Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWa
rning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/Excel/Desktop/vins/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 50
0000 data points
# you can change the number to any other number based on your computing
 power
```

```python
# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Sco
re != 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 LIMIT 50000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text

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

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

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

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

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

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

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

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

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

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

Out[9]:
```
(46072, 10)
```

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

Out[10]:
```
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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

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

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

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

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

```
(46071, 10)
```

Out[13]:
```
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 [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)
```

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

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

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

```
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 bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, 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 tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast).  I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:<br /><br />-Quality:  Fi
rst, this tea offers a smooth quality without any harsh or bitter after
tones, which often turns people off from many green teas.  I've found m
y ideal brewing time to be between 3-5 minutes, giving you a light 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 g
```

```
uality of this tea is such that you still get a smooth but deeper flavo
r 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 bags also include chunks of tropical fruit and o
ther 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 wonder what it is you are actually drinking.<br
/><br />-Taste:  This tea offers notes of real pineapple and other hint
s of tropical fruits, yet isn't sweet or artificially flavored.  You ha
ve the foundation of a high-quality young hyson green tea for those tru
e "tea flavor" lovers, yet the subtle hints of fruit make this a truly
unique tea that I believe most will enjoy.  If you want it sweet, you c
an add sugar, splenda, etc but this really is not necessary as this tea
offers an inherent warmth of flavor through it's ingredients.<br /><br
/>-Price:  This tea offers an excellent product at an exceptional price
(especially when purchased at the prices Amazon offers).  Compared to o
ther brands which I believe to be of similar quality (Mighty Leaf, Rish
i, Two Leaves, etc.), Revolution offers a superior product at an outsta
nding price.  I have been purchasing this through Amazon for less per b
ox than I would be paying at my local grocery store for Lipton, etc.<br
/><br />Overall, this is a wonderful tea that is comparable, and even b
etter than, other teas that are priced much higher.  It offers a well-b
alanced cup of green tea that I believe many will enjoy.  In terms of t
aste, quality, and price, I would argue you won't find a better combina
tion that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [15]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
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 bec
```

```
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [16]:

```python
# 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 bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
```

great bargain 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 briefly highlight what makes this tea attractive to a wide range of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through over 12 boxes of this tea myself, and highly recommend it for the following reasons:-Quality: First, this tea offers a smooth quality without 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, giving you a light but flavorful cup of tea. However, if you get distracted or forget about your 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 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 wonder what it is you are actually drinking.-Taste: This tea offers notes 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 make this a truly unique tea 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 excellent product at an exceptional price (especially when purchased at the prices Amazon offers). Compared to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product 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, etc.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 green tea that I believe many will enjoy. In terms of taste, quality, and price, I would argue you wo

```
n't find a better combination that that offered by Revolution's Tropica
l Green Tea.
```

In [17]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

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

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

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
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 bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

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

```
Great flavor low in calories high in nutrients high in protein Usually
protein powders are high priced and high in calories this one is a grea
t bargain and tastes great I highly recommend for the lady gym rats pro
bably not macho enough for guys since it is soy based
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
```

```
                'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
                "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
                'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████| 46071/46071 [00:25<00:00, 182
6.37it/s]
```

In [24]:
```
preprocessed_reviews[1500]
```

Out[24]:
```
'great flavor low calories high nutrients high protein usually protein
powders high priced high calories one great bargain tastes great highly
recommend lady gym rats probably not macho enough guys since soy based'
```

[3.2] Preprocessing Review Summary

In [25]:
```
final["Time"] = pd.to_datetime(final["Time"], unit = "s")
final = final.sort_values(by = "Time")
```

[4] Featurization

[4.1] AVG-W2V(BRUTE_FORCE)

In [26]:
```
X = preprocessed_reviews
```

In [27]:
```
Y=final["Score"]
Y.shape
```

Out[27]:
```
(46071,)
```

In [28]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33)


print(np.shape(X_train), y_train.shape)

print(np.shape(X_cv), y_cv.shape)
print(np.shape(X_test), y_test.shape)
```

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
```

In [29]:
```
i=0
list_of_sentance=[]
```

```
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [30]:
```
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())
```

In [31]:
```
sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

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

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

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

In [32]:
```
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

cv_vectors = [];
for sent in sent_of_cv:
```

```
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)




# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)
```

In [33]:
```
X_train=train_vectors
```

In [34]:
```
np.shape(X_train)
```

Out[34]: (20680, 50)

In [35]:
```
X_cv=cv_vectors
```

In [36]:
```
np.shape(X_cv)
```

Out[36]: (10187, 50)

In [37]:
```
X_test=test_vectors
```

In [38]:
```
np.shape(X_test)
```

Out[38]: (15204, 50)

In [39]:
```
final["preprocessed_reviews"]=preprocessed_reviews
```

In [40]:
```
length_text=final["preprocessed_reviews"].apply(lambda x: len(str(x).sp
lit(" ")))
length_text
```

Out[40]:
```
1146      27
1145      12
28086     13
28087     10
38740     39
38889     17
38888     34
10992     17
28085      9
39671    135
48952     81
24061     36
24220     33
7427      12
25005     31
10116     37
3481      42
42951     83
45634     43
6790     105
36849     41
37320     37
16330     35
20580     35
```

```
36851      8
36853     46
17537     12
10994     52
1112      82
13578     85
        ...
14526     74
7156      15
42269     35
1005      57
45413     15
30235     15
41621     53
30998     10
13539     30
9         54
7451      32
43268     34
25112     22
22401     94
24496     70
7620     246
5472      84
6548      21
39050     52
9513      32
8731      25
37074     14
29158     19
38043     20
32585      9
19181     13
14299     16
14300     17
16026     29
5259      19
Name: preprocessed_reviews, Length: 46071, dtype: int64
```

In [41]:
```
type(length_text)
```
Out[41]: pandas.core.series.Series

In [42]:
```
d=(length_text)
df=pd.DataFrame(d)
```

In [43]:
```
type(df)
```
Out[43]: pandas.core.frame.DataFrame

In [44]:
```
df_train=df[:20680]
```

In [45]:
```
np.shape(df_train)
```
Out[45]: (20680, 1)

In [46]:
```
np.shape(X_train)
```
Out[46]: (20680, 50)

In [47]:
```
X_train=np.concatenate((X_train,df_train),axis=1)
```

In [48]:
```
np.shape(X_train)
```
Out[48]: (20680, 51)

In [49]:
```
np.shape(X_cv)
```
Out[49]: (10187, 50)

In [50]:
```
df_cv=df[20680:30867]
```

In [51]:
```
np.shape(df_cv)
```
Out[51]: (10187, 1)

In [52]:
```
X_cv=np.concatenate((X_cv,df_cv),axis=1)
```

In [53]:
```
np.shape(X_cv)
```

Out[53]: (10187, 51)

In [54]:
```
np.shape(X_test)
```

Out[54]: (15204, 50)

In [55]:
```
df_test=df[30867:46071]
```

In [56]:
```
np.shape(df_test)
```

Out[56]: (15204, 1)

In [57]:
```
X_test=np.concatenate((X_test,df_test),axis=1)
```

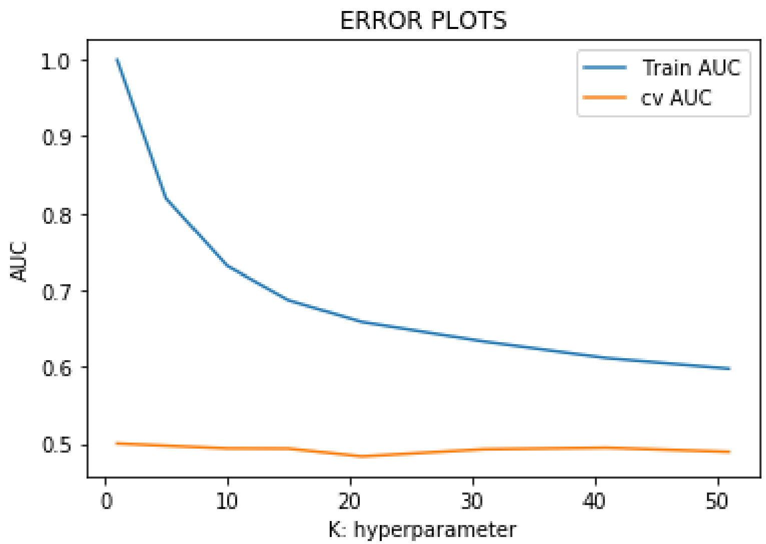
In [58]:
```
np.shape(X_test)
```

Out[58]: (15204, 51)

FINDING THE BEST K USING SIMPLE FORLOOP

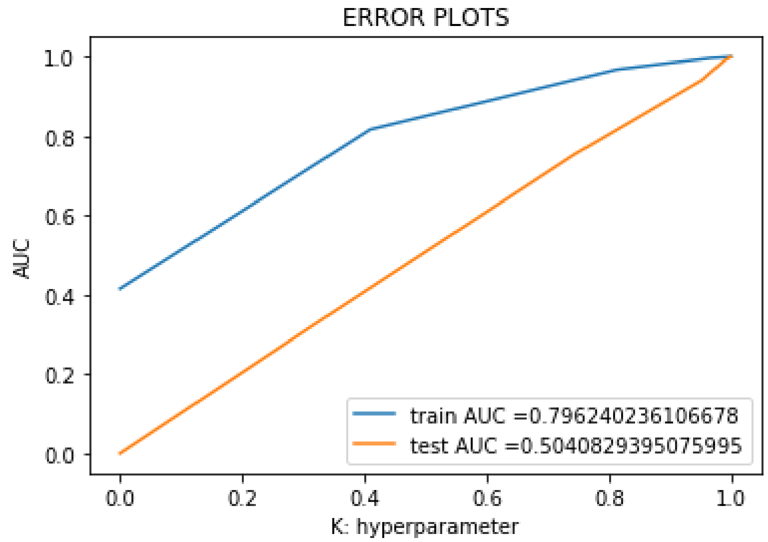
In [59]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(X_train, y_train)
    y_train_pred = []
```

```
    n = len(X_train)
    for i in range(0 ,n, 1000):
        y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1
])
    n = len(X_cv)
    y_cv_pred = []
    for i in range(0 ,n, 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv[i:i+1000])[:,1])
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='cv AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [67]:
```
from sklearn.neighbors import  KNeighborsClassifier
# pick my best k as 10
neigh = KNeighborsClassifier(n_neighbors=6,algorithm='brute')
neigh.fit(X_train, y_train)
```

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs
y_train_pred = []
n=len(X_train)
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1])
y_test_pred=[]
n=len(X_test)
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict_proba(X_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, y_test_pred)

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



observation: my model is not so good

confusion matrix

In [68]:
```
from sklearn.metrics import confusion_matrix
print("train confusion matrix")
y_train_pred = []
n=len(X_train)
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict(X_train[i:i+1000]))

y_test_pred=[]
n=len(X_test)
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict(X_test[i:i+1000]))
cm_trainw2v=confusion_matrix(y_train,y_train_pred)
cm_testw2v=confusion_matrix(y_test,y_test_pred)
print(cm_trainw2v)
print("="*100)
```

```python
print("test confusion matrix")
print(cm_testw2v)
```

