Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [3]:

```
%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,roc_auc_score,a
ccuracy_score

from tqdm import tqdm
import os
```

```
C:\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunk
ize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [4]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/sesha/OneDrive/Desktop/ICONS/IMP/before/MINIPJ/Personal/AMAZON foo
d review 2/database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

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

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

```
Number of data points in our data (100000, 10)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [5]:

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

In [6]:

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

```
(80668, 7)
```

Out[6]:

	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 [7]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[7]:

	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 [8]:

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

Out[8]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

[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 [9]:

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

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

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 [10]:

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

In [11]:

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

Out[11]:

```
(87775, 10)
```

In [12]:

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

Out[12]:

```
87.775
```

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 [13]:

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

display.head()
```

Out[13]:

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

In [14]:

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

In [15]:

```
#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()
```

```
(87773, 10)
```

Out[15]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1] Preprocessing Review Text

[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 [16]:

```
# 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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [17]:

```
# 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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [18]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an
```

```
-element
from bs4 import BeautifulSoup

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
```

In [19]:

```
# 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 [20]:

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

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [21]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
```

```
#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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [22]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [23]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [24]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 87773/87773
[00:25<00:00, 3468.51it/s]
```

In [25]:

```
preprocessed_reviews[1500]
```

Out[25]:

```
'way hot blood took bite jig lol'
```

[4] Featurization

[4.1] BAG OF WORDS

In [26]:

```
Y = final['Score'].values
X = final['Text'].values
```

In [32]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

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


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()
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)
print("="*100)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(39400,) (39400,)
(19407,) (19407,)
(28966,) (28966,)
====================================================================================================

After vectorizations
(39400, 38368) (39400,)
(19407, 38368) (19407,)
(28966, 38368) (28966,)
====================================================================================================

NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

[4.3] TF-IDF

In [33]:

```
# TRAIN DATA
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
tfidf_train= tf_idf_vect.transform(X_train)
print("The shape of TEXT DATA ",tfidf_train.get_shape())

# CV DATA
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
tfidf_cv = tf_idf_vect.transform(X_cv)
print("The shape of CV DATA",tfidf_cv.get_shape())

#TEST DATA
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
tfidf_test = tf_idf_vect.transform(X_test)
print("The shape of TRAIN DATA ",tfidf_test.get_shape())
```

```
The shape of TEXT DATA  (39400, 44377)
The shape of CV DATA (19407, 44377)
The shape of TRAIN DATA  (28966, 44377)
```

[4.4] Word2Vec

In [34]:

```
# Train your own Word2Vec model using X_train
i=0
sent_of_train=[]
for sentance in X_train:
    sent_of_train.append(sentance.split())

# Train your own Word2Vec model using X_test
i=0
sent_of_test=[]
for sentance in X_test:
    sent_of_test.append(sentance.split())
```

In [35]:

```
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'))
```

```
[('wonderful', 0.8880136609077454), ('fantastic', 0.8658082485198975), ('good',
0.8550300598144531), ('perfect', 0.7955553531646729), ('awesome', 0.7660056352615356),
('excellent', 0.7464603781700134), ('decent', 0.7436758875846863), ('delicious',
0.7390892505645752), ('terrific', 0.7388360500335693), ('great,', 0.728641152381897)]
==================================================
[('best', 0.8488404750823975), ('tastiest', 0.7427985668182373), ('closest', 0.7329630851745605),
('best.', 0.705921471118927), ('greatest', 0.7029646635055542), ('BEST', 0.6975589990615845), ('ho
ttest', 0.6854673624038696), ('ever', 0.6781014204025269), ('smoothest', 0.6739946603775024),
('tasted.', 0.6604142189025879)]
```

In [36]:

```
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  23290
sample words  ['<span', 'class="tiny">', 'Length::', 'Mins<br', '/><br', 'with', 'a', 'cup', 'of',
'kibble,', 'this', 'toy', 'my', '60', 'lb', 'German', 'Shepherd', 'puppy,', 'for', '30',
'minutes,', 'which', 'is', 'impressive.', "It's", 'our', 'favorite', 'the', 'Busy', 'Buddy', 'toys
.', 'I', 'only', 'wish', 'it', "weren't", 'so', 'likes', 'to', 'hard', 'plastic', 'bottle',
'around', 'and', 'hit', 'things.', 'found', 'dog', 'food', 'after']
```

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

[4.4.1.1] Avg W2v

In [37]:

```
#TEST VECTORS
# average Word2Vec
# compute average word2vec for each review.
test_vectors_avgw2v= []; # the avg-w2v for each sentence/review is stored in this list
sent_vectors
for sent in tqdm(X_test): # for each review/sentence
    test_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            test_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        test_vec /= cnt_words
    test_vectors_avgw2v.append(test_vec)
print(len(test_vectors_avgw2v))
print(len(test_vectors_avgw2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 28966/28966 [39
:00<00:00, 12.38it/s]
```

```
28966
50
```

In [38]:

```
#TRAIN VECTORS
# average Word2Vec
# compute average word2vec for each review.
train_vectors_avgw2v= []; # the avg-w2v for each sentence/review is stored in this list
sent_vectors
for sent in tqdm(X_train): # for each review/sentence
    train_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change
this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            train_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        train_vec /= cnt_words
    train_vectors_avgw2v.append(train_vec)
print(len(train_vectors_avgw2v))
print(len(train_vectors_avgw2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 39400/39400 [50
:46<00:00, 12.93it/s]
```

```
39400
50
```

In [39]:

```
#CV VECTORS
# average Word2Vec
# compute average word2vec for each review.
cv_vectors_avgw2v= []; # the avg-w2v for each sentence/review is stored in this list sent_vectors
for sent in tqdm(X_cv): # for each review/sentence
```

```
    cv_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            cv_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        cv_vec /= cnt_words
    cv_vectors_avgw2v.append(cv_vec)
print(len(cv_vectors_avgw2v))
print(len(cv_vectors_avgw2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 19407/19407 [24
:42<00:00, 18.55it/s]
```

```
19407
50
```

[4.4.1.2] TFIDF weighted W2v

In [40]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix_train = model.fit_transform(X_train)
# 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 TRAIN SET
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_w2v_train_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_train): # for each review/sentence
    train_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_w2v_train_vectors.append(train_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████████████████| 39400/39400
[2:35:59<00:00,  3.03it/s]
```

In [42]:

```
print(len(tfidf_w2v_train_vectors))
```

```
39400
```

In [43]:

```
# TF-IDF weighted Word2Vec TEST SET
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
```

```
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_w2v_test_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_test): # for each review/sentence
    test_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_w2v_test_vectors.append(test_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████████| 28966/28966
[1:53:01<00:00,  4.27it/s]
```

In [44]:

```
print(len(tfidf_w2v_test_vectors))
```

```
28966
```

In [45]:

```
# TF-IDF weighted Word2Vec CV SET
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_w2v_cv_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_cv): # for each review/sentence
    cv_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_w2v_cv_vectors.append(cv_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████████| 19407/19407
[1:35:46<00:00,  3.94it/s]
```

In [46]:

```
print(len(tfidf_w2v_cv_vectors))
```

```
19407
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) on these feature sets**

1. **Apply Knn(brute force version) on these feature sets**

   - SET 1:Review text, preprocessed one converted into vectors using (BOW)
   - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Apply Knn(kd tree version) on these feature sets**
   NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this link

   - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

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

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

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

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

3. **The hyper paramter tuning(find best K)**

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

4. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

5. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

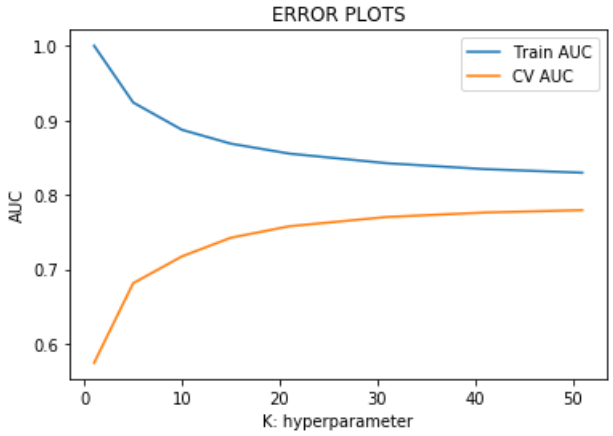
[5.1] Applying KNN brute force

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

In [47]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
```

```
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

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

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

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



In [88]:

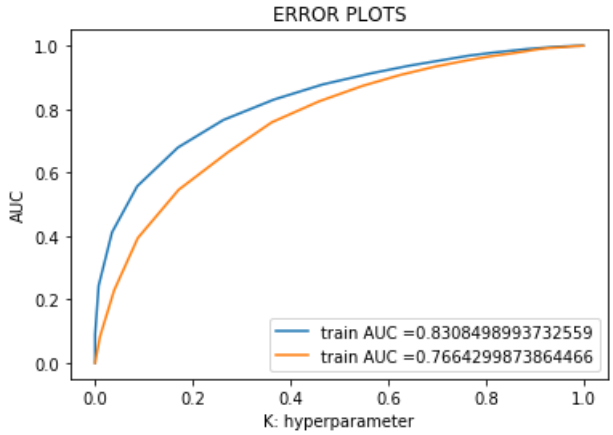
```
best_k=47
```

TESTING WITH TEST DATA

In [90]:

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


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

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

```
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="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(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))

# Summarizing the values
bow_brute_K = best_k
bow_brute_train= train_auc
bow_brute_test = cv_auc
```



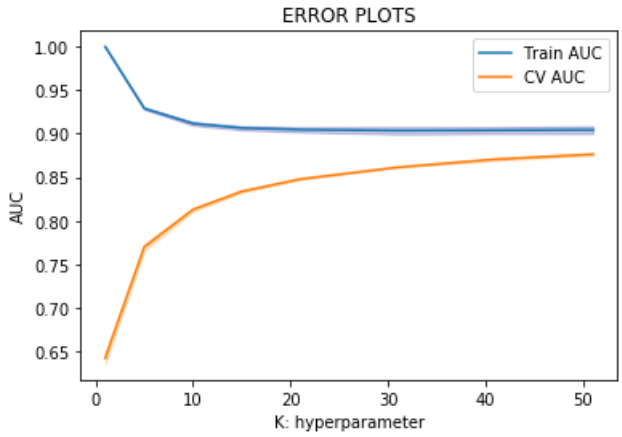

```
====================================================================================================

Train confusion matrix
[[   47  6309]
 [    6 33038]]
Test confusion matrix
[[   20  4659]
 [    8 24279]]
```

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

In [50]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(tfidf_train, 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=0.2,color='darkb
lue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
```

```
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



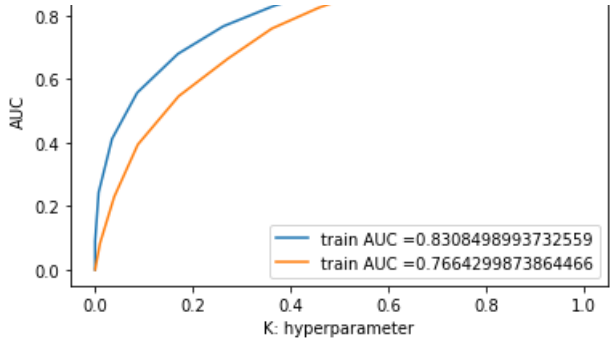
In [ ]:

```
best_k=45
```

TESTING WITH TEST DATA

In [91]:

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


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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="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(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))

# Summarizing the values

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



==========

```
Train confusion matrix
[[   47  6309]
 [    6 33038]]
Test confusion matrix
[[   20  4659]
 [    8 24279]]
```

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

Working with word2vec

Preparing Reviews for gensim model

In [53]:

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

In [54]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

In [55]:

```
# 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 [56]:

```
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  23290
sample words  ['<span', 'class="tiny">', 'Length::', 'Mins<br', '/><br', 'with', 'a', 'cup', 'of',
'kibble,', 'this', 'toy', 'my', '60', 'lb', 'German', 'Shepherd', 'puppy,', 'for', '30',
'minutes,', 'which', 'is', 'impressive.', "It's", 'our', 'favorite', 'the', 'Busy', 'Buddy', 'toys
.', 'I', 'only', 'wish', 'it', "weren't", 'so', 'likes', 'to', 'hard', 'plastic', 'bottle',
'around', 'and', 'hit', 'things.', 'found', 'dog', 'food', 'after']
```

Converting Reviews into Numerical Vectors using W2V vectors

Algorithm: Avg W2V

Algorithm: Avg W2v

In [57]:

```
from tqdm import tqdm
import numpy as np
```

Converting Train data text

In [58]:

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

```
100%|██████████████████████████████████████████████████████████████████████████████| 39400/39400 [02:
39<00:00, 247.72it/s]
```

```
(39400, 50)
[ 0.09936383  0.32583968 -0.84825908  0.17466137 -0.4919007  -0.20970294
  0.68205659 -0.44839912  0.06595315 -0.0502705   0.14605214 -0.0970234
 -0.76191734  0.26188279 -0.53403658 -0.48783047 -0.01867198 -0.21292501
 -0.41253758  0.13731108  0.22870046  0.20682593 -0.38671847 -0.09141498
 -0.24214392  0.22971407  0.01019453 -0.10017818  0.20843573 -0.26555206
  0.0406852  -0.5928359   0.44081797  0.33154109  0.16329402 -0.2468434
 -0.5020074   0.7022068   0.76981334 -0.06564124 -0.58529191 -0.15675232
  0.66597863 -0.18712556 -0.35134968 -0.66587853  0.06388933 -0.46898245
  0.13819367 -0.37983168]
```

Converting CV data text

In [59]:

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

In [60]:

```
# 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 to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
```

```
    sent_vectors_cv.append(sent_vec)
sent_vectors_cv = np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
100%|████████████████████████████████████████████████████████████████| 19407/19407 [01:13<00:00, 263.92it/s]
```

```
(19407, 50)
[-0.20771698  0.77564057 -0.7408383   0.21118608 -0.56562494  0.26279409
  0.52145747 -0.33939505  0.40667276  0.04780574 -0.10242904 -0.5917149
 -0.96501295  0.10307733 -0.2062607  -0.23353849  0.11439712 -0.20677446
  0.24156997 -0.07904129  0.23421204  0.06298535 -0.8965666   0.2570546
 -0.09917909  0.23263699  0.32832351 -0.05508822  0.3014569  -0.32415604
 -0.31880579 -0.50346375  0.49230307  0.16564708 -0.07458735 -0.20528457
 -0.39325897  0.74844276  0.35573095 -0.08681086 -0.47312074 -0.36188322
  0.75835275 -0.23161657 -0.06797857 -0.01819318 -0.50817776 -0.42608685
  0.19104849 -0.18522249]
```

Converting Test data text

In [61]:

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

In [62]:

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

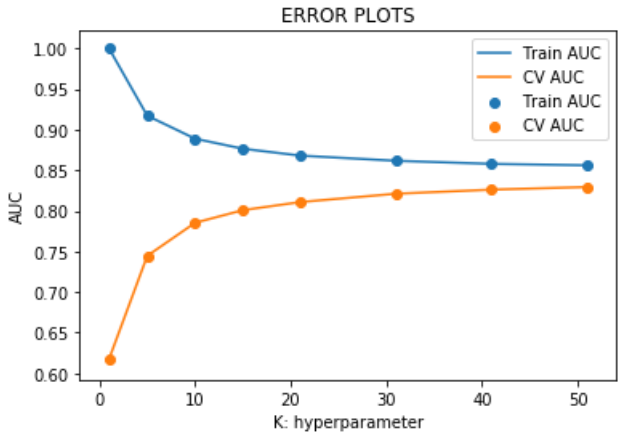
```
100%|████████████████████████████████████████████████████████████████| 28966/28966 [01:49<00:00, 228.91it/s]
```

```
(28966, 50)
[ 0.39787429  0.27554153 -1.00829184  0.27919239 -0.91170146  0.28580303
  0.72916656 -0.2727758  -0.40675347 -0.13259418  0.09724499 -0.36348884
 -0.67020501  0.90835531  0.07008341 -0.32119013  0.26746514 -0.00910771
 -0.05215402  0.24674246  0.13876777  0.05471492 -0.65686616 -0.00471886
 -0.09334091  0.48711172 -0.2593969  -0.66707474  0.4974678  -1.09543606
  0.54860576 -0.40232833  0.44430536  0.4543359   0.4638557  -0.21726917
 -0.33558476  0.46082919  0.31149242  0.12330095 -0.28365576  0.11278309
  0.67586516 -0.23836512 -0.72076703 -0.65771766 -0.46369162 -0.46848176
  0.1535684  -0.72622334]
```

Till now, we have trained the W2V model only one time with train data
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

In [63]:

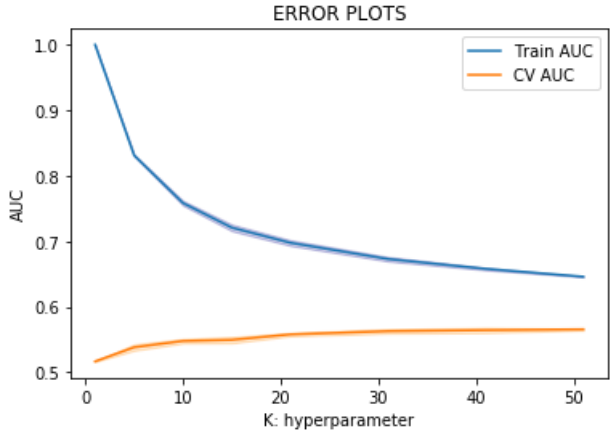
```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted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

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

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



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

In [94]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
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_avgw2v, 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=0.2,color='darkb
lue')

