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:\Users\user\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWar
ning: 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('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
```

```
# 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 100000""", 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 (100000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

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

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

```
(80668, 7)
```

Out[4]:

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

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

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

Out[5]:

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

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

Out[6]:
```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

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

```
""", con)
display.head()
```

Out[7]:

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

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

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

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

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

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

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, 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]:
```
(87775, 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]:
```
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 [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	HelpfulnessDenom
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

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

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

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

```
(87773, 10)
```

Out[13]:
```
1    73592
0    14181
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.
==================================================
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 w
ere 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 sme
ll. So if you are afraid of the fishy smell, don'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 the
se 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 bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [15]:
```
# 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]:
```
# 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.
==================================================
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 w
ere 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 sme
ll. So if you are afraid of the fishy smell, don'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 the
```

```
se 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 bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
```

In [17]:
```python
# 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]:
```python
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 [19]:
```python
#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)
```

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

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%|██████████| 87773/87773 [01:04<00:00, 1356.64it/s]
```

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

Out[23]:
```
'way hot blood took bite jig lol'
```

In [24]:
```
final['CleanedText']=preprocessed_reviews
final.head(5)
```

Out[24]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	
70677	76870	B00002N8SM	A19Q006CSFT011	Arlielle	0	
70676	76869	B00002N8SM	A1FYH4S02BW7FN	wonderer	0	
70675	76868	B00002N8SM	AUE8TB5VHS6ZV	eyeofthestorm	0	

[3.2] Preprocessing Review Summary

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

[4] Featurization

[4.1] BAG OF WORDS

In [25]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [26]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
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  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [27]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_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) ['ability', 'able', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [28]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [42]:
```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

In [36]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

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

[4.4.1.1] Avg W2v

In [38]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [39]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [41]:
```
# 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_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored 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/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_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 4986/4986 [00:20<00:00, 245.63it/s]
```

[5] Assignment 7: SVM

1. **Apply SVM 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. **Procedure**

- You need to work with 2 versions of SVM
  - Linear kernel
  - RBF kernel
- When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
- When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use CalibratedClassifierCV
- Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

- Find the best hyper parameter which will give the maximum AUC 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. **Feature importance**

- When you are working on the linear kernel with BOW or TFIDF please print the top 10 best features for each of the positive and negative classes.

5. **Feature engineering**

- To increase the performance of your model, you can also experiment with with feature engineering like :
  - Taking length of reviews as another feature.
  - Considering some features from review summary as well.

6. **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 with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

7. **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

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.

Applying SVM

[5.1] Linear SVM

[5.1.1] Applying Linear SVM on BOW, SET 1

In [3]:
```
# Please write all the code with proper documentation
```

In [26]:
```
X = final["CleanedText"]
print("shape of X:", X.shape)
```

```
shape of X: (87773,)
```

In [27]:
```
y = final["Score"]
print("shape of y:", y.shape)
```

```
shape of y: (87773,)
```

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=
0.33, shuffle=Flase): this is for time series 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) # this is random splitting


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

print("="*100)

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

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

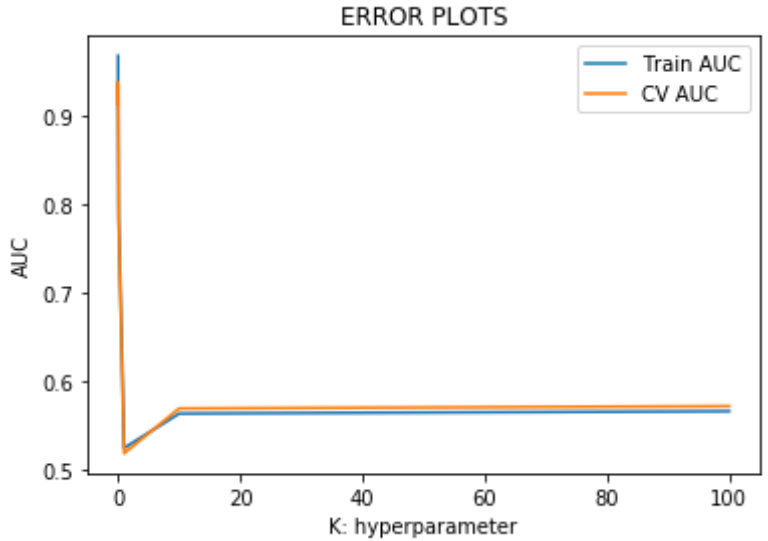
```
X_test_bow = vectorizer.transform(X_test)

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

```
(39400,) (39400,)
(19407,) (19407,)
(28966,) (28966,)
============================================================================
============================
After vectorizations
(39400, 37411) (39400,)
(19407, 37411) (19407,)
(28966, 37411) (28966,)
============================================================================
============================
```

