In [ ]:

Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
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 [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data p
# 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 LIM
# for tsne assignment you can take 5k data points

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

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

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

```
Number of data points in our data (200000, 10)
```

Out[2]:

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

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

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

In [6]:
```
display['COUNT(*)'].sum()
```

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
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	

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 [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplac
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},
final.shape
```

Out[9]:
```
(160178, 10)
```

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]:
```
80.089
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entries le
print(final.shape)

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

```
(160176, 10)
```

Out[13]:
```
1    134799
0     25377
Name: Score, dtype: int64
```

segragating datapoints w.r.t calss labels

In [14]:
```
zero_class=final[final.Score==0]
print(zero_class['Score'].value_counts())
print(zero_class.shape)
one_class=final[final.Score==1]
print(one_class['Score'].value_counts())
print(one_class.shape)
```

```
0    25377
Name: Score, dtype: int64
(25377, 10)
1    134799
Name: Score, dtype: int64
(134799, 10)
```

selecting 25377 random points

In [15]:
```
one_class1=one_class.sample(n=25377)
print(one_class1.shape)
```

```
(25377, 10)
```

combining both the datapoints to obtain a balanced dataset

In [16]:
```
print(zero_class.shape)
print(one_class1.shape)
combined_frame=pd.concat([zero_class,one_class1])
print(combined_frame.shape)
```

```
(25377, 10)
(25377, 10)
(50754, 10)
```

Shuffeling the data ponts randomly

In [17]:
```
final_new_frame=combined_frame.sample(frac=1)
```

In [18]:
```
print(type(final_new_frame))
print(final_new_frame.shape)
print(final_new_frame['Score'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
(50754, 10)
1    25377
0    25377
Name: Score, dtype: int64
```

PreProcessing

In [19]:
```
import re
from bs4 import BeautifulSoup

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


stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            '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'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"])



from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final_new_frame['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 s
    preprocessed_reviews.append(sentance.strip())

j=0
for i in tqdm(preprocessed_reviews):
    j=j+1
print(j)
```

```
100%|██████████████████████████████████████████████████████████████████
| 50754/50754 [00:20<00:00, 2441.86it/s]
100%|█████████████████████████████████████████████████████████████████| 5
0754/50754 [00:00<00:00, 2111861.05it/s]

50754
```

In [20]:
```python
print(len(preprocessed_reviews))
print(type(final_new_frame))
print(final_new_frame.shape)
```

```
50754
<class 'pandas.core.frame.DataFrame'>
(50754, 10)
```

End of preprocessing

In [ ]:

Below Preprcesing is not used

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [21]:
```
# 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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as
good as I remembered and my kids love it too.  My older daughter now reads it t
o her sister.  Good rhymes and nice pictures.
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his
stomach, you get 10 more pounds and it is cheaper wich is a plus for me. You ca
n always ad your own rice and veggies. Its fresher that way and better for him
in my opinion. Plus if you you can get it deliverd to your house for free its e
ven better.  Gotta love pitbulls
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word,
and&quot;ko-&quot;  is &quot;child of&quot; or of &quot;derived from&quot;.) Pa
nko are used for katsudon, tonkatsu or cutlets served on rice or in soups. The
cutlets, pounded chicken or pork, are coated with these light and crispy crumbs
and fried. They are not gritty and dense like regular crumbs. They are very nic
e on deep fried shrimps and decorative for a more gourmet touch.
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, i
t's perfect for me.  I like this flavor best with my Senseo... It has a nice da
rk full body flavor without the burt bean taste I tend sense with starbucks.  I
t's a shame most americans haven't bought into single serve coffe makers as our
Dutch counter parts have.  Every cup is fresh brewed and doesn't sit long enoug
h on my desk to get that old taste either.
==================================================
```

In [22]:
```
# 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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as
good as I remembered and my kids love it too.  My older daughter now reads it t
o her sister.  Good rhymes and nice pictures.
```

In [23]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove
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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as
good as I remembered and my kids love it too.  My older daughter now reads it t
o her sister.  Good rhymes and nice pictures.
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his
stomach, you get 10 more pounds and it is cheaper wich is a plus for me. You ca
n always ad your own rice and veggies. Its fresher that way and better for him
in my opinion. Plus if you you can get it deliverd to your house for free its e
ven better.  Gotta love pitbulls
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word,
and"ko-"  is "child of" or of "derived from".) Panko are used for katsudon, ton
katsu or cutlets served on rice or in soups. The cutlets, pounded chicken or po
rk, are coated with these light and crispy crumbs and fried. They are not gritt
y and dense like regular crumbs. They are very nice on deep fried shrimps and d
ecorative for a more gourmet touch.
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, i
t's perfect for me.  I like this flavor best with my Senseo... It has a nice da
rk full body flavor without the burt bean taste I tend sense with starbucks.  I
t's a shame most americans haven't bought into single serve coffe makers as our
Dutch counter parts have.  Every cup is fresh brewed and doesn't sit long enoug
h on my desk to get that old taste either.
```

In [24]:
```
# 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 [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were o
rdering; the other wants crispy cookies.  Hey, I am sorry; but these reviews do
nobody any good beyond reminding us to look  before ordering.<br /><br />These
are chocolate-oatmeal cookies.  If you do not like that combination, do not ord
er this type of cookie.  I find the combo quite nice, really.  The oatmeal sort
of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-typ
e consistency.  Now let is also remember that tastes differ; so, I have given m
y opinion.<br /><br />Then, these are soft, chewy cookies -- as advertised.  Th
ey are not "crispy" cookies, or the blurb would say "crispy," rather than "chew
y."  I happen to like raw cookie dough; however, I do not see where these taste
like raw cookie dough.  Both are soft, however, so is this the confusion?  And,
yes, they stick together.  Soft cookies tend to do that.  They are not individu
ally wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies ten
d to be somewhat sweet.<br /><br />So, if you want something hard and crisp, I
suggest Nabiso is Ginger Snaps.  If you want a cookie that is soft, chewy and t
astes like a combination of chocolate and oatmeal, give these a try.  I am here
to place my second order.
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> />
<br />The Victor  and  traps are unreal, of course -- total fly genocide. Prett
y stinky, but only right nearby.
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were orderi
ng the other wants crispy cookies Hey I am sorry but these reviews do nobody an
y good beyond reminding us to look before ordering br br These are chocolate oa
tmeal cookies If you do not like that combination do not order this type of coo
kie I find the combo quite nice really The oatmeal sort of calms the rich choco
late flavor and gives the cookie sort of a coconut type consistency Now let is
also remember that tastes differ so I have given my opinion br br Then these ar
e soft chewy cookies as advertised They are not crispy cookies or the blurb wou
ld say crispy rather than chewy I happen to like raw cookie dough however I do
not see where these taste like raw cookie dough Both are soft however so is thi
s the confusion And yes they stick together Soft cookies tend to do that They a
re not individually wrapped which would add to the cost Oh yeah chocolate chip
cookies tend to be somewhat sweet br br So if you want something hard and crisp
I suggest Nabiso is Ginger Snaps If you want a cookie that is soft chewy and ta
stes like a combination of chocolate and oatmeal give these a try I am here to
place my second order
```

In [21]:
```
# 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 st

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            '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'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# 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 s
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████
█| 4986/4986 [00:01<00:00, 3137.37it/s]
```

In [23]:
```
preprocessed_reviews[1500]
```

Out[23]:
```
'wow far two two star reviews one obviously no idea ordering wants crispy cooki
es hey sorry reviews nobody good beyond reminding us look ordering chocolate oa
tmeal cookies not like combination not order type cookie find combo quite nice
really oatmeal sort calms rich chocolate flavor gives cookie sort coconut type
consistency let also remember tastes differ given opinion soft chewy cookies ad
vertised not crispy cookies blurb would say crispy rather chewy happen like raw
cookie dough however not see taste like raw cookie dough soft however confusion
yes stick together soft cookies tend not individually wrapped would add cost oh
yeah chocolate chip cookies tend somewhat sweet want something hard crisp sugge
st nabiso ginger snaps want cookie soft chewy tastes like combination chocolate
oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

In [6]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
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', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'a
bby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [26]:
```
#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/

# 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_bi
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [27]:

```
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_
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
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able fin
d', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely l
ove', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [28]:

```
# 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 [42]:
```
# 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-negative3
        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 = Tr
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful',
0.9946032166481018), ('excellent', 0.9944332838058472), ('especially', 0.994114
4585609436), ('baked', 0.9940600395202637), ('salted', 0.994047224521637), ('al
ternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('healthy', 0.9
936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn',
0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.9992451071739197),
('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('american', 0.
9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.999156713485717
8)]
```

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky',
'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'shipmen
t', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'removed', 'ea
sily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautifull
y', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'comp
uter', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'everybody',
'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [38]:
```
# 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 ne
    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%|██████████████████████████████████████████████████████████████████
█| 4986/4986 [00:03<00:00, 1330.47it/s]

4986
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]:
```python
# 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 = t

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
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]
```

[5] Assignment 3: KNN

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

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

2. **Apply Knn(kd tree version) on these feature sets**
   NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this link (https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.sparse.csr_matrix.toarray.html)

   - 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) 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 (http://zetcode.com/python/prettytable/)](http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)](https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

[5.1] Applying KNN brute force

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

All the vectorizer codes are same are obtaned or reused wherever can be from previous assignments

In [35]:
```
print(len(preprocessed_reviews))
print(type(final_new_frame))
print(final_new_frame.shape)
```

```
50754
<class 'pandas.core.frame.DataFrame'>
(50754, 10)
```

SPlitting the data into train and test

In [36]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews, final_n
```

In [37]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train)

X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)
```

In [38]:
```
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
```

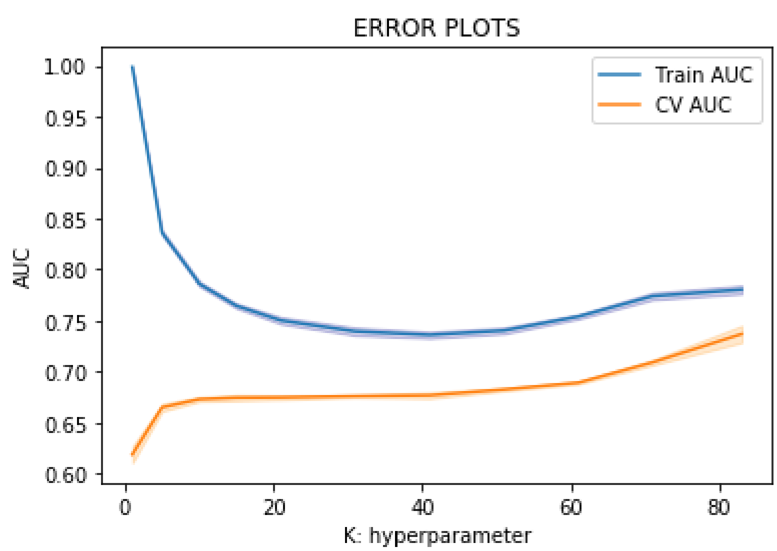
```
After vectorizations
(34005, 36643) (34005,)
(16749, 36643) (16749,)
```

In [39]:
```
print(type(X_train_bow))
print(X_train_bow.get_shape())
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(34005, 36643)
```

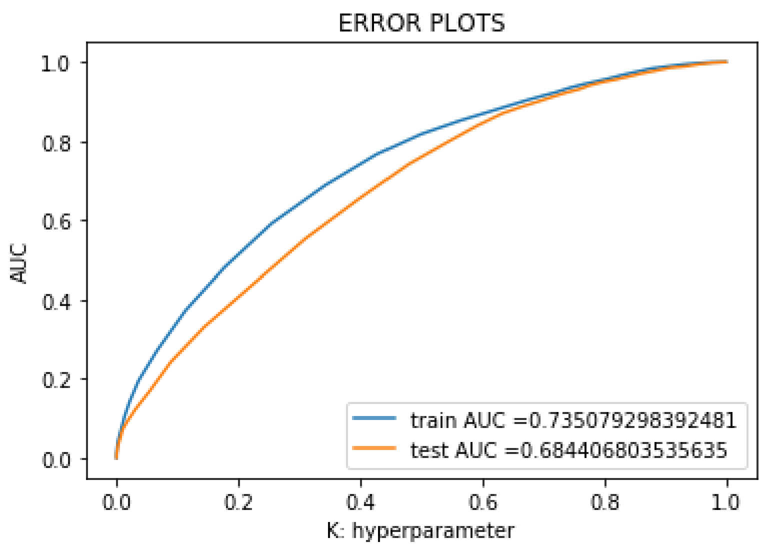
Hyper Parameter Tunning using Grid Search CV

In [40]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,61,71,83]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_bow, y_train)

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

K = [1, 5, 10, 15, 21, 31, 41, 51,61,71,83]


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,alph

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



Applying KNN to n selected from Hyper parameter tunning

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=40)
neigh.fit(X_train_bow, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh.predict_proba(X_train_
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()
```



In [42]:
```
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
[[ 9694  7231]
 [ 3984 13096]]
Test confusion matrix
[[4402 4050]
 [2144 6153]]
```

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

Splitting the data and fitting the TFIDF Vectorizer

In [30]:
```
from sklearn.model_selection import train_test_split
print(len(preprocessed_reviews))
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews, final_n
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
```

```
50754
```

Out[30]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=10,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

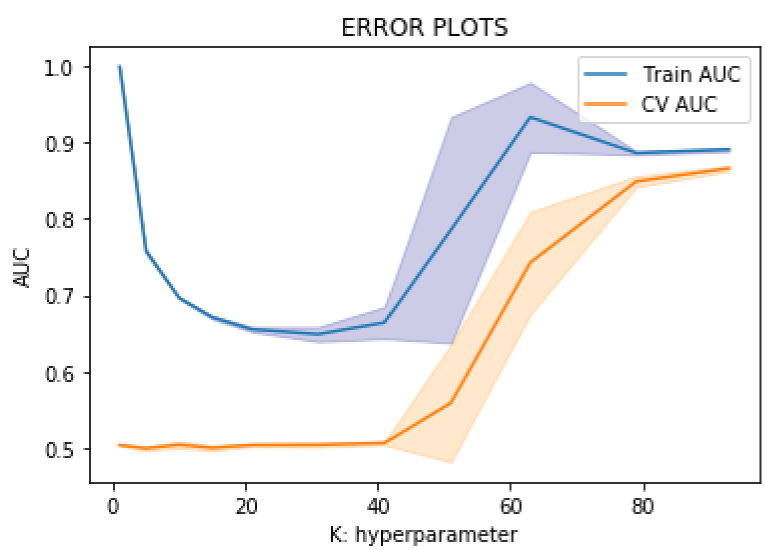
In [31]:
```
X_train_tfidf=tf_idf_vect.transform(X_train)
X_test_tfidf=tf_idf_vect.transform(X_test)
```

In [32]:
```
print(type(X_train_tfidf))
print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(34005, 20716)
(16749, 20716)
```

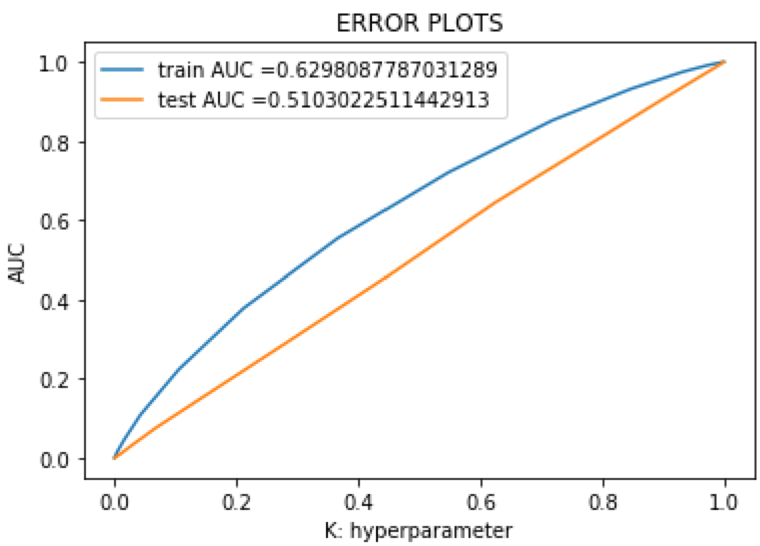
Hyper Parameter Tunning using Grid Search CV

In [33]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,63,79,93]}
#parameters = {'n_neighbors':[1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf, y_train)

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

K =[1, 5, 10, 15, 21, 31, 41, 51,63,79,93]


plt.plot(K, train_auc, label='Train AUC')
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alph
plt.plot(K, cv_auc, label='CV AUC')

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



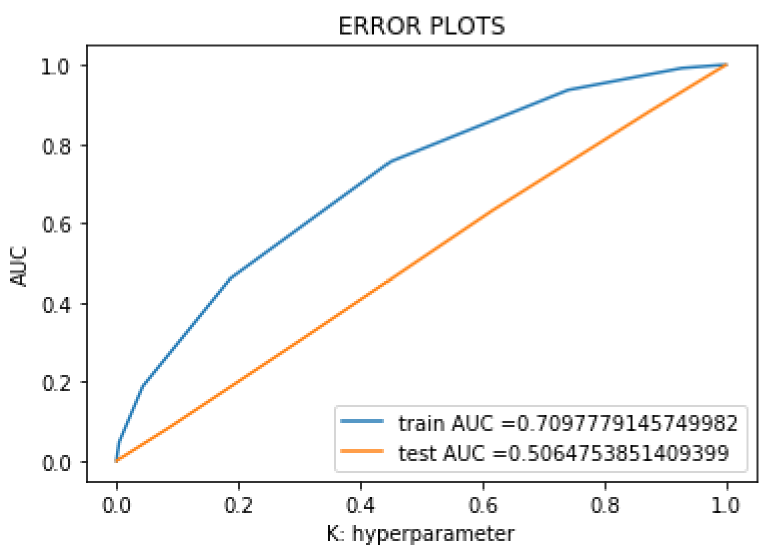
Applying KNN Brute force

In [34]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix

neigh = KNeighborsClassifier(n_neighbors=50)
neigh.fit(X_train_tfidf, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh.predict_proba(X_train_t
test_fpr, test_tpr, thresholds = roc_curve(y_test,neigh.predict_proba(X_test_tfid

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



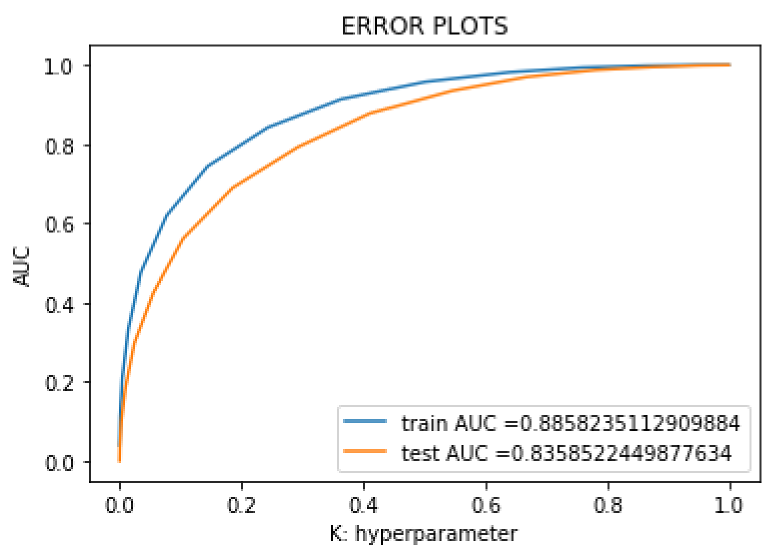

```
Train confusion matrix
[[10834  6276]
 [ 7524  9371]]
Test confusion matrix
[[4527 3740]
 [4554 3928]]
```

In [36]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix

neigh = KNeighborsClassifier(n_neighbors=80)
neigh.fit(X_train_tfidf, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh.predict_proba(X_train_
test_fpr, test_tpr, thresholds = roc_curve(y_test,neigh.predict_proba(X_test_tfid

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




```
Train confusion matrix
[[ 9401  7709]
 [ 4117 12778]]
Test confusion matrix
[[3161 5106]
 [3112 5370]]
```

In [37]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix

neigh = KNeighborsClassifier(n_neighbors=100)
neigh.fit(X_train_tfidf, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh.predict_proba(X_train_
test_fpr, test_tpr, thresholds = roc_curve(y_test,neigh.predict_proba(X_test_tfid

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




```
Train confusion matrix
[[12942  4168]
 [ 2672 14223]]
Test confusion matrix
[[5845 2422]
 [1762 6720]]
```

Checking For n values of 50,80 and 100 we get the best test AUC for n=100 this is in accordance with the hyper parameter tunning result where CV AUC is high and also the dist between CV AUC and Train AUC is less for n=100

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

Vectorizer code refrenced from previous assignment

In [21]:
```python
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews, final_n

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)
```

In [22]:
```python
w2v_words = list(w2v_model.wv.vocab)
```

In [23]:
```python
sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

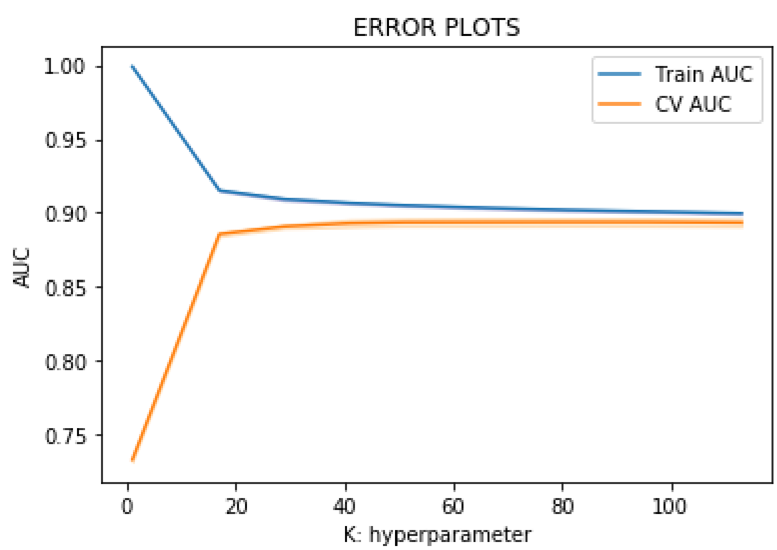
```
100%|██████████████████████████████████████████████████████████████████████
█| 34005/34005 [00:56<00:00, 603.41it/s]

(34005, 50)
[ 0.80395827  0.34084704 -0.04717524 -0.52602211  0.90565988 -0.33880063
  0.01070994 -0.10385925  0.2314673  -0.08582932 -0.00949383  0.22381872
 -0.02368514 -0.0337261  -0.04225203 -0.31601828  0.20311441 -0.13736973
  0.30905328 -0.25564512  0.32558733  0.39170077  0.09300012 -0.27253868
  0.4120241  -0.49218326  0.14699364 -0.62260617  0.13854463 -0.19762198
 -0.33287454 -0.46288454 -0.11524113  0.30399964  0.02425819 -0.15556378
 -0.5770385   0.57380712 -0.41859786 -0.96712761 -0.02562932 -0.53420726
  0.41898786 -0.27531041  0.49739465 -0.10356262 -0.48503531  0.05577242
  0.33640851  0.19985892]
```

In [24]:
```
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

print(type(list_of_sentance_test[0]))
sent_vectors_test = [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
<class 'list'>

100%|████████████████████████████████████████████████████████████
█| 16749/16749 [00:26<00:00, 625.80it/s]

(16749, 50)
[ 1.11455491e-01  8.70539741e-02 -8.57706903e-01 -3.36829973e-01
  5.59557737e-01 -2.24106914e-01  6.58206622e-01  2.96810611e-02
  3.67863611e-01 -6.46198983e-01 -2.58352914e-01 -1.79916478e-01
 -1.08552289e-01 -9.12926334e-01 -5.00331470e-01 -7.74618084e-01
  7.13361213e-04 -5.96841741e-01  6.81761704e-01 -8.11262749e-01
  5.24108489e-01  3.41333755e-01 -1.52924094e-01 -5.04475496e-01
  5.61114556e-02 -6.02913978e-01 -6.67297029e-02 -6.39484508e-01
 -3.94518304e-01 -1.72372180e-01  6.41962525e-01 -7.81751360e-02
 -4.60887676e-01 -2.32047305e-01 -6.46963893e-01 -5.60662672e-01
 -4.31593508e-01  6.66555756e-01 -4.98968795e-01 -8.52756998e-01
 -8.47998737e-01  1.01653123e-01  1.32381014e-02  3.25876372e-01
  1.29832693e-01  3.61877718e-01 -5.40207806e-01 -9.36953331e-01
 -4.03456610e-02 -7.26745275e-01]
```

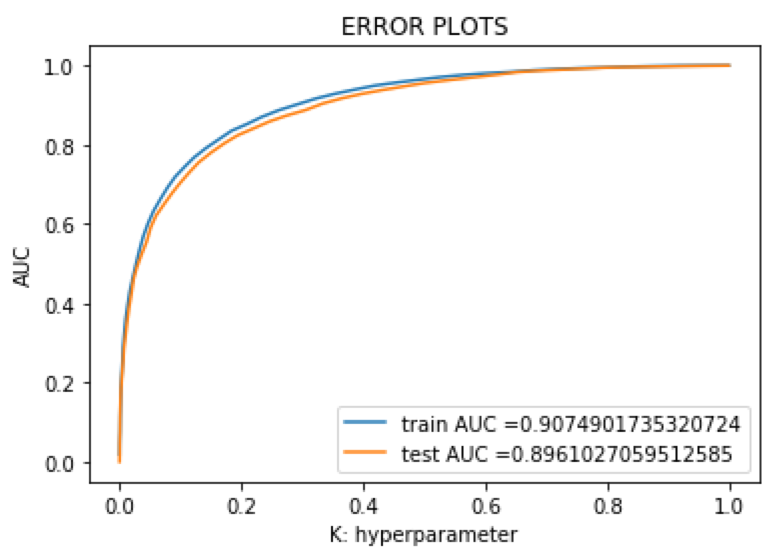
Hyper Parameter Tunning

In [25]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 17,29,41,51,67,81,96,113]}
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']

K = [1, 17,29,41,51,67,81,96,113]


plt.plot(K, train_auc, label='Train AUC')
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alph
plt.plot(K, cv_auc, label='CV AUC')

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



Applying KNN for n=50

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(n_neighbors=50)
neigh.fit(sent_vectors_train, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent_ve
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vecto

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)))
```

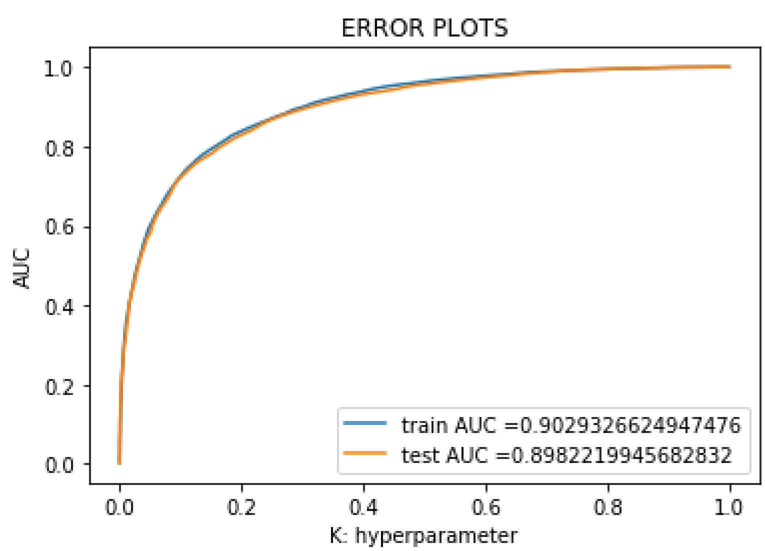



```
================================================================================
====================
Train confusion matrix
[[14572  2409]
 [ 3552 13472]]
Test confusion matrix
[[7128 1268]
 [1830 6523]]
```

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

Applying KNN for n=100

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(n_neighbors=100)
neigh.fit(sent_vectors_train, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent_ve
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vecto

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
[[14471  2510]
 [ 3565 13459]]
Test confusion matrix
[[7112 1284]
 [1808 6545]]
```

Checking For n values of 50, and 100 we do not see much of a diffrence

in terms of Test AUC score this is in accordance with the hyper parameter tunning result where Train AUC and CV AUC error plots are similar in the range for k=40 to 100

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

Vectorizer code refrenced from Previous assignments

In [32]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

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)
```

In [33]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [34]:
```
w2v_words = list(w2v_model.wv.vocab)
```

In [35]:
```
tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████
█| 34005/34005 [07:05<00:00, 79.87it/s]
```

In [36]:
```
print(tfidf_sent_vectors[0])
```

```
[ 0.81067913  0.2112177  -0.13891514 -0.49902968  0.79983206 -0.42251336
 -0.02293323 -0.26061624  0.02929149 -0.00192107 -0.01562334  0.40462623
 -0.06005639 -0.13277111 -0.28161164 -0.14503571  0.10863212 -0.32146697
  0.04591432 -0.29439214  0.52164997  0.20060198  0.49195202 -0.28277213
  0.0929979  -0.40429216  0.11310623 -0.68481343 -0.01609555 -0.04648923
 -0.50385808 -0.430047   -0.05057494  0.13189331 -0.03260945 -0.40887159
 -0.55438698  0.22519873 -0.24616429 -0.98497784 -0.34688246 -0.40446599
  0.12135677 -0.42180905  0.3659215  -0.23468598 -0.7183316   0.37251256
  0.48801083  0.09426283]
```

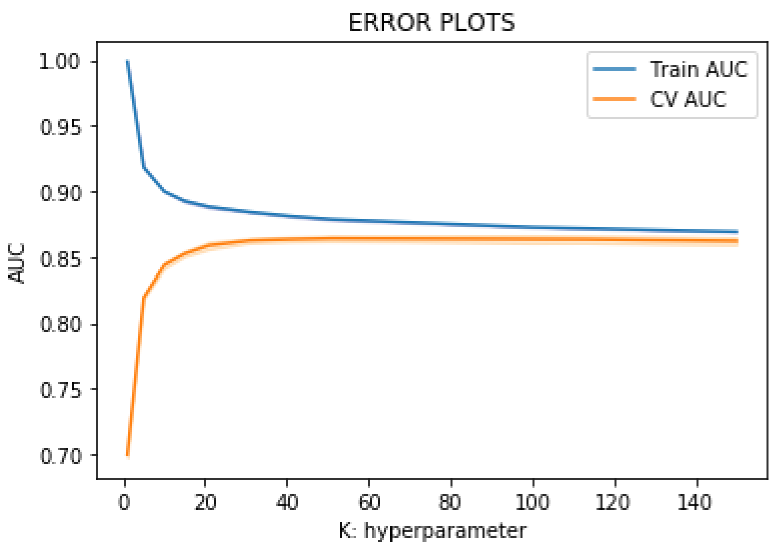
In [37]:
```
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [38]:
```
tfidf_sent_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████████████████
█| 16749/16749 [03:25<00:00, 81.56it/s]
```

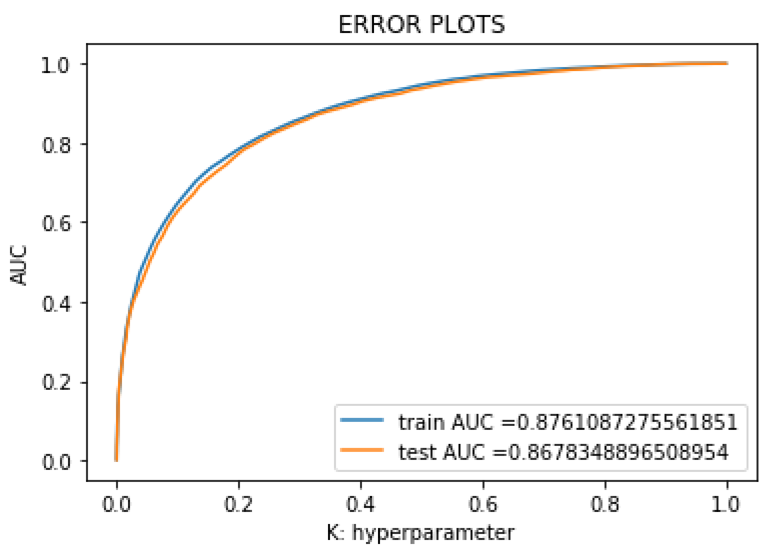
Hyper Parameter Tunning

In [39]:
```
neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,100,113,129,137,150]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(tfidf_sent_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']

K = [1, 5, 10, 15, 21, 31, 41, 51,100,113,129,137,150]


plt.plot(K, train_auc, label='Train AUC')
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alph
plt.plot(K, cv_auc, label='CV AUC')

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



Applying KNN for n=100

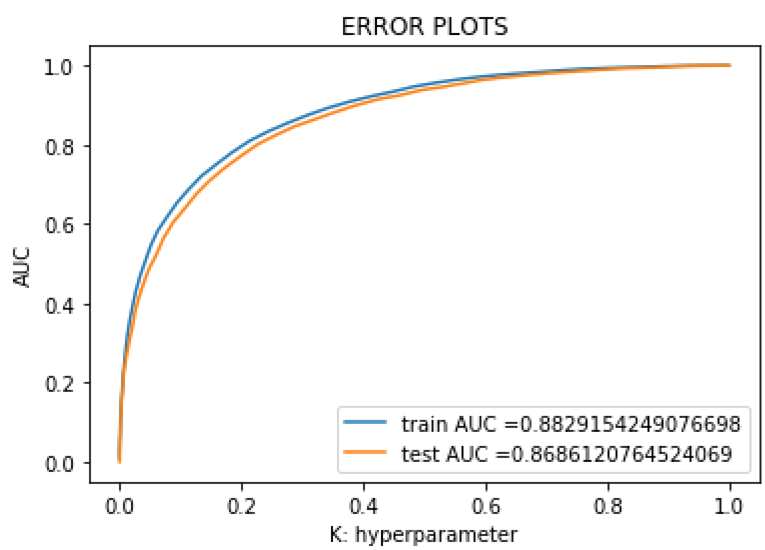
In [40]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(n_neighbors=100)
neigh.fit(tfidf_sent_vectors, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_s
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_sent_

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




```
================================================================================
====================
Train confusion matrix
[[13501  3480]
 [ 3604 13420]]
Test confusion matrix
[[6645 1751]
 [1814 6539]]
```

Applying KNN for n=50

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(n_neighbors=50)
neigh.fit(tfidf_sent_vectors, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_s
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_sent

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




```
================================================================================
====================
Train confusion matrix
[[13853  3128]
 [ 3746 13278]]
Test confusion matrix
[[6723 1673]
 [1909 6444]]
```

Checking For n values of 50, and 100 we do not see much of a diffrence in terms of Test AUC score this is in accordance with the hyper

parameter tunning result where Train AUC and CV AUC error plots are similar in the range for k=40 to 100¶

[5.2] Applying KNN kd-tree

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

Vectorizer code refrenced from prebious assignments

In [43]:
```
vectorizer = CountVectorizer(min_df=10,max_features=500)
vectorizer.fit(X_train)

X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)
```

In [44]:
```
print(type(X_train_bow),X_train_bow.shape)
print(type(X_test_bow),X_test_bow.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'> (34005, 500)
<class 'scipy.sparse.csr.csr_matrix'> (16749, 500)
```

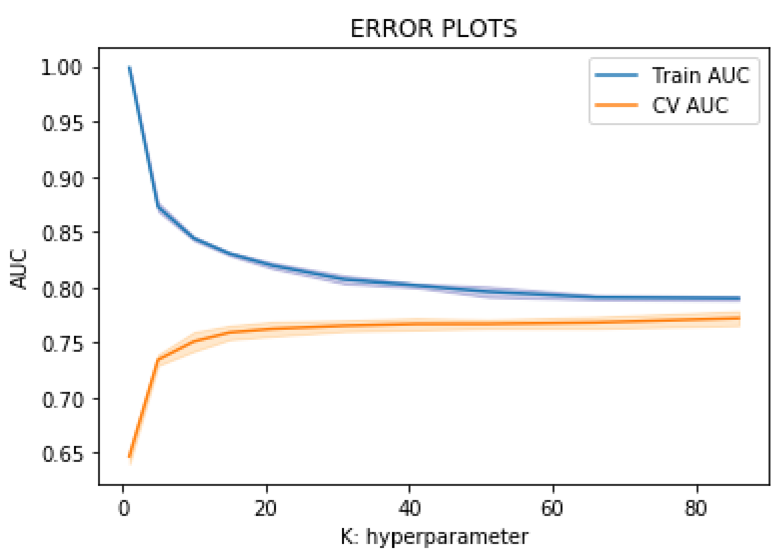
Converting sparse matrix because KD tree implementation of Sparse matrix will be interpreted as Brute force iplementation internally

In [45]:
```
from scipy import sparse
X_train_bow1=sparse.csr_matrix.toarray(X_train_bow)
X_test_bow1=sparse.csr_matrix.toarray(X_test_bow)
print(type(X_train_bow1),X_train_bow1.shape)
print(type(X_test_bow1),X_test_bow1.shape)
```

```
<class 'numpy.ndarray'> (34005, 500)
<class 'numpy.ndarray'> (16749, 500)
```

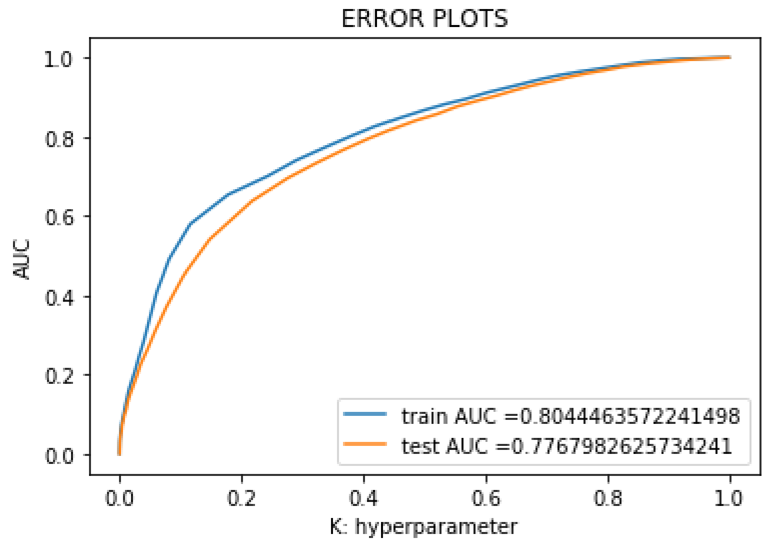
Hyper Parameter Tunning

In [46]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(algorithm="kd_tree")
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,66,86]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_bow1, 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']

K = [1, 5, 10, 15, 21, 31, 41, 51,66,86]


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,alph

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



Applying KNN KD tree for n=50 selectedd from error plots

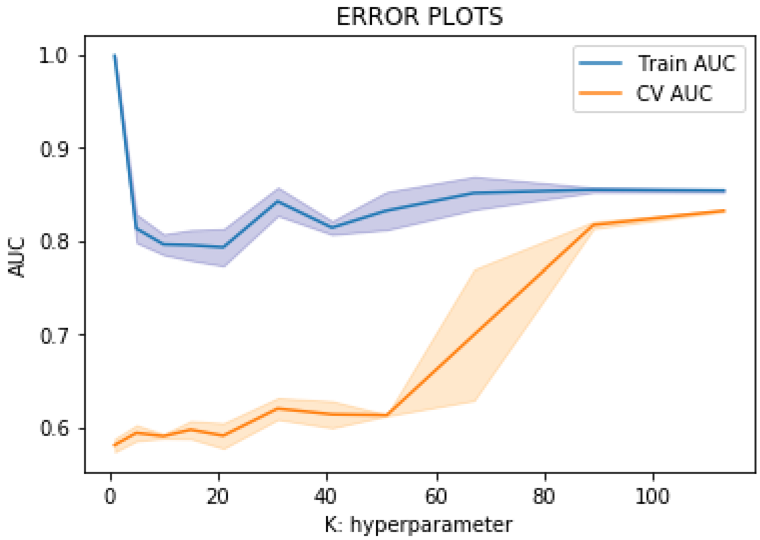
In [47]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix

neigh = KNeighborsClassifier(n_neighbors=50,algorithm='kd_tree')
neigh.fit(X_train_bow1, y_train)

"""y_train_pred = []
for i in range(0, X_train_bow1.shape[0],100):
    y_train_pred.extend(neigh.predict_proba(X_train_bow1[i:i+100,:])[:,1])

Y_test_pred = []
for i in range(0, X_test_bow1.shape[0],2):
    Y_test_pred.extend(neigh.predict_proba(X_test_bow1[i:i+2,:])[:,1])"""

#print(len(y_train_pred))
#print(len(Y_test_pred))




#train_fpr, train_tpr, thresholds = roc_curve(y_train,y_train_pred)
#test_fpr, test_tpr, thresholds = roc_curve(y_test,Y_test_pred)

train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh.predict_proba(X_train_
test_fpr, test_tpr, thresholds = roc_curve(y_test,neigh.predict_proba(X_test_bow1

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("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow1)))
```




```
Train confusion matrix
[[13909  3016]
 [ 5913 11167]]
Test confusion matrix
[[6614 1838]
 [3007 5290]]
```

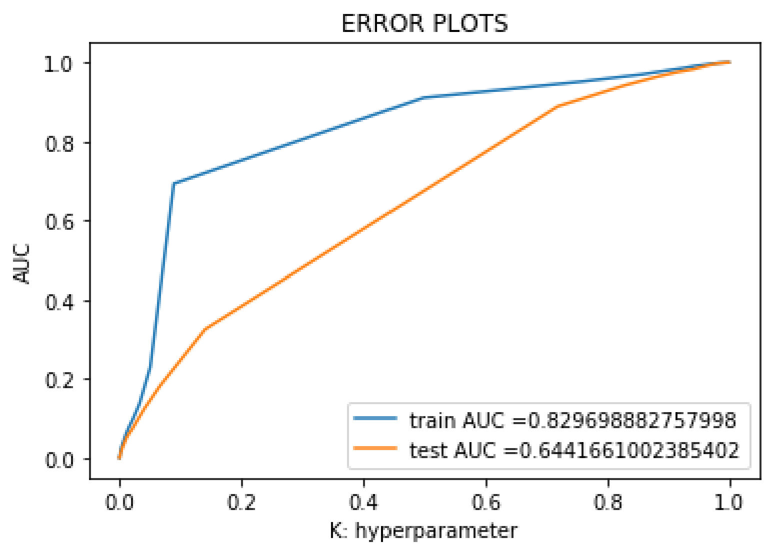
[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [38]:
```
from scipy import sparse
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
tf_idf_vect.fit(X_train)
X_train_tfidf=tf_idf_vect.transform(X_train)
X_test_tfidf=tf_idf_vect.transform(X_test)


X_train_tfidf1=sparse.csr_matrix.toarray(X_train_tfidf)
X_test_tfidf1=sparse.csr_matrix.toarray(X_test_tfidf)


neigh = KNeighborsClassifier(algorithm='kd_tree')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,67,89,113]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf1, 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']

K =[1, 5, 10, 15, 21, 31, 41, 51,67,89,113]

plt.plot(K, train_auc, label='Train AUC')
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alph
plt.plot(K, cv_auc, label='CV AUC')

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



Applying KNN KD tree for n=50

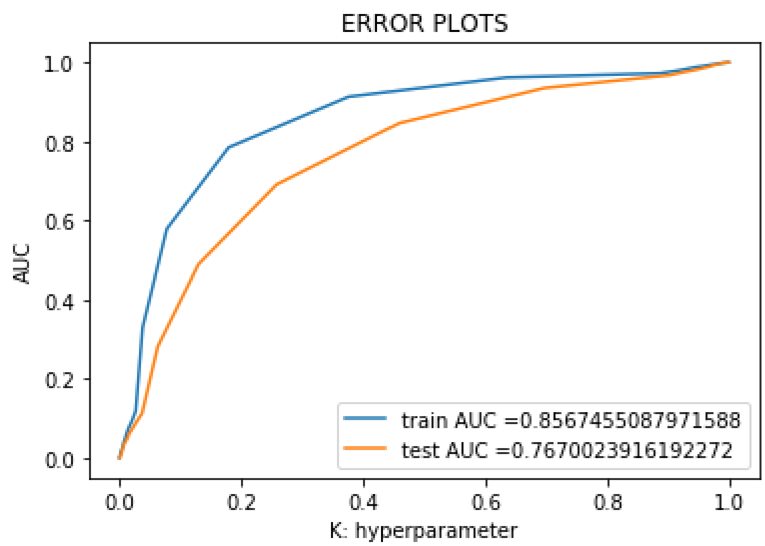
In [40]:
```
neigh = KNeighborsClassifier(n_neighbors=50,algorithm='kd_tree')
neigh.fit(X_train_tfidf1, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh.predict_proba(X_train_
test_fpr, test_tpr, thresholds = roc_curve(y_test,neigh.predict_proba(X_test_tfid

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("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfidf1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf1)))
```




```
Train confusion matrix
[[15585  1525]
 [ 5185 11710]]
Test confusion matrix
[[7105 1162]
 [5720 2762]]
```

Applying KNN KD tree for n=50

In [42]:
```
neigh1 = KNeighborsClassifier(n_neighbors=100,algorithm='kd_tree')
neigh1.fit(X_train_tfidf1, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train,neigh1.predict_proba(X_train
test_fpr, test_tpr, thresholds = roc_curve(y_test,neigh1.predict_proba(X_test_tfi

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("Train confusion matrix")
print(confusion_matrix(y_train, neigh1.predict(X_train_tfidf1)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh1.predict(X_test_tfidf1)))
```




```
Train confusion matrix
[[10671  6439]
 [ 1467 15428]]
Test confusion matrix
[[4471 3796]
 [1303 7179]]
```

Checking For n values of 50 and 100 we get the best test AUC for n=100 this is in accordance with the hyper parameter tunning result where CV AUC is high and also the dist between CV AUC and Train AUC is less for n=100

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

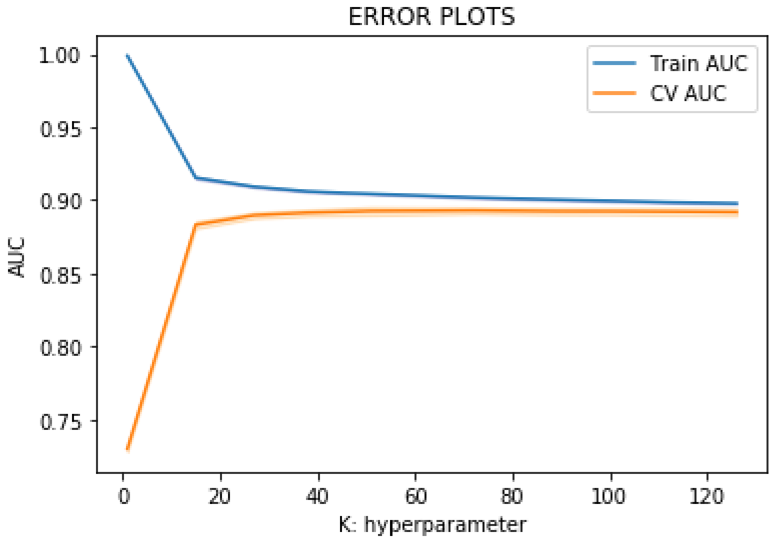
Vectorizer code refrenced from previous asignments

In [46]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
```

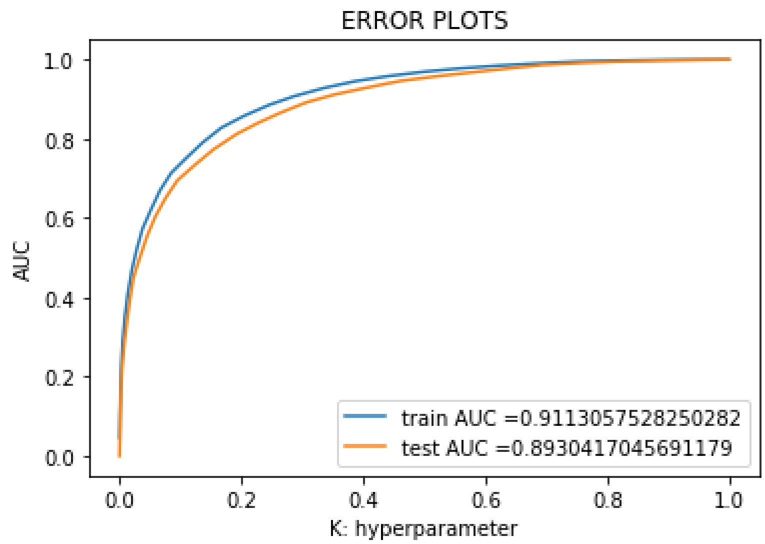
In [47]:
```
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])


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

print(type(list_of_sentance_test[0]))
sent_vectors_test = [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])



neigh = KNeighborsClassifier(algorithm='kd_tree')
parameters = {'n_neighbors':[1,15, 27, 38, 46, 51,72,89,100,113,126]}
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']
```

```
K = [1,15, 27, 38, 46, 51,72,89,100,113,126]


plt.plot(K, train_auc, label='Train AUC')
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alph
plt.plot(K, cv_auc, label='CV AUC')

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

```
100%|██████████████████████████████████████████████████
█| 34005/34005 [00:58<00:00, 580.43it/s]

(34005, 50)
[ 0.50586602  0.3061913  -0.08240696 -0.50578571  0.49237934 -0.55960288
  0.07193047 -0.16187727  0.36533053 -0.22463803  0.03311529  0.18101004
 -0.20580879  0.01221181 -0.05313694 -0.19886963  0.06438196  0.06688096
  0.20019419 -0.06500789  0.54308846  0.00464618  0.16405928 -0.10144309
  0.21782434 -0.51869821  0.23597123 -0.68866324  0.07655553 -0.19367655
 -0.34406461 -0.41421708 -0.33860347  0.39380102 -0.11631725 -0.23547268
 -0.79801995  0.59040077 -0.31856142 -1.03047795 -0.19925957 -0.62826622
  0.08085848 -0.17309224  0.55994121 -0.11338763 -0.67628629  0.16155288
  0.50261697  0.00940539]
<class 'list'>

100%|██████████████████████████████████████████████████
█| 16749/16749 [00:28<00:00, 581.67it/s]

(16749, 50)
[-0.16393338  0.13735049 -0.84379346 -0.63234403  0.60764675 -0.3421442
  0.63146576 -0.16803266  0.34924566 -0.81239995 -0.29901268 -0.36243477
 -0.24479751 -0.93964417 -0.21212988 -0.50596931  0.01885001 -0.53835658
  0.6008047  -0.91906296  0.30371708 -0.15578093 -0.28023856 -0.4180097
 -0.19505965 -0.43768684 -0.33160493 -0.39825683 -0.12457685 -0.23016674
  0.66667309  0.23535212 -0.67365746 -0.10840427 -0.7179515  -0.71012305
 -0.55379123  0.96933477 -0.37119959 -0.68444496 -0.65664589 -0.09328728
 -0.16575895  0.14669829  0.29213838  0.34844925 -0.7375138  -0.68735491
  0.08297015 -0.65553044]
```

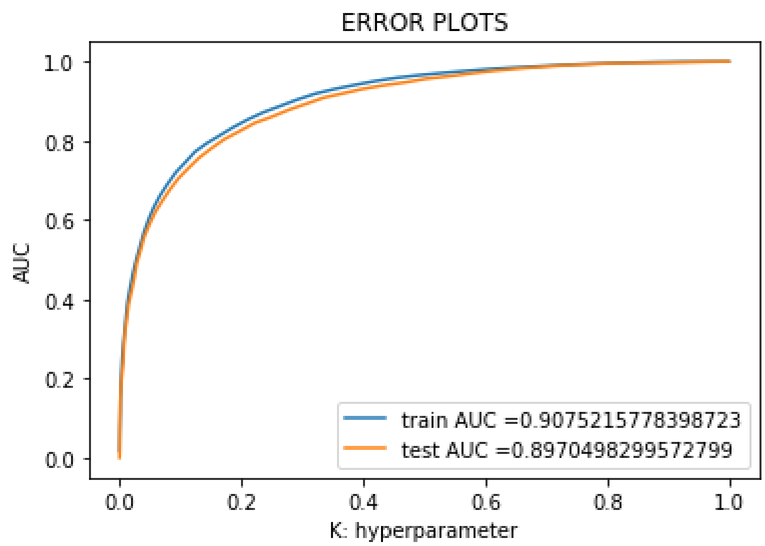

ERROR PLOTS
Train AUC
CV AUC
AUC
1.00
0.95
0.90
0.85
0.80
0.75
0
20
40
60
80
100
120
K: hyperparameter

In [49]:
```
neigh = KNeighborsClassifier(n_neighbors=30,algorithm='kd_tree')
neigh.fit(sent_vectors_train, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent_ve
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vecto

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)


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
[[14664  2317]
 [ 3571 13453]]
Test confusion matrix
[[7104 1292]
 [1901 6452]]
```

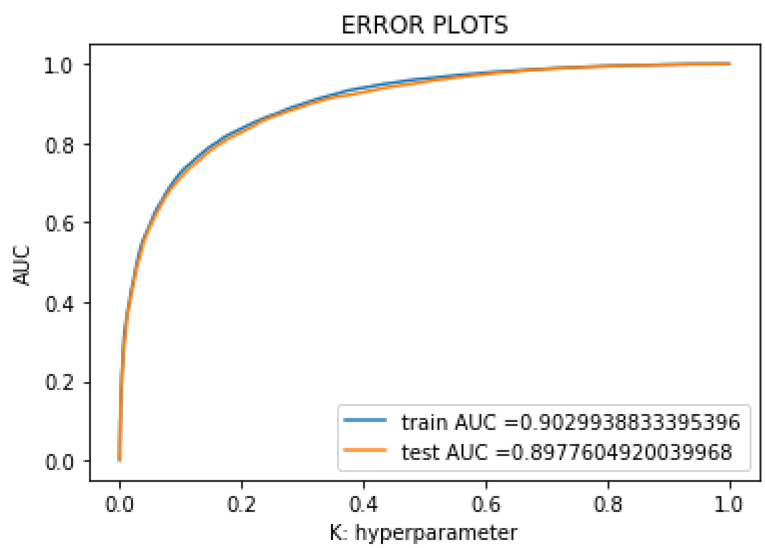
In [30]:
```
neigh = KNeighborsClassifier(n_neighbors=50,algorithm='kd_tree')
neigh.fit(sent_vectors_train, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent_ve
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vecto

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)


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
[[14549  2432]
 [ 3516 13508]]
Test confusion matrix
[[7132 1264]
 [1835 6518]]
```

In [31]:

```
neigh = KNeighborsClassifier(n_neighbors=100,algorithm='kd_tree')
neigh.fit(sent_vectors_train, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent_ve
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_vecto

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)


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
[[14464  2517]
 [ 3561 13463]]
Test confusion matrix
[[7118 1278]
 [1794 6559]]
```

Checking For n values of 30,50 and 100 we do not see much of a diffrence in terms of Test AUC score this is in accordance with the hyper parameter tunning result where Train AUC and CV AUC error plots are similar in the range for k=40 to 100

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

Vectorizer code refrenced from previous assignments

In [42]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

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)
model = TfidfVectorizer(max_features=500)
tf_idf_matrix = model.fit_transform(X_train)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
w2v_words = list(w2v_model.wv.vocab)


tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1

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

tfidf_sent_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

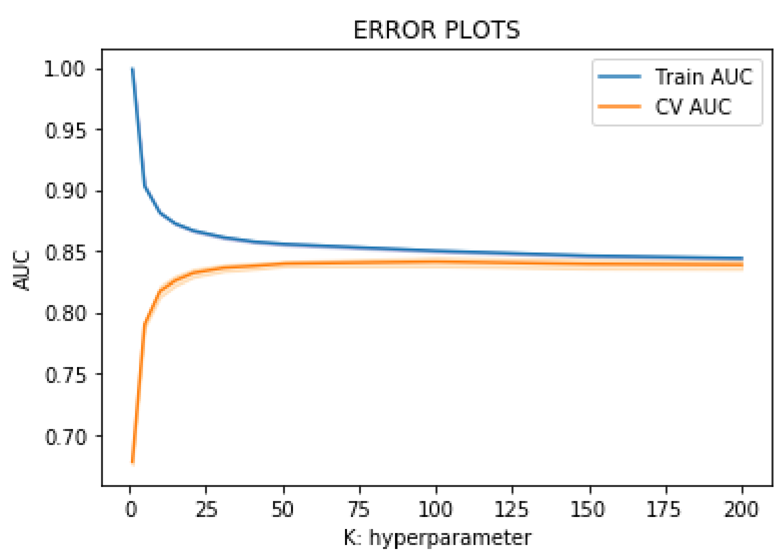
```
100%|██████████████████████████████
```

```
█| 34005/34005 [01:25<00:00, 398.21it/s]
100%|████████████████████████████████████████
█| 16749/16749 [00:46<00:00, 356.53it/s]
```

Hyper Parameter Tunning

In [43]:

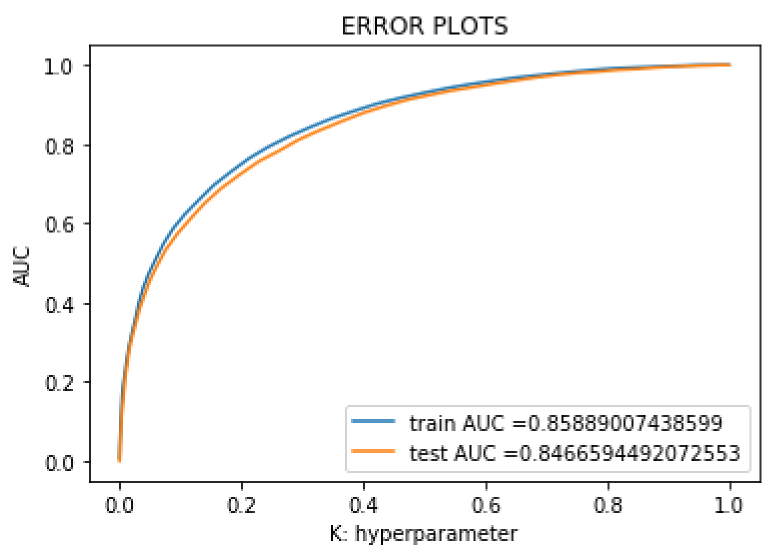
```python
neigh = KNeighborsClassifier(algorithm='kd_tree')
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,100,150,200]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(tfidf_sent_vectors, y_train)

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']

K = [1, 5, 10, 15, 21, 31, 41, 51,100,150,200]


plt.plot(K, train_auc, label='Train AUC')
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alph
plt.plot(K, cv_auc, label='CV AUC')

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



Applying KNN for n=50

In [44]:

```
neigh = KNeighborsClassifier(n_neighbors=50,algorithm='kd_tree')
neigh.fit(tfidf_sent_vectors, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_s
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_sent


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

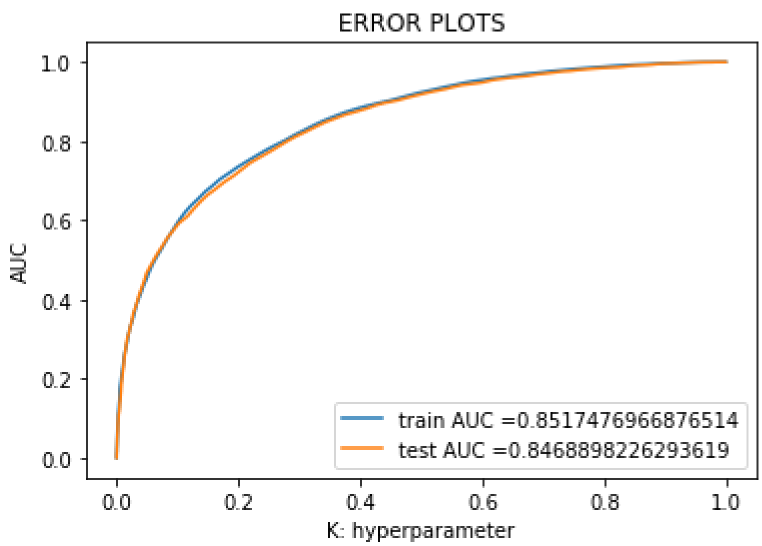



```
================================================================================
====================
Train confusion matrix
[[13372  3609]
 [ 4024 13000]]
Test confusion matrix
[[6468 1928]
 [2030 6323]]
```

Applying KNN for n=125

In [45]:

```
neigh = KNeighborsClassifier(n_neighbors=125,algorithm='kd_tree')
neigh.fit(tfidf_sent_vectors, y_train)


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfidf_s
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_sent_


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




```
================================================================================
====================
Train confusion matrix
[[12849  4132]
 [ 3824 13200]]
Test confusion matrix
[[6287 2109]
 [1899 6454]]
```

Checking For n values of 50 and 125 we do not see much of a diffrence in terms of Test AUC score this is in accordance with the hyper parameter tunning result where Train AUC and CV AUC error plots are

similar in the range for k=50 to 200

[6] Conclusions

In [50]:

```
from prettytable import PrettyTable


x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["BOW","Brute","40","0.684"])
x.add_row(["TFIDF","Brute","100","0.835"])
x.add_row(["avgW2V","Brute","50","0.896"])
x.add_row(["TFIDF W2V","Brute","50","0.868"])
x.add_row(["========","========","========","========"])
x.add_row(["","","",""])
x.add_row(["========","========","========","========"])
x.add_row(["BOW","KDTree","50","0.776"])
x.add_row(["TFIDF","KDTree","100","0.767"])
x.add_row(["avgW2V","KDTree","30","0.8930"])
x.add_row(["TFIDF W2V","KDTree","50","0.846"])


print(x)
```

```
+------------+----------+-----------------+----------+
| Vectorizer |  Model   | Hyper Parameter |   AUC    |
+------------+----------+-----------------+----------+
|    BOW     |  Brute   |        40       |  0.684   |
|   TFIDF    |  Brute   |       100       |  0.835   |
|   avgW2V   |  Brute   |        50       |  0.896   |
| TFIDF W2V  |  Brute   |        50       |  0.868   |
|  ========  | ======== |     ========    | ======== |
|            |          |                 |          |
|  ========  | ======== |     ========    | ======== |
|    BOW     |  KDTree  |        50       |  0.776   |
|   TFIDF    |  KDTree  |       100       |  0.767   |
|   avgW2V   |  KDTree  |        30       |  0.8930  |
| TFIDF W2V  |  KDTree  |        50       |  0.846   |
+------------+----------+-----------------+----------+
```