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

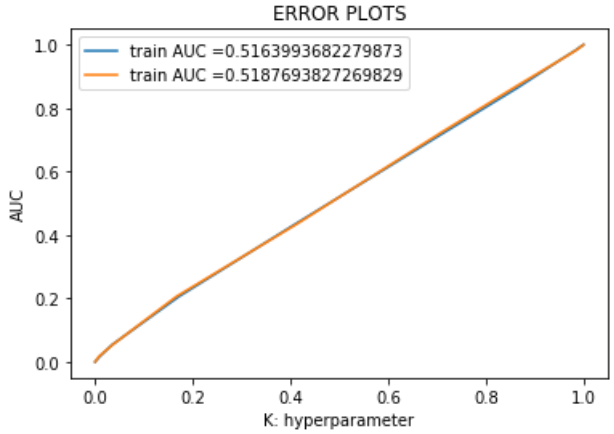


In [95]:

```
best_k=33
```

In [98]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(sent_vectors_train, y_train)

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


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(train_vectors_avgw2v)[:,
1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(test_vectors_avgw2v)[:,1])

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_avgw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(test_vectors_avgw2v)))


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

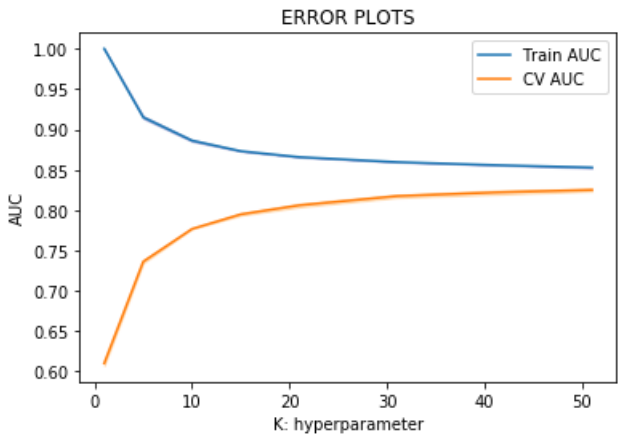


```
==================================================================================================

Train confusion matrix
[[    0  6356]
 [    0 33044]]
Test confusion matrix
[[    0  4679]
 [    0 24287]]
```

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

In [65]:

```
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(sent_vectors_train, 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=0.2,color='darkb
lue')

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



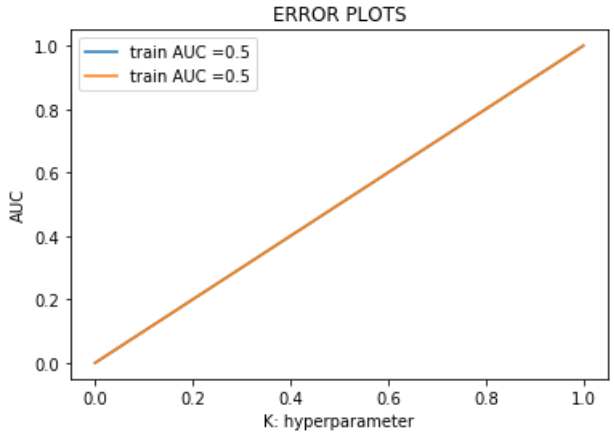
In [66]:

```
best_k=39
```

In [99]:

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


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(sent_vectors_test, y_test)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs12)[:,1]
```

```
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_w2v_train_vectors)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_w2v_test_vectors)[:,1]
)

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_w2v_train_vectors)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(tfidf_w2v_test_vectors)))

# Summarizing the values

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



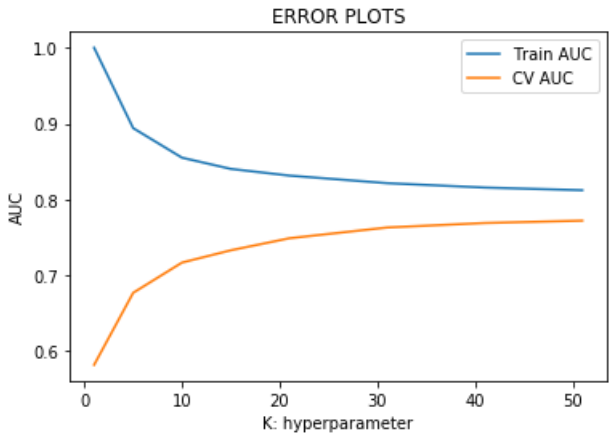

```
====================================================================================================

Train confusion matrix
[[    0  6356]
 [    0 33044]]
Test confusion matrix
[[    0  4679]
 [    0 24287]]
```

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

In [73]:

```
from sklearn import tree

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)
Xcv_bow = svd.transform(X_cv_bow)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
```

```
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(Xtrain_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(Xtrain_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(Xcv_bow)[:,1]

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

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



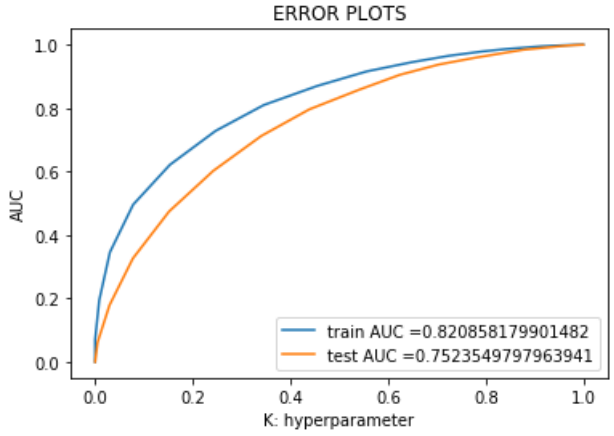
In [75]:

```
best_k=43
```

In [100]:

```
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 of the positive
class
# not the predicted outputs

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

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

print("="*100)

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)))

# Summarizing the values

BOW_kd_tree_K = best_k
BOW_kd_tree_train = train_auc
BOW_kd_tree_test = cv_auc
```



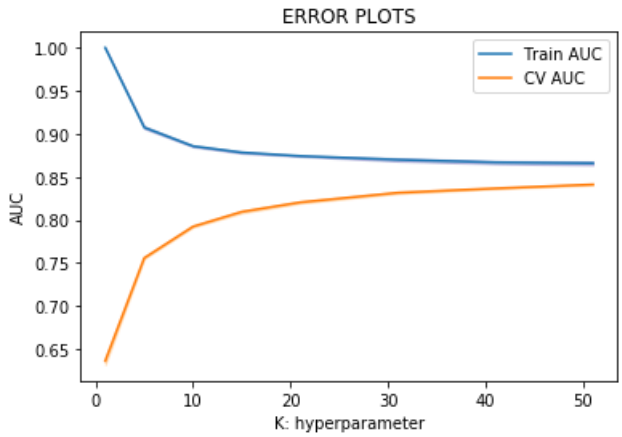
==========

```
Train confusion matrix
[[  267  6089]
 [   58 32986]]
Test confusion matrix
[[  145  4534]
 [   59 24228]]
```

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

In [78]:

```
svd = TruncatedSVD(n_components=100)
Xtrain_tfidf = svd.fit_transform(tfidf_train)
Xtest_tfidf = svd.transform(tfidf_test)
Xcv_tfidf = svd.transform(tfidf_cv)

# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
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=0.2,color='darkb
lue')

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

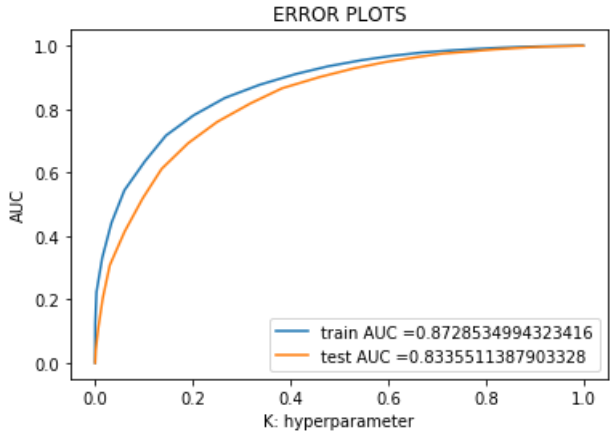


In [79]:

```
best_k= 41
```

In [101]:

```
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 of the positive
class
# not the predicted outputs

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

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

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)))

# Summarizing the values

TFIDF_kd_tree_K = best_k
TFIDF_kd_tree_train = train_auc
TFIDF_kd_tree_test = cv_auc
```



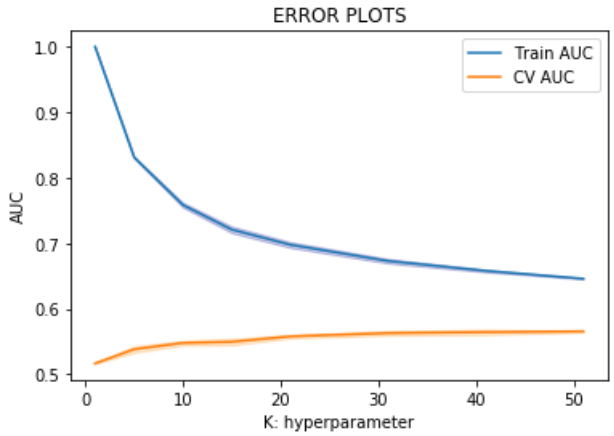
==========

```
Train confusion matrix
[[ 1695  4661]
 [  472 32572]]
Test confusion matrix
[[ 1062  3617]
 [  421 23866]]
```

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

In [81]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
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_avgw2v, 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=0.2,color='darkb
lue')

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



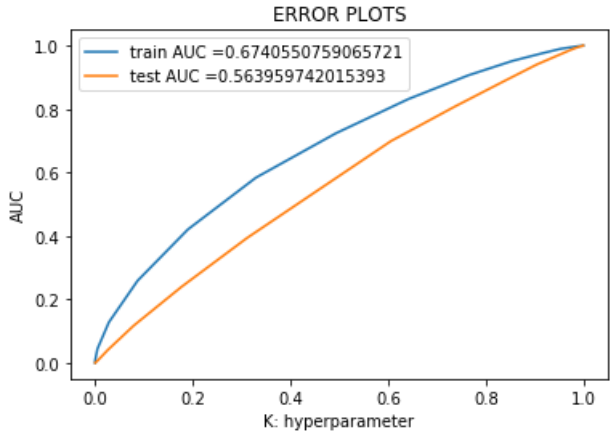
In [82]:

```
best_k= 37
```

In [102]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(train_vectors_avgw2v, y_train)

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(train_vectors_avgw2v)[:,
1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(test_vectors_avgw2v)[:, 1])
```

```
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(test_vectors_avgw2v)[:,1])

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

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(train_vectors_avgw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(test_vectors_avgw2v)))

# Summarizing the values

AVG_Word2Vec_kd_tree_K = best_k
AVG_Word2Vec_kd_tree_train = train_auc
AVG_word2Vec_kd_tree_test = cv_auc
```



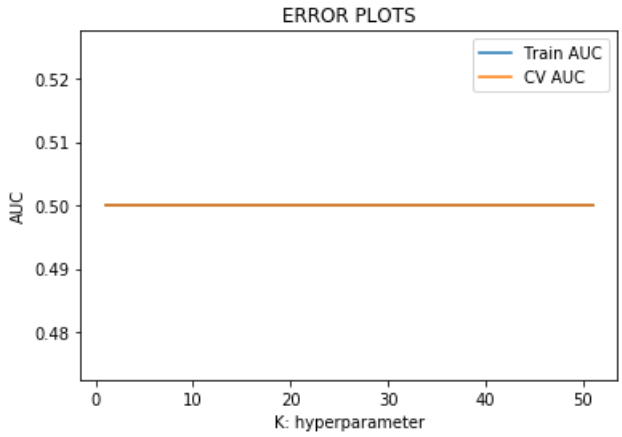

```
====================================================================================================

Train confusion matrix
[[   15  6341]
 [   11 33033]]
Test confusion matrix
[[    5  4674]
 [    7 24280]]
```

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

In [84]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
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(tfidf_w2v_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=0.2,color='darkb
lue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
```

```
# this code is copied from here: https://stackoverflow.com/a/48003501/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

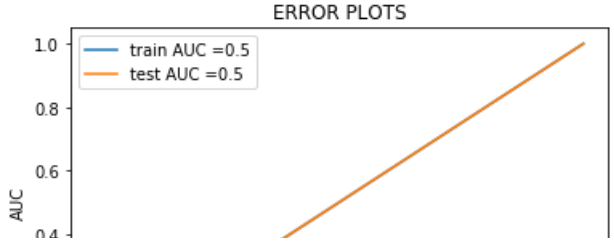


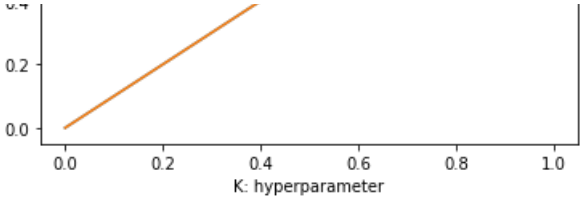
In [87]:

```
best_k=49
```

In [103]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(tfidf_w2v_train_vectors, y_train)

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_w2v_train_vectors)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_w2v_test_vectors)[:,1]
)

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

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(tfidf_w2v_train_vectors)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(tfidf_w2v_test_vectors)))

# Summarizing the values

TFIDF_Word2Vec_kd_tree_K = best_k
TFIDF_Word2Vec_kd_tree_train = train_auc
TFIDF_word2Vec_kd_tree_test = cv_auc
```





==========

```
Train confusion matrix
[[    0  6356]
 [    0 33044]]
Test confusion matrix
[[    0  4679]
 [    0 24287]]
```

[6] Conclusions

In [115]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["VECTORIZER", "MODEL", "HYPERPARAMETER", "TRAIN AUC","TEST AUC"]

x.add_row(["BRUTE","BOW", bow_brute_K,bow_brute_train,bow_brute_test])
x.add_row(["BRUTE","TFIDF", tfidf_brute_K,tfidf_brute_train,tfidf_brute_test])
x.add_row(["BRUTE","TFIDF AVGW2V",
Avg_Word2Vec_brute_K,Avg_Word2Vec_brute_train,Avg_word2Vec_brute_test])
x.add_row(["BRUTE","TFIDF W2V ",
TFIDF_Word2Vec_brute_K,TFIDF_Word2Vec_brute_train,TFIDF_word2Vec_brute_test])
x.add_row(["KD TREE","BOW", BOW_kd_tree_K,BOW_kd_tree_train,BOW_kd_tree_test])
x.add_row(["KD TREE","TFIDF", TFIDF_kd_tree_K,TFIDF_kd_tree_train,TFIDF_kd_tree_test])
x.add_row(["KD TREE","TFIDF AVG W2v",
AVG_Word2Vec_kd_tree_K,AVG_Word2Vec_kd_tree_train,AVG_word2Vec_kd_tree_test])
x.add_row(["KD TREE","TFIDF W2V",
TFIDF_Word2Vec_kd_tree_K,TFIDF_Word2Vec_kd_tree_train,TFIDF_word2Vec_kd_tree_test])

# Printing the Table
print(x)
```

```
+------------+---------------+----------------+-------------------------------------------------------------
------------+-------------------------------------------------------------------------+
| VECTORIZER |     MODEL     | HYPERPARAMETER |                                  TRAIN AUC
|                                TEST AUC                                 |
+------------+---------------+----------------+-------------------------------------------------------------
------------+-------------------------------------------------------------------------+
|   BRUTE    |      BOW      |       47       |                 [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]
|                [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]                |
|   BRUTE    |     TFIDF     |       47       |                 [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]
|                [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]                |
|   BRUTE    |  TFIDF AVGW2V |       33       | [1.         0.83112319 0.75847551 0.72088606 0.6977
566 0.67303253 | [0.51681355 0.53836529 0.54797142 0.54965779 0.55802418 0.56298117 |
|            |               |                |                      0.65808045 0.6460741 ]
|                      0.564522   0.56577151]                      |
|   BRUTE    |   TFIDF W2V   |       33       | [1.         0.83112319 0.75847551 0.72088606 0.6977
566 0.67303253 | [0.51681355 0.53836529 0.54797142 0.54965779 0.55802418 0.56298117 |
|            |               |                |                      0.65808045 0.6460741 ]
|                      0.564522   0.56577151]                      |
|  KD TREE   |      BOW      |       33       | [1.         0.83112319 0.75847551 0.72088606 0.6977
566 0.67303253 | [0.51681355 0.53836529 0.54797142 0.54965779 0.55802418 0.56298117 |
|            |               |                |                      0.65808045 0.6460741 ]
|                      0.564522   0.56577151]                      |
|  KD TREE   |     TFIDF     |       33       | [1.         0.83112319 0.75847551 0.72088606 0.6977
566 0.67303253 | [0.51681355 0.53836529 0.54797142 0.54965779 0.55802418 0.56298117 |
|            |               |                |                      0.65808045 0.6460741 ]
|                      0.564522   0.56577151]                      |
|  KD TREE   | TFIDF AVG W2v |       33       | [1.         0.83112319 0.75847551 0.72088606
0.69775566 0.67303253 | [0.51681355 0.53836529 0.54797142 0.54965779 0.55802418 0.56298117 |
|            |               |                |                      0.65808045 0.6460741 ]
|                      0.564522   0.56577151]                      |
```

```
|                      0.564522   0.56577151]                      |
|  KD TREE   |   TFIDF W2V   |       33       | [1.         0.83112319 0.75847551 0.72088606 0.6977
566 0.67303253 | [0.51681355 0.53836529 0.54797142 0.54965779 0.55802418 0.56298117 |
|            |               |                |                       0.65808045 0.6460741 ]
|                      0.564522   0.56577151]                      |
+------------+---------------+----------------+--------------------------------------------------
------------+-----------------------------------------------------------------------+
```

In [ ]: