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.

In [1]:
```
#mounting google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649
1hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fww
w.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%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]:
```
#giving all permissions on files
!chmod 777 /content/drive/My\ Drive/applied\ ai\ assignment\ notebooks/k-nearest\ neighbors/database.sqlite
!chmod 777 /content/drive/My\ Drive/applied\ ai\ assignment\ notebooks/k-nearest\ neighbors/03\ Amazon\ Fine\ Food\ Re
views\ Analysis_KNN.ipynb
```

In [2]:
```
#installing scikit plot (used here for plotting confusion matrix)
!pip install scikit-plot
```

```
Requirement already satisfied: scikit-plot in /usr/local/lib/python3.6/dist-packages (0.3.7)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.6/dist-packages (from scikit-plot) (0.13.1)
Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/python3.6/dist-packages (from scikit-plot) (3.0.2)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.6/dist-packages (from scikit-plot) (0.20.
2)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.6/dist-packages (from scikit-plot) (1.1.0)
Requirement already satisfied: numpy>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.0->sciki
t-plot) (1.14.6)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (fr
om matplotlib>=1.4.0->scikit-plot) (2.3.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.0->s
cikit-plot) (1.0.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.0
->scikit-plot) (2.5.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.0->scikit
-plot) (0.10.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotl
ib>=1.4.0->scikit-plot) (40.8.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.1->matplot
lib>=1.4.0->scikit-plot) (1.11.0)
```

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [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, roc_auc_score
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

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
```

```
from sklearn.model_selection import cross_val_score
import matplotlib.cm as cm
```

In [0]:
```
# using SQLite Table to read data.
#con = sqlite3.connect('database.sqlite')
#con = sqlite3.connect('/content/drive/My\ Drive/applied\ ai\ assignment\ notebooks/k-nearest\ neighbors/database.sqli
te')
con=sqlite3.connect('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/database.sqlite')
# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3

#Selecting 100K points
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

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

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

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

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summar
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertise
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

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

In [0]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[0]:

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

In [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	HelpfulnessDenominator	Score	Time	Sum
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF

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, inplace=False, kind='quicksort', na_positio
n='last')
```

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

Out[0]: (87775, 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]: 87.775

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

In [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	HelpfulnessDenominator	Score	Time	Summa
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste wi crunchy almond inside

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

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

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

```
(87773, 10)
```

Out[0]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

In [0]:
```
#saving the cleaned data as database file

#con=sqlite3.connect('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/cleaneddata.sqlite')
#final.to_sql('Reviews',con)
```

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

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 they are out there, but this one isnt.  Its too bad too because its a good produ
ct but I wont take any chances till they know what is going on with the china imports.
```

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

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

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

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

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

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

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)
```

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

In [0]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

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

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

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', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'furthe
r',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
e',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
 \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "were
n'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 in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:35<00:00, 2501.99it/s]
```

In [0]:
```
preprocessed_reviews[100]
```

Out[0]:
```
'frenchbull given nylabone chew since weeks old safe strong bite not break large pieces could choke dinosaur chew per
fect many places hold bite dylabone product buy'
```

In [0]:
```
#saving the preprocessed reviews as pickle file for later use
with open('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/preprocessed_reviews.pkl','wb')
 as f:
    pickle.dump(preprocessed_reviews,f)
```

In [0]:
```
#saving the 100k sample dataset as database
#dbcon=sqlite3.connect('100kdataset_final.sqlite')
#final.to_sql('final100k',dbcon)
#dbcon.close()
```

[3.2] Preprocessing Review Summary

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

[4] Featurization

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/preprocessed_reviews.pkl','rb') as f:
  preprocessed_reviews=pickle.load(f)
with open('/content/drive/My Drive/Applied AI assignments/list_of_scores.pkl','rb') as f:
  scores=pickle.load(f)
```

In [0]:
```
X=preprocessed_reviews
y=scores
```

[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])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaahhhhhh', 'aaaaaaarrrrrg
gghhh', 'aaaaaawwwwwwwww', 'aaaaah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 54904)
the number of unique words  54904
```

[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/stable/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])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[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_feature_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])
```

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'abandon', 'abandoned', 'abdominal', 'abili
ty', 'able', 'able add', 'able brew']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 51709)
the number of unique words including both unigrams and bigrams  51709
```

[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/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.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 ")
```

```
[('fantastic', 0.8574245572090149), ('awesome', 0.8441421389579773), ('good', 0.8213924169540405), ('wonderful', 0.79
99011874198914), ('excellent', 0.7962841391563416), ('terrific', 0.7957799434661865), ('perfect', 0.750534415245056
2), ('amazing', 0.7475045919418335), ('nice', 0.7322636842727661), ('decent', 0.6936917304992676)]
==================================================
[('greatest', 0.8162376880645752), ('best', 0.7067530155181885), ('tastiest', 0.7064868211746216), ('disgusting', 0.6
875180602073669), ('nastiest', 0.6804093718528748), ('horrible', 0.6466028690338135), ('closest', 0.624045431613922
1), ('experienced', 0.6234358549118042), ('freshest', 0.621214747428894), ('vile', 0.6140969395637512)]
```

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])
```

```
number of words that occured minimum 5 times  17386
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore', 'hard', 'find', 'product
s', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'imports', 'love', 'sa
w', 'pet', 'store', 'tag', 'attached', 'regarding', 'satisfied', 'safe', 'infestation', 'literally', 'everywhere', 'f
lying', 'around', 'kitchen', 'bought', 'hoping', 'least', 'get', 'rid', 'weeks', 'fly', 'stuck', 'squishing', 'bugger
s', 'success', 'rate']
```

In [0]:
```
#Saving all the above generated vectors as pickle file
with open('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/final_counts_BoW.pkl','wb') as
f:
    pickle.dump(final_counts,f)
with open('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/final_bigram_counts.pkl','wb')
as f:
    pickle.dump(final_bigram_counts,f)
with open('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/final_tf_idf.pkl','wb') as f:
    pickle.dump(final_tf_idf,f)
with open('/content/drive/My Drive/applied ai assignment notebooks/k-nearest neighbors/list_of_sentance.pkl','wb') as
f:
    pickle.dump(list_of_sentance,f)
```

[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 list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
oogle'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]))
```

```
100%|██████████| 87773/87773 [03:27<00:00, 422.60it/s]

87773
50
```

