in range(1,30):
              neigh = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
              neigh.fit(X_train_bow, y_train)
              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(range(1,30), train_auc, label='Train AUC')
          #plt.plot(range(1,30), cv_auc, label='CV AUC')
          #plt.legend()
          #plt.xlabel("K: hyperparameter")
          #plt.ylabel("AUC")
          #plt.title("ERROR PLOTS")
          #plt.show()
```

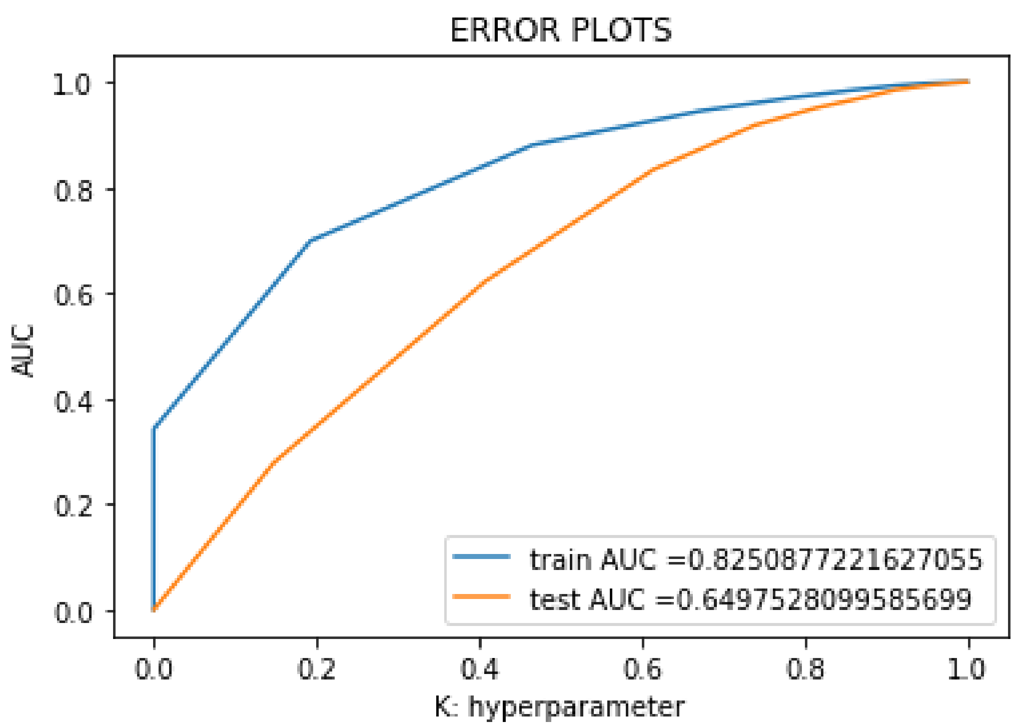
```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
================================================================================================
```

```
In [151]: plt.plot(range(1,30), train_auc, label='Train AUC')
          plt.scatter(range(1,30), train_auc, label='Train AUC')
          plt.plot(range(1,30), cv_auc, label='CV AUC')
          plt.scatter(range(1,30), cv_auc, label='CV AUC')
          plt.legend()
          plt.xlabel("K: hyperparameter")
          plt.ylabel("AUC")
          plt.title("ERROR PLOTS")
          plt.show()
```




```
In [98]: #From the above plot we can see that after k=16 AUC is almost stabalizes. So taking b
         best_k = 16

In [152]: from sklearn.metrics import roc_curve, auc


          neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
          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_bow
          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)))
```




```
===============================================================================================
Train confusion matrix
[[  477  2891]
 [  268 17044]]
Test confusion matrix
[[  334  2326]
 [  358 12186]]
```

Here from confusion Matrix from Test data TN = 334, TP=12186, Total N = (334+358) and Total P = 12186+2326. So, TPR = TP/P = 0.8397 TNR = 334/(334+358) = 0.482

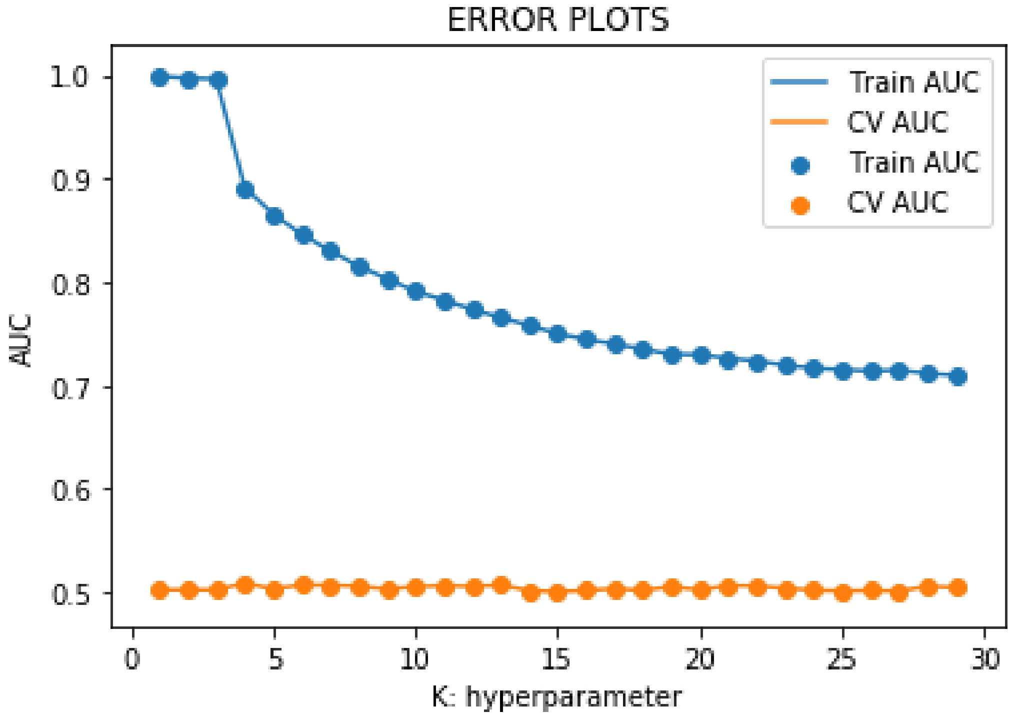
TPR is good but TNR is just around 50%. Model is not that well performing for -ve cases.

6.1.3 [5.1.2] Applying KNN brute force on TFIDF, SET 2

In [153]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
#Apply tf-idf on splitted data sets
X_train_bow = tf_idf_vect.fit_transform(X_train)
X_cv_bow = tf_idf_vect.transform(X_cv)
X_test_bow = tf_idf_vect.transform(X_test)
```

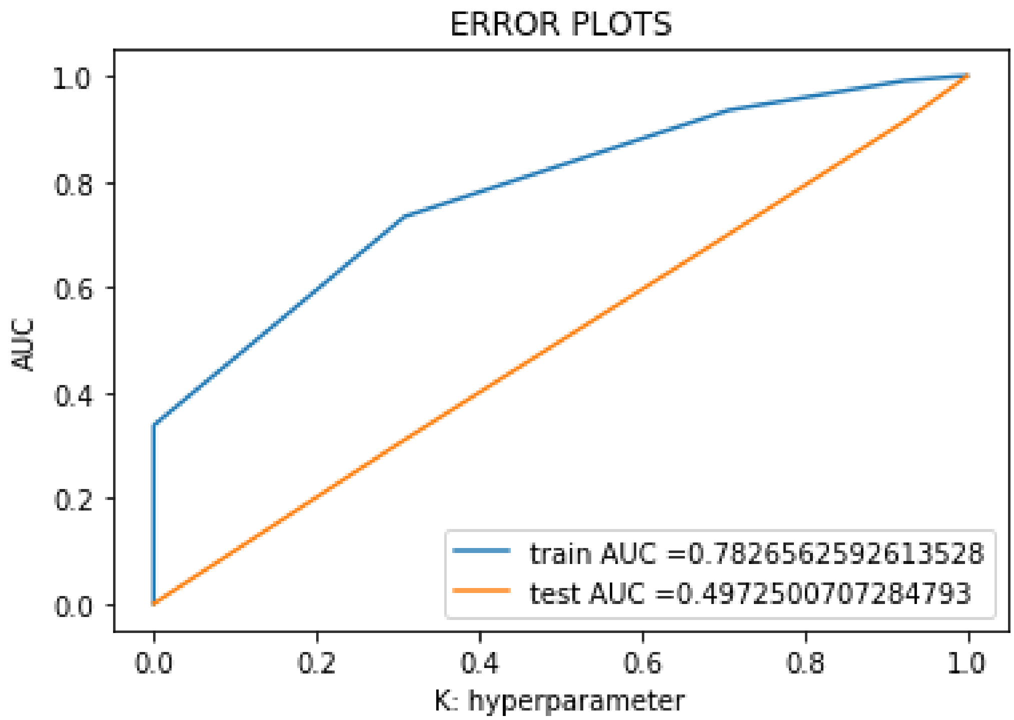
In [155]:
```
train_auc = []
cv_auc = []

for i in range(1,30):
    neigh = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    neigh.fit(X_train_bow, y_train)
    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))
```

In [156]:
```
plt.plot(range(1,30), train_auc, label='Train AUC')
plt.scatter(range(1,30), train_auc, label='Train AUC')
plt.plot(range(1,30), cv_auc, label='CV AUC')
plt.scatter(range(1,30), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [121]: #From the above plot we can see that AUC is almost consistent throughout, though at 
          best_k = 29

In [157]: from sklearn.metrics import roc_curve, auc


          neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
          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_bow
          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)))
```




```
====================================================================================================
Train confusion matrix
[[    5  3363]
 [    0 17312]]
Test confusion matrix
[[    0  2660]
 [    0 12544]]
```

Conclusion: AUC for test data is just 0.5. From confusion matrix there is no -ve clasification. TPR = 12544/(12544+2660)= 0.825. Rest of the parameter is zero. From the confusion matirx none of the -ve cases were identified. So this model also not that acceptable.

6.1.4 [5.1.3] Applying KNN brute force on AVG W2V, SET 3

```
In [131]: i=0
          list_of_sentance_train=[]
          for sentance in X_train:
              list_of_sentance_train.append(sentance.split())
```

```
In [132]: from tqdm import tqdm
          import numpy as np
          from gensim.models import Word2Vec
          from gensim.models import KeyedVectors

In [133]: # 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)

In [134]: w2v_words = list(w2v_model.wv.vocab)

In [135]: # average Word2Vec
          # compute average word2vec for each review.
          sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this li
          for sent in tqdm(list_of_sentance_train): # for each review/sentence
              sent_vec = np.zeros(50)
              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)

100%|| 22574/22574 [00:57<00:00, 392.46it/s]


(22574, 50)


In [136]: #converting cv data
          i=0
          list_of_sentance_cv=[]
          for sentance in X_cv:
              list_of_sentance_cv.append(sentance.split())

          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)
```

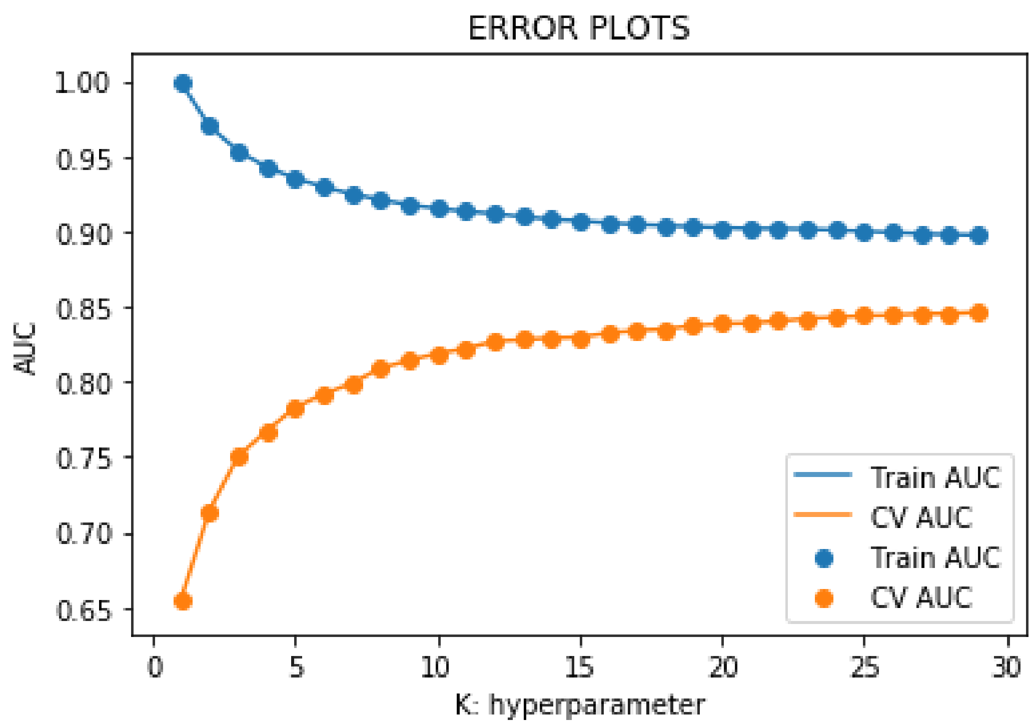
```
100%|| 9675/9675 [00:27<00:00, 401.92it/s]


(9675, 50)
```

```
In [137]: #Converting for test data
          i=0
          list_of_sentance_test=[]
          for sentance in X_test:
              list_of_sentance_test.append(sentance.split())

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

```
100%|| 13822/13822 [00:41<00:00, 329.94it/s]


(13822, 50)
```

```
In [141]: 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 range(1,30):
              neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
```

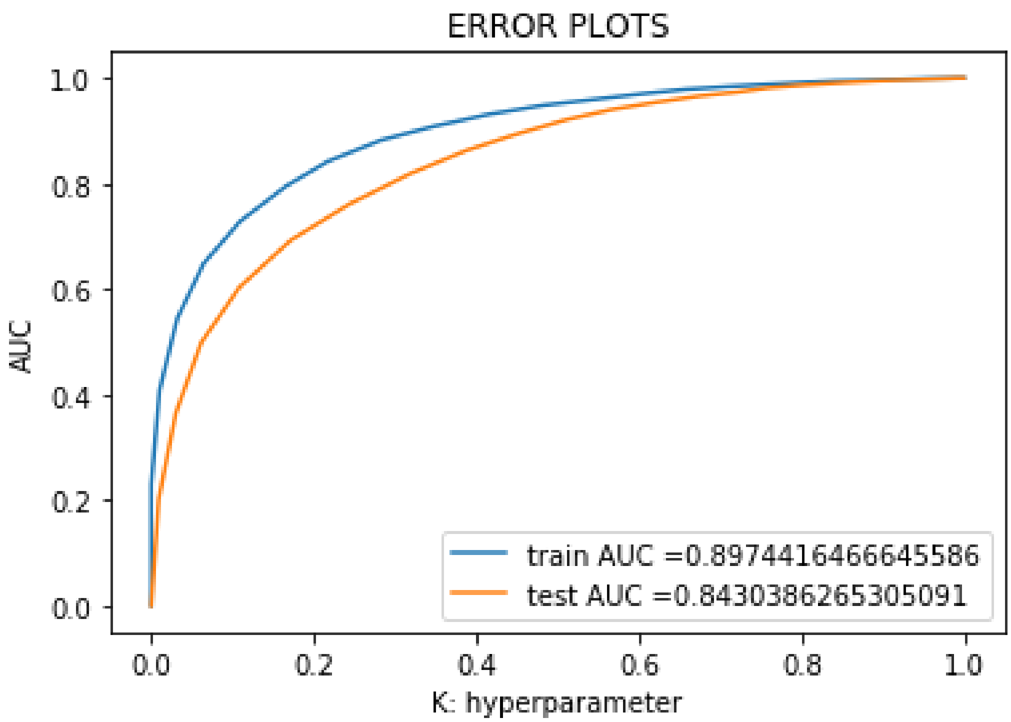
```
    neigh.fit(sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(sent_vectors_cv)[:,1]

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

plt.plot(range(1,30), train_auc, label='Train AUC')
plt.scatter(range(1,30), train_auc, label='Train AUC')
plt.plot(range(1,30), cv_auc, label='CV AUC')
plt.scatter(range(1,30), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



From the above plot, we can see that at k = 29 we are finding the best result.

```
In [143]: neigh = KNeighborsClassifier(n_neighbors=29,algorithm='brute')
          neigh.fit(sent_vectors_train, 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(sent_vecto
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vectors_

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




```
================================================================================
Train confusion matrix
[[ 1048  2580]
 [  304 18642]]
```

```
Test confusion matrix
[[  568  1807]
 [  208 11239]]
```

Conclusion : From the confusion matrix TPR = 0.86 and TNR = 0.732. So this is better model than the earlier two.

6.1.5 [5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

```
In [158]: i=0
          list_of_sentance_train=[]
          for sentance in X_train:
              list_of_sentance_train.append(sentance.split())
```

```
In [162]: model = TfidfVectorizer()
          tf_idf_matrix = model.fit_transform(preprocessed_reviews)
          # we are converting a dictionary with word as a key, and the idf as a value
          dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

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

          tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in
          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/review
              for word in sent: # for each word in a review/sentence
                  if word in w2v_words and word in tfidf_feat:
                      vec = w2v_model.wv[word]
          #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                      # to reduce the computation we are
                      # dictionary[word] = idf value of word in whole courpus
                      # sent.count(word) = tf valeus of word in this review
                      tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                      sent_vec += (vec * tf_idf)
                      weight_sum += tf_idf
              if weight_sum != 0:
                  sent_vec /= weight_sum
              tfidf_sent_vectors_train.append(sent_vec)
              row += 1
```

```
100%|| 20680/20680 [17:41<00:00, 19.49it/s]
```

```
In [163]: i=0
          list_of_sentance_cv=[]
```

```
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())

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

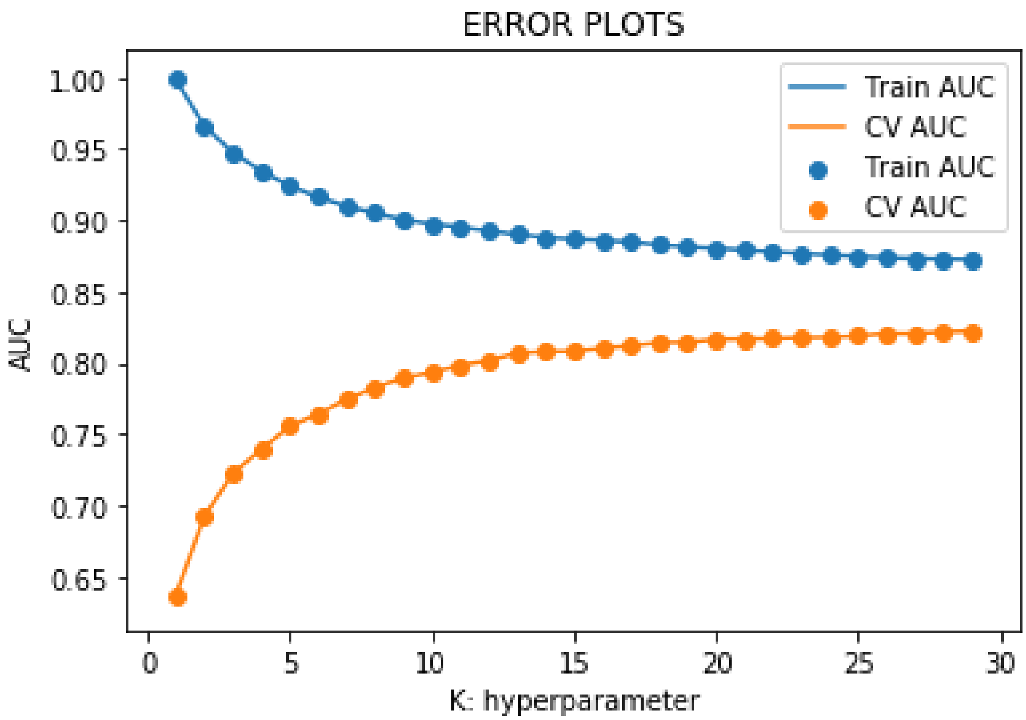
100%|| 10187/10187 [08:33<00:00, 20.30it/s]

In [164]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

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

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

```
100%|| 15204/15204 [13:56<00:00, 18.19it/s]
```

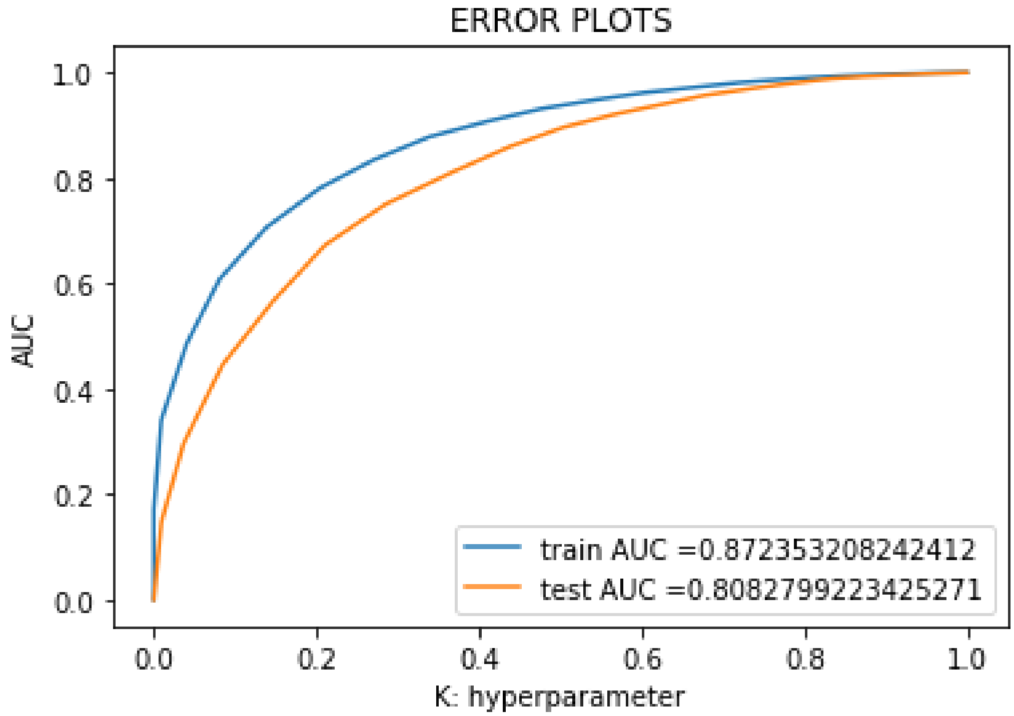
In [166]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []

for i in range(1,30):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(tfidf_sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
    # 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(range(1,30), train_auc, label='Train AUC')
plt.scatter(range(1,30), train_auc, label='Train AUC')
plt.plot(range(1,30), cv_auc, label='CV AUC')
plt.scatter(range(1,30), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [167]: #from the above plot best_k = 29
          best_k=29
```

```
In [168]: neigh = KNeighborsClassifier(n_neighbors=29,algorithm='brute')
          neigh.fit(tfidf_sent_vectors_train, 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_sent
          test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_sent_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_sent_vectors_train)))
```

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




```
====================================================================================================
Train confusion matrix
[[  785  2583]
 [  241 17071]]
Test confusion matrix
[[  529  2131]
 [  216 12328]]
```

Conclusion : This model is having TPR = 0.853 and TNR = 0.71. This model is also quite acceptable.

6.2 [5.2] Applying KNN kd-tree

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

For K-NN with KD-tree we will be considering 20k data set so that our memory not get flooded

```
In [175]: preprocessed_data = preprocessed_reviews[:20000]
          print(preprocessed_data[:3])
          print(len(preprocessed_data))
```

```
['dogs loves chicken product china wont buying anymore hard find chicken products made usa one
20000
```

```
In [180]: Y = final['Score'].values
          print(Y.shape)
          print(type(Y))
          Y = Y[:20000]
          print(Y.shape)
          print(type(Y))
          X = np.asarray(preprocessed_data)
```

```
(46071,)
<class 'numpy.ndarray'>
(20000,)
<class 'numpy.ndarray'>
```

```
In [187]: from sklearn.model_selection import train_test_split
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import roc_auc_score

          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,shuffle=Fals
          X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,shu
          print(X_train.shape, y_train.shape)
          print(X_cv.shape, y_cv.shape)
          print(X_test.shape, y_test.shape)
          print("="*100)

          from sklearn.feature_extraction.text import CountVectorizer
          vectorizer = CountVectorizer(min_df=10, max_features=500)

          X_train_bow = vectorizer.fit_transform(X_train)
          X_cv_bow = vectorizer.transform(X_cv)
          X_test_bow = vectorizer.transform(X_test)

          print(type(X_train_bow))
```

```
(8978,) (8978,)
(4422,) (4422,)
(6600,) (6600,)
==========================================================================================
<class 'scipy.sparse.csr.csr_matrix'>
```

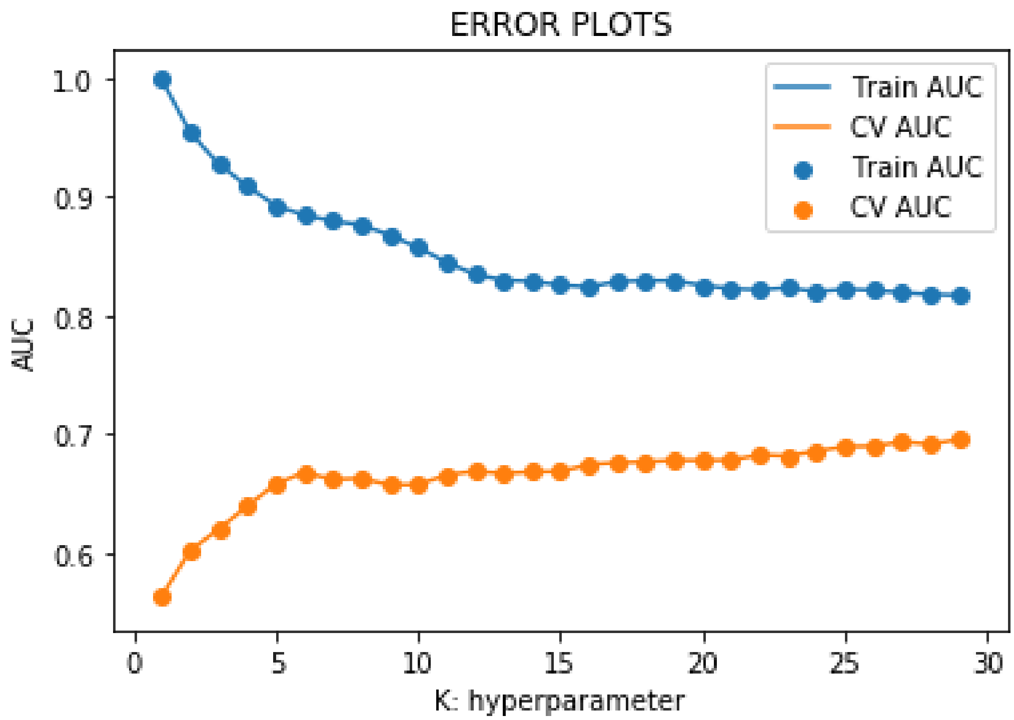
```
In [188]: #converting sparse to dense matrix as K-NN with algo KD-tree in skitlearn accept den
          X_train_bow_dense = X_train_bow.toarray()
          X_cv_bow_dense = X_cv_bow.toarray()
          X_test_bow_dense = X_test_bow.toarray()
```

```
print(type(X_train_bow_dense))
print(X_train_bow_dense.shape)
```

```
<class 'numpy.ndarray'>
(8978, 500)
```

```
In [189]: train_auc = []
          cv_auc = []

          for i in range(1,30):
              neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
              neigh.fit(X_train_bow_dense, y_train)
              y_train_pred =  neigh.predict_proba(X_train_bow_dense)[:,1]
              y_cv_pred =  neigh.predict_proba(X_cv_bow_dense)[:,1]

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

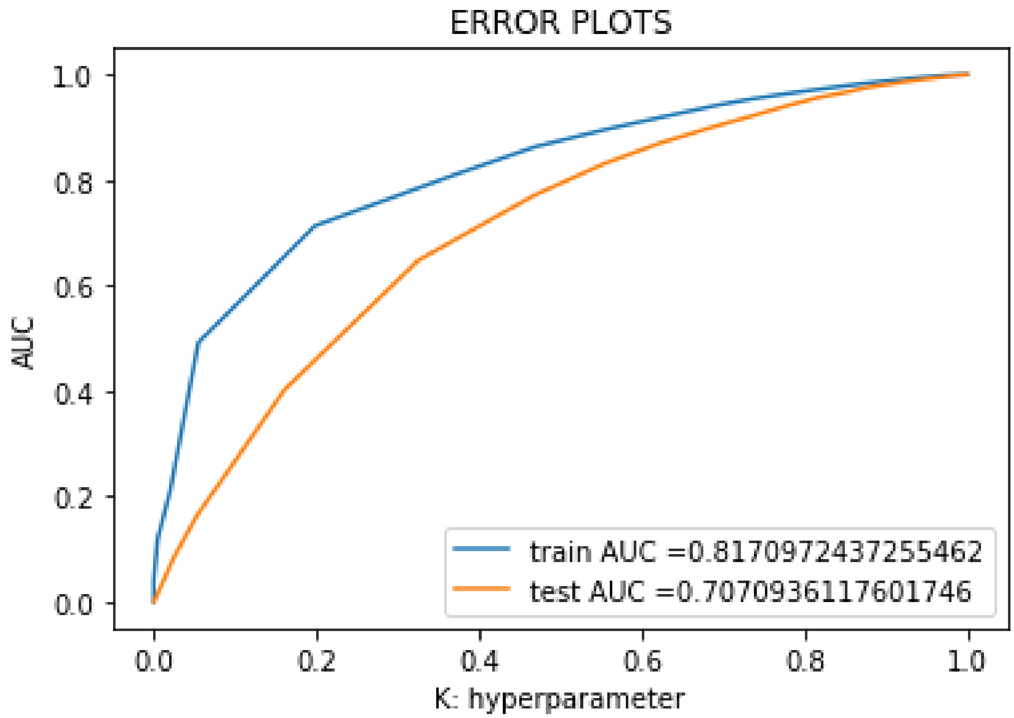
```
In [190]: plt.plot(range(1,30), train_auc, label='Train AUC')
          plt.scatter(range(1,30), train_auc, label='Train AUC')
          plt.plot(range(1,30), cv_auc, label='CV AUC')
          plt.scatter(range(1,30), cv_auc, label='CV AUC')
          plt.legend()
          plt.xlabel("K: hyperparameter")
          plt.ylabel("AUC")
          plt.title("ERROR PLOTS")
          plt.show()
```



In [191]:
```
#From the AUC plot above we can see that k=29 we have best AUC
best_k = 29
```

In [192]:
```
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
neigh.fit(X_train_bow_dense, 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_bow
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow_den

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




```
====================================================================================================
Train confusion matrix
[[ 119 1213]
 [  83 7563]]
Test confusion matrix
[[  95  968]
 [  93 5444]]
```

From the confusion matrix TPR = 0.849 and TNR = is almost 0.5. This model is somewhat ok.

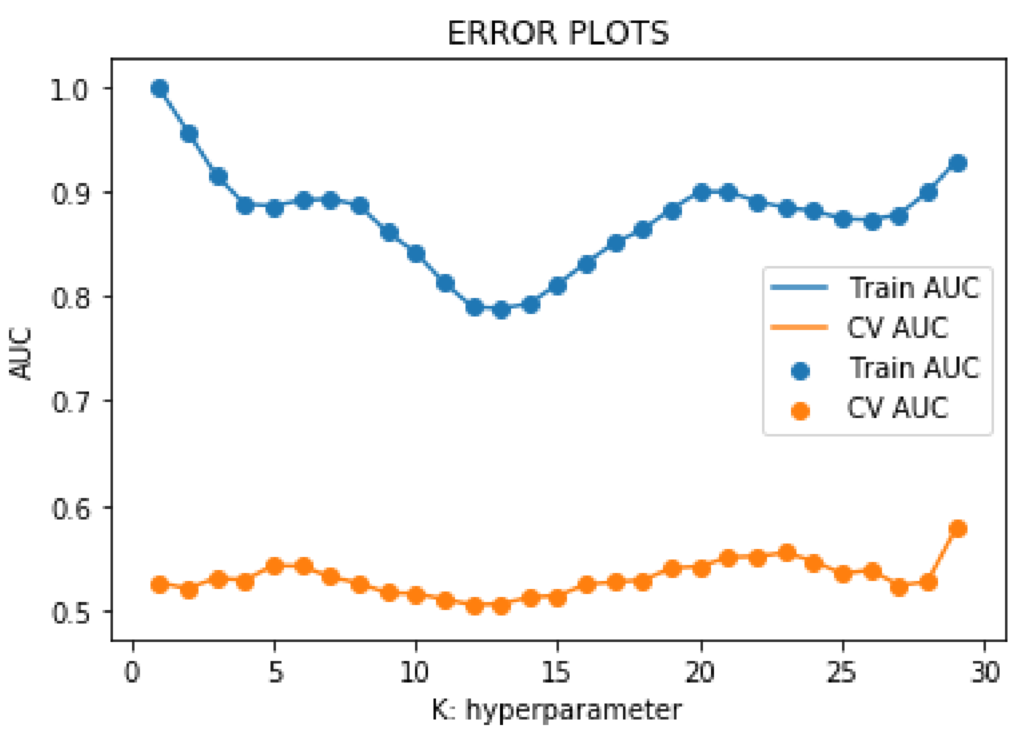
6.2.2 [5.2.2] Applying KNN kd-tree on TFIDF, SET 6

```
In [193]: from sklearn.feature_extraction.text import TfidfVectorizer
          tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=500)
          #Apply tf-idf on splitted data sets
          X_train_tfidf = tf_idf_vect.fit_transform(X_train)
          X_cv_tfidf = tf_idf_vect.transform(X_cv)
```

```
X_test_tfidf = tf_idf_vect.transform(X_test)

X_train_tfidf_dense = X_train_tfidf.toarray()
X_cv_tfidf_dense = X_cv_tfidf.toarray()
X_test_tfidf_dense = X_test_tfidf.toarray()
```

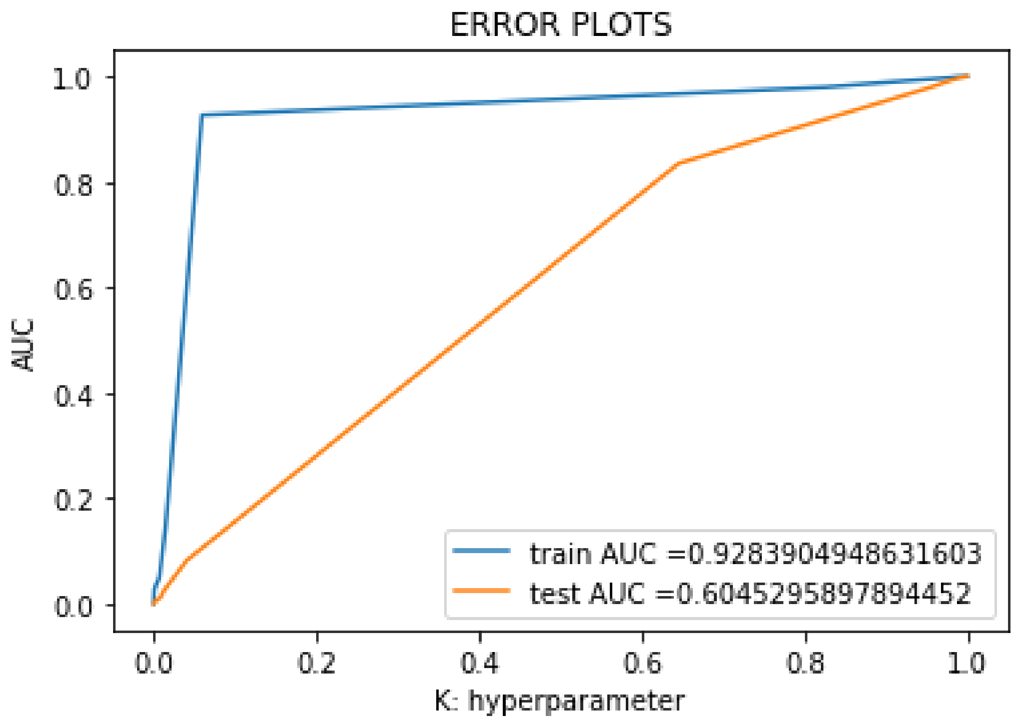
In [194]:
```
train_auc = []
cv_auc = []

for i in range(1,30):
    neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    neigh.fit(X_train_tfidf_dense, y_train)
    y_train_pred =  neigh.predict_proba(X_train_tfidf_dense)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tfidf_dense)[:,1]

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

In [195]:
```
plt.plot(range(1,30), train_auc, label='Train AUC')
plt.scatter(range(1,30), train_auc, label='Train AUC')
plt.plot(range(1,30), cv_auc, label='CV AUC')
plt.scatter(range(1,30), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [ ]:
```
#best_k = 29 from AUC graph
best_k = 29
```

In [196]:
```
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
neigh.fit(X_train_tfidf_dense, 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_dense)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf_dense)))
```




```
==========================================================================================
Train confusion matrix
[[    5 1327]
 [    2 7644]]
Test confusion matrix
[[    0 1063]
 [    0 5537]]
```

Conclusion: TPR is acceptable but with this model no -ve classification is done rightly. So this model is not acceptable.

6.2.3 [5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

```
In [197]: from tqdm import tqdm
          import numpy as np
          from gensim.models import Word2Vec
          from gensim.models import KeyedVectors

          i=0
          list_of_sentance_train=[]
          for sentance in X_train:
              list_of_sentance_train.append(sentance.split())
```

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

w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this li
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50)
    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)
```

```
100%|| 8978/8978 [00:19<00:00, 451.14it/s]


(8978, 50)
```

```
In [198]: i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())

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)
```

```
100%|| 4422/4422 [00:09<00:00, 442.28it/s]


(4422, 50)
```

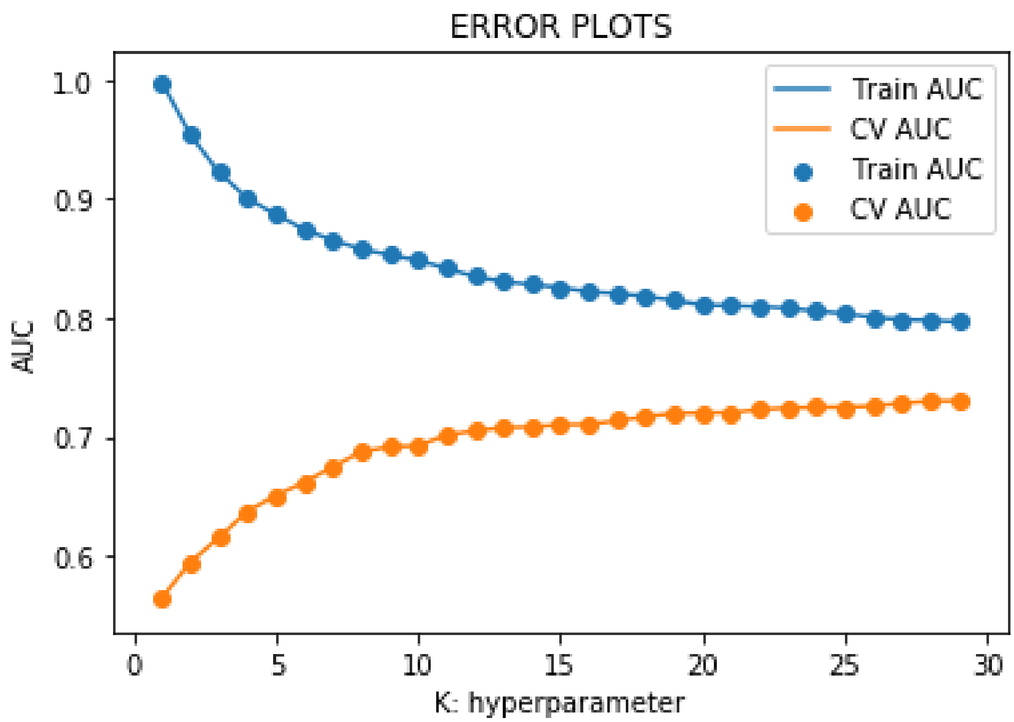
```
In [199]: #Converting for test data
          i=0
          list_of_sentance_test=[]
          for sentance in X_test:
              list_of_sentance_test.append(sentance.split())

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

```
100%|| 6600/6600 [00:14<00:00, 453.20it/s]
```

```
(6600, 50)
```

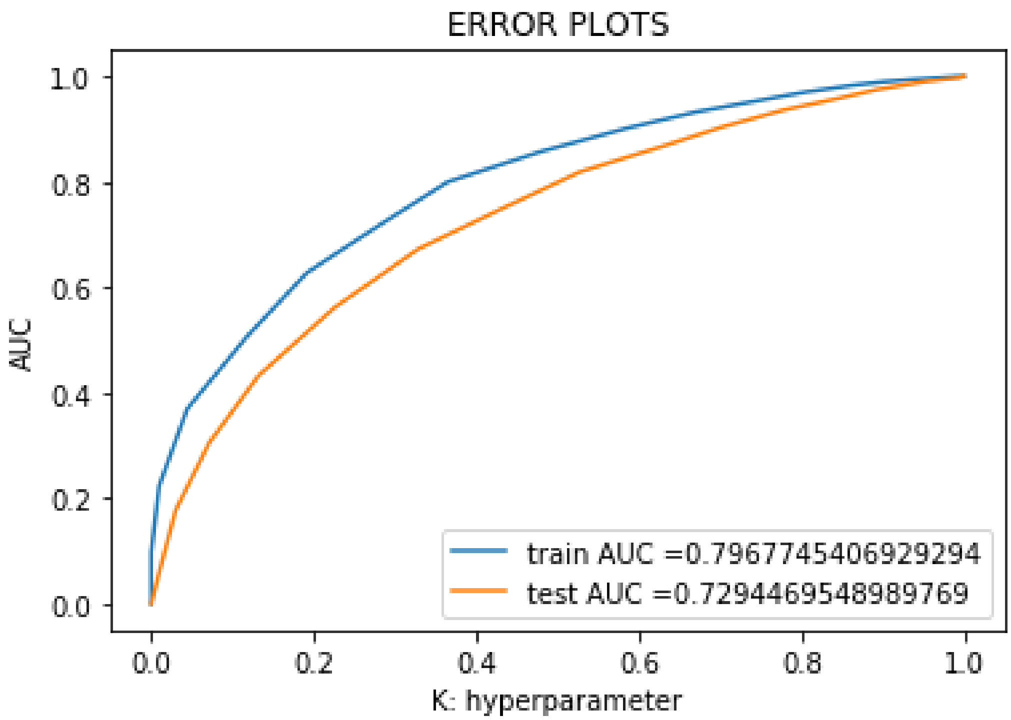
```
In [200]: from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import roc_auc_score
          import matplotlib.pyplot as plt

          train_auc = []
          cv_auc = []

          for i in range(1,30):
              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 probability estimat
              # 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(range(1,30), train_auc, label='Train AUC')
```

```
plt.scatter(range(1,30), train_auc, label='Train AUC')
plt.plot(range(1,30), cv_auc, label='CV AUC')
plt.scatter(range(1,30), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [201]:
```
#From the AUC plot above 29 is the best k
neigh = KNeighborsClassifier(n_neighbors=29,algorithm='kd_tree')
neigh.fit(sent_vectors_train, 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(sent_vecto
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vectors_

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




```
====================================================================================================
Train confusion matrix
[[  85 1247]
 [  44 7602]]
Test confusion matrix
[[  48 1015]
 [  51 5486]]
```

Conclusion: TPR = 0.84 and TNR = 0.52, this model is quite acceptable.

6.2.4 [5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

```
In [202]: i=0
          list_of_sentance_train=[]
```

```
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())

model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_data)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

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

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in
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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_train.append(sent_vec)
    row += 1
```

```
100%|| 8978/8978 [03:23<00:00, 44.04it/s]
```

In [203]:
```
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())

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

```
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors_cv.append(sent_vec)
        row += 1
```

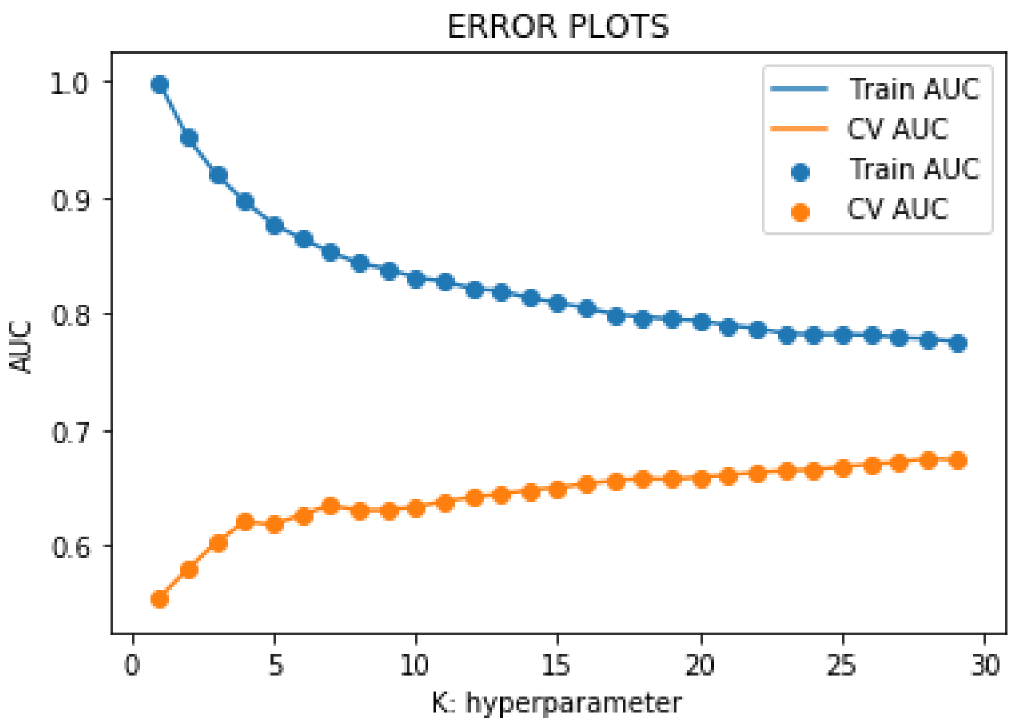
```
100%|| 4422/4422 [01:42<00:00, 43.17it/s]
```

In [204]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

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

```
100%|| 6600/6600 [03:50<00:00, 28.66it/s]
```

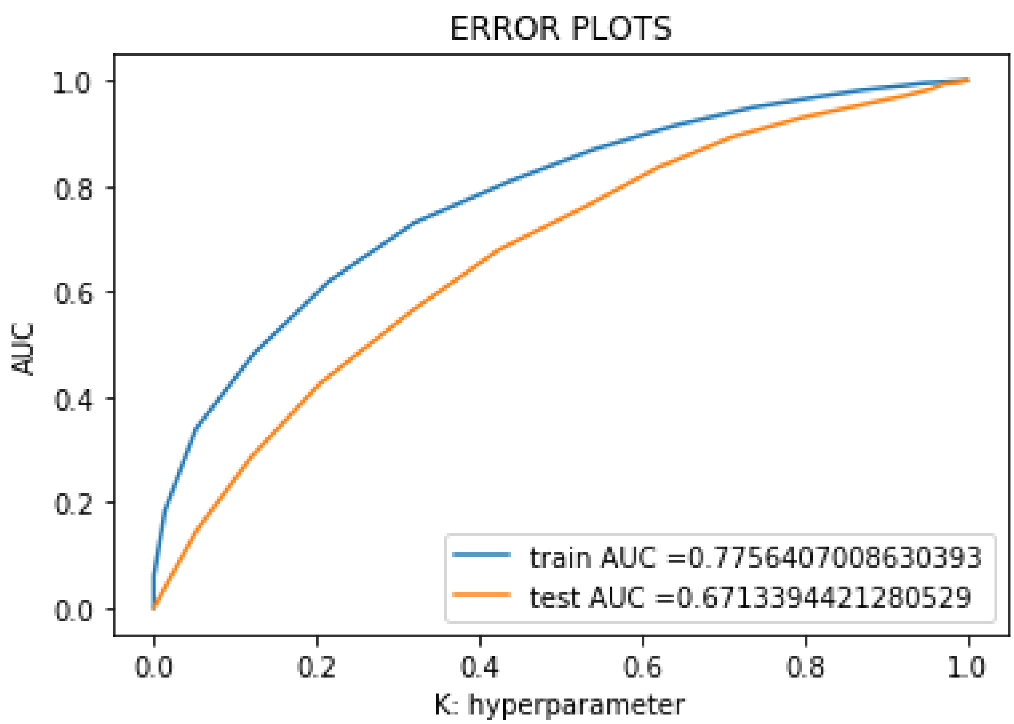
In [205]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
```

```
for i in range(1,30):
    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 probability estimat
    # 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(range(1,30), train_auc, label='Train AUC')
plt.scatter(range(1,30), train_auc, label='Train AUC')
plt.plot(range(1,30), cv_auc, label='CV AUC')
plt.scatter(range(1,30), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [206]:
```
#Best k = 29 from AUC graph
neigh = KNeighborsClassifier(n_neighbors=29,algorithm='kd_tree')
```

```
neigh.fit(tfidf_sent_vectors_train, 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_sent_
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_sent_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_sent_vectors_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(tfidf_sent_vectors_test)))
```




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

```
Train confusion matrix
[[  41 1291]
 [  21 7625]]
Test confusion matrix
[[   9 1054]
 [  18 5519]]
```

Conclusion : TPR = 0.84 but TNR is just 9/27=0.33. So this model is not that good.

7 [6] Conclusions

In [213]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper parameter(K)", "AUC"]

x.add_row(["BOW", "brute", 16, 0.65])
x.add_row(["TFIDF", "brute", 29, 0.50])
x.add_row(["AVG W2V", "brute", 29, 0.84])
x.add_row(["TFIDF W2V", "brute", 29, 0.81])

x.add_row(["BOW", "kd-tree", 29, 0.71])
x.add_row(["TFIDF", "kd-tree", 29, 0.60])
x.add_row(["AVG W2V", "kd-tree", 29, 0.73])
x.add_row(["TFIDF W2V", "kd-tree", 29, 0.67])

print(x)
```

```
+------------+---------+--------------------+------+
| Vectorizer |  Model  | Hyper parameter(K) | AUC  |
+------------+---------+--------------------+------+
|    BOW     |  brute  |         16         | 0.65 |
|   TFIDF    |  brute  |         29         | 0.5  |
|  AVG W2V   |  brute  |         29         | 0.84 |
| TFIDF W2V  |  brute  |         29         | 0.81 |
|    BOW     | kd-tree |         29         | 0.71 |
|   TFIDF    | kd-tree |         29         | 0.6  |
|  AVG W2V   | kd-tree |         29         | 0.73 |
| TFIDF W2V  | kd-tree |         29         | 0.67 |
+------------+---------+--------------------+------+
```

Seeing the above table of comparition of different model, it is quite evident that AVG W2V is performing best with Bruteforce method. Other than that TFIDF W2V (Brute) and AVG W2V (kd-tree) also quite well performing model.

In [ ]: