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

In [0]:
```
%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]:
```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2F
www.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fau
th%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=
code

Enter your authorization code:
··********··
Mounted at /content/drive
```

In [0]:
```
%cd ./drive/'My Drive'
```

```
/content/drive/My Drive
```

[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 [0]:
```
# 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 da
ta points
# you can change the number to any other number based on your computing power

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

filtered_data = pd.read_sql_query("""" SELECT * FROM Reviews WHERE Score != 3 L
IMIT 50000""", 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 (50000, 10)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
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 [0]:

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

In [0]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[0]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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 [0]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[0]:

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

In [0]:
```
display['COUNT(*)'].sum()
```

Out[0]: 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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
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 [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inp
lace=False, kind='quicksort', na_position='last')
```

In [0]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
}, keep='first', inplace=False)
final.shape
```

Out[0]: (46072, 10)

In [0]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[0]: 92.144

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

In [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [0]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

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

```
(46071, 10)
```

Out[0]:
```
1    38479
0     7592
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 [0]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 =  re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying
it anymore.  Its very hard to find any chicken products made in the USA but t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
```

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying
it anymore.  Its very hard to find any chicken products made in the USA but t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cob
bler. home made is better, but a heck of a lot more work. this is great to ha
ve on hand for last minute dessert needs where you really want to impress wih
your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually pr
otein powders are high priced and high in calories, this one is a great barga
in and tastes great, I highly recommend for the lady gym rats, probably not
"macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should
definitely give this one a try. Let me first start by saying that everyone is
looking for something different for their ideal tea, and I will attempt to br
iefly highlight what makes this tea attractive to a wide range of tea drinker
s (whether you are a beginner or long-time tea enthusiast).  I have gone thro
ugh over 12 boxes of this tea myself, and highly recommend it for the followi
ng reasons:<br /><br />-Quality:  First, this tea offers a smooth quality wit
hout any harsh or bitter after tones, which often turns people off from many
green teas.  I've found my ideal brewing time to be between 3-5 minutes, givi
ng you a light but flavorful cup of tea.  However, if you get distracted or f
orget about your tea and leave it brewing for 20+ minutes like I sometimes d
o, the quality of this tea is such that you still get a smooth but deeper fla
vor without the bad after taste.  The leaves themselves are whole leaves (not
powdered stems, branches, etc commonly found in other brands), and the high-q
uality nylon bags also include chunks of tropical fruit and other discernible
ingredients.  This isn't your standard cheap paper bag with a mix of unknown
ingredients that have been ground down to a fine powder, leaving you to wonde
r what it is you are actually drinking.<br /><br />-Taste:  This tea offers n
otes of real pineapple and other hints of tropical fruits, yet isn't sweet or
artificially flavored.  You have the foundation of a high-quality young hyson
green tea for those true "tea flavor" lovers, yet the subtle hints of fruit m
ake this a truly unique tea that I believe most will enjoy.  If you want it s
weet, you can add sugar, splenda, etc but this really is not necessary as thi
s tea offers an inherent warmth of flavor through it's ingredients.<br /><br
/>-Price:  This tea offers an excellent product at an exceptional price (espe
cially when purchased at the prices Amazon offers).  Compared to other brands
which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, et
c.), Revolution offers a superior product at an outstanding price.  I have be
en purchasing this through Amazon for less per box than I would be paying at
my local grocery store for Lipton, etc.<br /><br />Overall, this is a wonderf
ul tea that is comparable, and even better than, other teas that are priced m
uch higher.  It offers a well-balanced cup of green tea that I believe many w
ill enjoy.  In terms of taste, quality, and price, I would argue you won't fi
nd a better combination that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [0]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
ove-all-tags-from-an-element
from bs4 import BeautifulSoup

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying
it anymore.  Its very hard to find any chicken products made in the USA but t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cob
bler. home made is better, but a heck of a lot more work. this is great to ha
ve on hand for last minute dessert needs where you really want to impress wih
your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually pr
otein powders are high priced and high in calories, this one is a great barga
in and tastes great, I highly recommend for the lady gym rats, probably not
"macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should
definitely give this one a try. Let me first start by saying that everyone is
looking for something different for their ideal tea, and I will attempt to br
iefly highlight what makes this tea attractive to a wide range of tea drinker
s (whether you are a beginner or long-time tea enthusiast).  I have gone thro
ugh over 12 boxes of this tea myself, and highly recommend it for the followi
ng reasons:-Quality:  First, this tea offers a smooth quality without any har
sh or bitter after tones, which often turns people off from many green teas.
I've found my ideal brewing time to be between 3-5 minutes, giving you a ligh
t but flavorful cup of tea.  However, if you get distracted or forget about y
our tea and leave it brewing for 20+ minutes like I sometimes do, the quality
of this tea is such that you still get a smooth but deeper flavor without the
bad after taste.  The leaves themselves are whole leaves (not powdered stems,
branches, etc commonly found in other brands), and the high-quality nylon bag
s also include chunks of tropical fruit and other discernible ingredients.  T
his isn't your standard cheap paper bag with a mix of unknown ingredients tha
t have been ground down to a fine powder, leaving you to wonder what it is yo
u are actually drinking.-Taste:  This tea offers notes of real pineapple and
other hints of tropical fruits, yet isn't sweet or artificially flavored.  Yo
u have the foundation of a high-quality young hyson green tea for those true
"tea flavor" lovers, yet the subtle hints of fruit make this a truly unique t
ea that I believe most will enjoy.  If you want it sweet, you can add sugar,
splenda, etc but this really is not necessary as this tea offers an inherent
warmth of flavor through it's ingredients.-Price:  This tea offers an excelle
nt product at an exceptional price (especially when purchased at the prices A
mazon offers).  Compared to other brands which I believe to be of similar qua
lity (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior pro
duct at an outstanding price.  I have been purchasing this through Amazon for
less per box than I would be paying at my local grocery store for Lipton, et
c.Overall, this is a wonderful tea that is comparable, and even better than,
other teas that are priced much higher.  It offers a well-balanced cup of gre
en tea that I believe many will enjoy.  In terms of taste, quality, and pric
e, I would argue you won't find a better combination that that offered by Rev
olution's Tropical Green Tea.
```

In [0]:
```
# 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 [0]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually pr
otein powders are high priced and high in calories, this one is a great barga
in and tastes great, I highly recommend for the lady gym rats, probably not
"macho" enough for guys since it is soy based...
==================================================
```

In [0]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/408403
9
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying
it anymore.  Its very hard to find any chicken products made in the USA but t
hey are out there, but this one isnt.  Its too bad too because its a good pro
duct but I wont take any chances till they know what is going on with the chi
na imports.
```

In [0]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually protei
n powders are high priced and high in calories this one is a great bargain an
d tastes great I highly recommend for the lady gym rats probably not macho en
ough for guys since it is soy based
```

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st
step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becau
se', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', '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", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:
```
# 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 i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 46071/46071 [00:24<00:00, 1898.32it/s]
```

In [0]:
```
preprocessed_reviews[1500]
```

Out[0]:
```
'great flavor low calories high nutrients high protein usually protein powder
s high priced high calories one great bargain tastes great highly recommend l
ady gym rats probably not macho enough guys since soy based'
```

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#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])
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#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/sta
ble/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

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

[4.3] TF-IDF

In [0]:
```
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_featu
re_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final
_tf_idf.get_shape()[1])
```

[4.4] Word2Vec

In [0]:
```
# 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 [0]:
```
# 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/#.W17SRFAzZP
Y
# 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-negati
ve300.bin', 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 ")
```

In [0]:
```
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])
```

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

[4.4.1.1] Avg W2v

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

[4.4.1.2] TFIDF weighted W2v

In [0]:
```
# 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 [0]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

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

[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/) 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

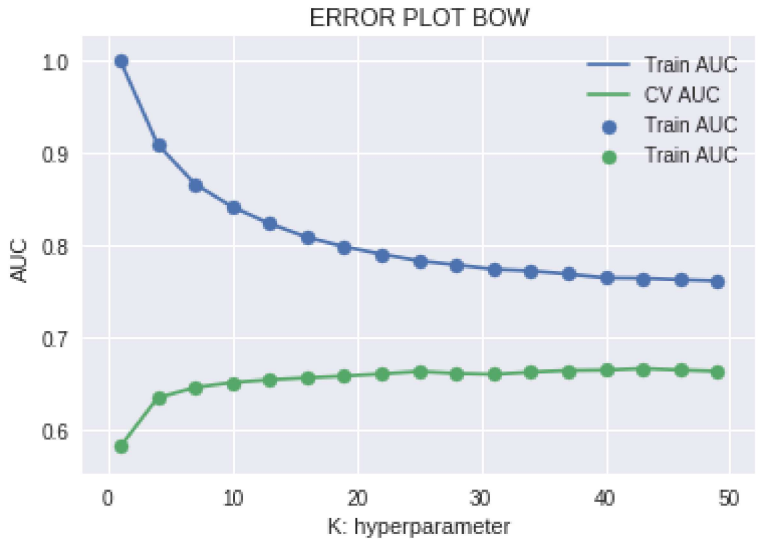
[5.1.1] Applying KNN brute force on BOW, SET 1

In [0]:
```
#Loading the data into X,Y
X = preprocessed_reviews
print(type(X))
Y = final['Score'].values
print(type(Y))

from sklearn.model_selection import train_test_split
#splitting the data into Train,Cross validate and test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train)# fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

```
<class 'list'>
<class 'numpy.ndarray'>
After vectorizations
(27642, 30898) (27642,)
(9214, 30898) (9214,)
(9215, 30898) (9215,)
================================================================================
=======================
```

Hyper parameter tuning BOW

In [0]:
```
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = range(1,50,3)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]

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

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



In [0]:
```
best_k=20
```

Testing with Test Data

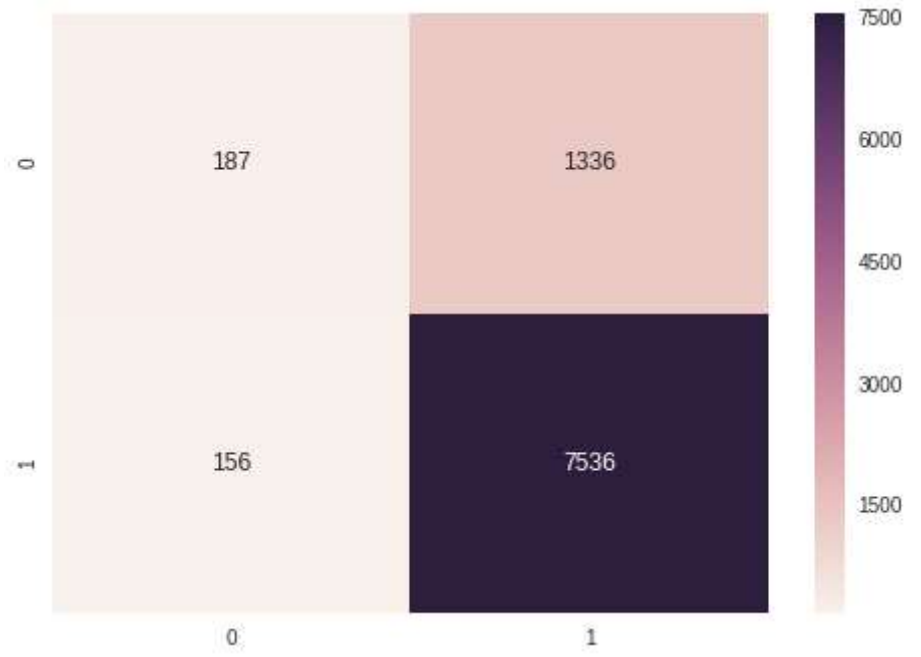
In [0]:
```
from sklearn.metrics import roc_curve, auc


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

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_tr
ain_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_
bow)[:,1])

#plotting the results
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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 PLOT TEST BOW")
plt.show()

print("="*100)
```

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

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(X_train_bow)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```



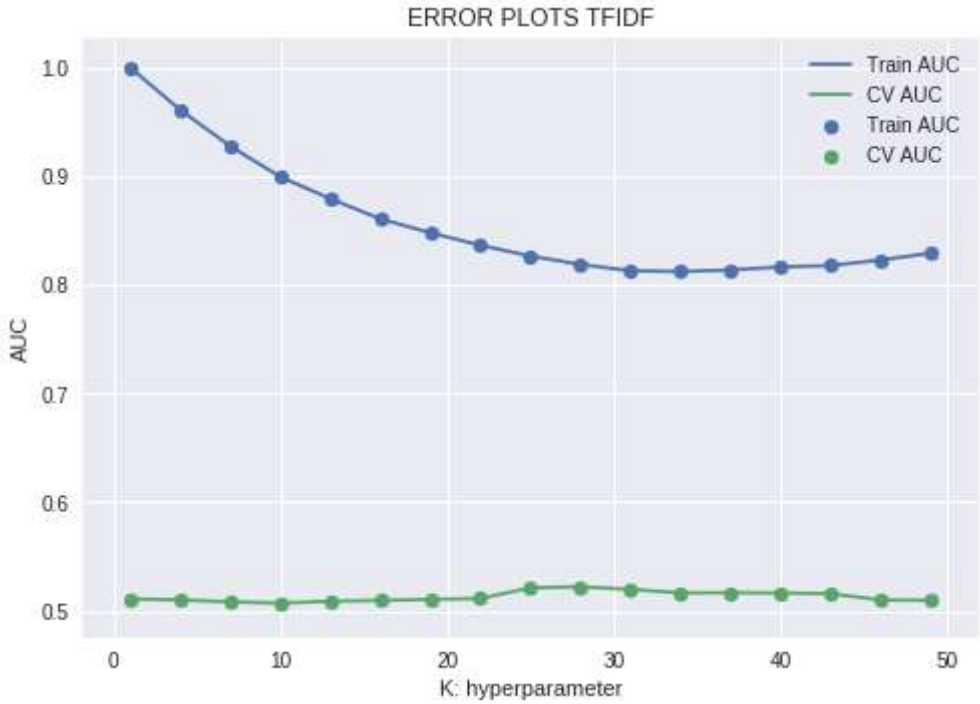
[5.1.2] Applying KNN brute force on TFIDF, SET 2

In [0]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#print(X_train.shape, Y_train.shape)
#print(X_cv.shape, Y_cv.shape)
#print(X_test.shape, Y_test.shape)

print("="*100)
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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)
print("="*100)
```

```
==============================================================================
=======================
After vectorizations
(27642, 16470) (27642,)
(9214, 16470) (9214,)
(9215, 16470) (9215,)
==============================================================================
=======================
```

Hyper-Parameter tuning TFIDF

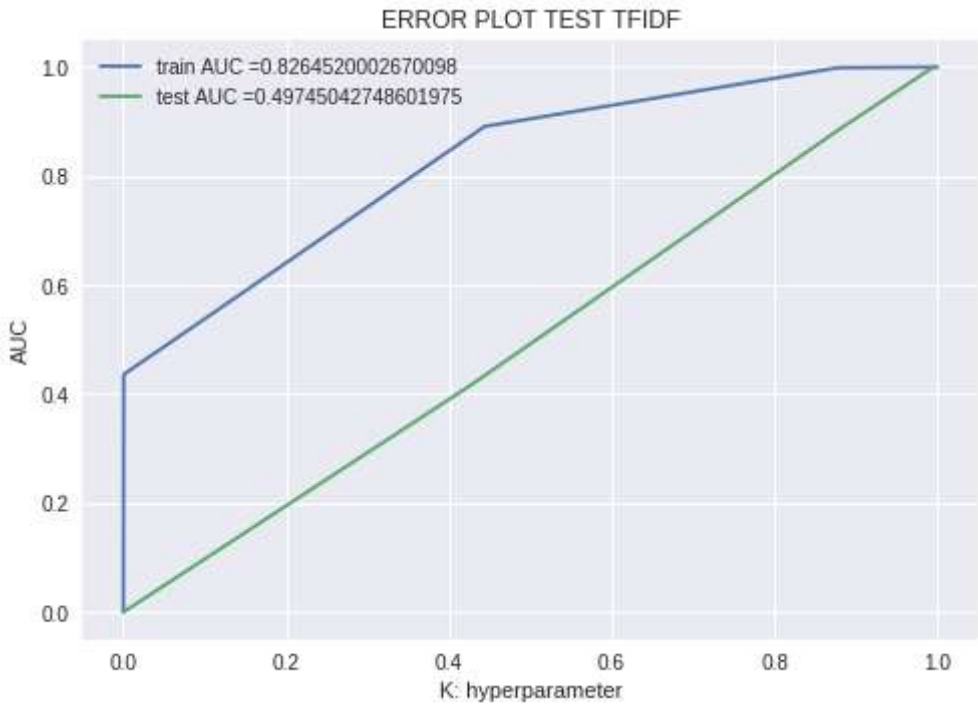
In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc_tf = []
cv_auc_tf = []
K = range(1,50,3)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_tf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tf)[:,1]

    train_auc_tf.append(roc_auc_score(y_train,y_train_pred))
    cv_auc_tf.append(roc_auc_score(y_cv, y_cv_pred))

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



In [0]:
```
best_k=25
```

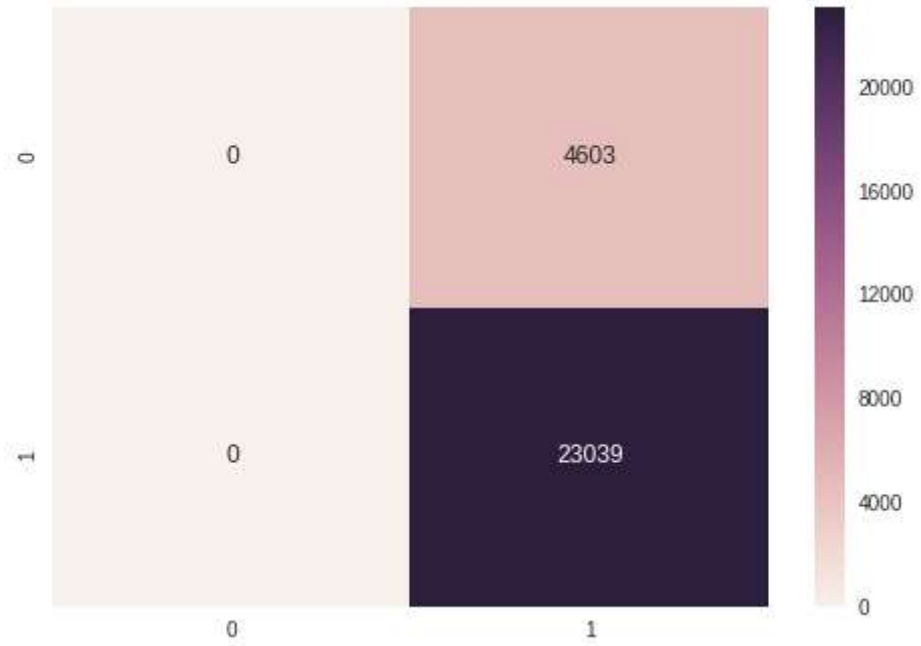
Test TFIDF with Test Data

In [0]:
```
from sklearn.metrics import roc_curve, auc


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

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_tr
ain_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_
tf)[:,1])

#plotting the results
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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 PLOT TEST TFIDF")
plt.show()
```

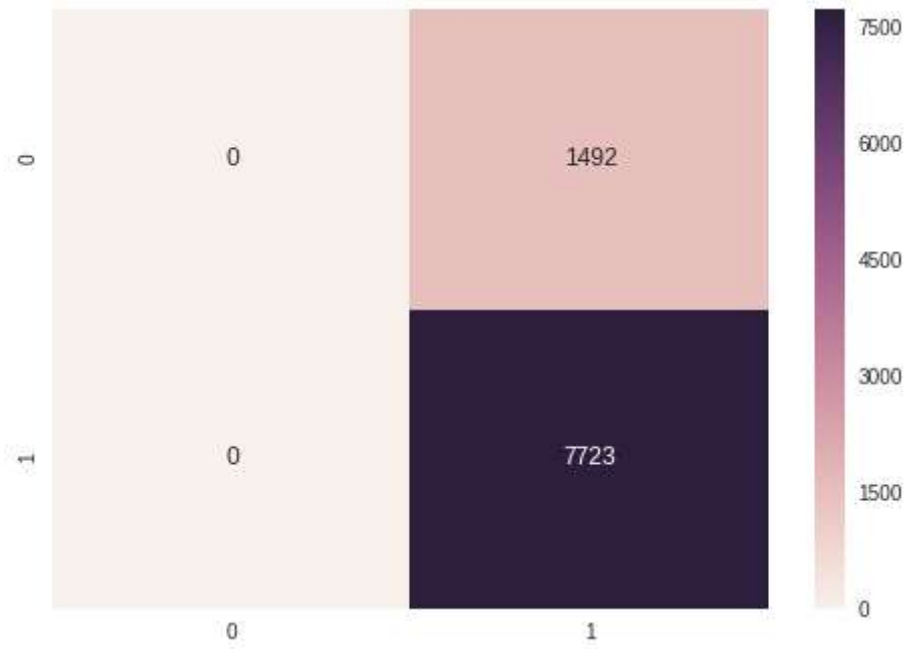


In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(X_train_tf)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



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

In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split
#Loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting

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

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

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  9982
sample words  ['fragile', 'pup', 'bit', 'right', 'rubber', 'rope', 'think',
'lasted', 'minutes', 'granted', 'still', 'higher', 'hopes', 'sticking', 'kong
s', 'filled', 'low', 'sodium', 'chicken', 'broth', 'frozen', 'overnight', 'al
lergies', 'family', 'ended', 'working', 'great', 'not', 'negative', 'reaction
s', 'ingredients', 'though', 'buy', 'many', 'pre', 'packaged', 'foods', 'wor
k', 'go', 'tried', 'several', 'varieties', 'prefer', 'cocoa', 'bars', 'almos
t', 'like', 'tootsie', 'roll', 'eat']
```

Converting Reviews into Numerical Vectors using W2V vectors

Algorithm: Avg W2V

In [0]:
```
from tqdm import tqdm
import numpy as np

# converting train data text

# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
his list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might
need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|██████████| 27642/27642 [00:44<00:00, 626.18it/s]

(27642, 50)
[ 0.50347267 -0.3577686  -0.07926292  0.13575505  0.22406996  0.01196603
 -0.11084776  0.37407058 -0.11845008 -0.45645243 -0.26519728  0.54780472
 -0.36561808  0.27867096 -0.12822265  0.71108093 -0.62197235 -0.29107584
  0.40845475  0.15912958 -0.2732986   0.22440454  0.20901338  0.19209682
  0.081383   -0.47663348 -0.01158237  0.09003615  0.70489767  0.30512914
 -0.05557746  0.52375869  0.11815198 -0.85411562  0.20364891  0.68488914
  0.24272164  0.33791799 -0.25620048  0.22621246 -0.06048601  0.61467127
  0.71380424 -0.18807816 -0.6869484  -0.21166347  0.7829895  -0.11158995
 -0.0245257  -0.0158259 ]
```

Converting CV data text

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

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

```
100%|██████████| 9214/9214 [00:15<00:00, 597.00it/s]

(9214, 50)
[ 0.05242311 -0.56653894 -0.73779737  0.21333677  0.51465294  0.11347399
  0.02804187  0.58370398 -0.00362269 -0.32131541 -0.00279314 -0.09449473
  0.32604416 -0.9749739  -0.21206839  0.83553149 -0.13427939 -0.22004984
 -0.62539248 -0.36303276  0.32617606  0.45779483  0.47717943 -0.85447177
  0.17810837 -0.12412346  0.7306913  -0.54386019  0.32995411  0.1282814
 -0.38569168  0.80017227  0.52024817 -0.43795009 -0.21975876 -0.35540144
  1.37839171 -0.16851676  0.33835824  0.25480358 -0.2296597   0.1893799
 -0.0716616  -0.03890469 -0.08955163  0.31674959  0.67655467 -0.58835905
  0.10004624  0.67696615]
```

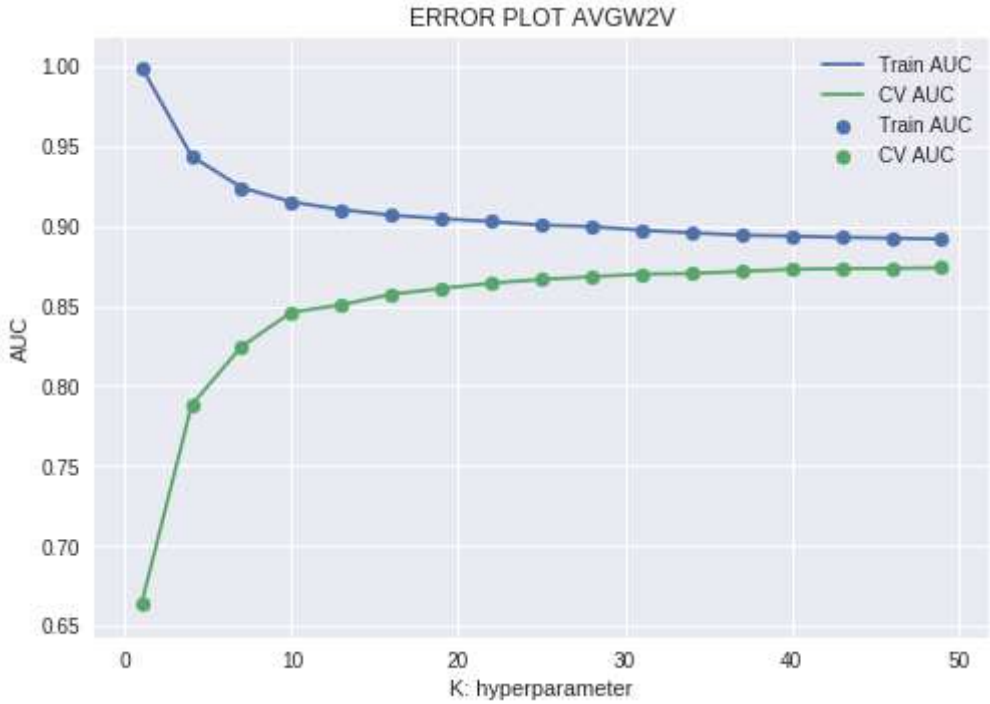
Converting Test Data Text

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

```
100%|██████████| 9215/9215 [00:14<00:00, 615.76it/s]

(9215, 50)
[ 0.33082493 -0.32736826 -0.72560754  0.02229255  0.7168271  -0.13228665
  0.1383338   0.22067253  0.30799425  0.0615083  -0.79526948 -0.05734002
 -0.20188906 -0.57055137 -0.14814083 -0.00155973 -0.12770092 -0.34050325
 -0.45771263  0.12665574  0.47915229  0.51689954 -0.24069701  0.0687601
  0.10890493 -0.50119132  0.76710782 -0.41149832  0.70168252  0.54700085
  0.10337205  0.30544083  0.07353469 -0.84284106  0.17087195  0.48157172
  0.44675198 -0.36333731  0.30287966  0.14656636 -0.2551075   0.58870431
  0.10247724  0.02251216 -0.5117915  -0.22140131  0.81012282 -0.59020667
 -0.00650185  0.42097952]
```

Hyper parameter tuning Avg W2V

In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = range(1,50,3)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(sent_vectors_cv)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
#plotting the error plot
plt.plot(K, train_auc, label='Train AUC')
plt.scatter(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.scatter(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AVGW2V")
plt.show()
```



In [0]:
```
best_k=25
```

Test with Test Data

In [0]:
```
neigh = KNeighborsClassifier(best_k)
neigh.fit(sent_vectors_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent
_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_ve
ctors_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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 PLOT TEST AVGW2V")
plt.show()

print("="*100)
```

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

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(sent_vectors_tra
in)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(sent_vectors_test
)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```

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

In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split
import numpy as np

a = preprocessed_reviews
X=np.array(preprocessed_reviews)
Y = final['Score'].values

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)

print(y_test.shape)
print(X_cv.shape)
print(y_cv.shape)
```

```
(27642,)
(27642,)
(9215,)
(9215,)
(9214,)
(9214,)
```

In [0]:
```
#preparing reviews for genism model
i=0
list_of_sentance_tf_train=[]
for sentance in X_train:
    list_of_sentance_tf_train.append(sentance.split())
```

In [0]:
```
#Training w2v models

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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tf_train,min_count=5,size=50, workers=4)

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  10020
sample words  ['favorite', 'gum', 'hard', 'find', 'retail', 'stores', 'grea
t', 'price', 'ordering', 'tea', 'directly', 'company', 'really', 'like', 'fla
vor', 'surprised', 'amazon', 'lower', 'cost', 'good', 'value', 'order', 'thre
e', 'boxes', 'total', 'bring', 'one', 'snack', 'work', 'almost', 'everyday',
'soooo', 'much', 'cheaper', 'buy', 'gas', 'station', 'reviews', 'say', 'packa
ges', 'far', 'single', 'package', 'tasted', 'little', 'rest', 'box', 'fine',
'jerky', 'strips']
```

Converting Train data text into TFIDFW2V vectors

In [0]:
```
from tqdm import tqdm
import numpy as np
```

In [0]:
```
model = TfidfVectorizer()
model.fit(X_train)
tf_idf_matrix = model.transform(X_train)
print(type(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_)))
```

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

In [0]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

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

```
100%|██████████| 27642/27642 [09:17<00:00, 53.29it/s]
```

Cross validation data

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

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

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

```
100%|██████████| 9214/9214 [03:02<00:00, 50.41it/s]
```

Test data

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

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

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

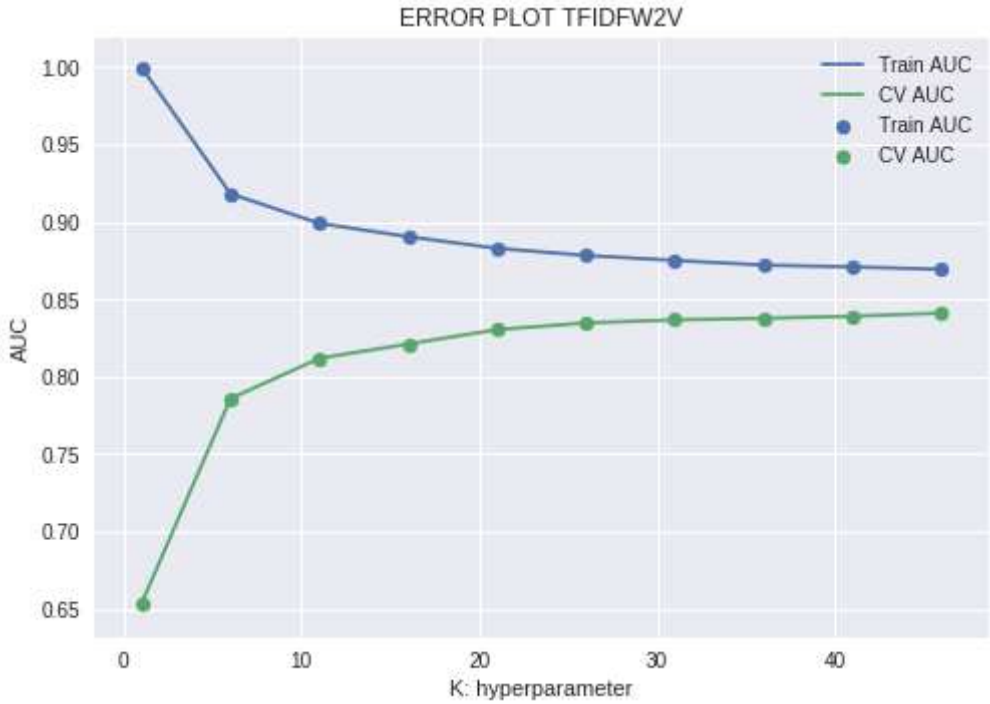
```
100%|██████████| 9215/9215 [03:03<00:00, 50.12it/s]
```

In [0]:
```
import numpy as np
tfidf_sent_train=np.array(tfidf_sent_train)
print(tfidf_sent_train.shape)
tfidf_sent_test=np.array(tfidf_sent_test)
print(tfidf_sent_test.shape)
tfidf_sent_cv=np.array(tfidf_sent_cv)
print(tfidf_sent_cv.shape)
print(y_train.shape)
print(y_test.shape)
print(y_cv.shape)
```

```
(27642, 50)
(9215, 50)
(9214, 50)
(27642,)
(9215,)
(9214,)
```

Hyper parameter Tuning TFIDFW2V

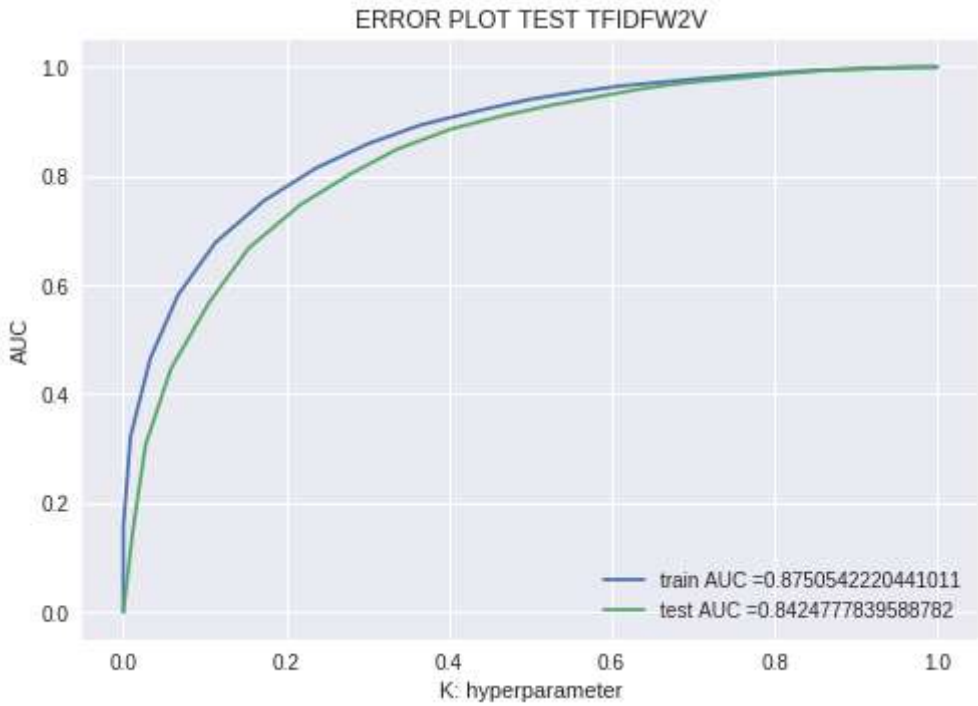
In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = range(1,50,5)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(tfidf_sent_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(tfidf_sent_train)[:,1]
    y_cv_pred =  neigh.predict_proba(tfidf_sent_cv)[:,1]

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

plt.plot(K, train_auc, label='Train AUC')
plt.scatter(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.scatter(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V")
plt.show()
```



In [0]:
```
best_k=31
```

Testing with Test Data

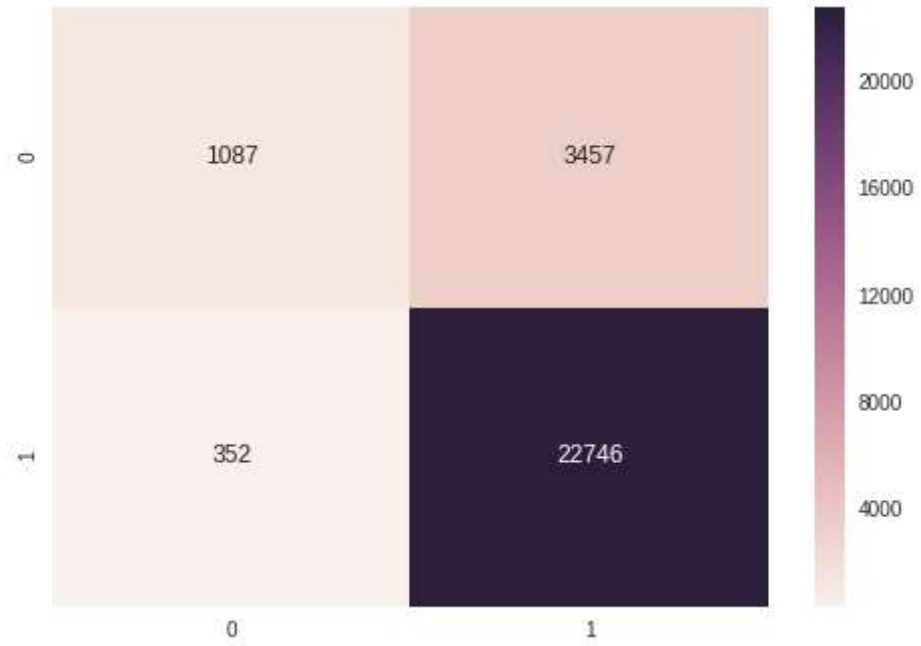
In [0]:
```
neigh = KNeighborsClassifier(best_k)
neigh.fit(tfidf_sent_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs


train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfid
f_sent_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_s
ent_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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 PLOT TEST TFIDFW2V")
plt.show()

print("="*100)
```



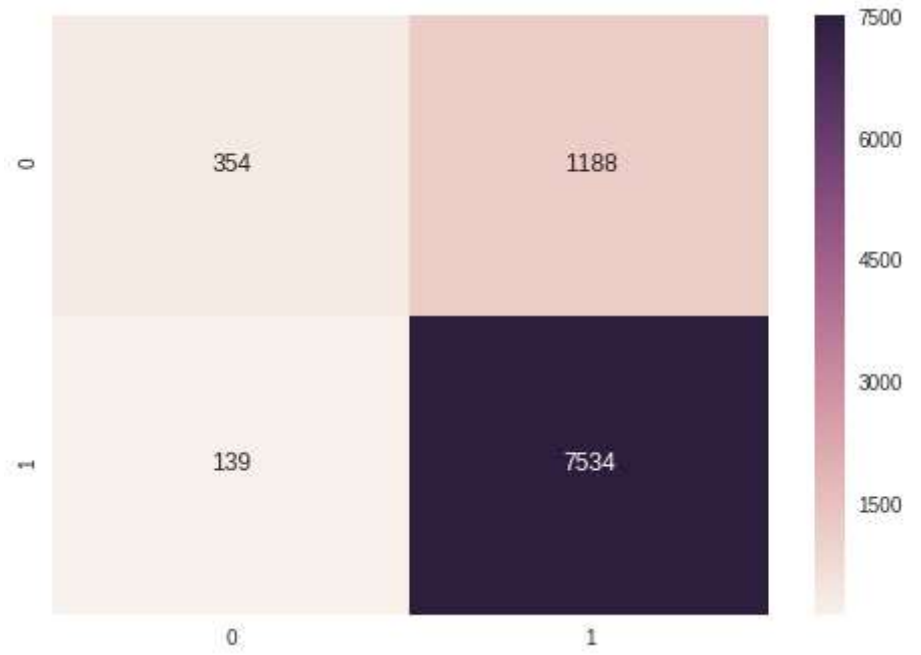

```
==================================================================================
========================
```

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(tfidf_sent_train
)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(tfidf_sent_test)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



[5.2] Applying KNN kd-tree

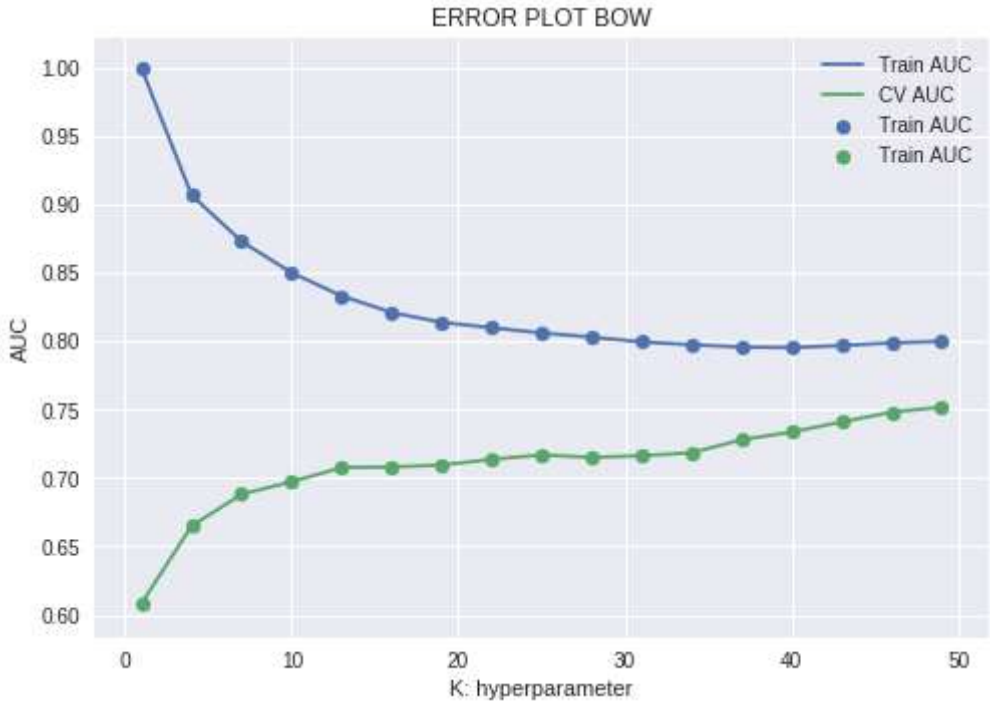
[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [0]:
```
#Loading 20000 points into X,Y

X = preprocessed_reviews[0:20000]
print(type(X))
Y = final['Score'].values[0:20000]
print(type(Y))

from sklearn.model_selection import train_test_split
from sklearn.decomposition import TruncatedSVD


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25)

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)
print("="*100)
svd = TruncatedSVD(n_components=10, n_iter=7, random_state=42)
svd.fit(X_train_bow)

print("After  Truncated vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

```
<class 'list'>
<class 'numpy.ndarray'>
After vectorizations
(12000, 500) (12000,)
(4000, 500) (4000,)
(4000, 500) (4000,)
================================================================================
========================
After  Truncated vectorizations
(12000, 500) (12000,)
(4000, 500) (4000,)
(4000, 500) (4000,)
================================================================================
========================
```

Hyper parameter Tuning BOW (KD -TREE)

In [0]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = range(1,50,3)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]

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

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



In [0]:
```
best_k=25
```

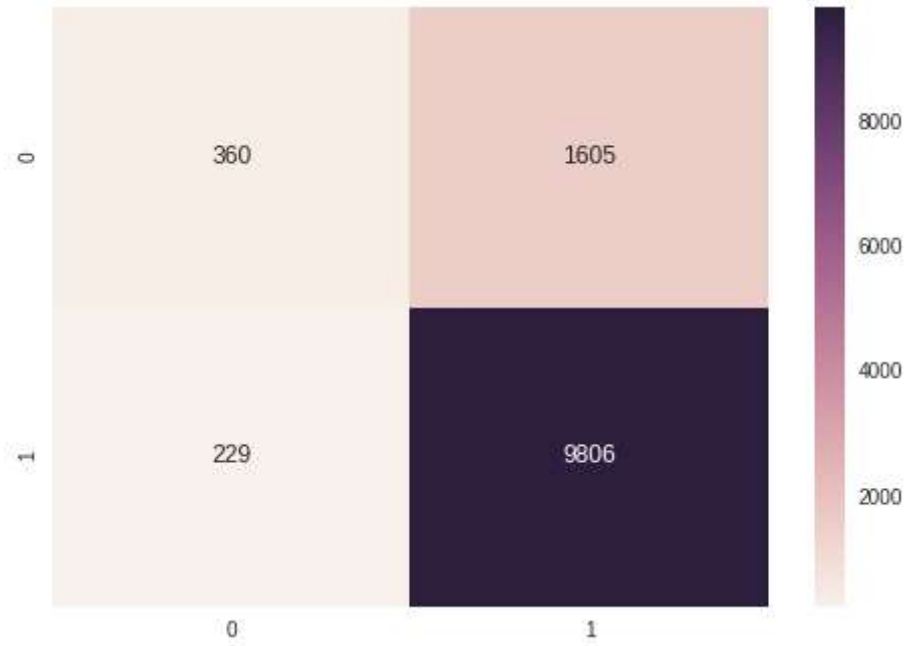
Testing with test data

In [0]:
```
from sklearn.metrics import roc_curve, auc


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

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_tr
ain_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_tp
r)))
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 PLOT TEST BOW")
plt.show()

print("="*100)
```

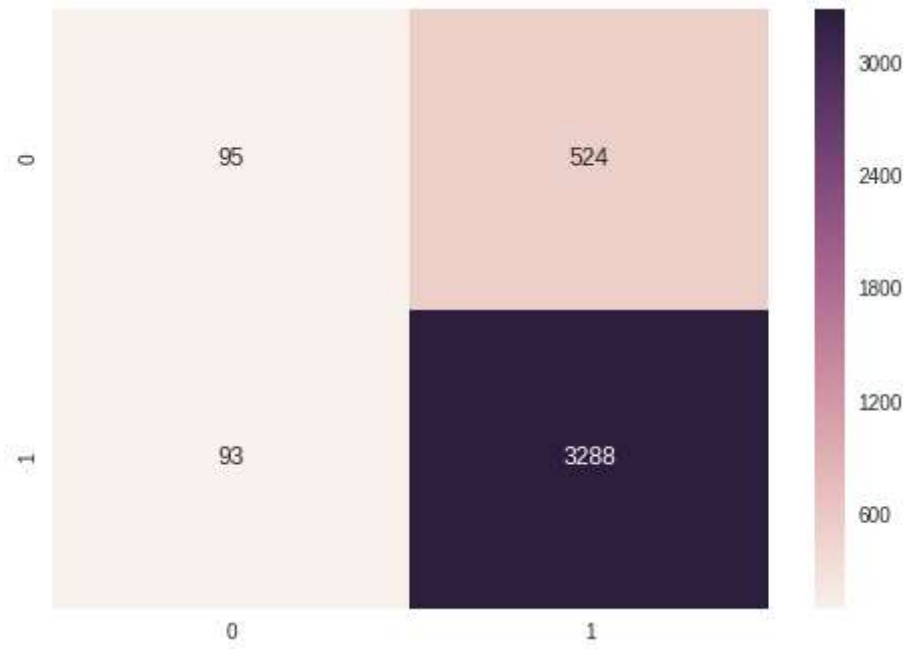
```
====================================================================================
========================
```

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(X_train_bow)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(X_test_bow)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



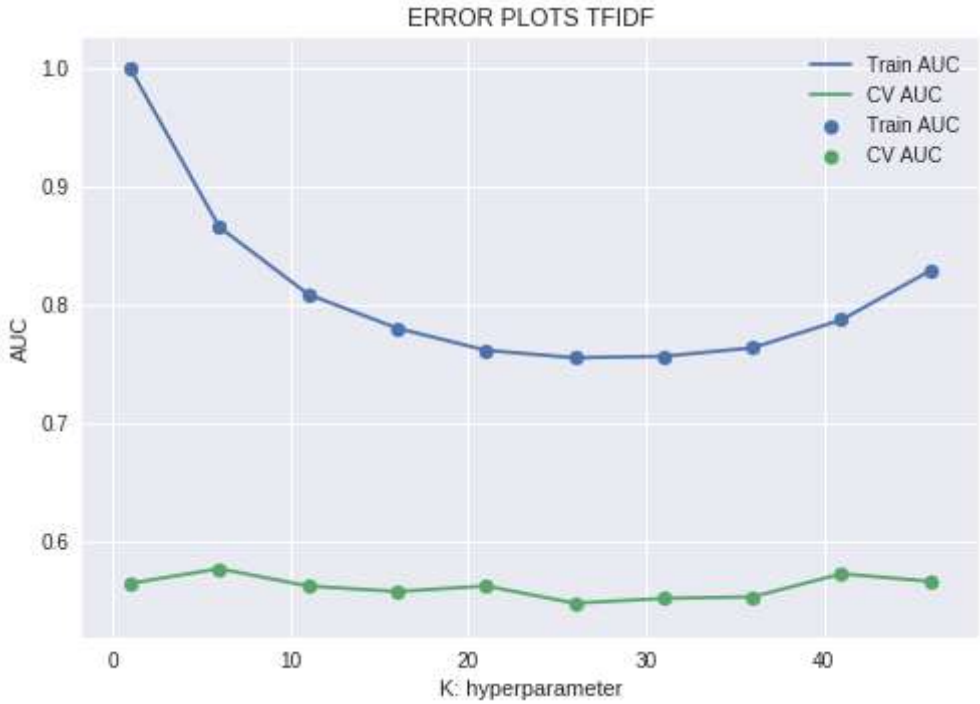
[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [0]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#Loading 20000 points into X,Y
X = preprocessed_reviews[0:20000]
Y = final['Score'].values[0:20000]

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#print(X_train.shape, Y_train.shape)
#print(X_cv.shape, Y_cv.shape)
#print(X_test.shape, Y_test.shape)

print("="*100)
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
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)
print("="*100)
```

```
==========================================================================
=======================
After vectorizations
(12000, 500) (12000,)
(4000, 500) (4000,)
(4000, 500) (4000,)
==========================================================================
=======================
```

Hyper parameter tuning TFIDF KD-TREE

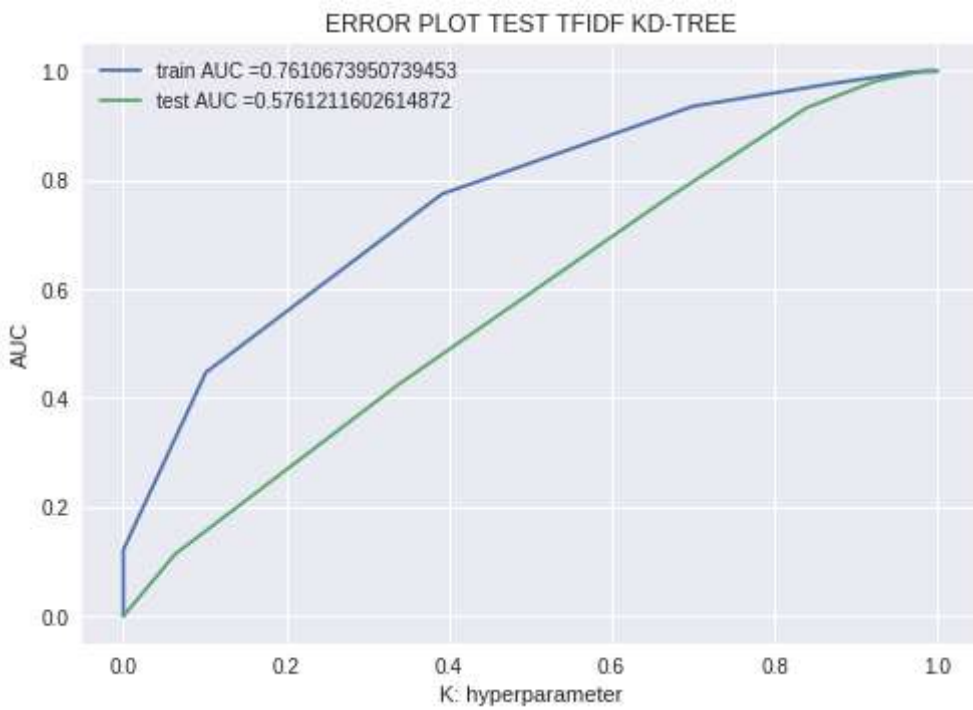
In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc_tf = []
cv_auc_tf = []
K = range(1,50,5)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(X_train_tf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tf)[:,1]

    train_auc_tf.append(roc_auc_score(y_train,y_train_pred))
    cv_auc_tf.append(roc_auc_score(y_cv, y_cv_pred))

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



In [0]:
```
best_k=21
```

Test with test data TFIDF KD-TREE

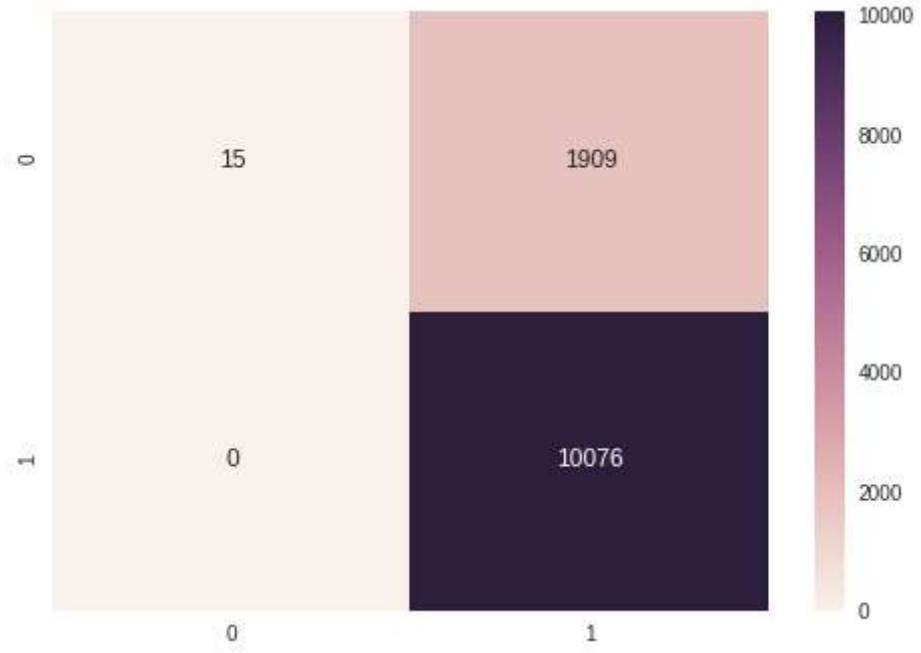
In [0]:
```
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_tr
ain_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_tp
r)))
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 PLOT TEST TFIDF KD-TREE")
plt.show()

print("="*100)
```



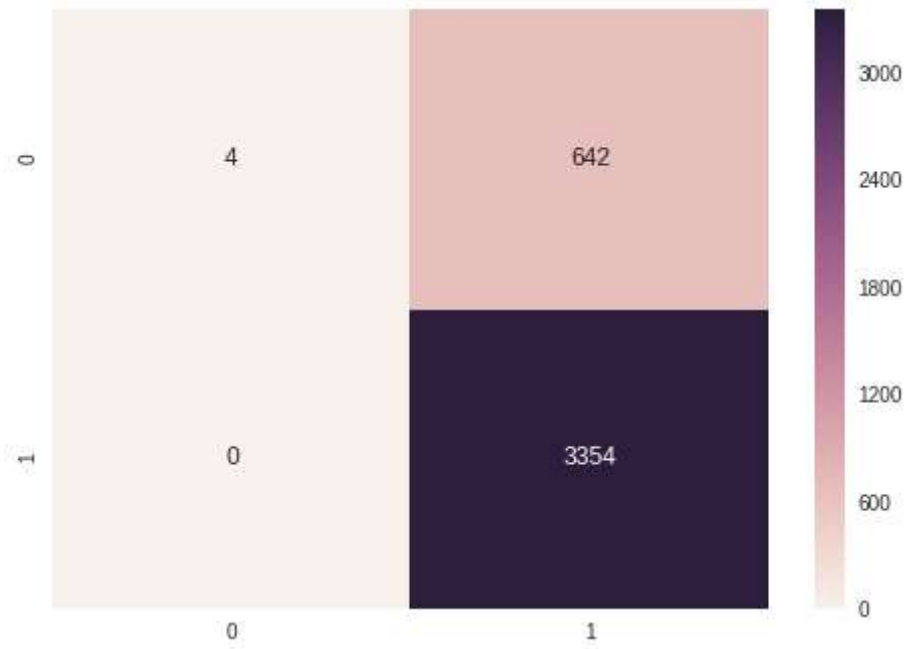

```
====================================================================================
========================
```

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(X_train_tf)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(X_test_tf)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



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

In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split
#Loading 20000 points into X,Y
X = preprocessed_reviews[0:20000]
Y = final['Score'].values[0:20000]

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, sh
uffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting

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

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

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  6647
sample words  ['beef', 'stew', 'almost', 'easy', 'make', 'admit', 'not', 'gre
at', 'seasonings', 'supplies', 'needed', 'spices', 'herbs', 'expensive', 'kee
p', 'putting', 'purchasing', 'list', 'keeps', 'getting', 'longer', 'eventuall
y', 'anything', 'work', 'planned', 'crock', 'pots', 'holidays', 'always', 'so
mething', 'hearty', 'warm', 'hand', 'family', 'came', 'went', 'lo', 'behold',
'finished', 'cutting', 'meat', 'veggies', 'loaded', 'nothing', 'actually', 'f
lavor', 'main', 'ingredients', 'sent', 'hubby']
```

Converting Reviews into Numerical Vectors using W2V vectors

Algorithm Average W2V

In [0]:
```
from tqdm import tqdm
import numpy as np

# converting train data text

# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
his list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might
need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|██████████| 12000/12000 [00:17<00:00, 670.86it/s]

(12000, 50)
[ 0.54461955 -0.20757164  0.02175738  0.20797643  0.18333802  0.04854276
 -0.45588623 -0.03854719 -0.21598298 -0.20736866 -0.32843685  0.17322387
 -0.23694588  0.33523099 -0.04355966  0.51453217 -0.42721338 -0.30704162
 -0.03880516 -0.14862375 -0.427977    0.16387746  0.16262336 -0.53495012
  0.24423521 -0.46447998  0.57438019 -0.3203047   0.52607501  0.34212611
  0.19013544  0.27845468  0.44731518 -0.80047406  0.18822067  0.35288975
  0.41359978 -0.21028322 -0.0060557   0.18985216 -0.22168091 -0.1246775
  0.55010039  0.06328048 -0.44213332 -0.14448465  0.13385749 -0.06979961
  0.32633421  0.26230269]
```

Converting CV data text

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

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

```
100%|██████████| 4000/4000 [00:06<00:00, 649.77it/s]

(4000, 50)
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
```

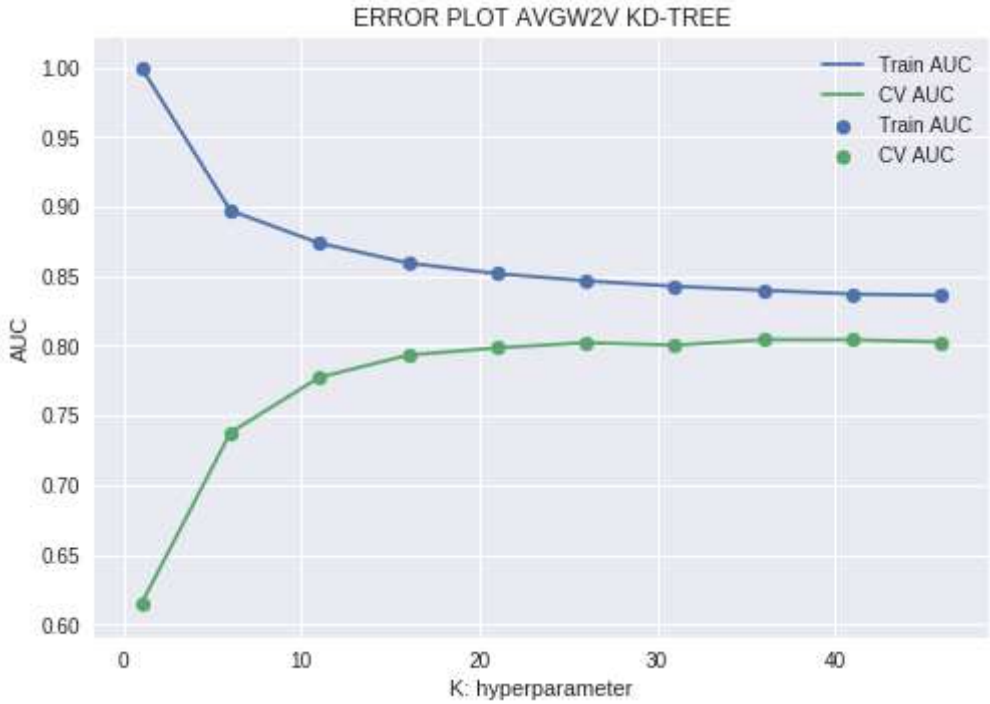
Converting Test data text

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

```
100%|██████████| 4000/4000 [00:06<00:00, 648.73it/s]

(4000, 50)
[ 0.45589239 -0.24459659  0.02367     0.27858559  0.6209352   0.56094986
 -0.69359772  0.06293064 -0.60175151 -0.48490833 -0.52433011  0.01735125
 -0.29704303  0.22989546  0.0131053   0.41887978  0.10922896 -0.49951675
 -0.33960553 -0.22653859 -0.00236882  0.30707247  0.14064264 -0.65429184
  0.41105296 -0.6121252   1.07457196 -0.52293084  0.13675529  0.40636085
 -0.02491634  0.21169928  0.90703775 -0.85690094  0.18312089 -0.19548247
  0.67162603 -0.31106986 -0.04409102  0.49070999 -0.29741538 -0.31839493
  0.81407568 -0.03448271 -0.40655181 -0.01334314  0.02565456  0.02165258
  0.70282039  0.32999196]
```

Hyper parameter Tuning AvgW2V KD-Treee

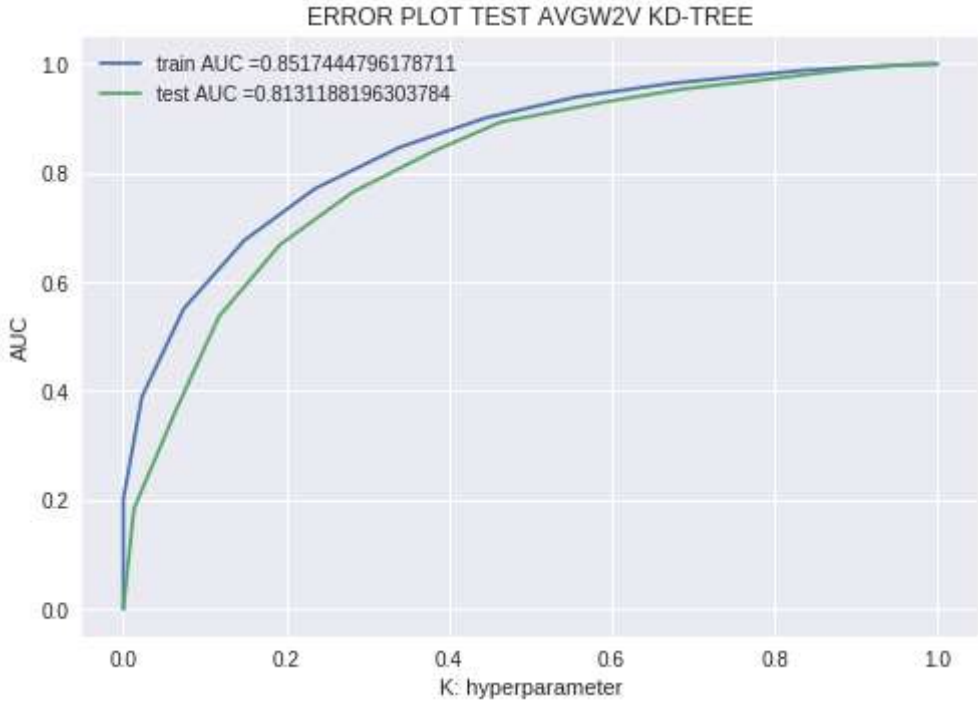
In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = range(1,50,5)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(sent_vectors_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(sent_vectors_train)[:,1]
    y_cv_pred =  neigh.predict_proba(sent_vectors_cv)[:,1]

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

plt.plot(K, train_auc, label='Train AUC')
plt.scatter(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.scatter(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT AVGW2V KD-TREE")
plt.show()
```



In [0]:
```
best_k=21
```

Testing with Text Data

In [0]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(sent_vectors_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(sent
_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(sent_ve
ctors_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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 PLOT TEST AVGW2V KD-TREE")
plt.show()

print("="*100)
```




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

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(sent_vectors_tra
in)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(sent_vectors_test
)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```

```
Test confusion matrix
```

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

In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tr
ain_test_split.html
from sklearn.model_selection import train_test_split
#Loading 20000 points into X,Y
X = preprocessed_reviews[0:20000]
Y = final['Score'].values[0:20000]

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # thi
s is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
25) # this is random splitting


#preparing reviews for genism model
i=0
list_of_sentance_tfkd_train=[]
for sentance in X_train:
    list_of_sentance_tfkd_train.append(sentance.split())



#Training w2v models

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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tfkd_train,min_count=5,size=50, workers=4)

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  6612
sample words  ['no', 'taste', 'heat', 'dish', 'turned', 'leathery', 'better',
'buying', 'high', 'price', 'truffles', 'elderly', 'pomeranian', 'loves', 'sen
ior', 'greenies', 'even', 'regular', 'absolutely', 'never', 'lets', 'us', 'fo
rget', 'clock', 'daily', 'treat', 'able', 'fully', 'understand', 'dogs', 'tel
l', 'time', 'know', 'good', 'surprisingly', 'accurate', 'little', 'bit', 'sof
ter', 'contain', 'chondroitin', 'glucosamine', 'joint', 'support', 'highly',
'recommended', 'typical', 'americans', 'eat', 'often']
```

In [0]:
```
from tqdm import tqdm
import numpy as np
```

In [0]:
```
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_)))
```

Converting Train data text to TFIDF vectors

In [0]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

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

```
100%|██████████| 12000/12000 [02:16<00:00, 88.13it/s]
```

Converting CV data Text

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

In [0]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

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

```
100%|██████████| 4000/4000 [00:45<00:00, 77.21it/s]
```

Converting TFIDF Test data text

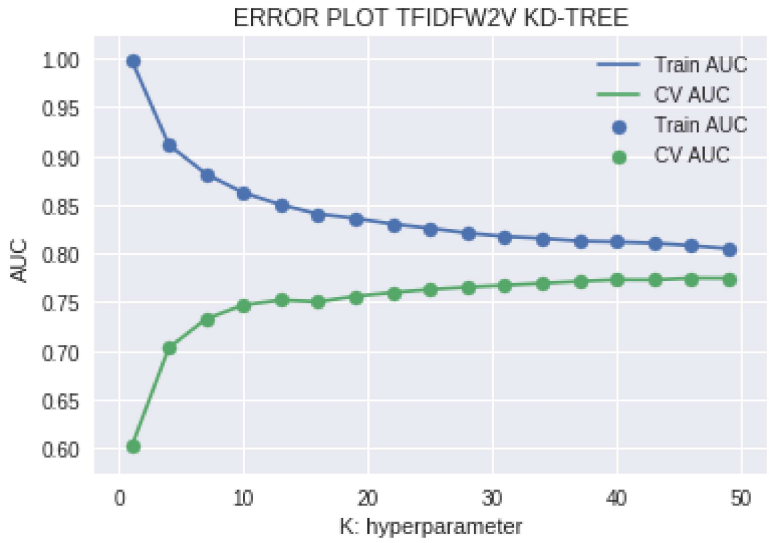
In [0]:
```
i=0
list_of_sentance_tfkd_test=[]
for sentance in X_test:
    list_of_sentance_tfkd_test.append(sentance.split())
```

In [0]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

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

```
100%|██████████| 4000/4000 [00:44<00:00, 89.07it/s]
```

Hyper parameter Tuning TFIDFW2V KD-TREE

In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the po3sitive class, con
fidence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = range(1,50,3)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(tfidf_sent_train_kd, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(tfidf_sent_train_kd)[:,1]
    y_cv_pred =  neigh.predict_proba(tfidf_sent_cv_kd)[:,1]

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

plt.plot(K, train_auc, label='Train AUC')
plt.scatter(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.scatter(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOT TFIDFW2V KD-TREE")
plt.show()
```



In [0]:
```
best_k=22
```

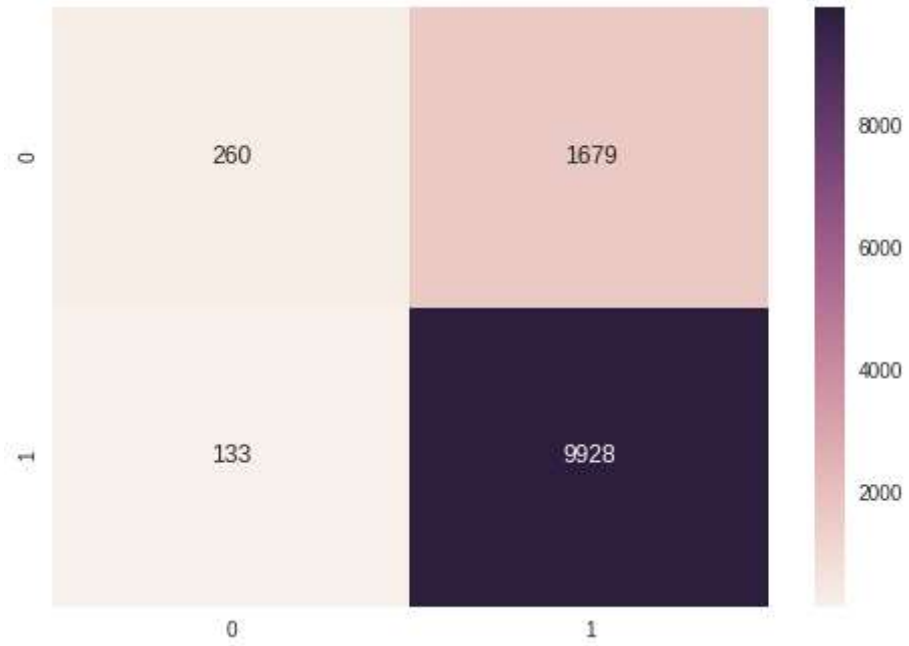
Testing with Test Data

In [0]:
```
neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='kd_tree')
neigh.fit(tfidf_sent_train_kd, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(tfid
f_sent_train_kd)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(tfidf_s
ent_test_kd)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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 PLOT TEST TFIDFW2V KD-TREE")
plt.show()

print("="*100)
```

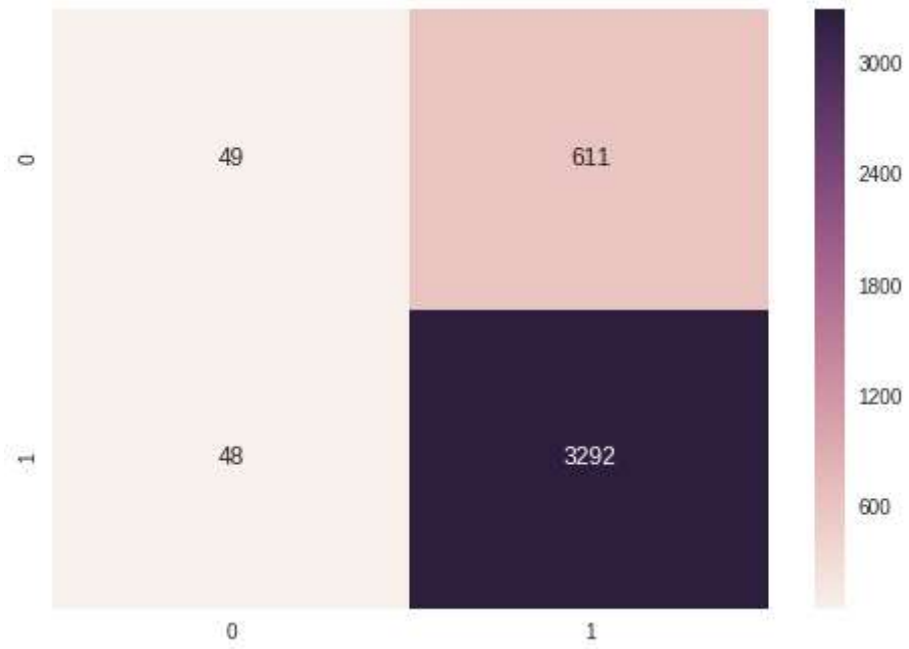
```
====================================================================================
=======================
```

In [0]:
```
#printing confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
print("Train confusion matrix")

df_train=pd.DataFrame(confusion_matrix(y_train, neigh.predict(tfidf_sent_train
_kd)))
sns.heatmap(df_train,annot=True,fmt="d")
plt.show()

print("Test confusion matrix")

df_test=pd.DataFrame(confusion_matrix(y_test,neigh.predict(tfidf_sent_test_kd
)))
sns.heatmap(df_test,annot=True,fmt="d")
plt.show()
```

```
Train confusion matrix
```




```
Test confusion matrix
```



[6] Conclusions

In [0]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper-Parameter", "AUC"]

x.add_row(["BOW", "BRUTE", 20,0.6644 ])
x.add_row(["TFIDF", "BRUTE",25,0.4974])
x.add_row(["W2V", "BRUTE", 25,0.8728])
x.add_row(["TFIDFW2V", "BRUTE",31,0.8424])
x.add_row(["BOW", "KD-TREE", 25,0.7157])
x.add_row(["TFIDF", "KD-TREE",21,0.5761])
x.add_row(["W2V", "KD-TREE", 21, 0.8131])
x.add_row(["TFIDFW2V", "KD-TREE",22,0.7535])


print(x)
```

```
+------------+---------+-----------------+--------+
| Vectorizer |  Model  | Hyper-Parameter |  AUC   |
+------------+---------+-----------------+--------+
|    BOW     |  BRUTE  |        20       | 0.6644 |
|   TFIDF    |  BRUTE  |        25       | 0.4974 |
|    W2V     |  BRUTE  |        25       | 0.8728 |
|  TFIDFW2V  |  BRUTE  |        31       | 0.8424 |
|    BOW     | KD-TREE |        25       | 0.7157 |
|   TFIDF    | KD-TREE |        21       | 0.5761 |
|    W2V     | KD-TREE |        21       | 0.8131 |
|  TFIDFW2V  | KD-TREE |        22       | 0.7535 |
+------------+---------+-----------------+--------+
```

Applying Brute force method on Average W2V yields the best AUC.