In [29]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix,roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
Alpha = [0.0001,0.001, 0.01, 0.1, 1, 10,100]
for i in Alpha:
    sgd=SGDClassifier(loss='hinge',alpha=i)
    sgd.fit(X_train_bow,y_train)
    clf=CalibratedClassifierCV(sgd, cv='prefit')
    clf.fit(X_train_bow, y_train)
    y_train_pred =  clf.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  clf.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(Alpha, train_auc, label='Train AUC')
plt.plot(Alpha, cv_auc, label='CV AUC')
plt.legend()
```

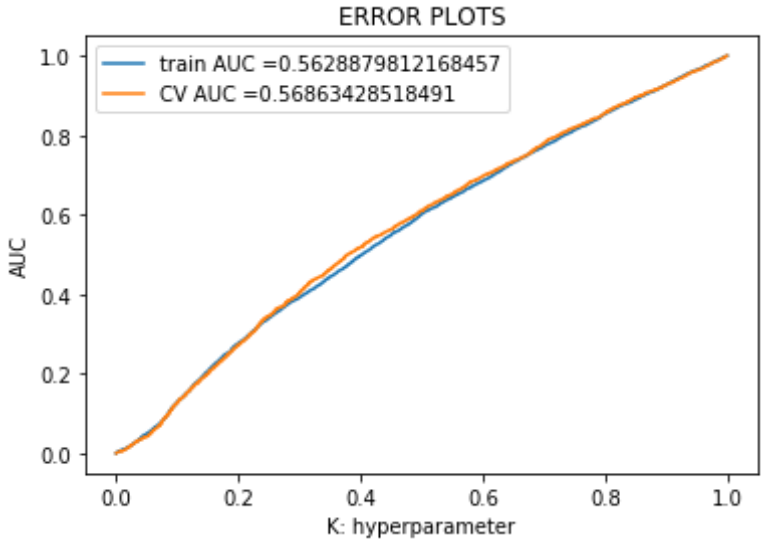
```
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [30]:
```
best_alpha=10
```

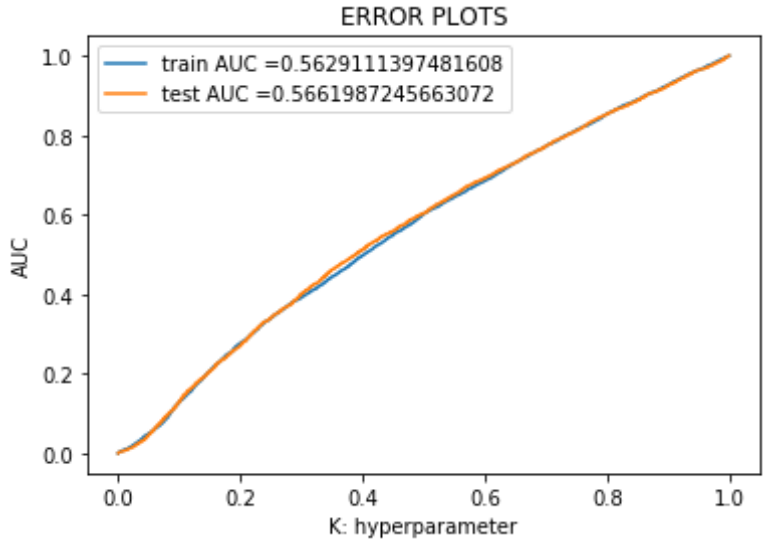
In [31]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train_bow,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, clf.predict_proba(X_cv
_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
```

```
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="CV AUC ="+str(auc(test_fpr, test_tp
r)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [32]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train_bow,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.fit(X_train_bow, 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

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_
test_bow)[:,1])
```

```
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 [63]:
```
feature_names = vectorizer.get_feature_names()
coefs_with_fns = sorted(zip(sgd.coef_[0], feature_names))
top = (coefs_with_fns[:-(10 + 1):-1])
print("\tPositive")
for (coef_2, fn_2) in top:
    print("\t%.4f\t%-15s\t\t\t\t" % (coef_2, fn_2))
```

```
        Positive
        0.4279  great
        0.3817  delicious
        0.3634  best
```

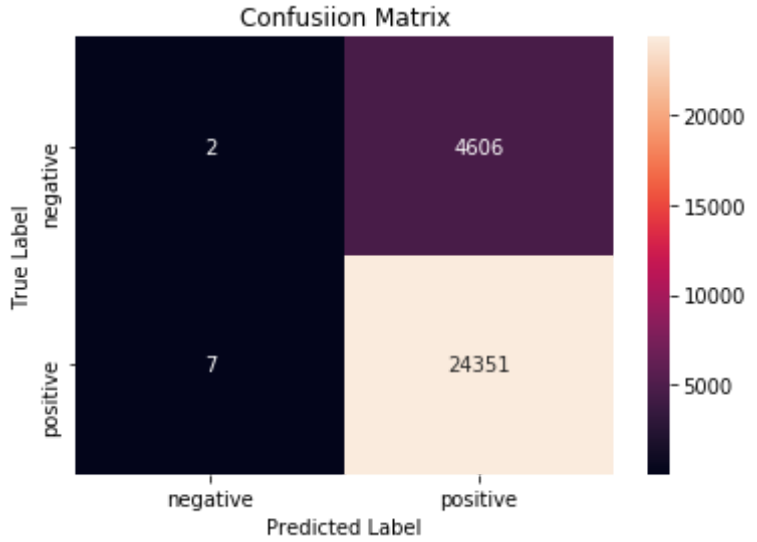
```
0.3497  loves
0.3233  perfect
0.3172  love
0.3106  excellent
0.2964  wonderful
0.2913  nice
0.2791  good
```