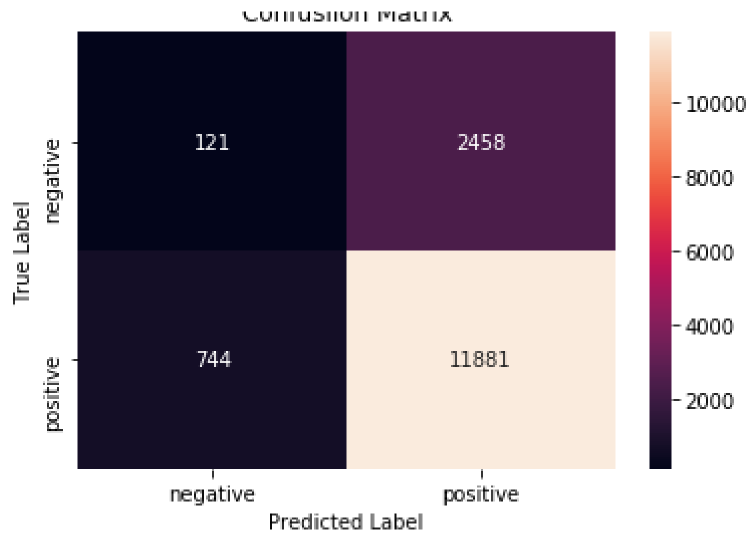
```
train confusion matrix
[[  621  2707]
 [  562 16790]]
===========================================================================
=============================
test confusion matrix
[[  121  2458]
 [  744 11881]]
```

In [69]:

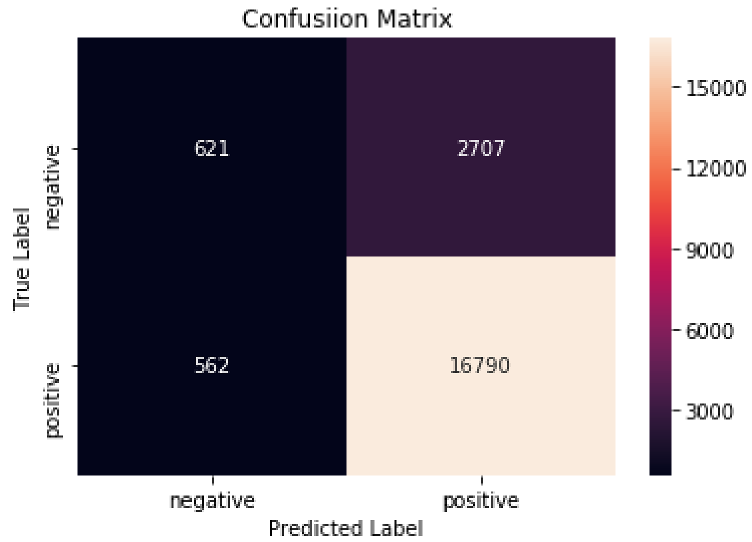
```python
import seaborn as sns
print("TEST CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_testw2v, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("TRAIN CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_trainw2v, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
TEST CONFUSION MATRIX
```

Confusiion Matrix



TRAIN CONFUSION MATRIX



[4.2] TFIDF-W2V(BRUTE_FORCE)

In [70]:
```
X = preprocessed_reviews
```

In [71]:
```
Y=final['Score']
```

In [72]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3,shuffle=False) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33,shuffle=False)


print(np.shape(X_train), y_train.shape)

print(np.shape(X_cv), y_cv.shape)
print(np.shape(X_test), y_test.shape)
```

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
```

In [73]:
```
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 v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [74]:
```
# 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 ce
ll_val = tfidf

tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is s
tored in this list
row=0;
for sent in tqdm(sent_of_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/r
eview
    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_train_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 20680/20680 [07:27<00:00, 4
6.25it/s]
```

In [75]:

```
tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in tqdm(sent_of_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/r
eview
    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_cv_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 10187/10187 [03:36<00:00, 4
7.04it/s]
```

In [76]:

```
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sent_of_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/r
eview
    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_test_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 15204/15204 [05:36<00:00, 4
5.18it/s]
```

In [77]:

```
X_test=tfidf_test_vectors
```

In [78]:

```
df_test=df[30867:46074]
```

In [79]:

```
np.shape(df_test)
```

Out[79]:

```
(15204, 1)
```

In [80]:
```
X_test=np.concatenate((X_test,df_test),axis=1)
```

In [81]:
```
np.shape(X_test)
```

Out[81]: (15204, 51)

In [82]:
```
X_train=tfidf_train_vectors
```

In [83]:
```
np.shape(X_train)
```

Out[83]: (20680, 50)

In [84]:
```
df_train=df[:20680]
```

In [85]:
```
X_train=np.concatenate((X_train,df_train),axis=1)
```

In [86]:
```
np.shape(X_train)
```

Out[86]: (20680, 51)

In [87]:
```
X_cv=tfidf_cv_vectors
```

In [88]:
```
np.shape(X_cv)
```

Out[88]: (10187, 50)

In [89]:
```
df_cv=df[20680:30867]
```

In [90]:
```
np.shape(df_cv)
```

Out[90]: (10187, 1)

In [91]:
```
X_cv=np.concatenate((X_cv,df_cv),axis=1)
```

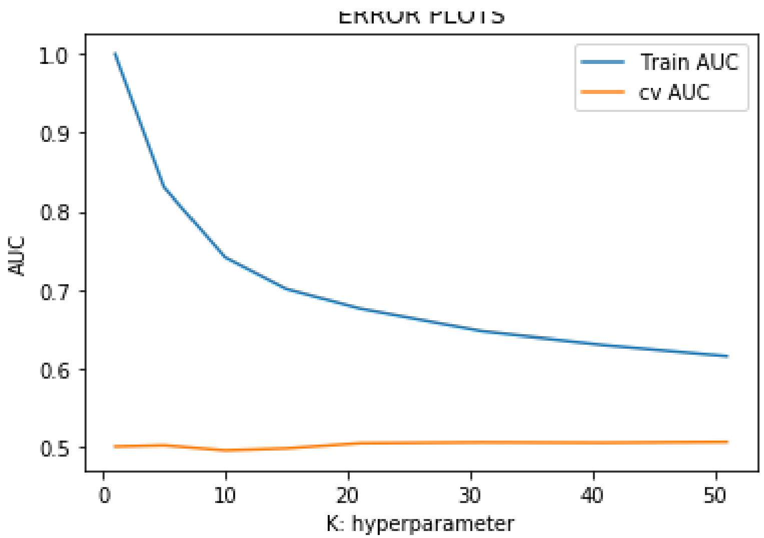
In [92]:
```
np.shape(X_cv)
```

Out[92]: (10187, 51)

forloop for KNN

In [93]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(X_train, y_train)
    y_train_pred = []
    n = len(X_train)
    for i in range(0 ,n, 1000):
        y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1
])
    n = len(X_cv)
    y_cv_pred = []
    for i in range(0 ,n, 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv[i:i+1000])[:,1])
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='cv AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

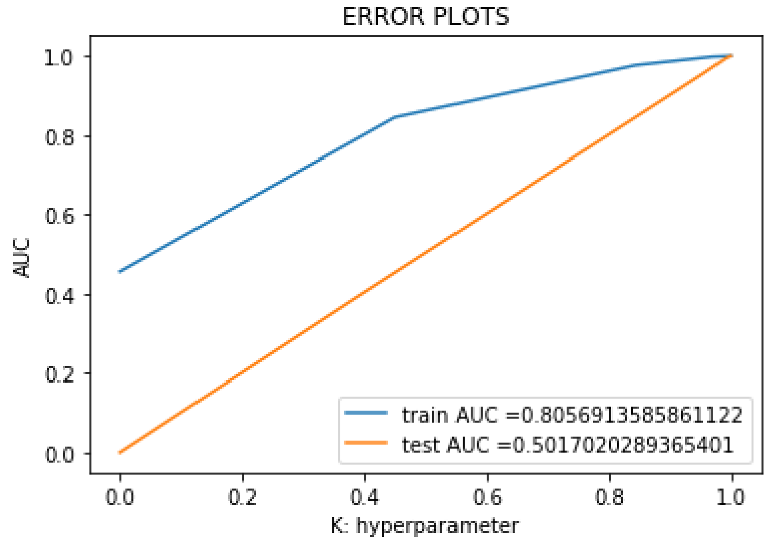
ERROR PLOTS



In [104]:

```
from sklearn.neighbors import  KNeighborsClassifier
# i know my model is not good but randomly pick my k
neigh = KNeighborsClassifier(n_neighbors=6,algorithm='brute')
neigh.fit(X_train, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs
y_train_pred = []
n=np.shape(X_train)[0]
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1])
y_test_pred=[]
n=np.shape(X_test)[0]
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict_proba(X_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
```

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



In [105]:
```
from sklearn.metrics import confusion_matrix
print("train confusion matrix")
y_train_pred = []
n=len(X_train)
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict(X_train[i:i+1000]))

y_test_pred=[]
n=len(X_test)
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict(X_test[i:i+1000]))
cm_traintfw2v=confusion_matrix(y_train,y_train_pred)
cm_testtfw2v=confusion_matrix(y_test,y_test_pred)
print(cm_traintfw2v)
print("="*100)
```

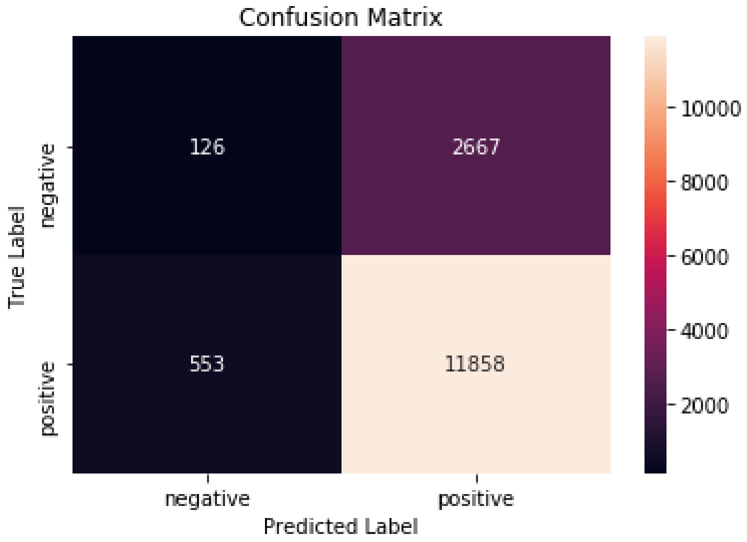
```
print("test confusion matrix")
print(cm_testtfw2v)
```

```
train confusion matrix
[[  470  2548]
 [  414 17248]]
========================================================================
==============================
test confusion matrix
[[  126  2667]
 [  553 11858]]
```

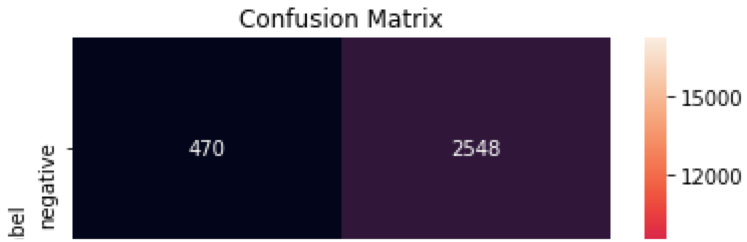
In [106]:

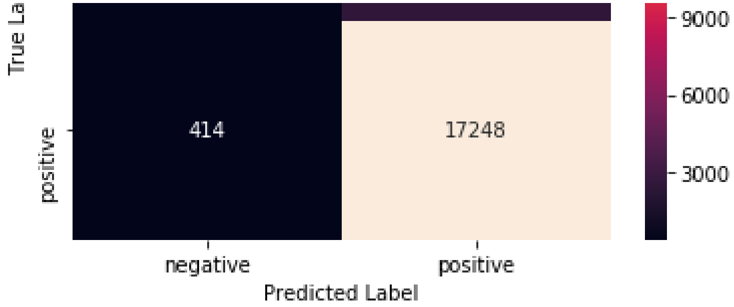
```
import seaborn as sns
print("TEST CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_testtfw2v, index = class_label, columns = class
_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("TRAIN CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_traintfw2v, index = class_label, columns = clas
s_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
TEST CONFUSION MATRIX
```



TRAIN CONFUSION MATRIX





AVG_W2V(KD_TREE)

In [107]:
```
final.shape
```
Out[107]: (46071, 11)

In [108]:
```
X=preprocessed_reviews[:20000]# we are first 20k points in sorted order
```

In [109]:
```
np.shape(X)
```
Out[109]: (20000,)

In [110]:
```
Y=final["Score"][:20000]
```

In [111]:
```
Y.shape
```
Out[111]: (20000,)

In [112]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3,shuffle=False) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33,shuffle=False)
```

```
print(np.shape(X_train), y_train.shape)

print(np.shape(X_cv), y_cv.shape)
print(np.shape(X_test), y_test.shape)
```

```
(8978,) (8978,)
(4422,) (4422,)
(6600,) (6600,)
```

In [113]:
```
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [114]:
```
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())
```

In [115]:
```
sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

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

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

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

In [116]:
```
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
```

```
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

cv_vectors = [];
for sent in sent_of_cv:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)




# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)
```

In [117]:
```
X_train=train_vectors
```

In [118]:
```
np.shape(X_train)
```

Out[118]: (8978, 50)

In [119]:
```
df_train=df[:8978]
```

In [120]:
```
np.shape(df_train)
```

Out[120]: (8978, 1)

In [121]:
```
X_train=np.concatenate((X_train,df_train),axis=1)
```

In [122]:
```
np.shape(X_train)
```

Out[122]: (8978, 51)

In [123]:
```
X_cv=cv_vectors
```

In [124]:
```
np.shape(X_cv)
```

Out[124]: (4422, 50)

In [125]:
```
df_cv=df[8978:13400]
```

In [126]:
```
np.shape(df_cv)
```

Out[126]: (4422, 1)

In [127]:
```
X_cv=np.concatenate((X_cv,df_cv),axis=1)
```

In [128]:
```
np.shape(X_cv)
```

Out[128]: (4422, 51)

In [129]:
```
X_test=test_vectors
```

In [130]:
```
np.shape(X_test)
```

Out[130]: (6600, 50)

In [131]:
```
df_test=df[13400:20000]
```

In [132]:
```
np.shape(df_test)
```

Out[132]: (6600, 1)

In [133]:
```
X_test=np.concatenate((X_test,df_test),axis=1)
```

In [134]:
```
np.shape(X_test)
```

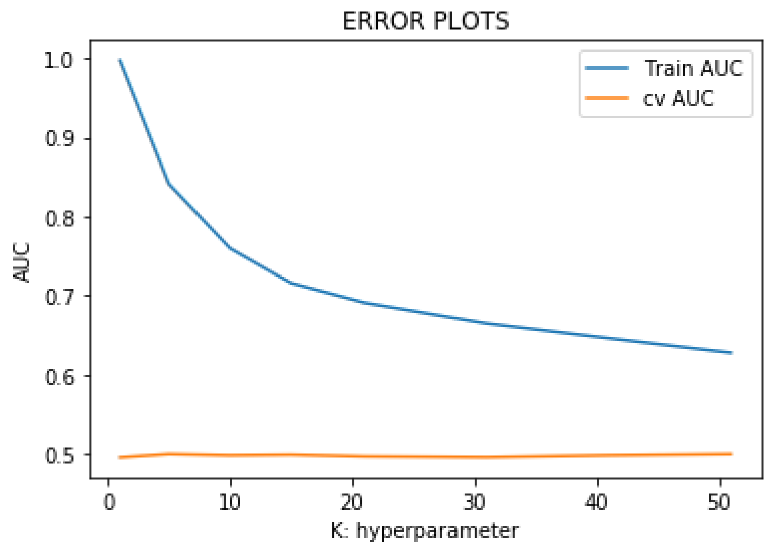
Out[134]: (6600, 51)

In [135]:
```
y_test.shape
```

Out[135]: (6600,)

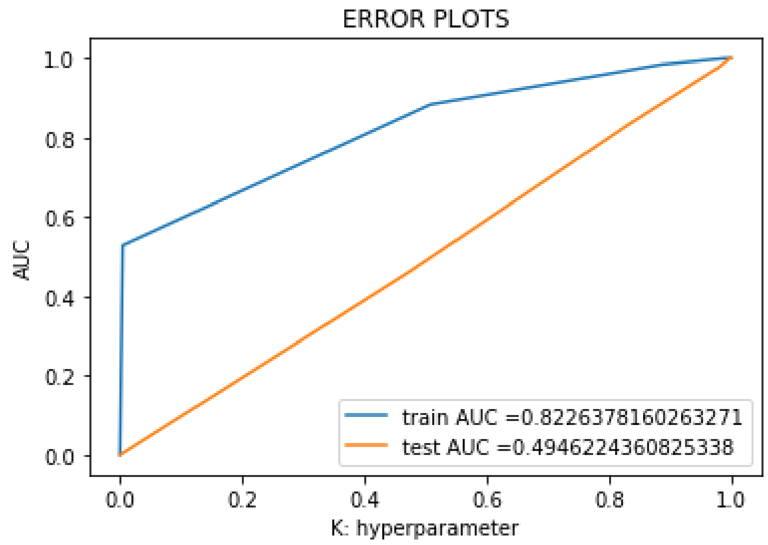
In [136]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(X_train, y_train)
    y_train_pred = []
    n = len(X_train)
    for i in range(0 ,n, 1000):
        y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1
])
    n = len(X_cv)
```

```
    y_cv_pred = []
    for i in range(0 ,n, 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv[i:i+1000])[:,1])
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='cv AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [145]:
```
from sklearn.neighbors import  KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=6,algorithm='kd_tree')
neigh.fit(X_train, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs
y_train_pred = []
```

```
n=len(X_train)
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1])
y_test_pred=[]
n=len(X_test)
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict_proba(X_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, y_test_pred)

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



CONFUSION MATRIX

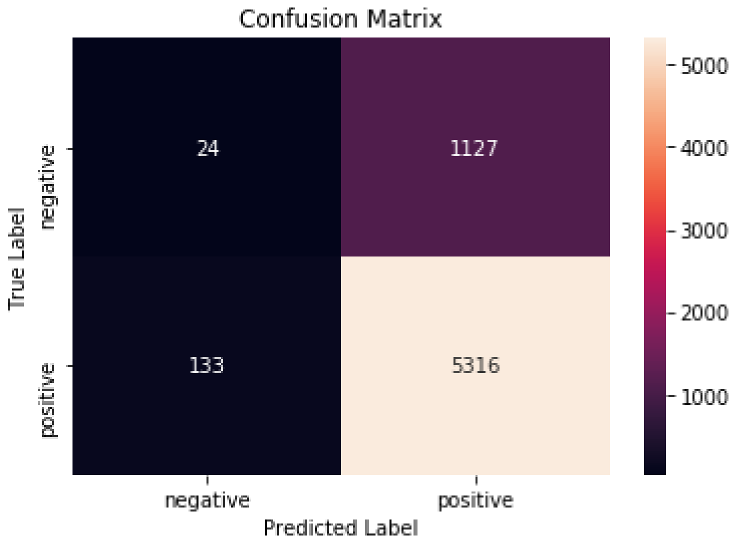
In [146]:
```
from sklearn.metrics import confusion_matrix
print("train confusion matrix")
y_train_pred = []
n=len(X_train)
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict(X_train[i:i+1000]))

y_test_pred=[]
n=len(X_test)
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict(X_test[i:i+1000]))
cm_trainw2v=confusion_matrix(y_train,y_train_pred)
cm_testw2v=confusion_matrix(y_test,y_test_pred)
print(cm_trainw2v)
print("="*100)
print("test confusion matrix")
print(cm_testw2v)
```

```
train confusion matrix
[[ 126 1000]
 [ 125 7727]]
======================================================================
==============================
test confusion matrix
[[  24 1127]
 [ 133 5316]]
```

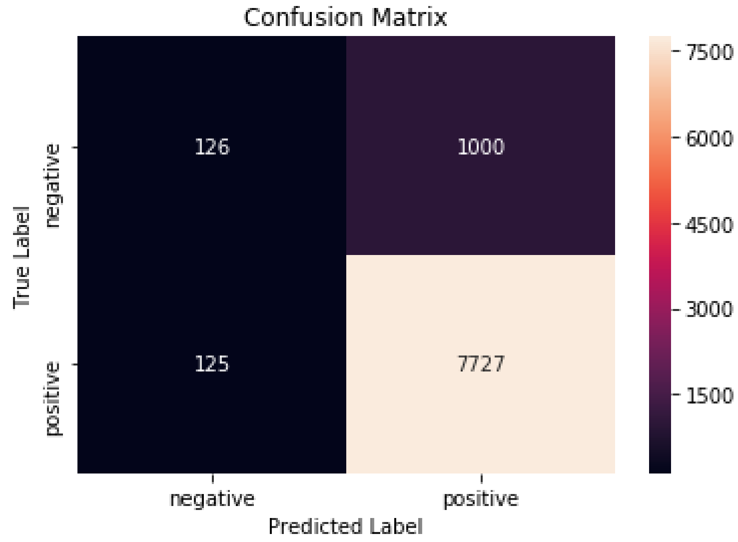
In [147]:
```
import seaborn as sns
print("TEST CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_testw2v, index = class_label, columns = class_l
abel)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
print("TRAIN CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_trainw2v, index = class_label, columns = class_
label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
TEST CONFUSION MATRIX
```




```
TRAIN CONFUSION MATRIX
```



TFIDF W2V (KD_TREE)

In [148]:
```
X=preprocessed_reviews[:20000]
```

In [149]:
```
Y=final["Score"][:20000]
```

In [150]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33)


print(np.shape(X_train), y_train.shape)

print(np.shape(X_cv), y_cv.shape)
print(np.shape(X_test), y_test.shape)
```

```
(8978,) (8978,)
(4422,) (4422,)
(6600,) (6600,)
```

In [151]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [152]:
```
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is s
tored in this list
row=0;
for sent in tqdm(sent_of_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/r
eview
    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_train_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 8978/8978 [01:57<00:00, 7
6.21it/s]
```

In [153]:

```
tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in tqdm(sent_of_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/r
eview
    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_cv_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 4422/4422 [00:58<00:00, 7
5.38it/s]
```

In [154]:

```
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sent_of_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/r
eview
    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_test_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 6600/6600 [01:32<00:00, 71.00it/s]
```

In [155]:
```
X_train=tfidf_train_vectors
```

In [156]:
```
np.shape(X_train)
```

Out[156]: (8978, 50)

In [157]:
```
df_train=df[:8978]
np.shape(df_train)
```

Out[157]: (8978, 1)

In [158]:
```
X_train=np.concatenate((X_train,df_train),axis=1)
```

In [159]:
```
np.shape(X_train)
```

Out[159]: (8978, 51)

In [160]:
```
X_cv=tfidf_cv_vectors
```

In [161]:
```
np.shape(X_cv)
```

Out[161]: (4422, 50)

In [162]:
```
df_cv=df[8978:13400]
np.shape(df_cv)
```

Out[162]: (4422, 1)

In [163]:
```
X_cv=np.concatenate((X_cv,df_cv),axis=1)
np.shape(X_cv)
```
Out[163]: (4422, 51)

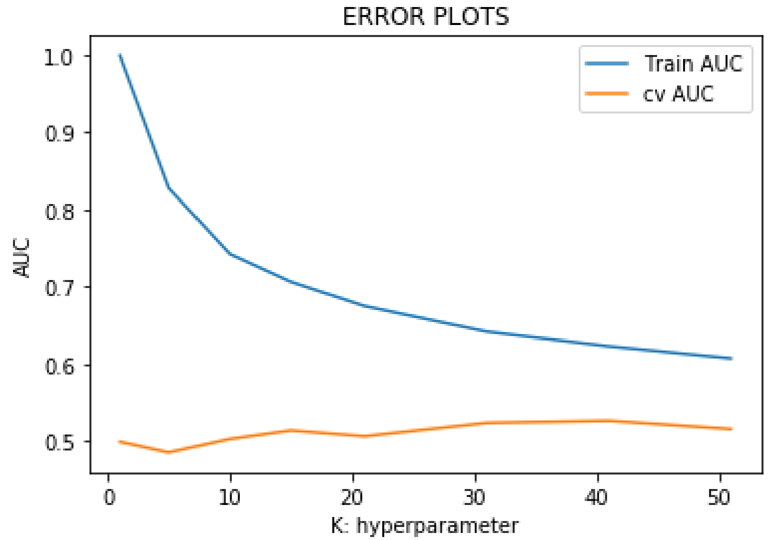
In [164]:
```
X_test=tfidf_test_vectors
```
In [165]:
```
np.shape(X_test)
```
Out[165]: (6600, 50)

In [166]:
```
df_test=df[13400:20000]
```
In [167]:
```
np.shape(df_test)
```
Out[167]: (6600, 1)

In [168]:
```
X_test=np.concatenate((X_test,df_test),axis=1)
```
In [169]:
```
np.shape(X_test)
```
Out[169]: (6600, 51)

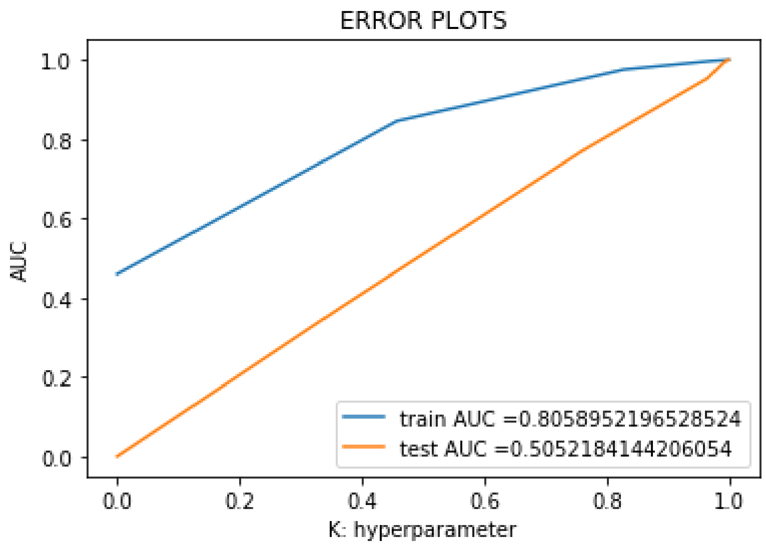
In [170]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    neigh.fit(X_train, y_train)
    y_train_pred = []
    n = len(X_train)
    for i in range(0 ,n, 1000):
        y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1
```

```
])
    n = len(X_cv)
    y_cv_pred = []
    for i in range(0 ,n, 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv[i:i+1000])[:,1])
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='cv AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [171]:
```
from sklearn.neighbors import  KNeighborsClassifier
# i know my model is not good but randomly pick my k
neigh = KNeighborsClassifier(n_neighbors=6,algorithm='kd_tree')
neigh.fit(X_train, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
```

```
# not the predicted outputs
y_train_pred = []
n=np.shape(X_train)[0]
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict_proba(X_train[i:i+1000])[:,1])
y_test_pred=[]
n=np.shape(X_test)[0]
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict_proba(X_test[i:i+1000])[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, y_test_pred)

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



CONFUSION MATRIX

In [172]:

```
from sklearn.metrics import confusion_matrix
print("train confusion matrix")
y_train_pred = []
n=len(X_train)
for i in range(0,n,1000):
    y_train_pred.extend(neigh.predict(X_train[i:i+1000]))

y_test_pred=[]
n=len(X_test)
for i in range(0,n,1000):
    y_test_pred.extend(neigh.predict(X_test[i:i+1000]))
cm_traintfw2v=confusion_matrix(y_train,y_train_pred)
cm_testtfw2v=confusion_matrix(y_test,y_test_pred)
print(cm_traintfw2v)
print("="*100)
print("test confusion matrix")
print(cm_testtfw2v)
```

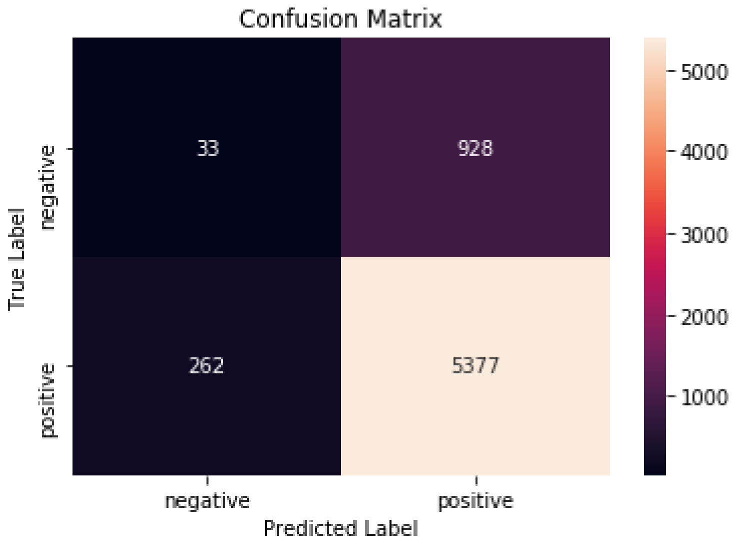
```
train confusion matrix
[[ 221 1067]
 [ 187 7503]]
========================================================================
============================
test confusion matrix
[[  33  928]
 [ 262 5377]]
```

In [173]:

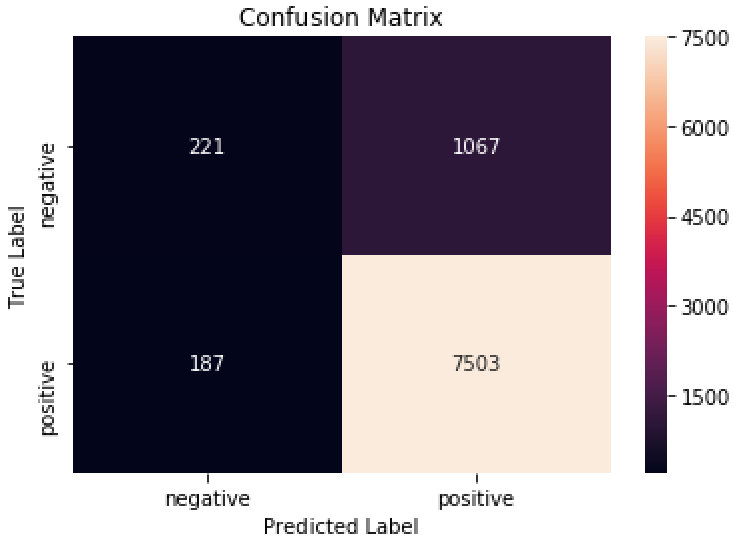
```
import seaborn as sns
print("TEST CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_testtfw2v, index = class_label, columns = class
_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
plt.show()

print("TRAIN CONFUSION MATRIX")
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_traintfw2v, index = class_label, columns = clas
s_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

TEST CONFUSION MATRIX



TRAIN CONFUSION MATRIX



NOTE : IN CONFUSION MATRIX PLEASE CHECK THE LABELS BECAUSE I HAVE TAKEN NEGATIVE AND POSITIVE

[6] Conclusions

In [4]:
```
#  compare all your models using Prettytable library
```

My prettytable is not working in my notebook

In [95]:
```
print("BRUTE ALGORITHM ")
from tabulate import tabulate
print(tabulate ([['BOW', 12, 0.69,0.82],['TFIDF',20,0.80,0.86],['AVG_W2V',10,0.49,0.73],['TFIDF-W2V',18,0.49,0.67]],    headers=['algorithm type', 'best_k','roc_score for test','roc_score for train']))
```

```
BRUTE ALGORITHM
algorithm type      best_k    roc_score for test    roc_score for train
----------------  --------  --------------------  ---------------------
BOW                     12                  0.69                   0.82
TFIDF                   20                  0.8                    0.86
AVG_W2V                 10                  0.49                   0.73
TFIDF-W2V               18                  0.49                   0.67
```

In [96]:
```
print("KD_TREE ALGORITHM ")
from tabulate import tabulate
print(tabulate ([['BOW', 15, 0.5,0.69],['TFIDF',12,0.49,0.73],['AVG_W2V',10,0.51,0.74],['TFIDF-W2V',10,0.49,0.73]],    headers=['algorithm type', 'best_k','roc_score for test','roc_score for train']))
```

```
KD_TREE ALGORITHM
algorithm type      best_k    roc_score for test    roc_score for train
----------------  --------  --------------------  ---------------------
BOW                     15                  0.5                    0.69
TFIDF                   12                  0.49                   0.73
AVG_W2V                 10                  0.51                   0.74
TFIDF-W2V               10                  0.49                   0.73
```

TFIDF WITH BRUTE FORCE IS WORKING BETTER AS COMPARED TO OTHER MODEL