[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
```

In [0]:
```
with open('avgw2v_vec.pkl','wb') as f:
    pickle.dump(sent_vectors,f)
with open('tfidfw2v_vec.pkl','wb') as f:
    pickle.dump(tfidf_sent_vectors,f)
```

[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)

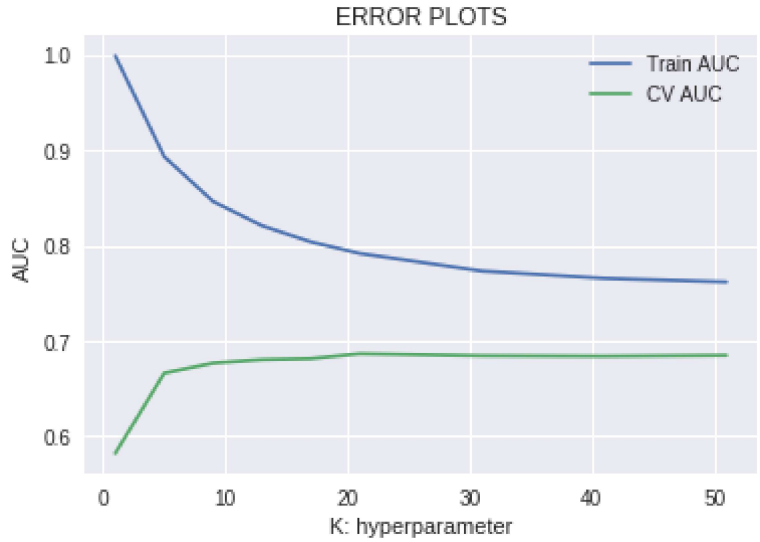
-## [5.1] Applying KNN brute force

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

In [0]:
```
#splitting the data into train,test and cross-validation datasets
#following simple cross validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3) # this is random splitting
```

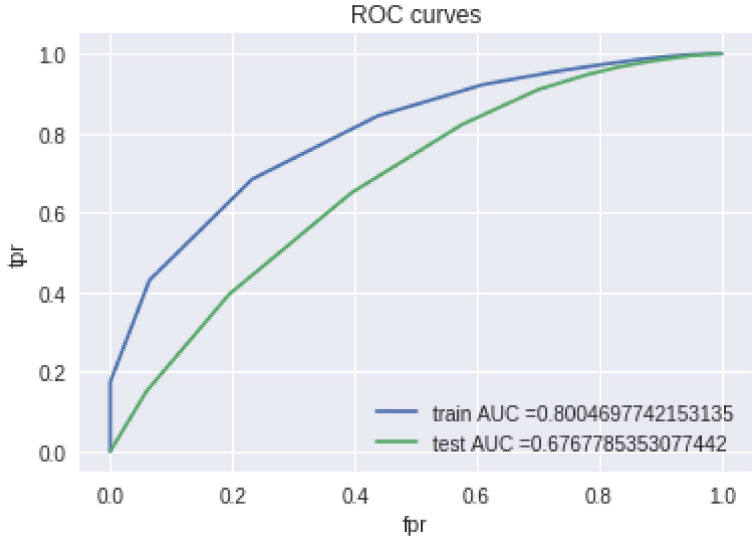
In [0]:
```
#vectorizing train,test and cv datasets using BoW
count_vect = CountVectorizer()
count_vect.fit(X_train)
X_train_bow=count_vect.transform(X_train)
X_test_bow=count_vect.transform(X_test)
X_cv_bow=count_vect.transform(X_cv)
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,50) if x%2!=0]

neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
  knn.fit(X_train_bow,y_train)

  y_train_pred=knn.predict_proba(X_train_bow)[:,1]
  y_cv_pred=knn.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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

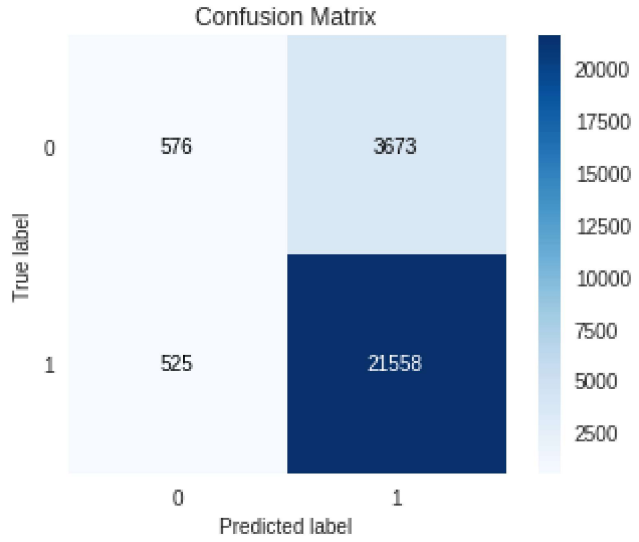


In [0]:
```
#from the above auc plots, the cv has max AUC from k=13 onwards but train AUC and cv AUC are close at k=51
#therfore best 'k' can be 51
best_k=13
```

In [0]:
```
#ROC curves for train and test data when trained on best k

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_bow)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



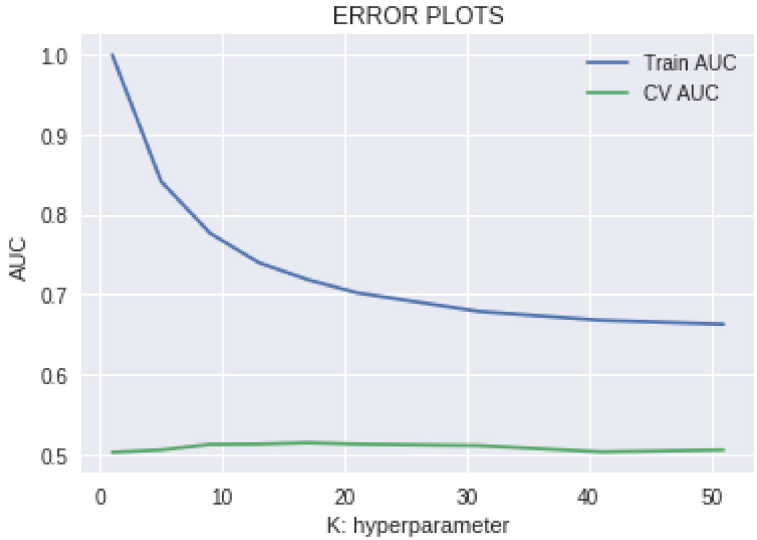
[5.1.2] Applying KNN brute force on TFIDF, SET 2

In [0]:
```
X=preprocessed_reviews
y=scores
```

In [0]:
```
#splitting the data into train,test and cross-validation datasets
#following simple cross validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3) # this is random splitting
```

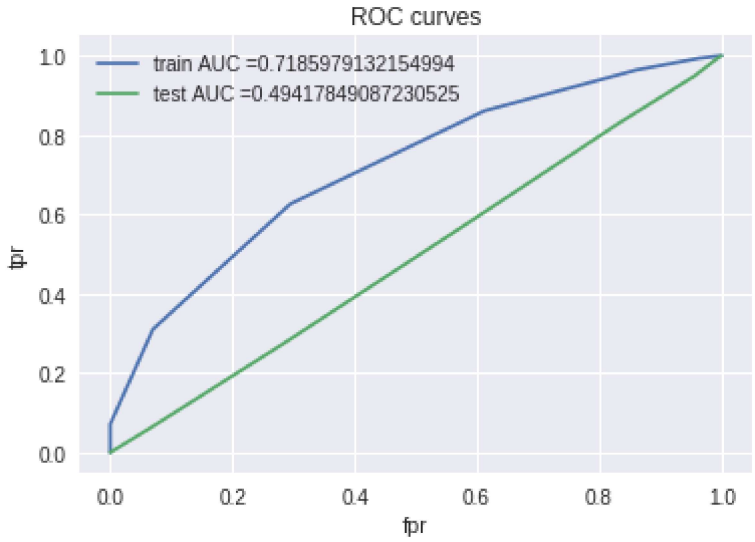
In [0]:
```
#vectorizing train,test,cv datasets using TF-IDF

tfidf_vect=TfidfVectorizer(ngram_range=(1,2), min_df=10)
X_train_tfidf=tfidf_vect.fit_transform(X_train)
X_test_tfidf=tfidf_vect.transform(X_test)
X_cv_tfidf=tfidf_vect.transform(X_cv)
```

In [11]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
  knn.fit(X_train_tfidf,y_train)

  y_train_pred=knn.predict_proba(X_train_tfidf)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_tfidf)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

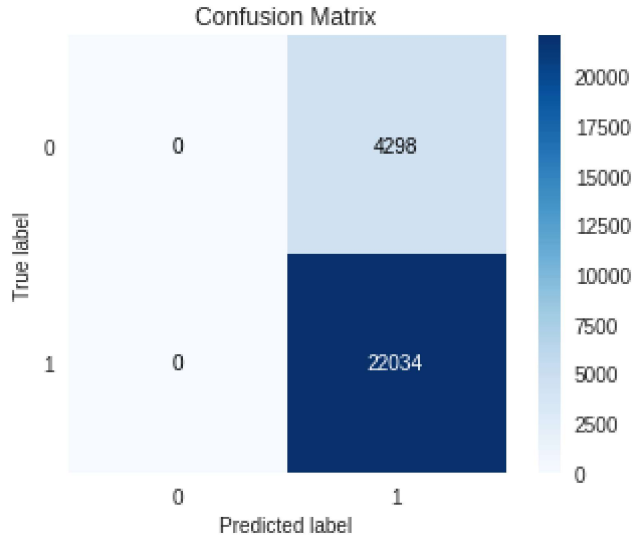


In [0]:
```
#from the above AUC plots, we observe that CV AUC is nearly constant and both train AUC and CV AUC are close at k=51
best_k=17
```

In [13]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_tfidf)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



Observations

For the tested 'k' values, the above model on tfidf vectorization is completely underfitting i.e it has complete bias towards positive class

Including new feature (length of reviews) to improve the performace of the model on tfidf vectors

In [0]:
```
X=[[review,len(review)] for review in preprocessed_reviews]
y=scores
```

In [0]:
```
#splitting the data into train,test and cross-validation datasets
#following simple cross validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3) # this is random splitting
```

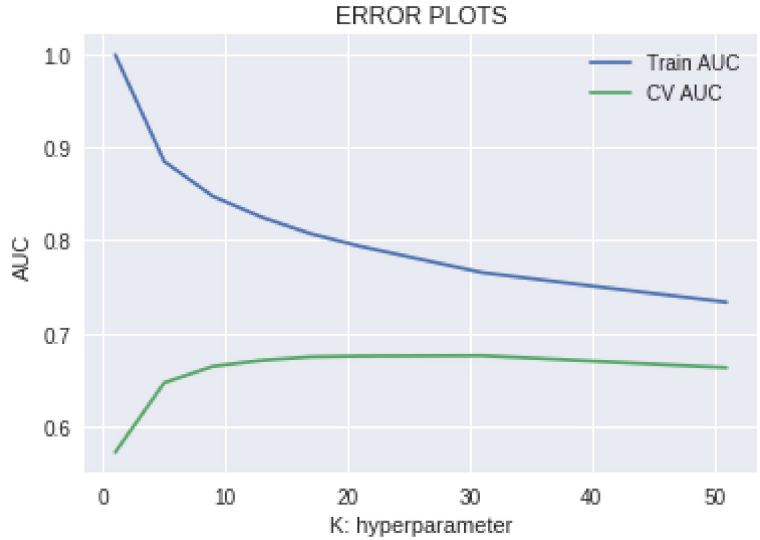
In [0]:
```
#seperating train reviews and their corresponding lenghts from a list of list
X_train_reviews=[]
X_train_review_len=[]
for l in X_train:
  X_train_reviews.append(l[0])
  X_train_review_len.append(l[1])
X_test_reviews=[]
X_test_review_len=[]
for l in X_test:
  X_test_reviews.append(l[0])
  X_test_review_len.append(l[1])
X_cv_reviews=[]
X_cv_review_len=[]
for l in X_cv:
  X_cv_reviews.append(l[0])
  X_cv_review_len.append(l[1])
```

In [0]:
```
#vectorizing train,test,cv datasets using TF-IDF

tfidf_vect=TfidfVectorizer(ngram_range=(1,2), min_df=10)
X_train_tfidf=tfidf_vect.fit_transform(X_train_reviews)
X_test_tfidf=tfidf_vect.transform(X_test_reviews)
X_cv_tfidf=tfidf_vect.transform(X_cv_reviews)
```

In [0]:
```
#converting list to numpy array (column vector)
X_train_review_len=np.array(X_train_review_len).reshape(len(X_train_review_len),1)
X_test_review_len=np.array(X_test_review_len).reshape(len(X_test_review_len),1)
X_cv_review_len=np.array(X_cv_review_len).reshape(len(X_cv_review_len),1)
```

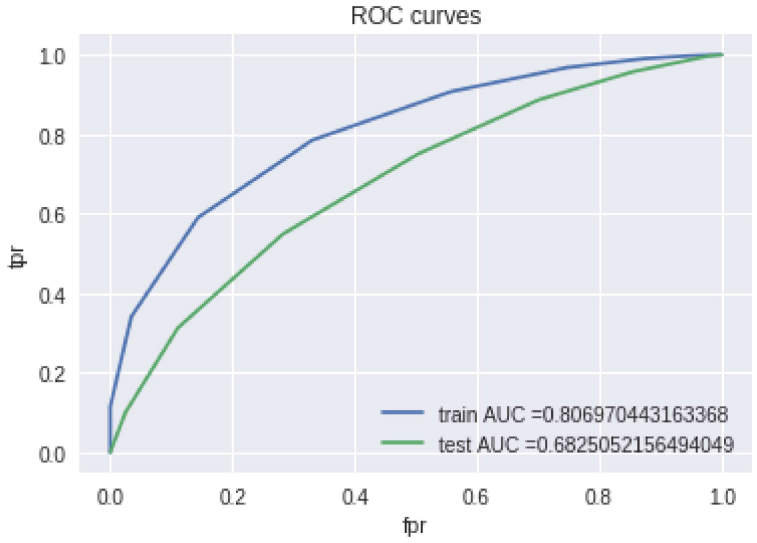
In [0]:
```
#using hstack from scipy for horizantal stacking review lengths with tfidf vectors
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.hstack.html

from scipy.sparse import hstack

X_train_tfidf_len_included=hstack([X_train_tfidf,X_train_review_len])
X_test_tfidf_len_included=hstack([X_test_tfidf,X_test_review_len])
X_cv_tfidf_len_included=hstack([X_cv_tfidf,X_cv_review_len])
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
  knn.fit(X_train_tfidf_len_included,y_train)

  y_train_pred=knn.predict_proba(X_train_tfidf_len_included)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_tfidf_len_included)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

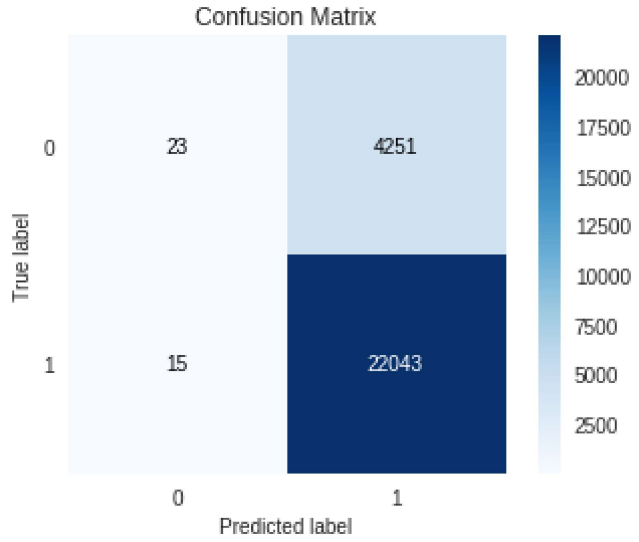


In [0]:
```
best_k=17
```

In [40]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_tfidf_len_included)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_tfidf_len_included)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [44]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_tfidf_len_included)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



Observations We observe that the performace of model is slightly improved when an extra feature (review length) is considered.

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

In [0]:
```
# list of tokenized sentences
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
#splitting the dataset into train,test and cross validation dataset
X_train,X_test,y_train,y_test=train_test_split(list_of_sentance,y,test_size=0.3,random_state=0)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.3,random_state=0)
```

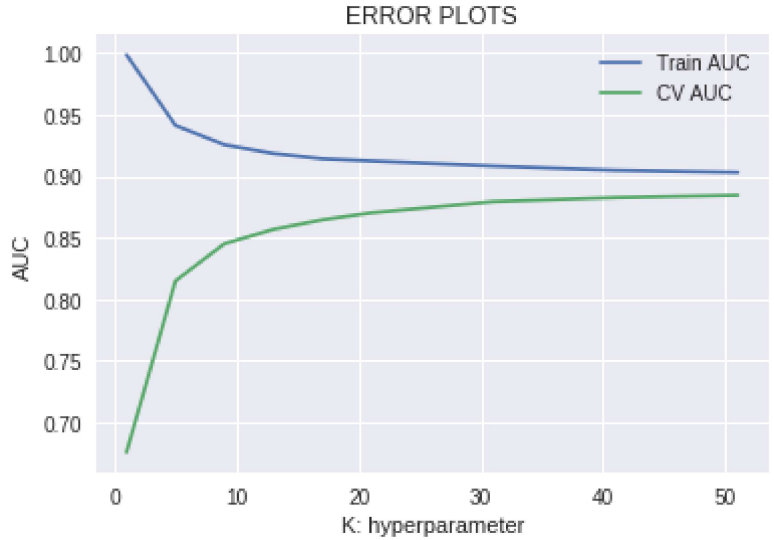
In [0]:
```
#W2V models for train, test and CV sets

X_train_w2v_model=Word2Vec(X_train,min_count=5,size=50,workers=4)
#X_test_w2v_model=Word2Vec(X_test,min_count=5,size=50,workers=4)
#X_cv_w2v_model=Word2Vec(X_cv,min_count=5,size=50,workers=4)
```

In [0]:
```
#vectorizing X_train using Avg W2V
X_train_w2v_words = list(X_train_w2v_model.wv.vocab)
X_train_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_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 g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_train_sent_vectors.append(sent_vec)

#vectorizing X_test using Avg W2V
#X_test_w2v_words = list(X_test_w2v_model.wv.vocab)
X_test_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_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 g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_test_sent_vectors.append(sent_vec)


#vectorizing X_cv using Avg W2V
#X_cv_w2v_words = list(X_cv_w2v_model.wv.vocab)
X_cv_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_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 g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_cv_sent_vectors.append(sent_vec)
```

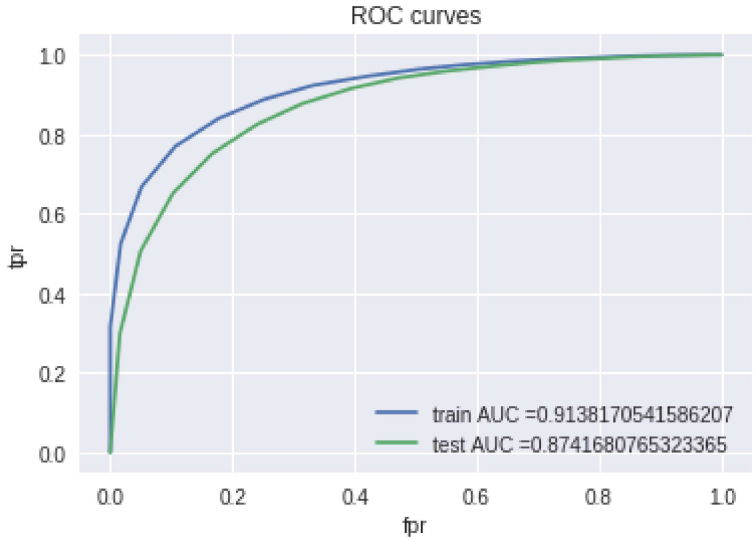
```
100%|██████████| 43008/43008 [01:31<00:00, 471.91it/s]
100%|██████████| 26332/26332 [00:55<00:00, 471.38it/s]
100%|██████████| 18433/18433 [00:39<00:00, 467.83it/s]
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
  knn.fit(X_train_sent_vectors,y_train)

  y_train_pred=knn.predict_proba(X_train_sent_vectors)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_sent_vectors)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

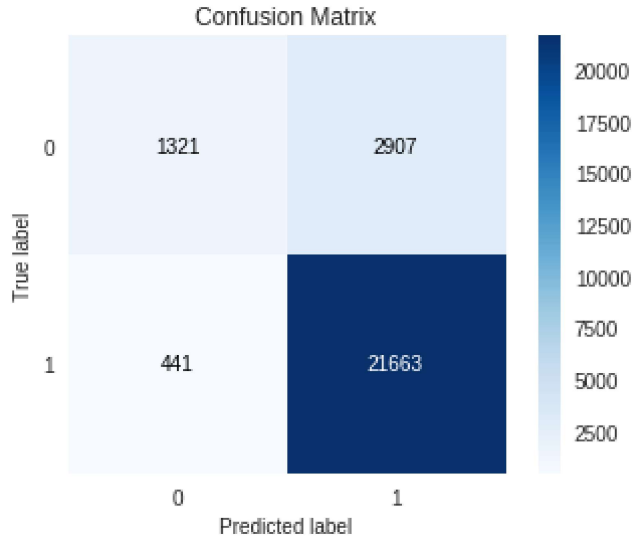


In [0]:
```
best_k=21
```

In [0]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_sent_vectors)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_sent_vectors)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



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

In [0]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.3,random_state=0)
```

In [0]:
```
model=TfidfVectorizer()
model.fit(X_train)
X_train_tfidf=model.transform(X_train)
X_test_tfidf=model.transform(X_test)
X_cv_tfidf=model.transform(X_cv)
```

In [0]:
```
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
#vectorizing X_train using tfidf w2v
X_train_list_of_sents=[]
for sent in X_train:
  X_train_list_of_sents.append(sent.split())

#build w2v model for X_train
X_train_w2v_model=Word2Vec(X_train_list_of_sents,min_count=5,size=50,workers=4)
words=list(X_train_w2v_model.wv.vocab)

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

X_train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_train_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 43008/43008 [13:51<00:00, 51.73it/s]
```

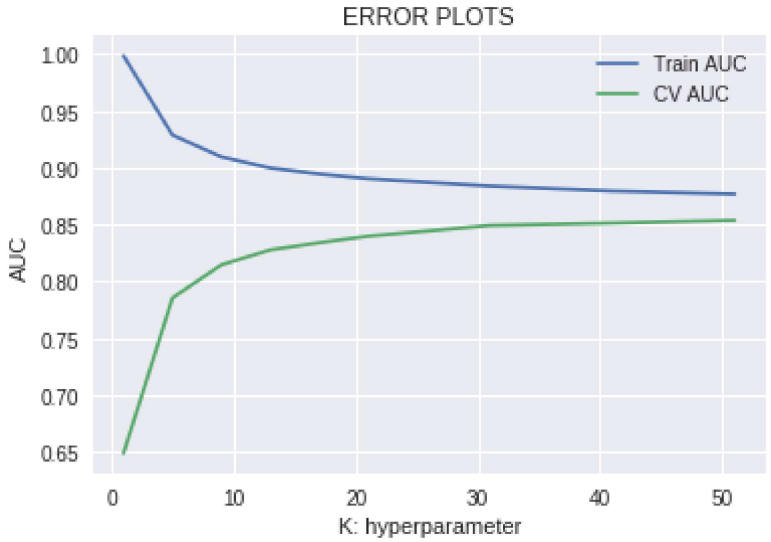
In [0]:
```
#vectorizing test dataset

X_test_list_of_sents=[]
for sent in X_test:
  X_test_list_of_sents.append(sent.split())

#build w2v model for X_test
#X_test_w2v_model=Word2Vec(X_test_list_of_sents,min_count=5,size=50,workers=4)
#X_test_words=list(X_test_w2v_model.wv.vocab)


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

X_test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_test_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 26332/26332 [08:25<00:00, 52.12it/s]
```

In [0]:
```
#vectorizing test dataset

X_cv_list_of_sents=[]
for sent in X_cv:
  X_cv_list_of_sents.append(sent.split())

#build w2v model for X_test
#X_cv_w2v_model=Word2Vec(X_cv_list_of_sents,min_count=5,size=50,workers=4)
#X_cv_words=list(X_cv_w2v_model.wv.vocab)


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

X_cv_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_cv_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_cv_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

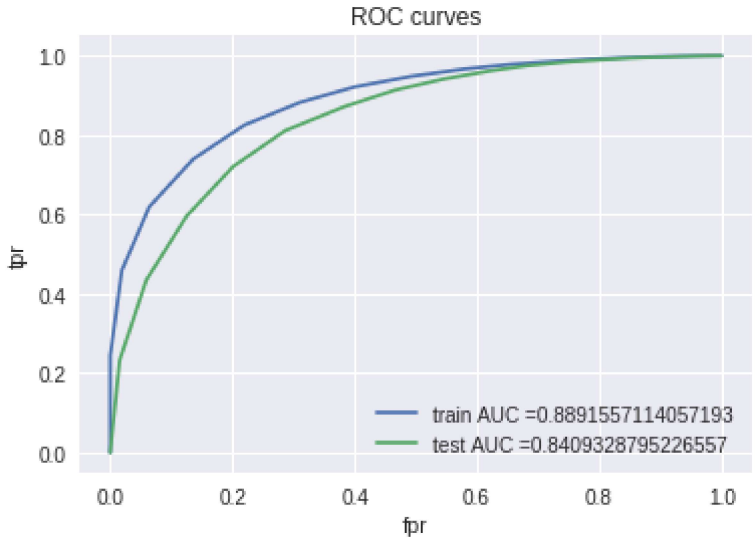
```
100%|██████████| 18433/18433 [05:57<00:00, 51.61it/s]
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,50) if x%2!=0]

neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
  knn.fit(X_train_tfidf_sent_vectors,y_train)

  y_train_pred=knn.predict_proba(X_train_tfidf_sent_vectors)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_tfidf_sent_vectors)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

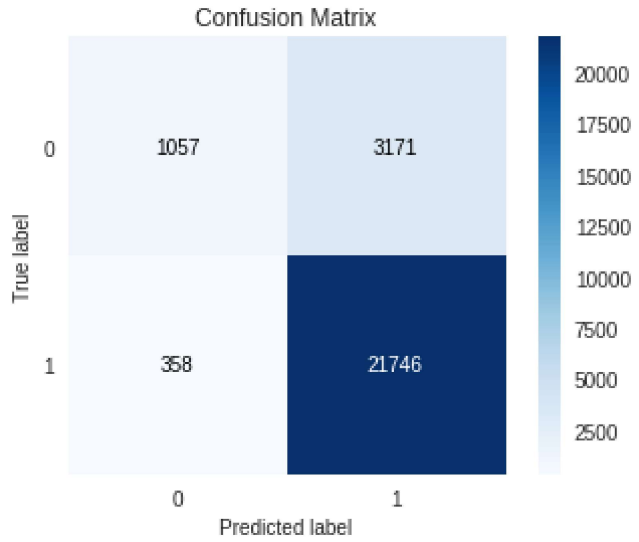


In [0]:
```
#from error plots, choose 'k' such that CV AUC is maximum and train AUC, CV AUC are close
best_k=21
```

In [0]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_tfidf_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_tfidf_sent_vectors)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_tfidf_sent_vectors)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



[5.2] Applying KNN kd-tree

Taking 40K reviews for kd-tree algorithm

In [0]:
```
#getting 40k points for kd-tree implementation

X=preprocessed_reviews[:40000]
y=scores[:40000]
```

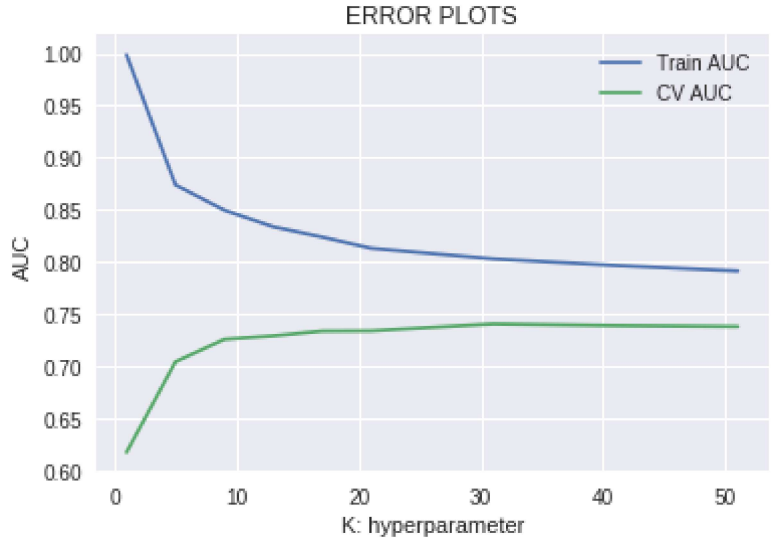
In [0]:
```
#splitting the data into train,test and cross-validation datasets
#following simple cross validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3) # this is random splitting
```

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

In [0]:
```
#vectorizing using BoW
count_vect = CountVectorizer(min_df=10, max_features=500)
count_vect.fit(X_train)
X_train_bow=count_vect.transform(X_train)
X_test_bow=count_vect.transform(X_test)
X_cv_bow=count_vect.transform(X_cv)
```

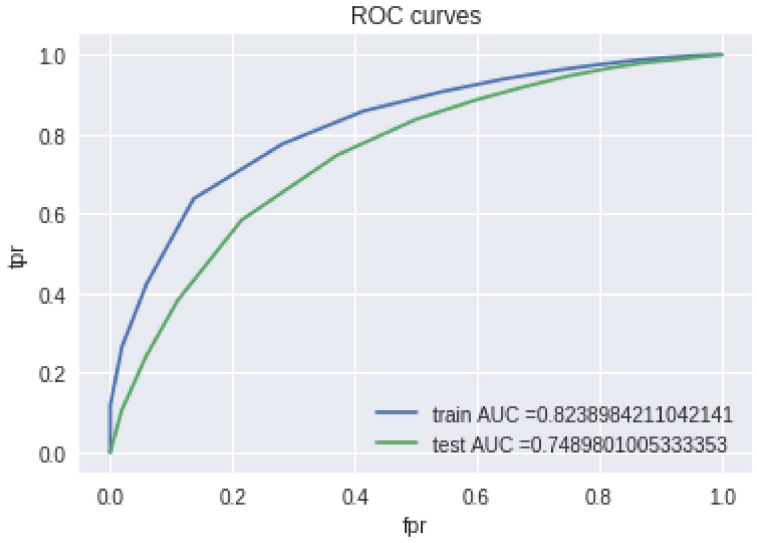
In [0]:
```
#converting sparse matrices into dense matrices as kd-tree algorithm accepts dense matrices

X_train_bow=X_train_bow.toarray()
X_test_bow=X_test_bow.toarray()
X_cv_bow=X_cv_bow.toarray()
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
  knn.fit(X_train_bow,y_train)

  y_train_pred=knn.predict_proba(X_train_bow)[:,1]
  y_cv_pred=knn.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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

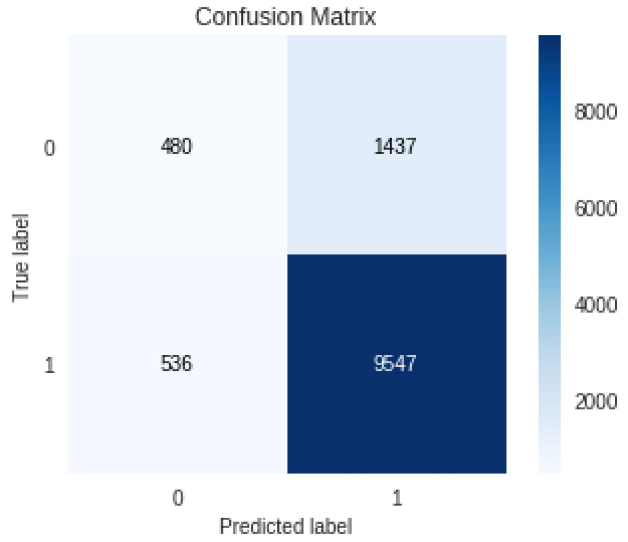


In [0]:
```
best_k=17
```

In [0]:
```
#ROC curves for train and test data when trained on best k

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_bow)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



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

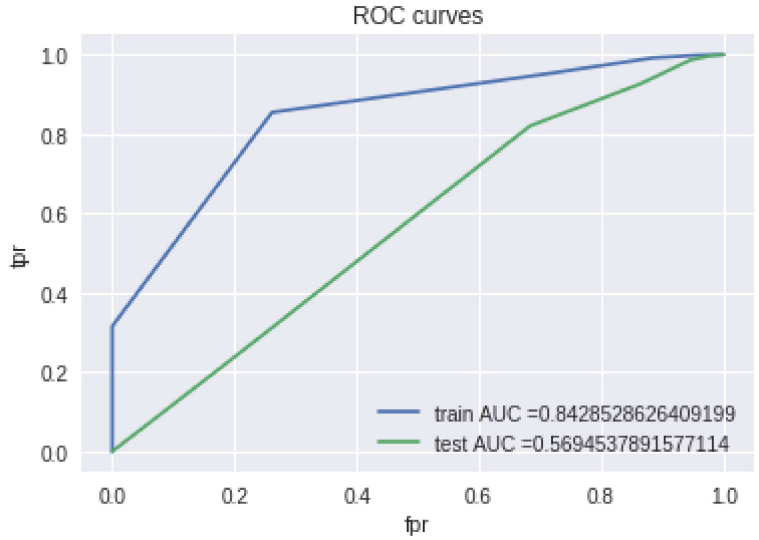
In [0]:
```
#vectorizing train,test,cv datasets using TF-IDF

tfidf_vect=TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
X_train_tfidf=tfidf_vect.fit_transform(X_train)
X_test_tfidf=tfidf_vect.transform(X_test)
X_cv_tfidf=tfidf_vect.transform(X_cv)
```

In [0]:
```
#converting sparse matrices into dense matrices as kd-tree algorithm accepts dense matrices

X_train_tfidf=X_train_tfidf.toarray()
X_test_tfidf=X_test_tfidf.toarray()
X_cv_tfidf=X_cv_tfidf.toarray()
```

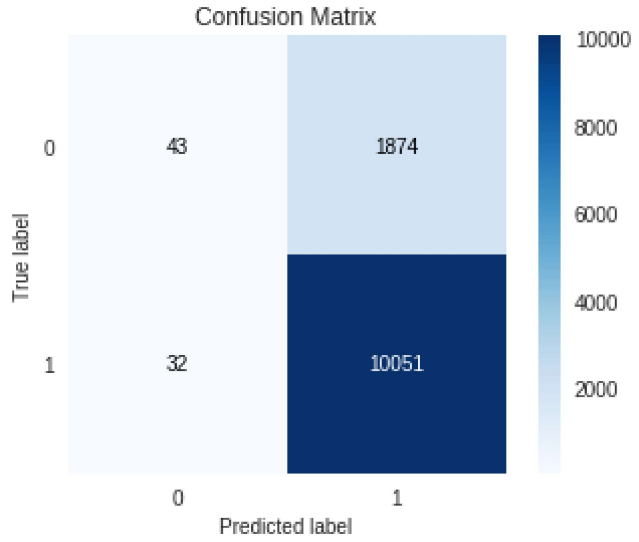
In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
  knn.fit(X_train_tfidf,y_train)

  y_train_pred=knn.predict_proba(X_train_tfidf)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_tfidf)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

In [0]:
```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the train and cv is l
ess
best_k=9
```

In [0]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_tfidf)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



Incluing the new feature (review length) to improve the performace of the model on tfidf vectors

In [0]:
```
#getting 40k points for kd-tree implementation

preprocessed_reviews=preprocessed_reviews[:40000]
scores=scores[:40000]
```

In [0]:
```
X=[[review,len(review)] for review in preprocessed_reviews]
y=scores
```

In [0]:
```
#splitting the data into train,test and cross-validation datasets
#following simple cross validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3) # this is random splitting
```

In [0]:
```
#seperating train reviews and their corresponding lenghts from a list of list
X_train_reviews=[]
X_train_review_len=[]
for l in X_train:
  X_train_reviews.append(l[0])
  X_train_review_len.append(l[1])
X_test_reviews=[]
X_test_review_len=[]
for l in X_test:
  X_test_reviews.append(l[0])
  X_test_review_len.append(l[1])
X_cv_reviews=[]
X_cv_review_len=[]
for l in X_cv:
  X_cv_reviews.append(l[0])
  X_cv_review_len.append(l[1])
```

In [0]:
```
#vectorizing train,test,cv datasets using TF-IDF

tfidf_vect=TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
X_train_tfidf=tfidf_vect.fit_transform(X_train_reviews)
X_test_tfidf=tfidf_vect.transform(X_test_reviews)
X_cv_tfidf=tfidf_vect.transform(X_cv_reviews)
```

In [0]:
```
#converting list to numpy array (column vector)
X_train_review_len=np.array(X_train_review_len).reshape(len(X_train_review_len),1)
X_test_review_len=np.array(X_test_review_len).reshape(len(X_test_review_len),1)
X_cv_review_len=np.array(X_cv_review_len).reshape(len(X_cv_review_len),1)
```

In [0]:
```
#using hstack from scipy for horizantal stacking review lengths with tfidf vectors
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.hstack.html

from scipy.sparse import hstack

X_train_tfidf_len_included=hstack([X_train_tfidf,X_train_review_len])
X_test_tfidf_len_included=hstack([X_test_tfidf,X_test_review_len])
X_cv_tfidf_len_included=hstack([X_cv_tfidf,X_cv_review_len])
```

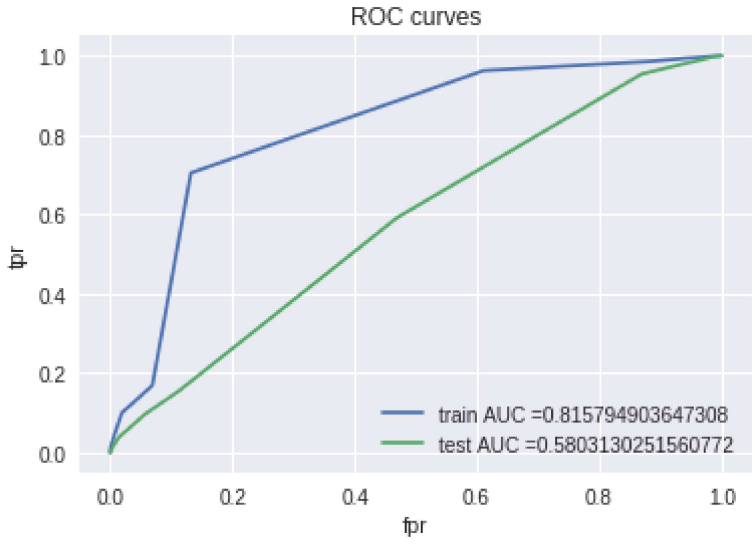
In [0]:
```
#converting sparse matrices into dense matrices as kd-tree algorithm accepts dense matrices

X_train_tfidf_len_included=X_train_tfidf.toarray()
X_test_tfidf_len_included=X_test_tfidf.toarray()
X_cv_tfidf_len_included=X_cv_tfidf.toarray()
```

In [12]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in tqdm(neighbors):
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
  knn.fit(X_train_tfidf_len_included,y_train)

  y_train_pred=knn.predict_proba(X_train_tfidf_len_included)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_tfidf_len_included)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

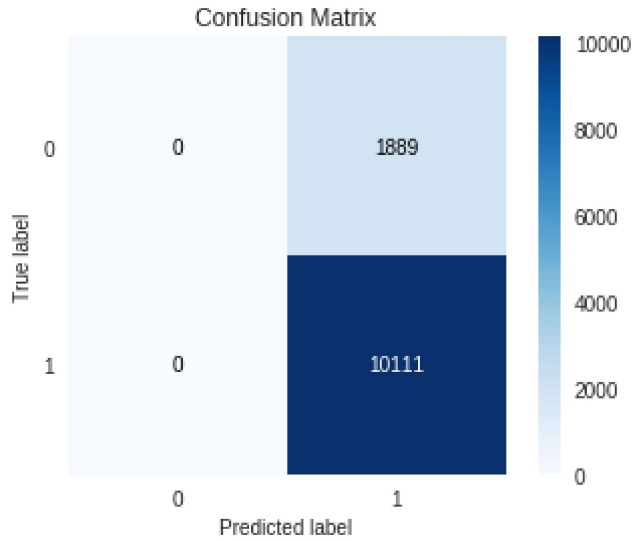
```
100%|██████████| 9/9 [1:29:21<00:00, 614.06s/it]
```

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

In [14]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_tfidf_len_included)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_tfidf_len_included)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [15]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_tfidf_len_included)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



Observation

We observe that even after the including the new feature (review length) there is not much change in the performance of the model. So it's better to consider the previous model.

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

In [0]:
```
preprocessed_reviews=preprocessed_reviews[:40000]
y=scores[:40000]
```

In [0]:
```
# list of tokenized sentences
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
#splitting the dataset into train,test and cross validation dataset
X_train,X_test,y_train,y_test=train_test_split(list_of_sentance,y,test_size=0.3,random_state=0)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.3,random_state=0)
```

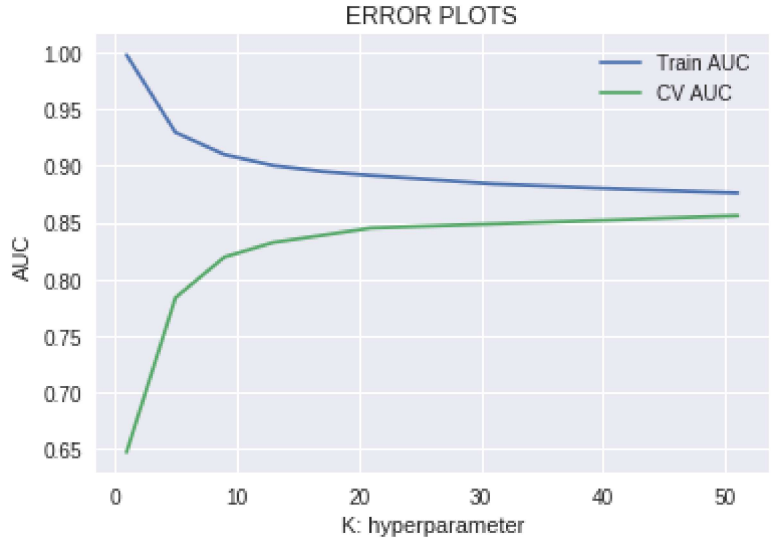
In [0]:
```
#W2V models for train, test and CV sets

X_train_w2v_model=Word2Vec(X_train,min_count=5,size=50,workers=4)
#X_test_w2v_model=Word2Vec(X_test,min_count=5,size=50,workers=4)
#X_cv_w2v_model=Word2Vec(X_cv,min_count=5,size=50,workers=4)
```

In [0]:
```
#vectorizing X_train using Avg W2V
X_train_w2v_words = list(X_train_w2v_model.wv.vocab)
X_train_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_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 g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_train_sent_vectors.append(sent_vec)

#vectorizing X_test using Avg W2V
#X_test_w2v_words = list(X_test_w2v_model.wv.vocab)
X_test_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_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 g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_test_sent_vectors.append(sent_vec)


#vectorizing X_cv using Avg W2V
#X_cv_w2v_words = list(X_cv_w2v_model.wv.vocab)
X_cv_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_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 g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_cv_sent_vectors.append(sent_vec)
```

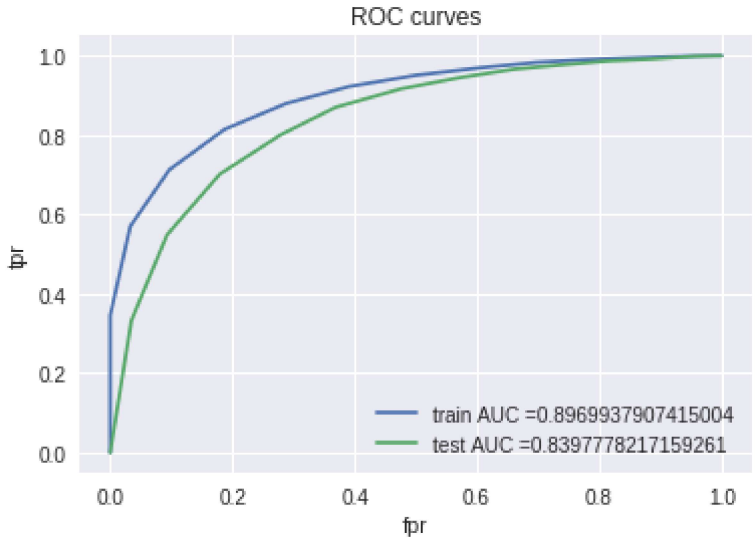
```
100%|██████████| 19600/19600 [00:34<00:00, 564.31it/s]
100%|██████████| 12000/12000 [00:22<00:00, 536.56it/s]
100%|██████████| 8400/8400 [00:15<00:00, 557.96it/s]
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,30) if x%2!=0]
neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
  knn.fit(X_train_sent_vectors,y_train)

  y_train_pred=knn.predict_proba(X_train_sent_vectors)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_sent_vectors)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

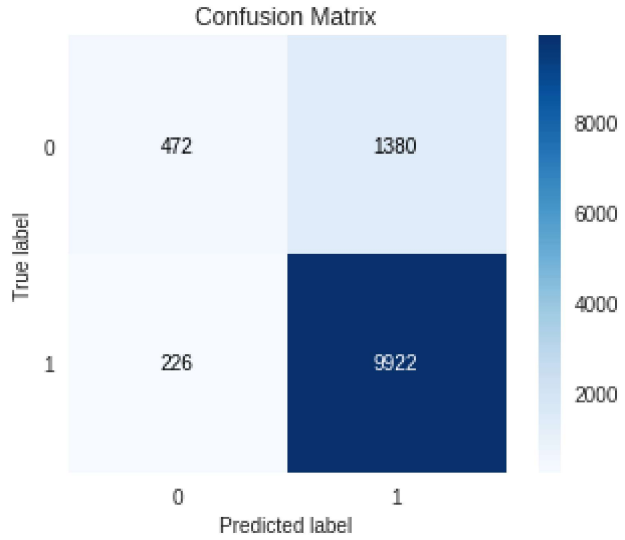


In [0]:
```
best_k=17
```

In [0]:
```
#ROC curves for train and test data when trained on best k


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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_sent_vectors)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_sent_vectors)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



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

In [0]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.3,random_state=0)
```

In [0]:
```
model=TfidfVectorizer()
model.fit(X_train)
X_train_tfidf=model.transform(X_train)
X_test_tfidf=model.transform(X_test)
X_cv_tfidf=model.transform(X_cv)
```

In [0]:
```
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
#vectorizing X_train using tfidf w2v
X_train_list_of_sents=[]
for sent in X_train:
  X_train_list_of_sents.append(sent.split())

#build w2v model for X_train
X_train_w2v_model=Word2Vec(X_train_list_of_sents,min_count=5,size=50,workers=4)
X_train_words=list(X_train_w2v_model.wv.vocab)

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

X_train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_train_list_of_sents): # 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 X_train_words and word in tfidf_feat:
            vec = X_train_w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 19600/19600 [04:59<00:00, 65.54it/s]
```

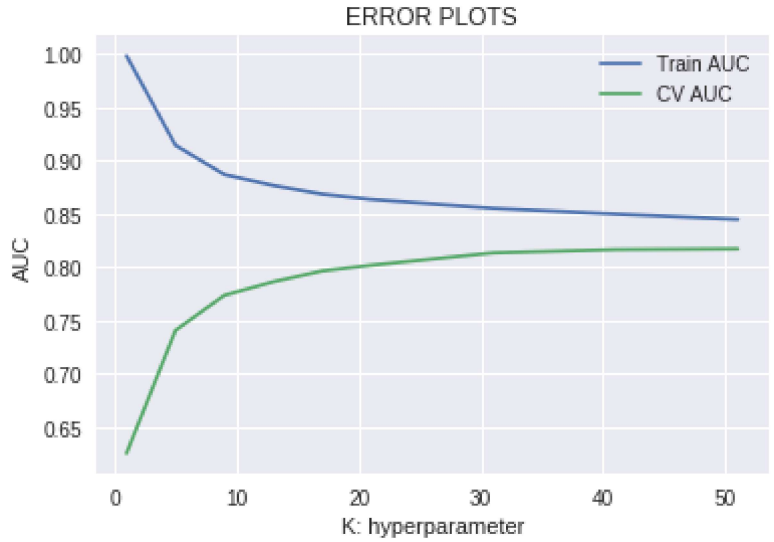
In [0]:
```
#vectorizing test dataset

X_test_list_of_sents=[]
for sent in X_test:
  X_test_list_of_sents.append(sent.split())

#build w2v model for X_test
#X_test_w2v_model=Word2Vec(X_test_list_of_sents,min_count=5,size=50,workers=4)
#X_test_words=list(X_test_w2v_model.wv.vocab)


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

X_test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_test_list_of_sents): # 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 X_train_words and word in tfidf_feat:
            vec = X_train_w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 12000/12000 [03:06<00:00, 64.51it/s]
```

In [0]:
```
#vectorizing test dataset

X_cv_list_of_sents=[]
for sent in X_cv:
  X_cv_list_of_sents.append(sent.split())

#build w2v model for X_test
#X_cv_w2v_model=Word2Vec(X_cv_list_of_sents,min_count=5,size=50,workers=4)
#X_cv_words=list(X_cv_w2v_model.wv.vocab)


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

X_cv_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_cv_list_of_sents): # 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 X_train_words and word in tfidf_feat:
            vec = X_train_w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_cv_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

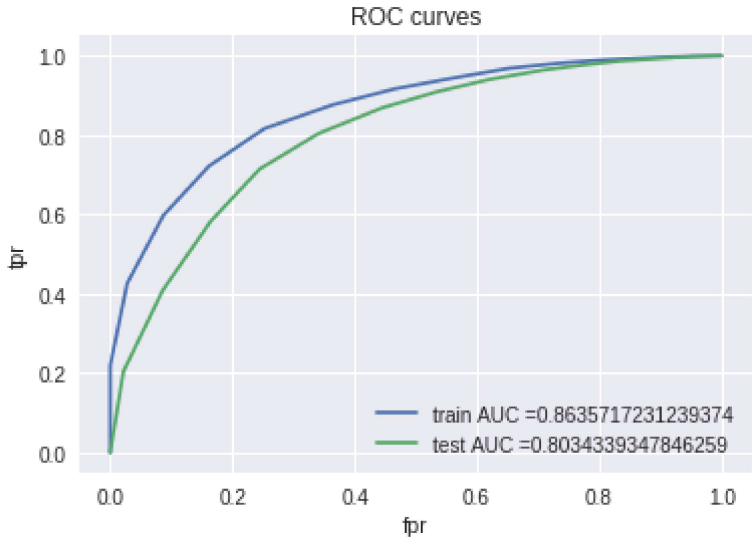
```
100%|██████████| 8400/8400 [02:04<00:00, 67.23it/s]
```

In [0]:
```
train_auc=[]
cv_auc=[]

#neighbors=[x for x in range(1,50) if x%2!=0]

neighbors=[1,5,9,13,17,21,31,41,51]

for k in neighbors:
  knn=KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
  knn.fit(X_train_tfidf_sent_vectors,y_train)

  y_train_pred=knn.predict_proba(X_train_tfidf_sent_vectors)[:,1]
  y_cv_pred=knn.predict_proba(X_cv_tfidf_sent_vectors)[:,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(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

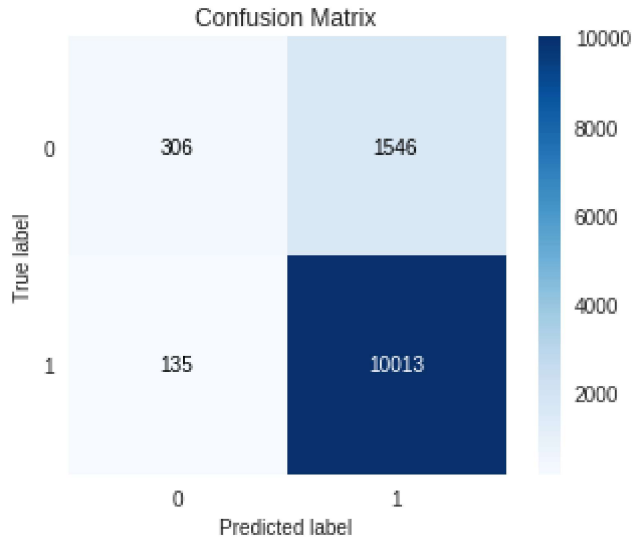


In [0]:
```
#from error plots, choose 'k' such that CV AUC is maximum and train AUC, CV AUC are close
best_k=21
```

In [0]:
```
#ROC curves for train and test data when trained on best k

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

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_tfidf_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_tfidf_sent_vectors)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]:
```
#confusion matrix
#code taken from https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
import scikitplot as skplt
y_pred=knn.predict(X_test_tfidf_sent_vectors)
skplt.metrics.plot_confusion_matrix(y_test,y_pred,normalize=False)
plt.show()
```



[6] Conclusions

In [16]:
```
#Installing pretty table
!pip install PrettyTable
```

```
Requirement already satisfied: PrettyTable in /usr/local/lib/python3.6/dist-packages (0.7.2)
```

In [17]:
```
#code refered from : http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Algorithm", "Optimal k", "Test AUC"]

x.add_row(["BoW", 'brute', 13, 0.68])
x.add_row(["Tfidf", 'brute', 17, 0.68])
x.add_row(["Avg W2V", 'brute', 21, 0.87])
x.add_row(["Tfidf W2V", 'brute', 21, 0.84])
x.add_row(["BoW", 'kd_tree', 17, 0.74])
x.add_row(["Tfidf", 'kd_tree', 9, 0.56])
x.add_row(["Avg W2V", 'kd_tree', 17, 0.83])
x.add_row(["Tfidf W2V",'kd_tree', 21, 0.80])

print(x)
```

```
+------------+-----------+-----------+----------+
| Vectorizer | Algorithm | Optimal k | Test AUC |
+------------+-----------+-----------+----------+
|    BoW     |   brute   |     13    |   0.68   |
|   Tfidf    |   brute   |     17    |   0.68   |
|  Avg W2V   |   brute   |     21    |   0.87   |
| Tfidf W2V  |   brute   |     21    |   0.84   |
|    BoW     |  kd_tree  |     17    |   0.74   |
|   Tfidf    |  kd_tree  |     9     |   0.56   |
|  Avg W2V   |  kd_tree  |     17    |   0.83   |
| Tfidf W2V  |  kd_tree  |     21    |   0.8    |
+------------+-----------+-----------+----------+
```