In [64]:
```
feature_names = vectorizer.get_feature_names()
coefs_with_fns = sorted(zip(sgd.coef_[0], feature_names))
top = (coefs_with_fns[:10])
print("\tNegative")

for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s\t\t\t\t" % (coef_1, fn_1))
```

```
        Negative
        -0.4182 disappointed
        -0.3720 horrible
        -0.3604 money
        -0.3360 worst
        -0.3355 terrible
        -0.3263 awful
        -0.3055 return
        -0.3000 waste
        -0.2954 stale
        -0.2883 bad
```

In [33]:
```
print("Test confusion matrix")
cm=confusion_matrix(y_test, clf.predict(X_test_bow))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [3]:
```
# Please write all the code with proper documentation
```

In [34]:
```
X = final["CleanedText"]
print("shape of X:", X.shape)
```

```
shape of X: (87773,)
```

In [35]:
```
X = final["CleanedText"]
print("shape of X:", X.shape)
```

```
shape of X: (87773,)
```

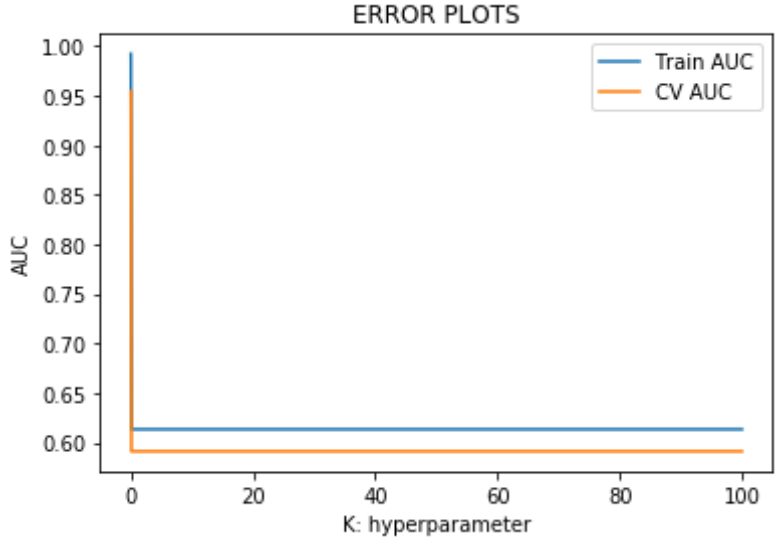
In [36]:
```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series 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) # this is random splitting


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

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1,2))
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = vectorizer.transform(X_train)
X_cv_tfidf = vectorizer.transform(X_cv)
X_test_tfidf = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
print("="*100)
```

```
(39400,) (39400,)
(19407,) (19407,)
(28966,) (28966,)
============================================================================
============================
After vectorizations
(39400, 773314) (39400,)
(19407, 773314) (19407,)
(28966, 773314) (28966,)
============================================================================
============================
```

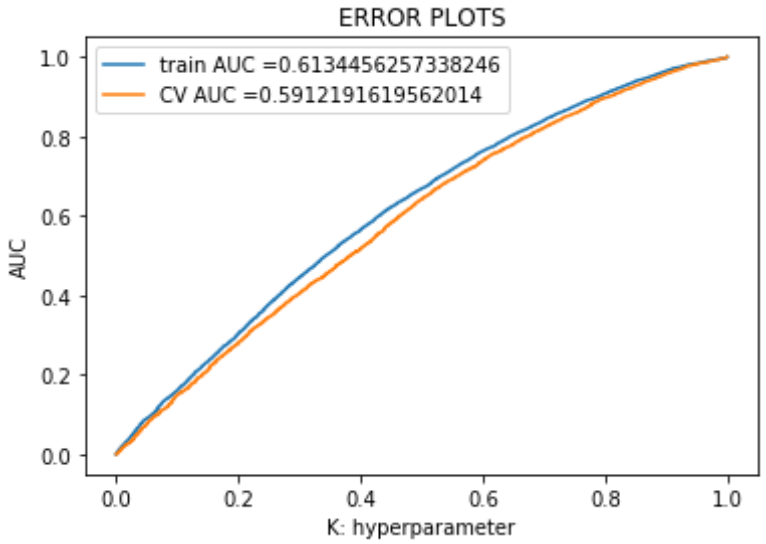
In [37]:
```
from sklearn.linear_model import SGDClassifier
```

