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 [0]:

```
!pip install kaggle
```

In [0]:

```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[0]:

```
{u'kaggle.json': '{"username":"shamimio","key":"8e71fd37342f4e78691bac8a27
26efc2"}'}
```

In [0]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

#changeb the permmission
!chmod 600 ~/.kaggle/kaggle.json
```

In [0]:

```
!kaggle datasets download -d snap/amazon-fine-food-reviews
```

```
Downloading amazon-fine-food-reviews.zip to /content
 96% 241M/251M [00:02<00:00, 138MB/s]
100% 251M/251M [00:02<00:00, 124MB/s]
```

In [0]:

```
from zipfile import ZipFile
file_name = "amazon-fine-food-reviews.zip"

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print("Done")
```

```
Done
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('../Logistic regression/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 50
0000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, 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 (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	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	C
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	
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recomme to try gre tea extra ...

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	Help
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(364173, 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]:

```
69.25890143662969
```

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	Help
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [12]:

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

In [13]:

```
#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()
```

```
(364171, 10)
```

Out[13]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

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

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
==================================================
I was really looking forward to these pods based on the reviews.  Starbuck
s is good, but I prefer bolder taste.... imagine my surprise when I ordere
d 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I
admit that Amazon agreed to credit me for cost plus part of shipping, but
geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe t
hat carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chick
en broth, the only thing I do not think belongs in it is Canola oil. Canol
a or rapeseed is not someting a dog would ever find in nature and if it di
d find rapeseed in nature and eat it, it would poison them. Today's Food i
ndustries have convinced the masses that Canola oil is a safe and even bet
ter oil than olive or virgin coconut, facts though say otherwise. Until th
e late 70's it was poisonous until they figured out a way to fix that. I s
till like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch t
he excellence of this product.<br /><br />Thick, delicious.  Perfect.  3 i
ngredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.
No garbage.<br /><br />Have numerous friends & family members hooked on th
is stuff.  My husband & son, who do NOT like "sugar free" prefer this over
major label regular syrup.<br /><br />I use this as my SWEETENER in bakin
g: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably
delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [15]:

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

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
```

In [16]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
==================================================
I was really looking forward to these pods based on the reviews.  Starbuck
s is good, but I prefer bolder taste.... imagine my surprise when I ordere
d 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I
admit that Amazon agreed to credit me for cost plus part of shipping, but
geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe t
hat carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chick
en broth, the only thing I do not think belongs in it is Canola oil. Canol
a or rapeseed is not someting a dog would ever find in nature and if it di
d find rapeseed in nature and eat it, it would poison them. Today's Food i
ndustries have convinced the masses that Canola oil is a safe and even bet
ter oil than olive or virgin coconut, facts though say otherwise. Until th
e late 70's it was poisonous until they figured out a way to fix that. I s
till like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch t
he excellence of this product.Thick, delicious.  Perfect.  3 ingredients:
Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbag
e.Have numerous friends & family members hooked on this stuff.  My husband
& son, who do NOT like "sugar free" prefer this over major label regular s
yrup.I use this as my SWEETENER in baking: cheesecakes, white brownies, mu
ffins, pumpkin pies, etc... Unbelievably delicious...Can you tell I like i
t? :)
```

In [17]:

```
# 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)
```

```
Great ingredients although, chicken should have been 1st rather than chick
en broth, the only thing I do not think belongs in it is Canola oil. Canol
a or rapeseed is not someting a dog would ever find in nature and if it di
d find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even be
tter oil than olive or virgin coconut, facts though say otherwise. Until t
he late 70 is it was poisonous until they figured out a way to fix that. I
still like it but it could be better.
==================================================
```

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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
```

In [20]:

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

```
Great ingredients although chicken should have been 1st rather than chicke
n broth the only thing I do not think belongs in it is Canola oil Canola o
r rapeseed is not someting a dog would ever find in nature and if it did f
ind rapeseed in nature and eat it it would poison them Today is Food indus
tries have convinced the masses that Canola oil is a safe and even better
oil than olive or virgin coconut facts though say otherwise Until the late
70 is it was poisonous until they figured out a way to fix that I still li
ke it but it could be better
```

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 step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
s', 'you', "you'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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            '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 stopwor
ds)
    preprocessed_reviews.append(sentance.strip())
```

100%|██████████| 364171/364171 [03:00<00:00, 2014.54
it/s]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great ingredients although chicken rather chicken broth thing not think b
elongs canola oil canola rapeseed not someting dog would ever find nature
find rapeseed nature eat would poison today food industries convinced mass
es canola oil safe even better oil olive virgin coconut facts though say o
therwise late poisonous figured way fix still like could better'
```

[3.2] Preprocessing Review Summary

In [0]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[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=5
00)
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_featu
res=500)
tf_idf_vect.fit(preprocessed_reviews)
```

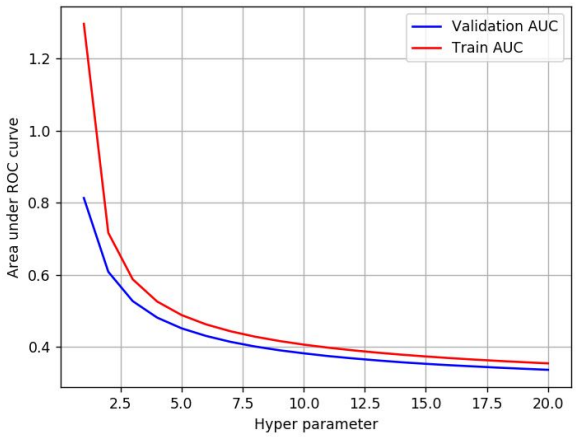
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

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

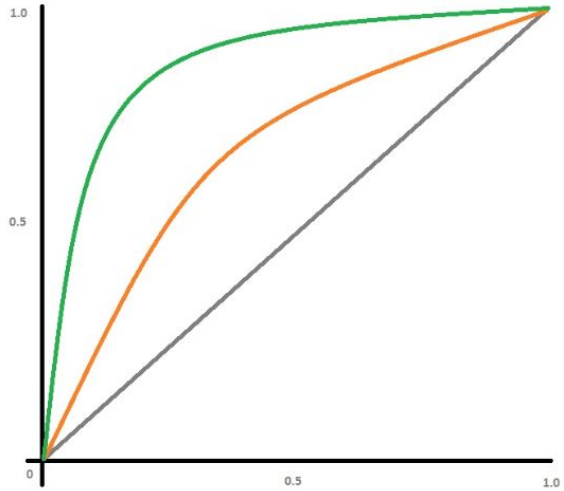
   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - 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/) 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/)

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)

[5.1] Applying KNN brute force

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

In [26]:

```
# Please write all the code with proper documentation
import warnings
warnings.filterwarnings("ignore")
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
#from sklearn.model_selection import cross_validation
```

In [0]:

```
df = pd.DataFrame({'Text':preprocessed_reviews})
X = df['Text'][:100000].values
y = final['Score'][:100000].values
```

In [0]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is ran
dom splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # thi
s is random splitting
```

In [0]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(44890,) (44890,)
(22110,) (22110,)
(33000,) (33000,)
```

In [0]:

```
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)
```

```
After vectorizations
(44890, 40685) (44890,)
(22110, 40685) (22110,)
(33000, 40685) (33000,)
```

Simple cross validation

In [0]:

```
for i in tqdm(range(1,30,2)):

  knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  # fitting the model on crossvalidation train
  knn.fit(X_train_bow, y_train)
  pred = knn.predict(X_cv_bow)
  # evaluate CV accuracy
  acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
  print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

```
  7%|▊          | 1/15 [00:44<10:25, 44.66s/it]
CV accuracy for k = 1 is 80%
 13%|█▎         | 2/15 [01:31<09:48, 45.26s/it]
CV accuracy for k = 3 is 84%
 20%|██         | 3/15 [02:26<09:39, 48.26s/it]
CV accuracy for k = 5 is 84%
 27%|██▋        | 4/15 [03:21<09:14, 50.38s/it]
CV accuracy for k = 7 is 85%
 33%|███▎       | 5/15 [04:17<08:40, 52.02s/it]
CV accuracy for k = 9 is 85%
 40%|████       | 6/15 [05:12<07:55, 52.82s/it]
CV accuracy for k = 11 is 85%
 47%|████▋      | 7/15 [06:07<07:07, 53.39s/it]
CV accuracy for k = 13 is 85%
 53%|█████▎     | 8/15 [07:01<06:16, 53.76s/it]
CV accuracy for k = 15 is 85%
 60%|██████     | 9/15 [07:56<05:24, 54.15s/it]
CV accuracy for k = 17 is 85%
 67%|██████▋    | 10/15 [08:51<04:31, 54.38s/it]
CV accuracy for k = 19 is 85%
 73%|███████▎   | 11/15 [09:46<03:37, 54.36s/it]
CV accuracy for k = 21 is 85%
 80%|████████   | 12/15 [10:40<02:43, 54.40s/it]
CV accuracy for k = 23 is 85%
 87%|████████▋  | 13/15 [11:34<01:48, 54.35s/it]
CV accuracy for k = 25 is 85%
 93%|█████████▎ | 14/15 [12:28<00:54, 54.27s/it]
CV accuracy for k = 27 is 85%
100%|██████████| 15/15 [13:23<00:00, 54.23s/it]
CV accuracy for k = 29 is 85%
```

In [0]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []

for i in range(1,30,2):
  neigh = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  neigh.fit(X_train_bow, y_train)
  #y_train_pred = []
  #for i in range(0, X_train.shape[0], 1000):
  # y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000])[:,1])
  #y_cv_pred = []
  #for i in range(0, X_cv.shape[0], 1000):
  # y_cv_pred.extend(neigh.predict_proba(X_cv_bow[i:i+1000])[:,1])
  y_train_pred = neigh.predict_proba(X_train_bow)[:,1]
  y_cv_pred = neigh.predict_proba(X_cv_bow)[:,1]
  train_auc.append(roc_auc_score(y_train,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

In [0]:

```
k = np.arange(1,30,2)
k
```

Out[0]:

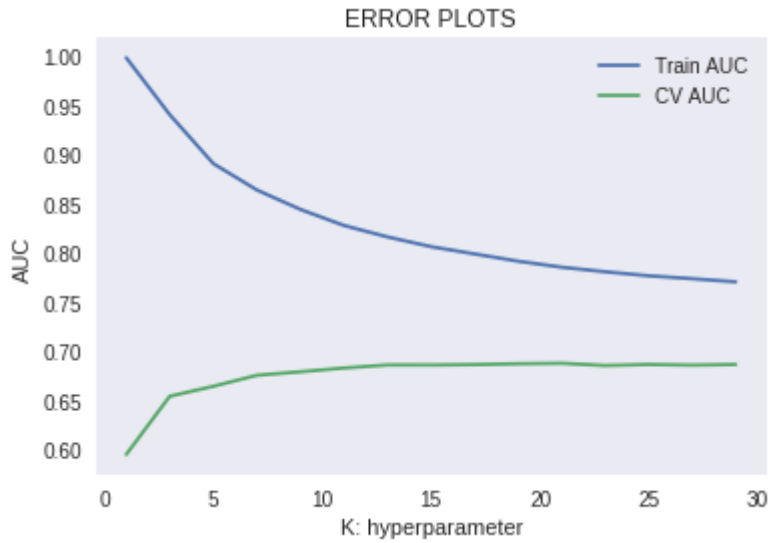
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29])
```

In [0]:

```
K = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[0]:

```
Text(0.5, 1.0, 'ERROR PLOTS')
```

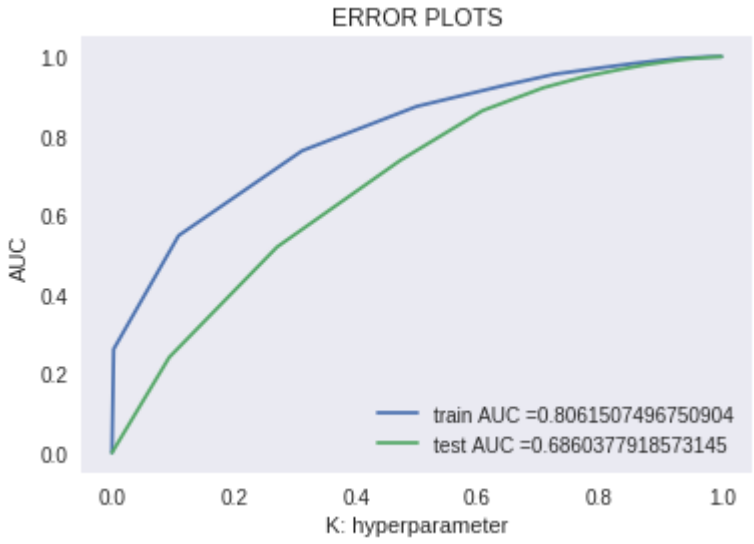


Best value of k is 15

Testing the data:-

In [0]:

```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=15, algorithm='brute')
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1
])
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
print("="*100)


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




```
====================================================================================
==========================
Train confusion matrix
[[  969  5601]
 [  687 37633]]
Test confusion matrix
[[  629  4328]
 [  595 27448]]
```

In [0]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.860
969
5601
687
37633
True label
0
1
0
1
Predicted label
32000
24000
16000
8000

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

In [0]:

```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train) # fit has to happen only on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_tf = tf_idf_vect.transform(X_train)
X_cv_tf = tf_idf_vect.transform(X_cv)
X_test_tf = tf_idf_vect.transform(X_test)
print("After vectorizations")
print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)
```

```
After vectorizations
((44890, 25406), (44890,))
((22110, 25406), (22110,))
((33000, 25406), (33000,))
```

Hyperparameter tuning

In [0]:

```
for i in tqdm(range(1,30,2)):

  knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  # fitting the model on crossvalidation train
  knn.fit(X_train_tf, y_train)
  pred = knn.predict(X_cv_tf)
  # evaluate CV accuracy
  acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
  print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

```
  7%|▉          | 1/15 [00:44<10:28, 44.90s/it]
CV accuracy for k = 1 is 85%
 13%|█▎         | 2/15 [01:31<09:51, 45.51s/it]
CV accuracy for k = 3 is 85%
 20%|██         | 3/15 [02:26<09:38, 48.23s/it]
CV accuracy for k = 5 is 84%
 27%|██▋        | 4/15 [03:20<09:10, 50.03s/it]
CV accuracy for k = 7 is 84%
 33%|███▎       | 5/15 [04:14<08:31, 51.16s/it]
CV accuracy for k = 9 is 85%
 40%|████       | 6/15 [05:08<07:49, 52.14s/it]
CV accuracy for k = 11 is 85%
 47%|████▋      | 7/15 [06:02<07:01, 52.70s/it]
CV accuracy for k = 13 is 85%
 53%|█████▎     | 8/15 [06:56<06:10, 52.97s/it]
CV accuracy for k = 15 is 85%
 60%|██████     | 9/15 [07:50<05:19, 53.25s/it]
CV accuracy for k = 17 is 85%
 67%|██████▋    | 10/15 [08:43<04:25, 53.17s/it]
CV accuracy for k = 19 is 85%
 73%|███████▎   | 11/15 [09:36<03:32, 53.15s/it]
CV accuracy for k = 21 is 85%
 80%|████████   | 12/15 [10:30<02:40, 53.38s/it]
CV accuracy for k = 23 is 85%
 87%|████████▋  | 13/15 [11:25<01:47, 53.76s/it]
CV accuracy for k = 25 is 85%
 93%|█████████▎ | 14/15 [12:20<00:54, 54.27s/it]
CV accuracy for k = 27 is 85%
100%|██████████| 15/15 [13:15<00:00, 54.63s/it]
CV accuracy for k = 29 is 85%
```

In [0]:

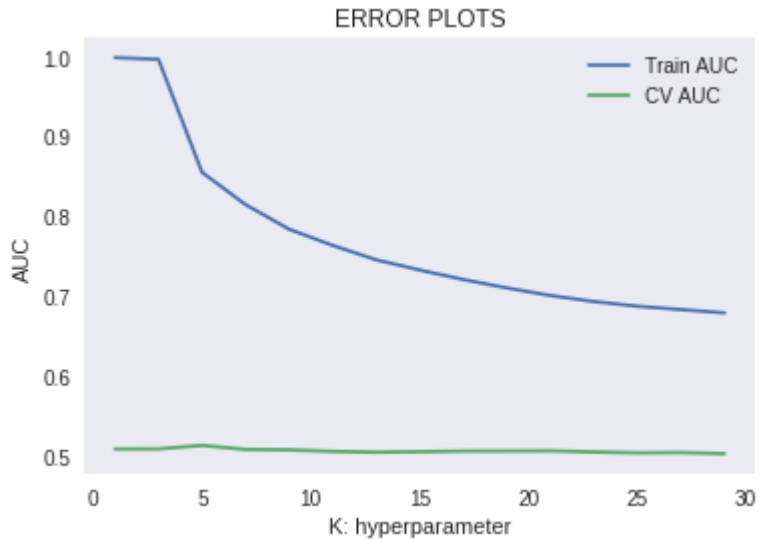
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []

k = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
for i in k:
  neigh = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  neigh.fit(X_train_tf, y_train)
  #y_train_pred = []
  #for i in range(0, X_train.shape[0], 1000):
  # y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000])[:,1])
  #y_cv_pred = []
  #for i in range(0, X_cv.shape[0], 1000):
  # y_cv_pred.extend(neigh.predict_proba(X_cv_bow[i:i+1000])[:,1])
  y_train_pred = neigh.predict_proba(X_train_tf)[:,1]
  y_cv_pred = neigh.predict_proba(X_cv_tf)[:,1]
  train_auc.append(roc_auc_score(y_train,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

In [0]:

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

Out[0]:

```
Text(0.5, 1.0, 'ERROR PLOTS')
```

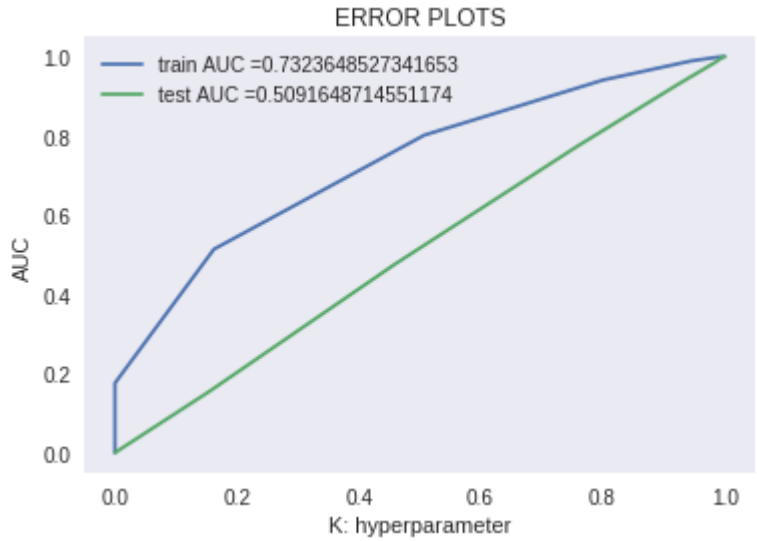


Best K = 15

Testing :-

In [0]:

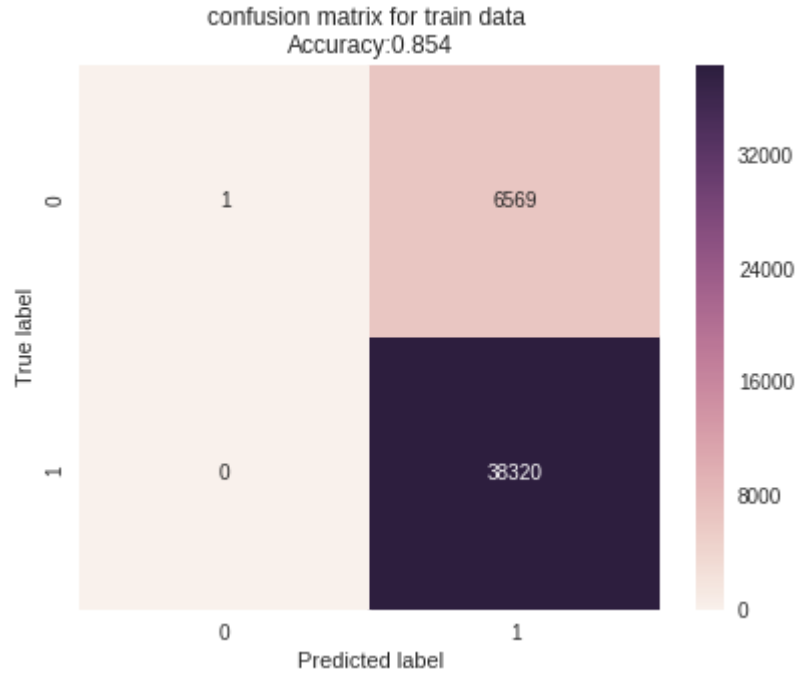
```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=15, algorithm='brute')
neigh.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tf)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tf)[:,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.grid()
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_tf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tf)))
```

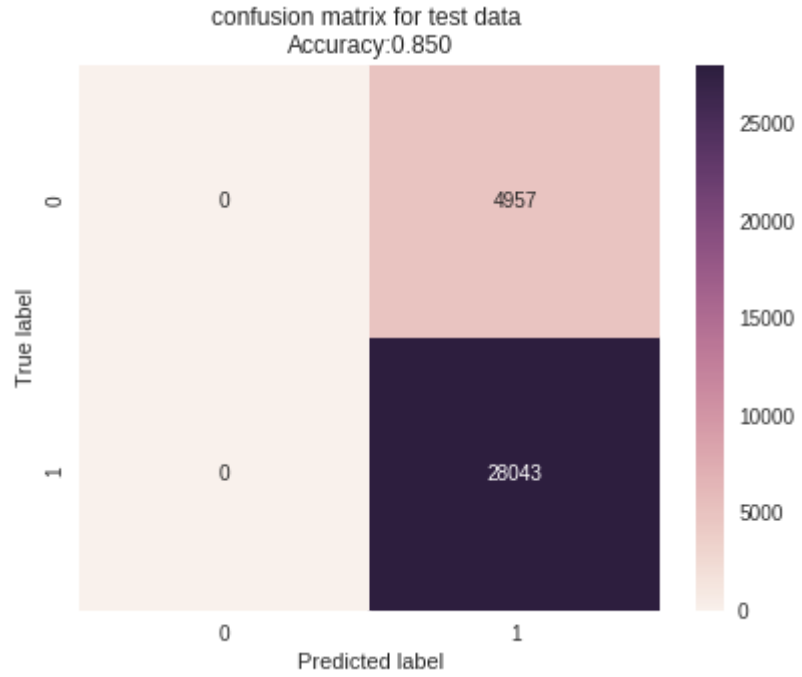



```
====================================================================================
==========================
Train confusion matrix
[[    1  6569]
 [    0 38320]]
Test confusion matrix
[[    0  4957]
 [    0 28043]]
```

In [0]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```





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

In [35]:

```
# Please write all the code with proper documentation
i=0
list_of_sentance=[]
for sentance in X_train:
      list_of_sentance.append(sentance.split())
```

In [37]:

```
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, size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
n', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
train your own w2v ")
```

```
[('good', 0.8653942942619324), ('excellent', 0.8313211798667908), ('wonder
ful', 0.8212835192680359), ('fantastic', 0.752105712890625), ('awesome',
0.7410129308700562), ('amazing', 0.734950840473175), ('delicious', 0.69350
98767280579), ('outstanding', 0.682363748550415), ('tasty', 0.661821186542
511), ('perfect', 0.6600908637046814)]
==================================================
[('nicest', 0.8918052911758423), ('finest', 0.8726404309272766), ('riches
t', 0.8524729013442993), ('favorites', 0.8465124368667603), ('eaten', 0.84
05107855796814), ('closest', 0.8393636345863342), ('tastiest', 0.831527650
3562927), ('humble', 0.8303540945053101), ('ive', 0.8290234804153442), ('s
moothest', 0.8284608125686646)]
```

In [38]:

```
w2v_words = list(w2v_model.wv.vocab)
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_aw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sentance: # for each review/sentence
  sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to ch
ange 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
  X_train_aw2v.append(sent_vec)
```

In [0]:

```
i=0
X_cv_w2v = []
for sentance in X_cv:
  X_cv_w2v.append(sentance.split())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
X_cv_aw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X_cv_w2v: # for each review/sentence
  sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to ch
ange 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
  X_cv_aw2v.append(sent_vec)
```

In [0]:

```
i=0
X_test_w2v = []
for sentance in X_test:
  X_test_w2v.append(sentance.split())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_aw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X_test_w2v: # for each review/sentence
  sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to ch
ange 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
  X_test_aw2v.append(sent_vec)
```

In [0]:

```
print(len(X_train_aw2v), y_train.shape)
print(len(X_cv_aw2v), y_cv.shape)
print(len(X_test_aw2v), y_test.shape)
```

```
44890 (44890,)
22110 (22110,)
33000 (33000,)
```

Simple Cross Validation:-

In [0]:

```
for i in tqdm(range(1,30,2)):

  knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  # fitting the model on crossvalidation train
  knn.fit(X_train_aw2v, y_train)
  pred = knn.predict(X_cv_aw2v)
  # evaluate CV accuracy
  acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
  print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

```
  7%|▉          | 1/15 [00:18<04:15, 18.28s/it]
CV accuracy for k = 1 is 83%
 13%|█▎         | 2/15 [00:38<04:06, 18.95s/it]
CV accuracy for k = 3 is 86%
 20%|██         | 3/15 [01:03<04:08, 20.69s/it]
CV accuracy for k = 5 is 87%
 27%|██▋        | 4/15 [01:28<04:01, 21.95s/it]
CV accuracy for k = 7 is 87%
 33%|███▎       | 5/15 [01:53<03:48, 22.83s/it]
CV accuracy for k = 9 is 88%
 40%|████       | 6/15 [02:18<03:30, 23.42s/it]
CV accuracy for k = 11 is 88%
 47%|████▋      | 7/15 [02:42<03:10, 23.85s/it]
CV accuracy for k = 13 is 88%
 53%|█████▎     | 8/15 [03:07<02:48, 24.07s/it]
CV accuracy for k = 15 is 87%
 60%|██████     | 9/15 [03:32<02:25, 24.32s/it]
CV accuracy for k = 17 is 87%
 67%|██████▋    | 10/15 [03:57<02:02, 24.46s/it]
CV accuracy for k = 19 is 87%
 73%|███████▎   | 11/15 [04:22<01:38, 24.60s/it]
CV accuracy for k = 21 is 87%
 80%|████████   | 12/15 [04:47<01:14, 24.70s/it]
CV accuracy for k = 23 is 87%
 87%|████████▋  | 13/15 [05:12<00:49, 24.80s/it]
CV accuracy for k = 25 is 87%
 93%|█████████▎ | 14/15 [05:37<00:24, 24.83s/it]
CV accuracy for k = 27 is 87%
100%|██████████| 15/15 [06:02<00:00, 24.87s/it]
CV accuracy for k = 29 is 87%
```

In [0]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []

k = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
for i in k:
  neigh = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  neigh.fit(X_train_aw2v, y_train)

  y_train_pred = neigh.predict_proba(X_train_aw2v)[:,1]
  y_cv_pred = neigh.predict_proba(X_cv_aw2v)[:,1]
  train_auc.append(roc_auc_score(y_train,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

In [0]:

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

Out[0]:

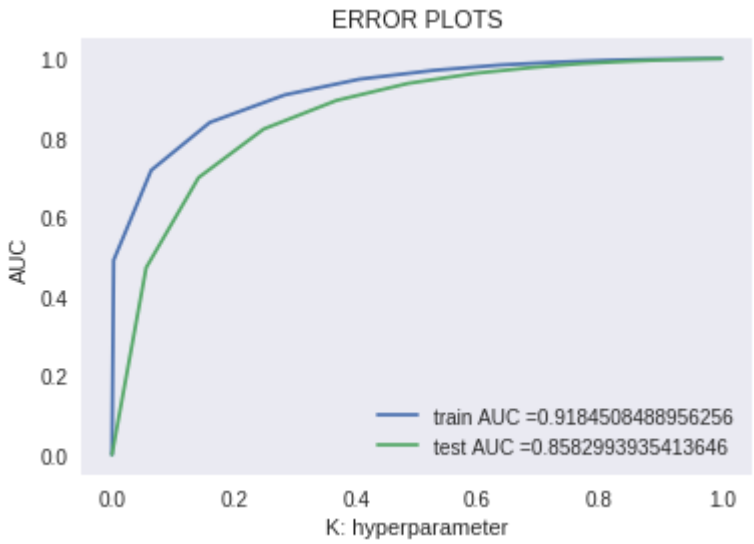
```
Text(0.5, 1.0, 'ERROR PLOTS')
```

In [0]:

```
Best k = 13
```

In [0]:

```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=13, algorithm='brute')
neigh.fit(X_train_aw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_aw2v)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_aw2v)[:,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.grid()
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_aw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_aw2v)))
```

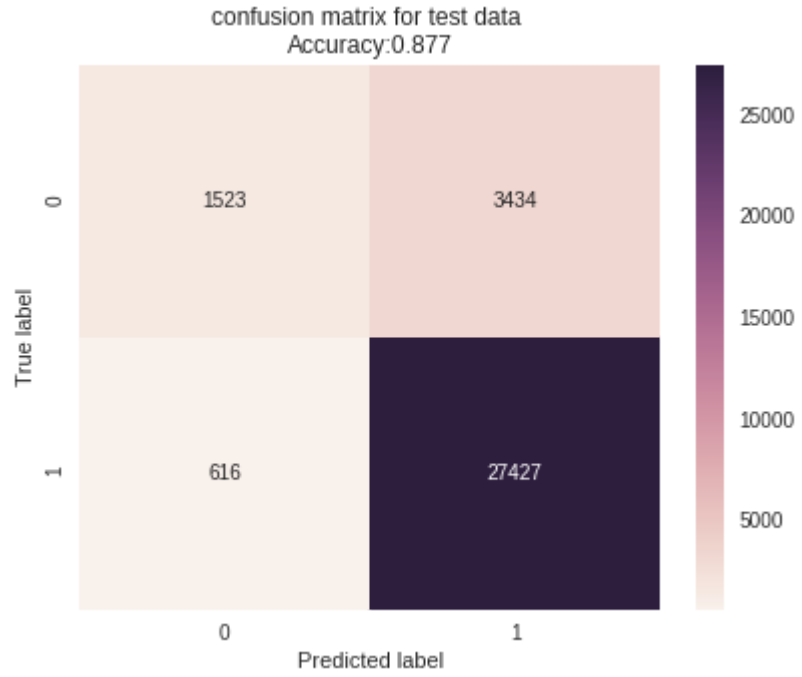



```
====================================================================================
==========================
Train confusion matrix
[[ 2388  4182]
 [  619 37701]]
Test confusion matrix
[[ 1523  3434]
 [  616 27427]]
```

In [0]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.893
2388
4182
619
37701
True label
0
1
0
1
Predicted label
32000
24000
16000
8000



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

In [29]:

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

In [30]:

```
model = TfidfVectorizer()
model.fit(X_train)
tf_idf_matrix = model.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 [39]:

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

X_train_tfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in 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
    X_train_tfw2v.append(sent_vec)
    row += 1
```

In [40]:

```
i=0
X_cv_w2v = []
for sentance in X_cv:
    X_cv_w2v.append(sentance.split())
```

In [41]:

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

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

X_cv_tfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in X_cv_w2v: # 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
    X_cv_tfw2v.append(sent_vec)
    row += 1
```

In [42]:

```
i=0
X_test_w2v = []
for sentance in X_test:
    X_test_w2v.append(sentance.split())
```

In [46]:

```
tf_idf_matrix = model.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_)))


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

X_test_tfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in X_test_w2v: # 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
    X_test_tfw2v.append(sent_vec)
    row += 1
```

Hyperparameter Tuning :-

In [47]:

```
for i in tqdm(range(1,50,5)):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
  # fitting the model on crossvalidation train
    knn.fit(X_train_tfw2v, y_train)
    pred = knn.predict(X_cv_tfw2v)
  # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

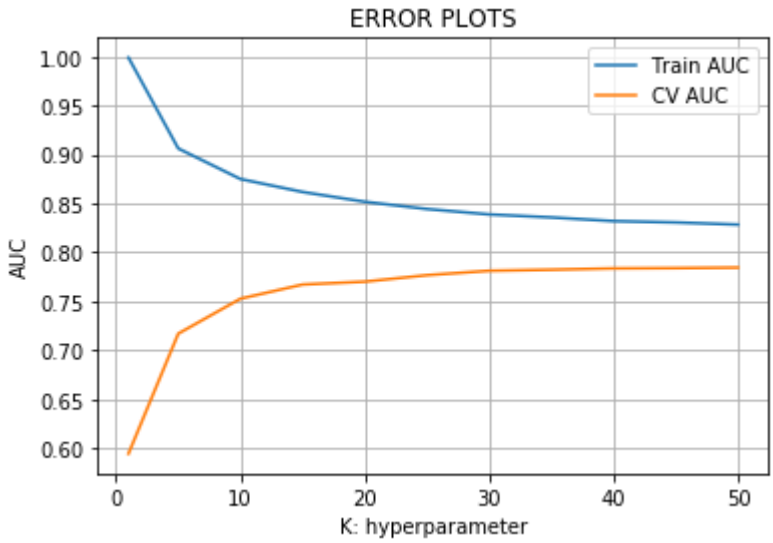
```
  0%|                                                    | 0/10 [00:00<?, ?
it/s]

CV accuracy for k = 1 is 78%

 10%|████                                      | 1/10 [00:04<00:36,  4.02
s/it]

CV accuracy for k = 6 is 83%

 20%|████████                                  | 2/10 [00:08<00:34,  4.30
s/it]

CV accuracy for k = 11 is 84%

 30%|████████████                              | 3/10 [00:13<00:30,  4.43
s/it]


CV accuracy for k = 16 is 84%

 40%|████████████████                          | 4/10 [00:18<00:27,  4.52
s/it]

CV accuracy for k = 21 is 84%

 50%|████████████████████                      | 5/10 [00:23<00:22,  4.60
s/it]

CV accuracy for k = 26 is 85%

 60%|████████████████████████                  | 6/10 [00:27<00:18,  4.63
s/it]

CV accuracy for k = 31 is 84%

 70%|████████████████████████████              | 7/10 [00:32<00:14,  4.75
s/it]

CV accuracy for k = 36 is 85%

 80%|████████████████████████████████          | 8/10 [00:37<00:09,  4.75
s/it]

CV accuracy for k = 41 is 84%

 90%|████████████████████████████████████      | 9/10 [00:42<00:04,  4.76
s/it]

CV accuracy for k = 46 is 84%

100%|████████████████████████████████████████| 10/10 [00:47<00:00,  4.79
s/it]
```

In [48]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [ 1, 5, 10, 15, 20, 25, 30, 35, 40 , 45, 50]

for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    neigh.fit(X_train_tfw2v, y_train)
    y_train_pred = neigh.predict_proba(X_train_tfw2v)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_tfw2v)[:,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.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[48]:

```
Text(0.5,1,'ERROR PLOTS')
```

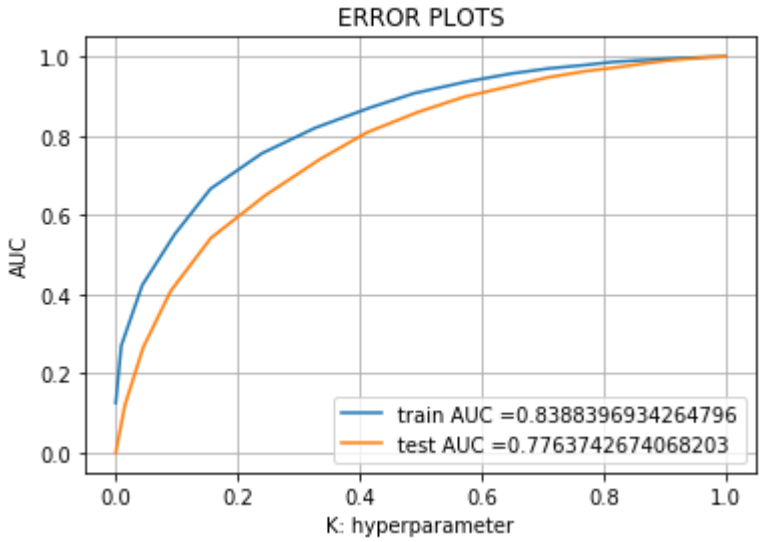


Best value of K is 30

Testing:

In [49]:

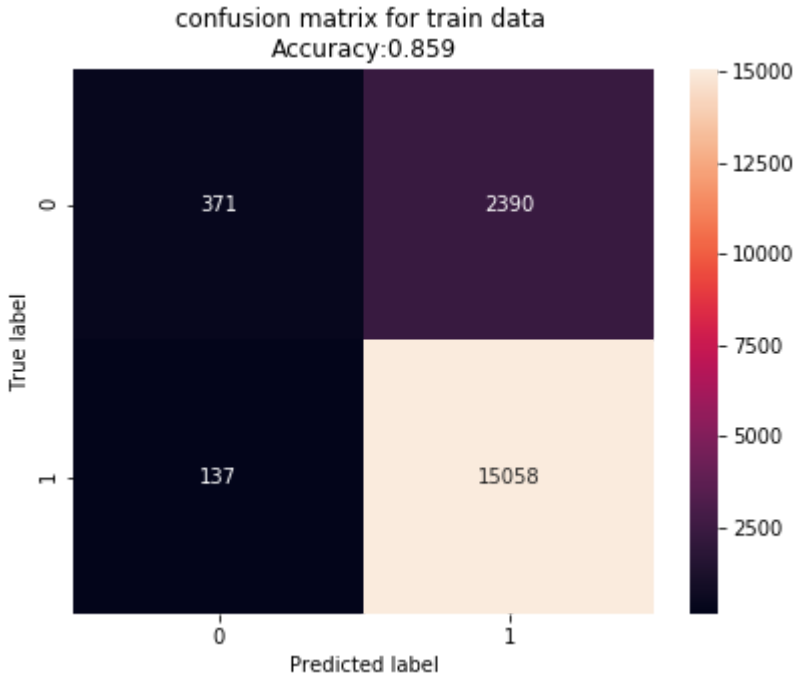
```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=30, algorithm='brute')
neigh.fit(X_train_tfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tfw2v
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tfw2v)[:,
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.grid()
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_tfw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfw2v)))
```

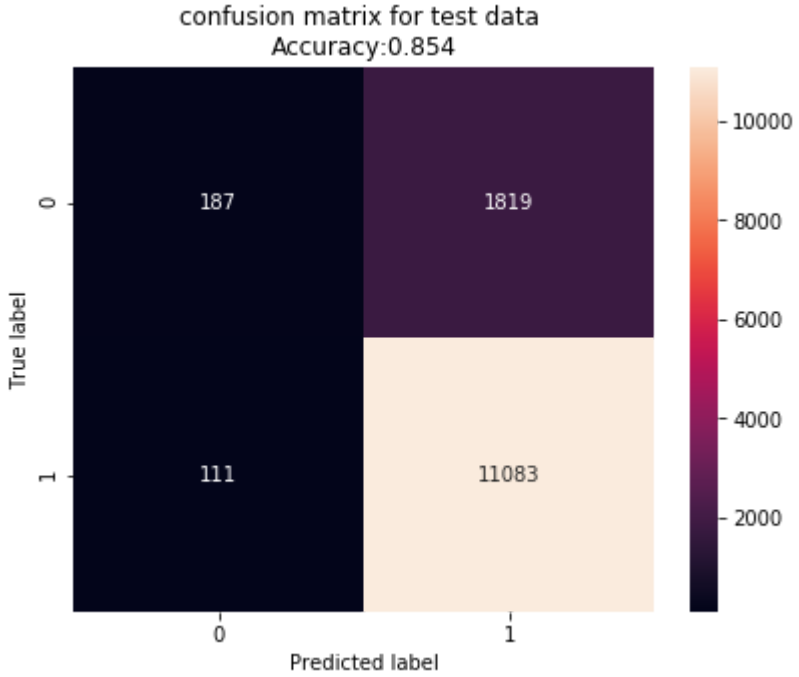



```
====================================================================================
==========================
Train confusion matrix
[[  371  2390]
 [  137 15058]]
Test confusion matrix
[[  187  1819]
 [  111 11083]]
```

In [50]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.859
371
2390
137
15058
True label
Predicted label
0
1
0
1
15000
12500
10000
7500
5000
2500



[5.2] Applying KNN kd-tree

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

In [27]:

```
# Please write all the code with proper documentation

df = pd.DataFrame({'Text':preprocessed_reviews})
X = df['Text'][:40000].values
y = final['Score'][:40000].values
```

In [28]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is ran
dom splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # thi
s is random splitting
```

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=500)
vectorizer.fit(X_train) # fit has to happen only on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
After vectorizations
((17956, 500), (17956,))
((8844, 500), (8844,))
((13200, 500), (13200,))
```

In [0]:

```
type(X_train_bow)
```

Out[0]:

```
scipy.sparse.csr.csr_matrix
```

In [0]:

```
# converting the sparse matrix to dense matrix

X_train_bow_sp = X_train_bow.toarray()
X_cv_bow_sp = X_cv_bow.toarray()
X_test_bow_sp = X_test_bow.toarray()
```

Hyperparameter Tuning using for loop:-

In [0]:

```
for i in tqdm(range(1,30,2)):

  knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  # fitting the model on crossvalidation train
  knn.fit(X_train_bow_sp, y_train)
  pred = knn.predict(X_cv_bow_sp)
  # evaluate CV accuracy
  acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
  print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

```
  7%|▉          | 1/15 [02:37<36:46, 157.60s/it]
CV accuracy for k = 1 is 74%
 13%|█▎         | 2/15 [05:20<34:29, 159.16s/it]
CV accuracy for k = 3 is 81%
 20%|██         | 3/15 [08:04<32:06, 160.57s/it]
CV accuracy for k = 5 is 83%
 27%|██▋        | 4/15 [10:49<29:42, 162.08s/it]
CV accuracy for k = 7 is 83%
 33%|███▎       | 5/15 [13:35<27:11, 163.10s/it]
CV accuracy for k = 9 is 84%
 40%|████       | 6/15 [16:21<24:36, 164.06s/it]
CV accuracy for k = 11 is 84%
 47%|████▋      | 7/15 [19:08<21:58, 164.78s/it]
CV accuracy for k = 13 is 84%
 53%|█████▎     | 8/15 [21:56<19:20, 165.79s/it]
CV accuracy for k = 15 is 84%
 60%|██████     | 9/15 [24:43<16:37, 166.33s/it]
CV accuracy for k = 17 is 84%
 67%|██████▋    | 10/15 [27:31<13:53, 166.79s/it]
CV accuracy for k = 19 is 84%
 73%|███████▎   | 11/15 [30:18<11:07, 166.91s/it]
CV accuracy for k = 21 is 84%
 80%|████████   | 12/15 [33:06<08:21, 167.22s/it]
CV accuracy for k = 23 is 84%
 87%|████████▋  | 13/15 [35:53<05:34, 167.20s/it]
CV accuracy for k = 25 is 85%
 93%|█████████▎ | 14/15 [38:41<02:47, 167.42s/it]
CV accuracy for k = 27 is 84%
100%|██████████| 15/15 [41:30<00:00, 167.63s/it]
CV accuracy for k = 29 is 85%
```

In [0]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [ 1, 5, 10, 15, 20, 25, 30, 35, 40 , 45, 50]

for i in K:
  neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  neigh.fit(X_train_bow_sp, y_train)
  y_train_pred = neigh.predict_proba(X_train_bow_sp)[:,1]
  y_cv_pred = neigh.predict_proba(X_cv_bow_sp)[:,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.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[0]:

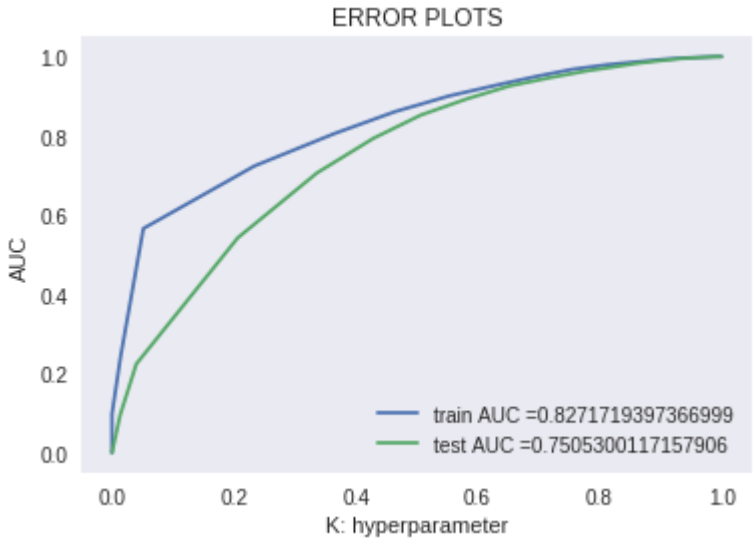
```
Text(0.5,1,'ERROR PLOTS')
```

Best value of K is 20

Testing:

In [0]:

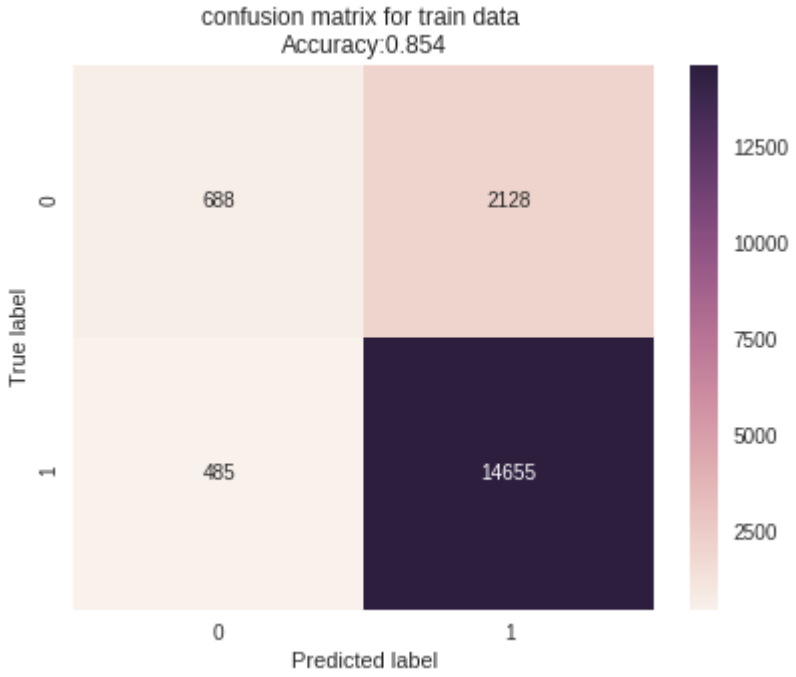
```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=20, algorithm='kd_tree')
neigh.fit(X_train_bow_sp, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow_s
p)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow_sp)
[:,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.grid()
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_sp)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow_sp)))
```




```
====================================================================================
==========================
Train confusion matrix
[[  688  2128]
 [  485 14655]]
Test confusion matrix
[[  441  1565]
 [  401 10793]]
```

In [0]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_bow_sp))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_bow_sp))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_bow_sp))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_bow_sp))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.854
688
2128
485
14655
0
1
True label
0
1
Predicted label
12500
10000
7500
5000
2500

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

In [0]:

```
# Please write all the code with proper documentation

from sklearn.feature_extraction.text import CountVectorizer
tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_vect.fit(X_train)
# we use the fitted CountVectorizer to convert the text to vector
X_train_tf = tf_idf_vect.transform(X_train)
X_cv_tf = tf_idf_vect.transform(X_cv)
X_test_tf = tf_idf_vect.transform(X_test)
print("After vectorizations")
print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)
```

```
After vectorizations
((17956, 500), (17956,))
((8844, 500), (8844,))
((13200, 500), (13200,))
```

In [0]:

```
# converting the sparse matrix to dense matrix

X_train_tf_sp = X_train_tf.toarray()
X_cv_tf_sp = X_cv_tf.toarray()
X_test_tf_sp = X_test_tf.toarray()
```

Hyperparameter Tuning:-

In [0]:

```
for i in tqdm(range(1,51,5)):

  knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  # fitting the model on crossvalidation train
  knn.fit(X_train_tf_sp, y_train)
  pred = knn.predict(X_cv_tf_sp)
  # evaluate CV accuracy
  acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
  print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

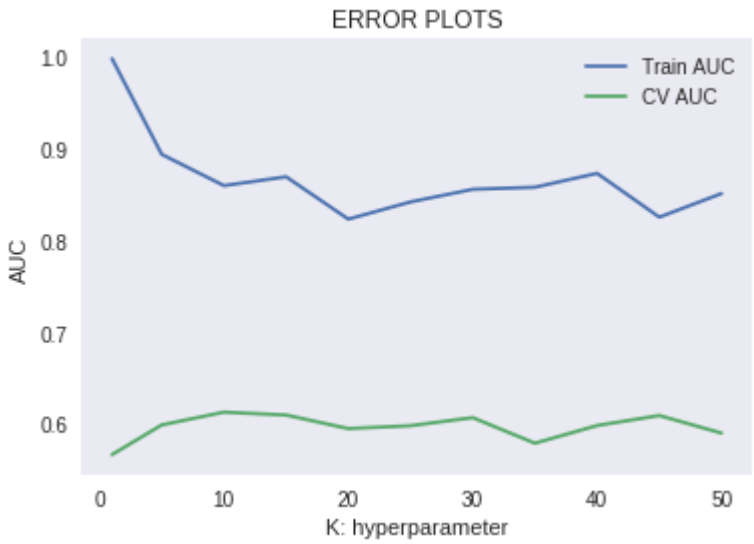
```
 10%|▉         | 1/10 [02:46<25:01, 166.80s/it]

CV accuracy for k = 1 is 74%

 20%|██        | 2/10 [05:34<22:16, 167.12s/it]

CV accuracy for k = 6 is 84%

 30%|███       | 3/10 [08:22<19:31, 167.31s/it]

CV accuracy for k = 11 is 85%

 40%|████      | 4/10 [11:09<16:43, 167.28s/it]

CV accuracy for k = 16 is 85%

 50%|█████     | 5/10 [13:57<13:57, 167.41s/it]

CV accuracy for k = 21 is 85%

 60%|██████    | 6/10 [16:46<11:11, 167.82s/it]

CV accuracy for k = 26 is 84%

 70%|███████   | 7/10 [19:36<08:26, 168.70s/it]

CV accuracy for k = 31 is 84%

 80%|████████  | 8/10 [22:28<05:39, 169.55s/it]

CV accuracy for k = 36 is 84%

 90%|█████████ | 9/10 [25:20<02:50, 170.22s/it]

CV accuracy for k = 41 is 84%

100%|██████████| 10/10 [28:11<00:00, 170.66s/it]

CV accuracy for k = 46 is 84%
```

In [0]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [ 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

for i in K:
  neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  neigh.fit(X_train_tf_sp, y_train)
  y_train_pred = neigh.predict_proba(X_train_tf_sp)[:,1]
  y_cv_pred = neigh.predict_proba(X_cv_tf_sp)[:,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.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[0]:

```
Text(0.5,1,'ERROR PLOTS')
```

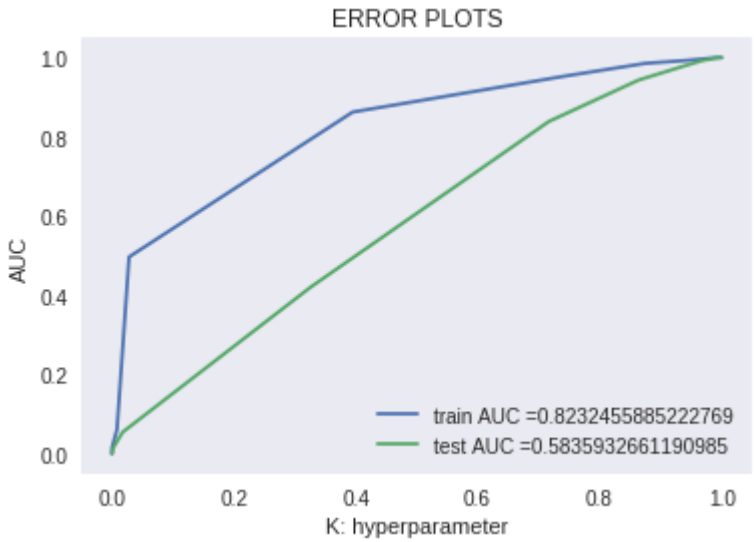


Best K = 20

Testing:-

In [0]:

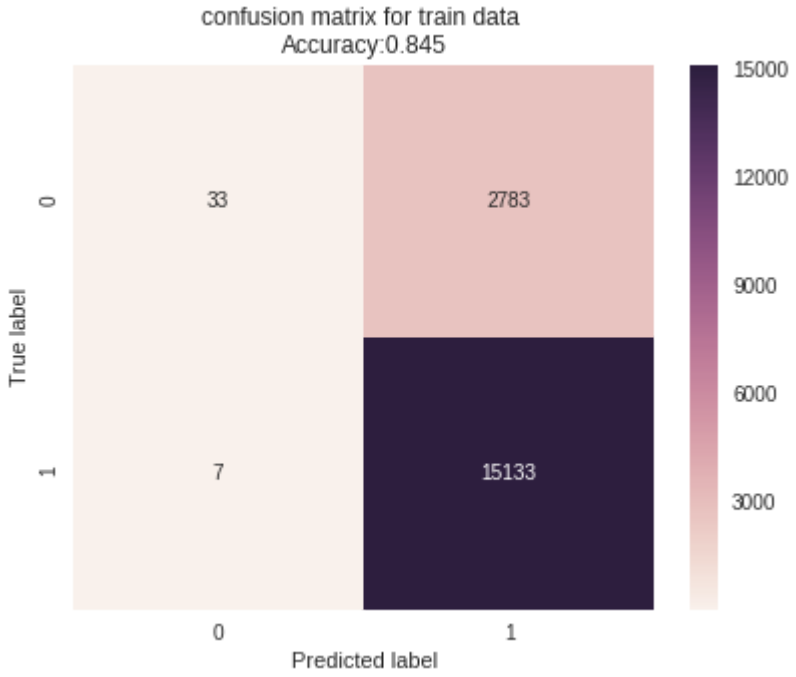
```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=20, algorithm='kd_tree')
neigh.fit(X_train_tf_sp, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tf_sp
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tf_sp)[:,
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.grid()
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_tf_sp)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tf_sp)))
```

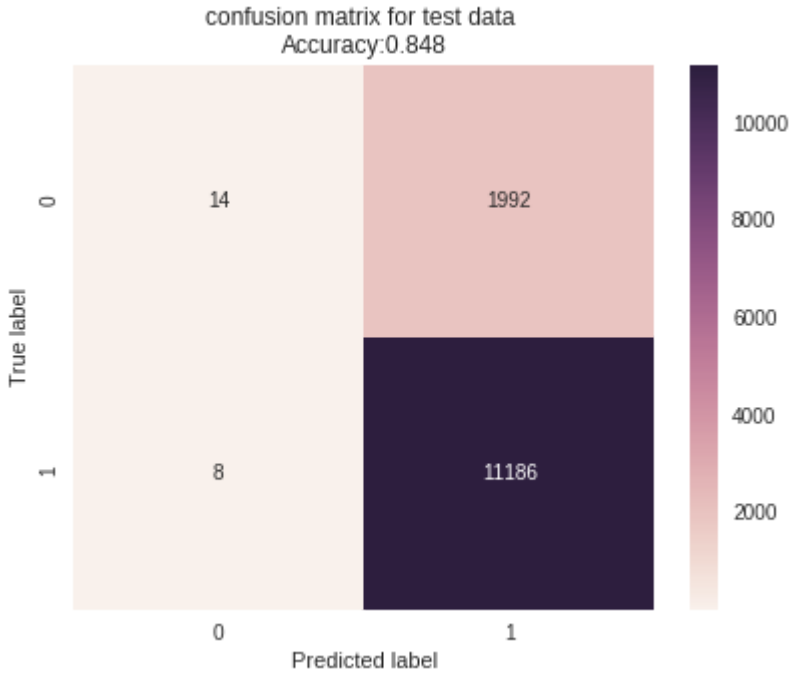



```
====================================================================================
==========================
Train confusion matrix
[[   33  2783]
 [    7 15133]]
Test confusion matrix
[[   14  1992]
 [    8 11186]]
```

In [0]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tf_sp))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tf_sp))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tf_sp))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tf_sp))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.845
True label
Predicted label
0
1
33
2783
7
15133
15000
12000
9000
6000
3000



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

Hyperparameter Tuning:-

In [0]:

```
for i in tqdm(range(1,51,5)):

  knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  # fitting the model on crossvalidation train
  knn.fit(X_train_aw2v, y_train)
  pred = knn.predict(X_cv_aw2v)
  # evaluate CV accuracy
  acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
  print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

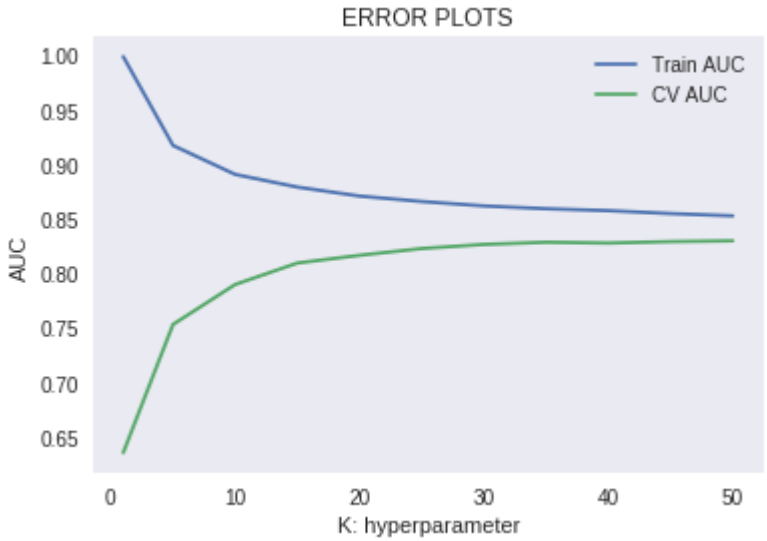
```
 10%|▉         | 1/10 [00:10<01:30, 10.06s/it]

CV accuracy for k = 1 is 81%

 20%|██        | 2/10 [00:22<01:25, 10.75s/it]

CV accuracy for k = 6 is 84%

 30%|███       | 3/10 [00:35<01:20, 11.47s/it]

CV accuracy for k = 11 is 85%

 40%|████      | 4/10 [00:49<01:12, 12.13s/it]

CV accuracy for k = 16 is 85%

 50%|█████     | 5/10 [01:03<01:03, 12.70s/it]

CV accuracy for k = 21 is 85%

 60%|██████    | 6/10 [01:17<00:52, 13.18s/it]

CV accuracy for k = 26 is 85%

 70%|███████   | 7/10 [01:32<00:40, 13.61s/it]

CV accuracy for k = 31 is 85%

 80%|████████  | 8/10 [01:46<00:27, 13.97s/it]

CV accuracy for k = 36 is 85%

 90%|█████████ | 9/10 [02:02<00:14, 14.28s/it]

CV accuracy for k = 41 is 85%

100%|██████████| 10/10 [02:17<00:00, 14.56s/it]

CV accuracy for k = 46 is 85%
```

In [0]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [ 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

for i in K:
  neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  neigh.fit(X_train_aw2v, y_train)
  y_train_pred = neigh.predict_proba(X_train_aw2v)[:,1]
  y_cv_pred = neigh.predict_proba(X_cv_aw2v)[:,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.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[0]:

```
Text(0.5,1,'ERROR PLOTS')
```

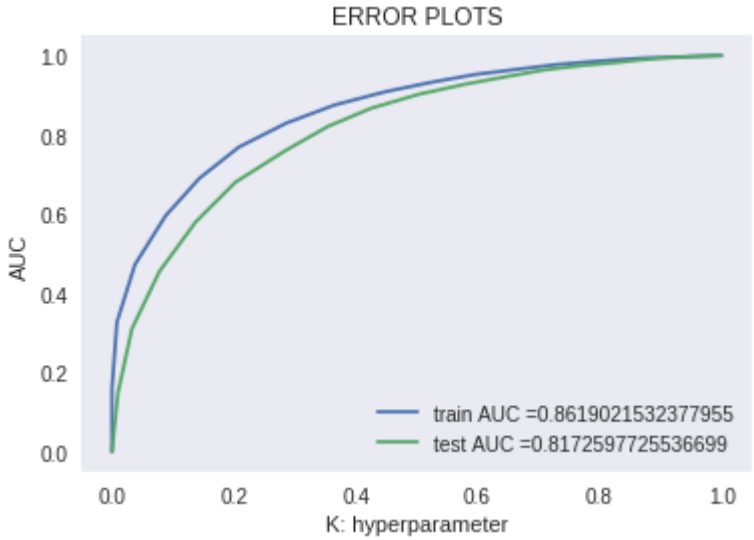


Best k is 30

Testing:-

In [0]:

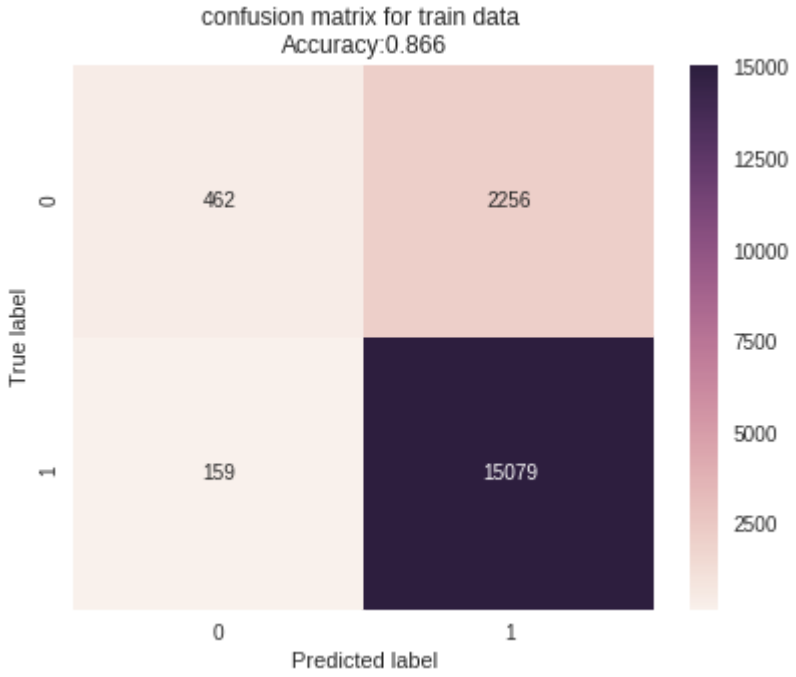
```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=30, algorithm='kd_tree')
neigh.fit(X_train_aw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_aw2v)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_aw2v)[:,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.grid()
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_aw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_aw2v)))
```




```
====================================================================================
==========================
Train confusion matrix
[[  462  2256]
 [  159 15079]]
Test confusion matrix
[[  283  1762]
 [  130 11025]]
```

In [0]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.866
462
2256
159
15079
0
1
True label
0
1
Predicted label
15000
12500
10000
7500
5000
2500

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

In [52]:

```
for i in tqdm(range(1,51,5)):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
  # fitting the model on crossvalidation train
    knn.fit(X_train_tfw2v, y_train)
    pred = knn.predict(X_cv_tfw2v)
  # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

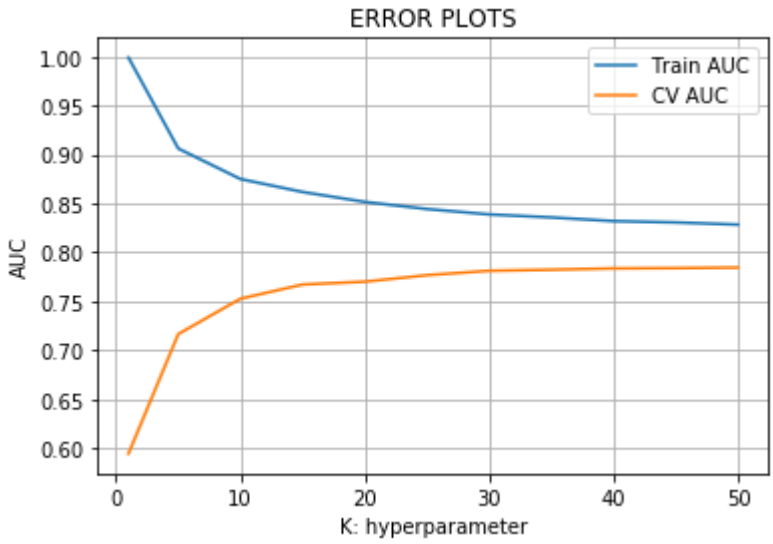
```
  0%|                                                    | 0/10 [00:00<?, ?
it/s]

CV accuracy for k = 1 is 78%

 10%|████                                      | 1/10 [00:15<02:16, 15.14
s/it]

CV accuracy for k = 6 is 83%

 20%|████████                                  | 2/10 [00:32<02:05, 15.69
s/it]

CV accuracy for k = 11 is 84%

 30%|████████████                              | 3/10 [00:50<01:56, 16.59
s/it]

CV accuracy for k = 16 is 84%

 40%|████████████████                          | 4/10 [01:10<01:45, 17.64
s/it]

CV accuracy for k = 21 is 84%

 50%|████████████████████                      | 5/10 [01:30<01:31, 18.28
s/it]

CV accuracy for k = 26 is 85%

 60%|████████████████████████                  | 6/10 [01:53<01:18, 19.73
s/it]

CV accuracy for k = 31 is 84%

 70%|████████████████████████████              | 7/10 [02:16<01:01, 20.53
s/it]

CV accuracy for k = 36 is 85%

 80%|████████████████████████████████          | 8/10 [02:37<00:41, 20.80
s/it]

CV accuracy for k = 41 is 84%

 90%|████████████████████████████████████      | 9/10 [03:00<00:21, 21.28
s/it]

CV accuracy for k = 46 is 84%

100%|████████████████████████████████████████| 10/10 [03:22<00:00, 21.64
s/it]
```

In [53]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [ 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    neigh.fit(X_train_tfw2v, y_train)
    y_train_pred = neigh.predict_proba(X_train_tfw2v)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_tfw2v)[:,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.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[53]:

```
Text(0.5,1,'ERROR PLOTS')
```

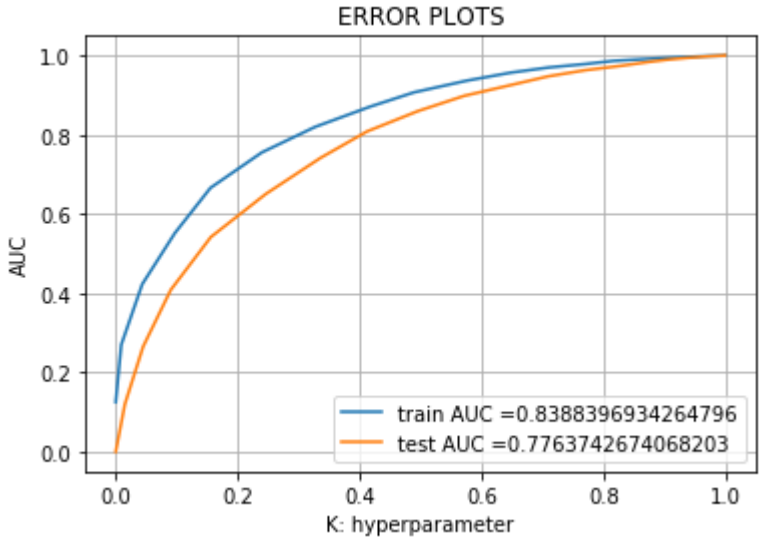


Observation: The best value of K is:- 30

Testing:-

In [54]:

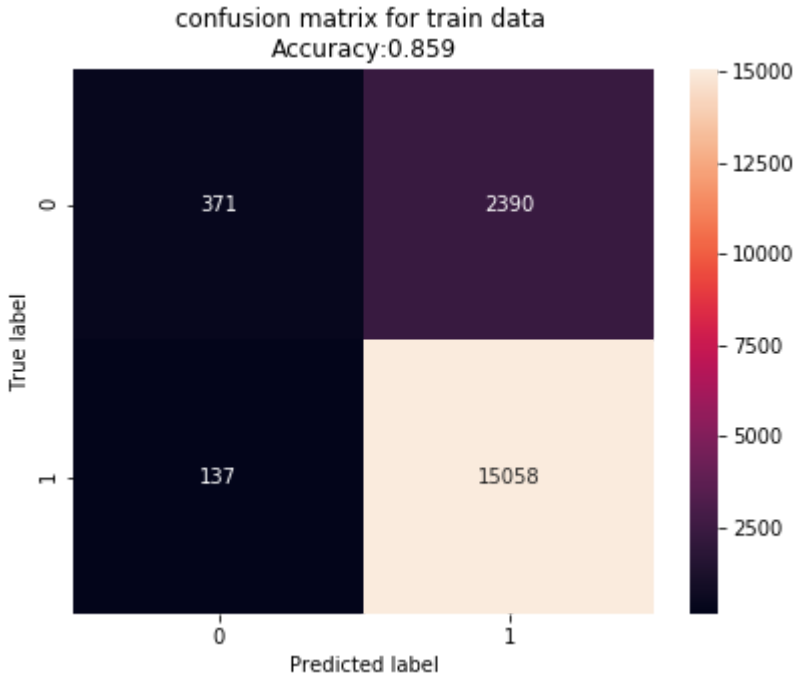
```
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=30, algorithm='kd_tree')
neigh.fit(X_train_tfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tfw2v
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tfw2v)[:,
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.grid()
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_tfw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfw2v)))
```

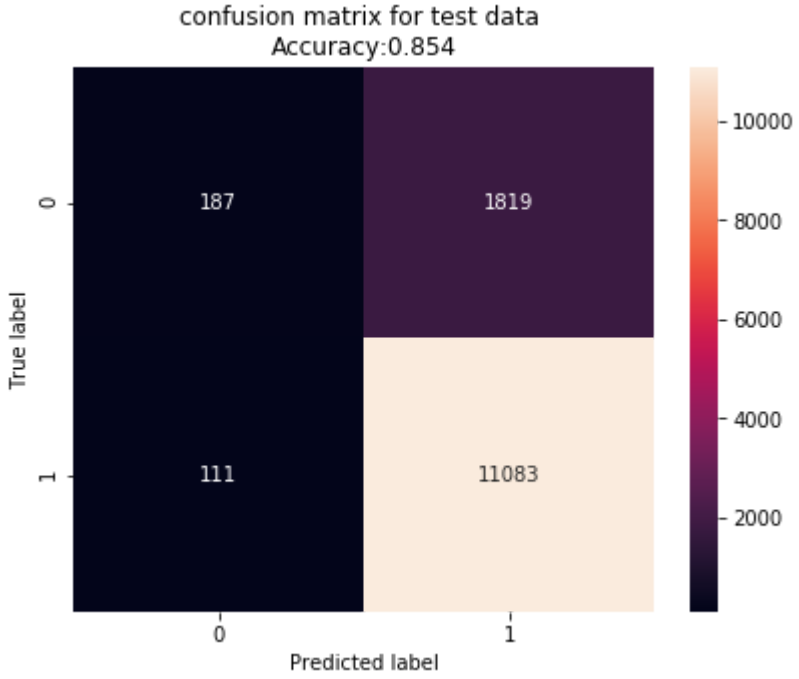



```
====================================================================================
==========================
Train confusion matrix
[[  371  2390]
 [  137 15058]]
Test confusion matrix
[[  187  1819]
 [  111 11083]]
```

In [55]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.859
371
2390
137
15058
True label
Predicted label
0
1
0
1
15000
12500
10000
7500
5000
2500



[6] Conclusions

In [56]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Hyperparameter(K)", "Train AUC", "Test AUC"]

x.add_row(["BOW(brute) ", 15, 0.80, 0.68])
x.add_row(["TFIDF(brute)", 15, 0.73, 0.509])
x.add_row(["AVGW2V(brute)", 13, 0.91, 0.85])
x.add_row(["TF-IDFW2V (brute)", 30, 0.83, 0.77])
x.add_row(["BOW(kd_tree) ", 20, 0.82, 0.75])
x.add_row(["TFIDF(kd_tree)", 20, 0.82, 0.58])
x.add_row(["AVGW2V(kd_tree)", 30, 0.86, 0.81])
x.add_row(["TF-IDFW2V (kd_tree)", 30, 0.83, 0.77])


print(x)
```

```
+--------------------+-------------------+-----------+----------+
|        Model       | Hyperparameter(K) | Train AUC | Test AUC |
+--------------------+-------------------+-----------+----------+
|     BOW(brute)     |         15        |    0.8    |   0.68   |
|     TFIDF(brute)   |         15        |    0.73   |  0.509   |
|    AVGW2V(brute)   |         13        |    0.91   |   0.85   |
|  TF-IDFW2V (brute) |         30        |    0.83   |   0.77   |
|    BOW(kd_tree)    |         20        |    0.82   |   0.75   |
|    TFIDF(kd_tree)  |         20        |    0.82   |   0.58   |
|   AVGW2V(kd_tree)  |         30        |    0.86   |   0.81   |
| TF-IDFW2V (kd_tree) |         30        |    0.83   |   0.77   |
+--------------------+-------------------+-----------+----------+
```