```
from sklearn.metrics import confusion_matrix,roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
Alpha = [0.0001,0.001, 0.01, 0.1, 1, 10,100]
for i in Alpha:
    sgd=SGDClassifier(loss='hinge',alpha=i)
    sgd.fit(X_train_tfidf,y_train)
    clf=CalibratedClassifierCV(sgd, cv='prefit')
    clf.fit(X_train_tfidf, y_train)
    y_train_pred =  clf.predict_proba(X_train_tfidf)[:,1]
    y_cv_pred =  clf.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(Alpha, train_auc, label='Train AUC')
plt.plot(Alpha, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

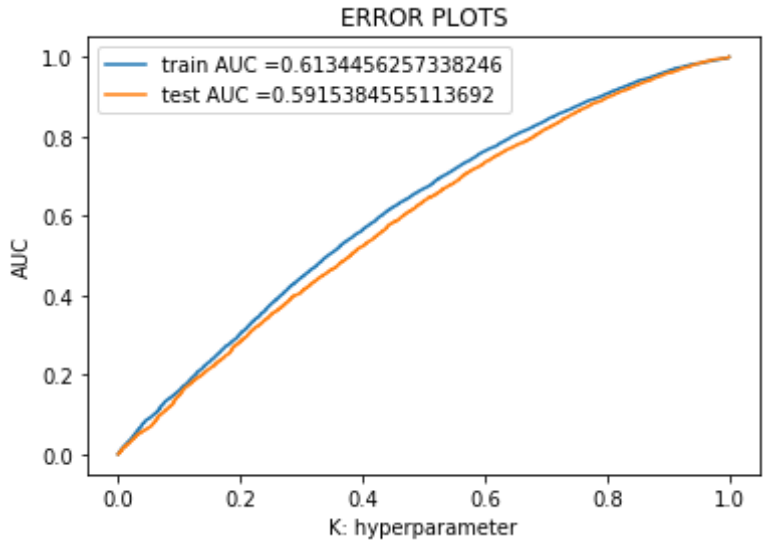


In [38]:
```
best_alpha=0.1
```

In [39]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train_tfidf,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.fit(X_train_tfidf, 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

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve( y_cv, clf.predict_proba(X_c
v_tfidf)[:,1])

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

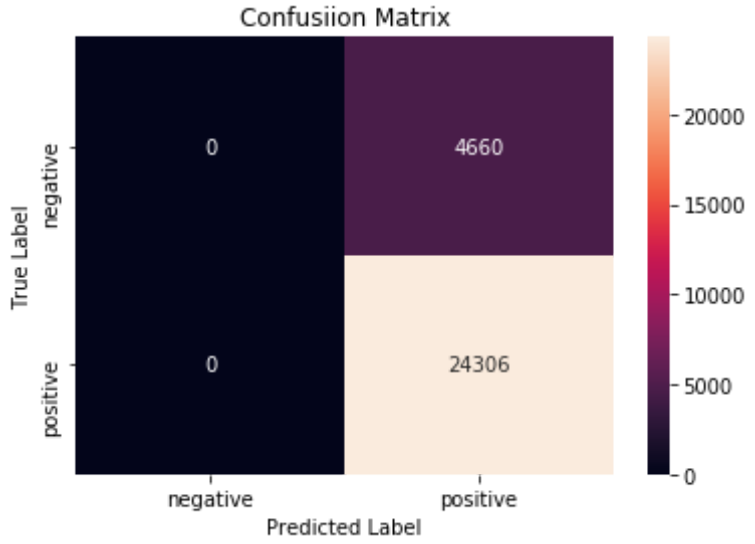


In [40]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train_tfidf,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_
test_tfidf)[:,1])

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 [49]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, clf.predict(X_test_tfidf))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[5.1.3] Applying Linear SVM on AVG W2V, SET 3

In [3]:
```
# Please write all the code with proper documentation
```

In [25]:
```
i=0
list_of_sentance=[]
for sentance in final['CleanedText']:
    list_of_sentance.append(sentance.split())
```

In [26]:
```
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'))
```

```
[('fantastic', 0.839952826499939), ('terrific', 0.8368092179298401),
('good', 0.8277462720870972), ('excellent', 0.8266530632972717), ('awes
ome', 0.8180069327354431), ('wonderful', 0.7559068202972412), ('perfec
t', 0.7529680728912354), ('nice', 0.7147786617279053), ('amazing', 0.70
95006108283997), ('decent', 0.7050840258598328)]
==================================================
[('greatest', 0.8100333213806152), ('tastiest', 0.7578601837158203),
('best', 0.7346643209457397), ('nastiest', 0.6857903003692627), ('disgu
sting', 0.641221284866333), ('horrible', 0.6114877462387085), ('closes
t', 0.6060494780540466), ('terrible', 0.5988964438438416), ('softest',
0.5912508964538574), ('hottest', 0.5888092517852783)]
```

In [27]:
```
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', 'products', 'made', 'usa', 'one',
'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'imp
orts', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding',
'satisfied', 'safe', 'infestation', 'literally', 'everywhere', 'flyin
g', 'around', 'kitchen', 'bought', 'hoping', 'least', 'get', 'rid', 'we
eks', 'fly', 'stuck', 'squishing', 'buggers', 'success', 'rate']
```

In [28]:
```
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, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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]))
from sklearn.model_selection import train_test_split
```

```
100%|██████████| 87773/87773 [06:07<00:00, 239.11it/s]
```

```
87773
50
```

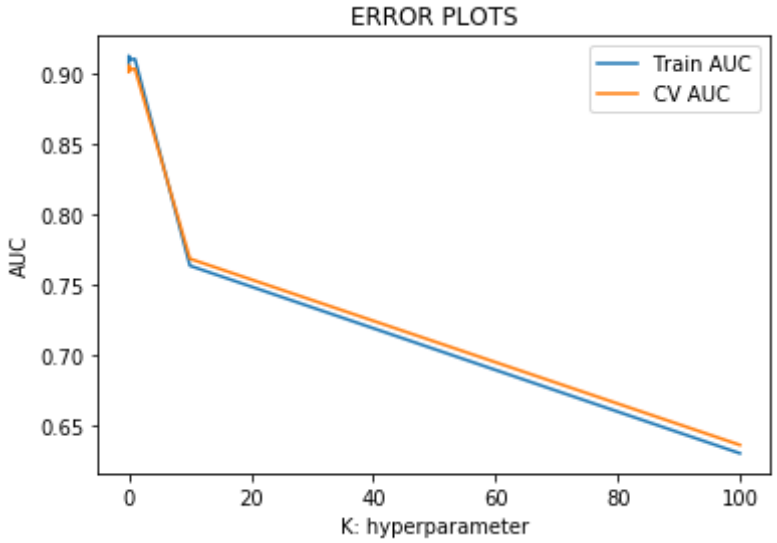
In [29]:

```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(sent_vectors, final
['Score'], test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

In [30]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix,roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
Alpha = [0.0001,0.001, 0.01, 0.1, 1, 10,100]
for i in Alpha:
    sgd=SGDClassifier(loss='hinge',alpha=i)
    sgd.fit(X_train,y_train)
    clf=CalibratedClassifierCV(sgd, cv='prefit')
    clf.fit(X_train, y_train)
    y_train_pred =  clf.predict_proba(X_train)[:,1]
    y_cv_pred =  clf.predict_proba(X_cv)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

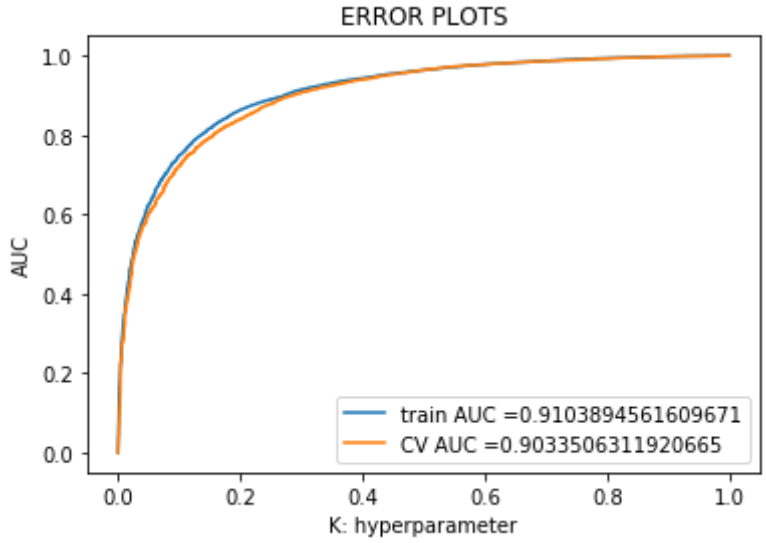
```
plt.plot(Alpha, train_auc, label='Train AUC')
plt.plot(Alpha, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [31]:
```
best_alpha=1
```

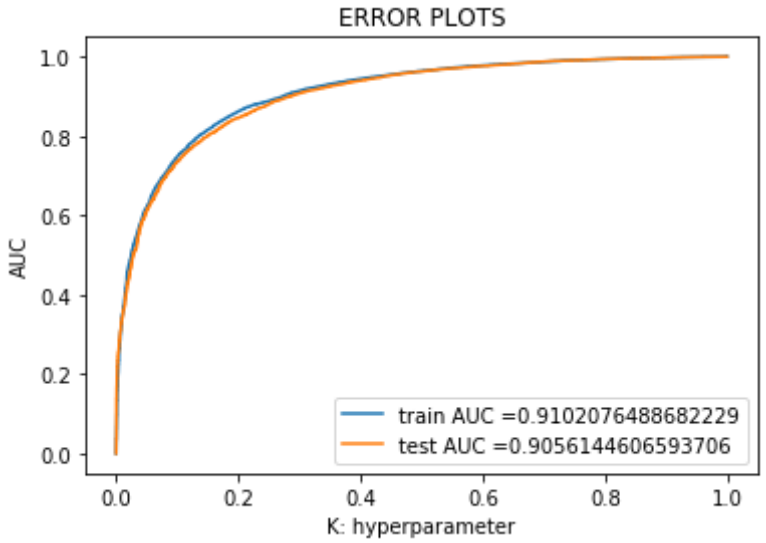
In [32]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.fit(X_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, clf.predict_proba(X_cv
```

```
)[:,1])

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



In [33]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.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

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train)[:,1])
```

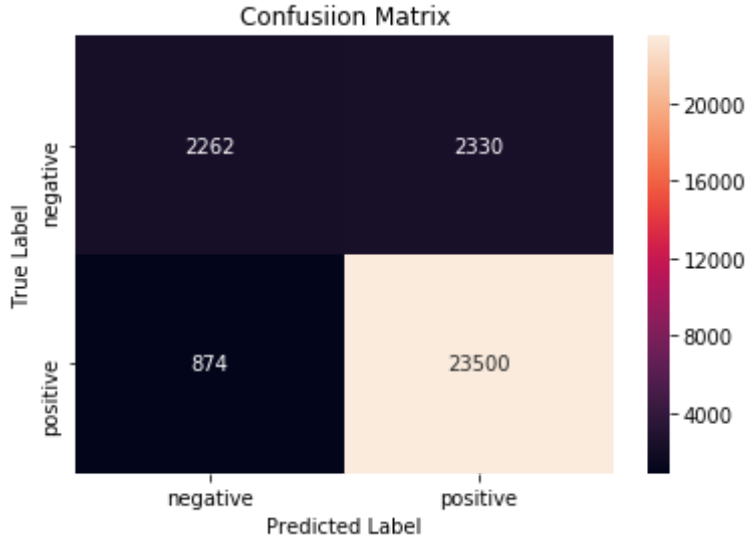
```
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_
test)[:,1])

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 [34]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, clf.predict(X_test))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

In [3]:
```
# Please write all the code with proper documentation
```

In [70]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'])
# 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 [71]:
```
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
```

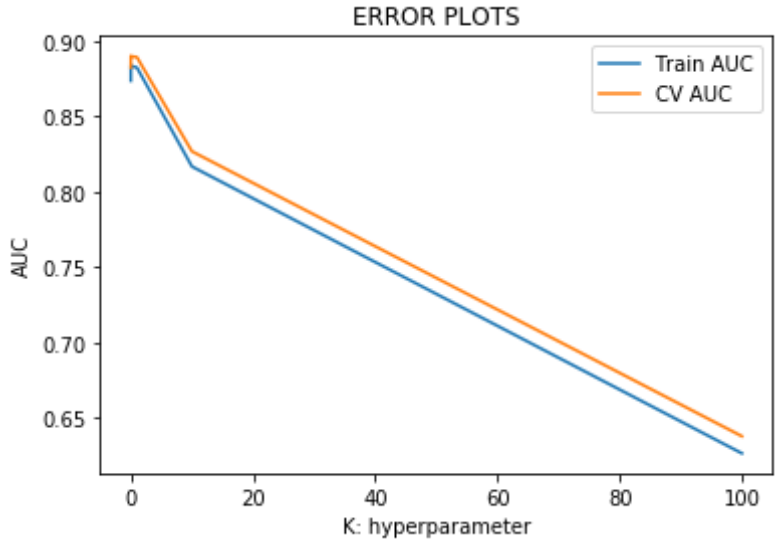
```
tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
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_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████
████████| 87773/87773 [1:36:17<00:00, 15.19it/s]
```

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=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(tfidf_sent_vectors,
 final['Score'], test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

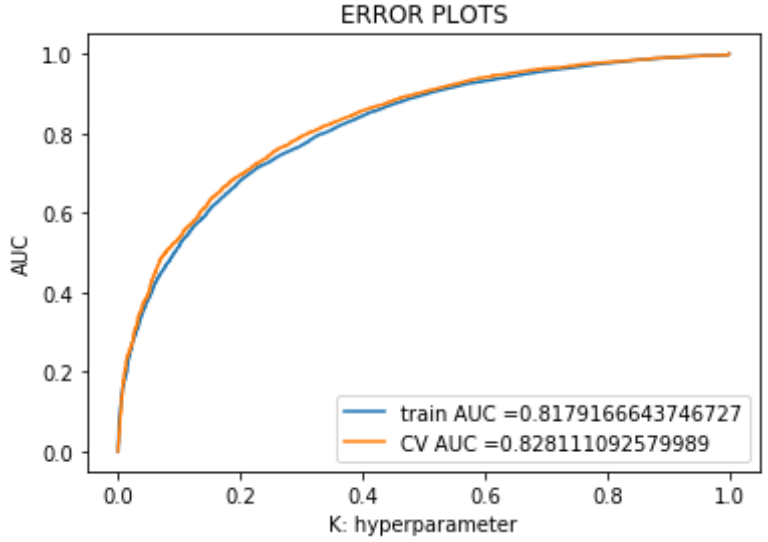
In [73]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix,roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
train_auc = []
```

```
cv_auc = []
Alpha = [0.0001,0.001, 0.01, 0.1, 1, 10,100]
for i in Alpha:
    sgd=SGDClassifier(loss='hinge',alpha=i)
    sgd.fit(X_train,y_train)
    clf=CalibratedClassifierCV(sgd, cv='prefit')
    clf.fit(X_train, y_train)
    y_train_pred =  clf.predict_proba(X_train)[:,1]
    y_cv_pred =  clf.predict_proba(X_cv)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(Alpha, train_auc, label='Train AUC')
plt.plot(Alpha, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

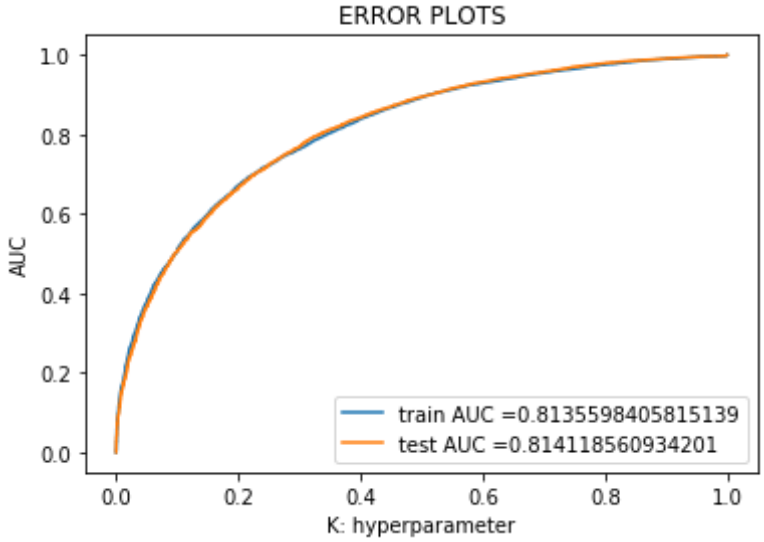


In [80]:
```
best_alpha=9
```

In [82]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.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

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, clf.predict_proba(X_cv
)[:,1])

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

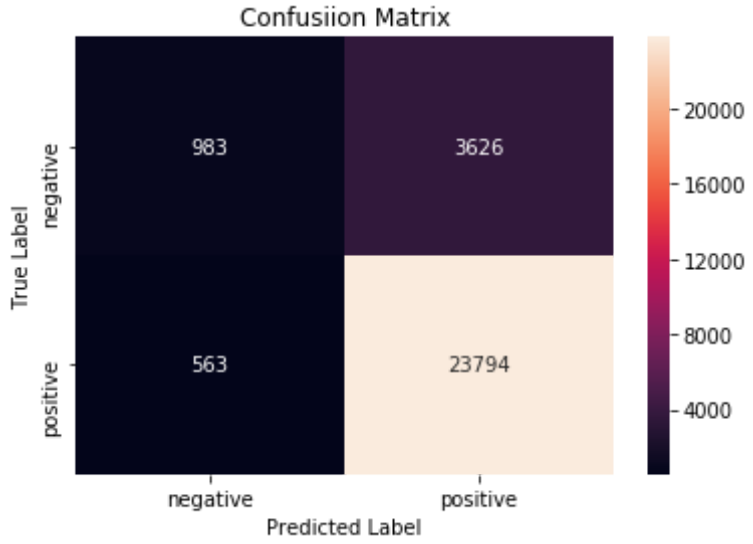


In [83]:
```
sgd=SGDClassifier(loss='hinge',alpha=best_alpha)
sgd.fit(X_train,y_train)
clf = CalibratedClassifierCV(sgd, cv='prefit')
clf.fit(X_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_
test)[:,1])

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 [86]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, clf.predict(X_test))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

In [3]:
```
# Please write all the code with proper documentation
```

In [87]:
```
final = final.iloc[:20000,:]
print(final.shape)
```

```
(20000, 11)
```

In [88]:
```
X = final["CleanedText"]
print("shape of X:", X.shape)
```

```
shape of X: (20000,)
```

In [89]:
```
y = final["Score"]
print("shape of y:", y.shape)
```

```
shape of y: (20000,)
```

In [90]:
```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series 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) # this is random splitting


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

print("="*100)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=500)


# we use the fitted CountVectorizer to convert the text to vector
```

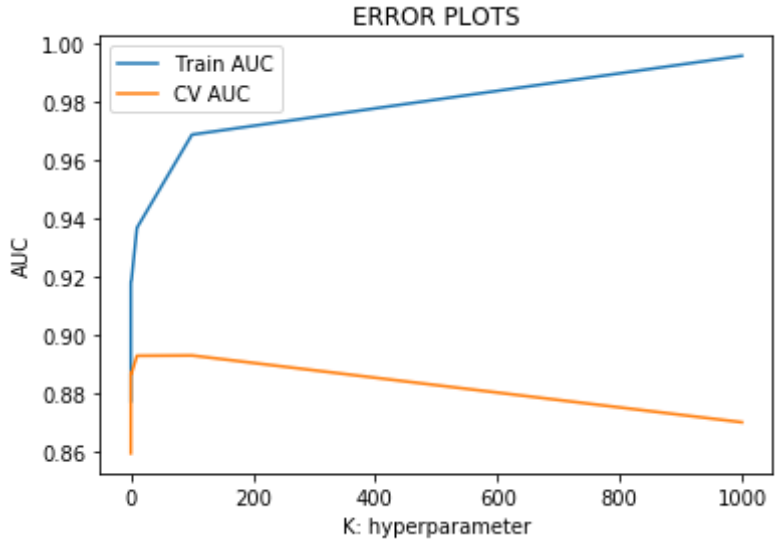
```
X_train_bo = vectorizer.fit_transform(X_train).toarray()
X_cv_bo = vectorizer.transform(X_cv).toarray()
X_test_bo = vectorizer.transform(X_test).toarray()

print("After vectorizations")
print(X_train_bo.shape, y_train.shape)
print(X_cv_bo.shape, y_cv.shape)
print(X_test_bo.shape, y_test.shape)
print("="*100)
```

```
(8978,) (8978,)
(4422,) (4422,)
(6600,) (6600,)
====================================================================================================
==============================
After vectorizations
(8978, 500) (8978,)
(4422, 500) (4422,)
(6600, 500) (6600,)
====================================================================================================
==============================
```

In [93]:

```
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [0.001,.01,.1,1,10,100,1000]
for i in K :
    svc = SVC(C=i,kernel='rbf',probability=True)
    svc.fit(X_train_bo, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  svc.predict_proba(X_train_bo)[:,1]
    y_cv_pred =  svc.predict_proba(X_cv_bo)[:,1]

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

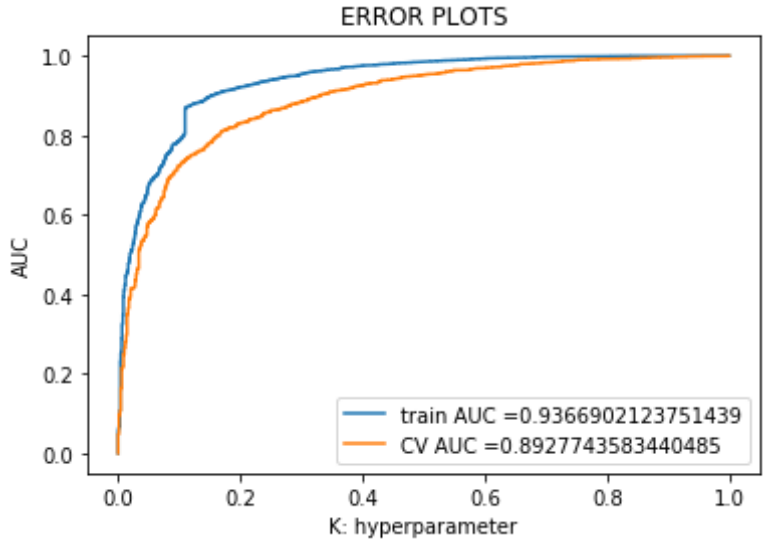
```
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [94]:
```
best_c=10
```

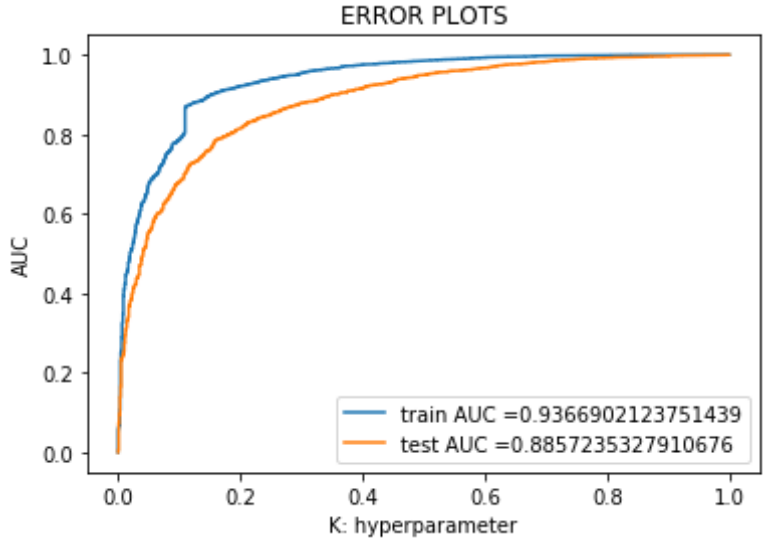
In [95]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.fit(X_train_bo, 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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
```

```
(X_train_bo)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, svc.predict_proba(X_cv
_bo)[:,1])

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



In [96]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.fit(X_train_bo, 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
```

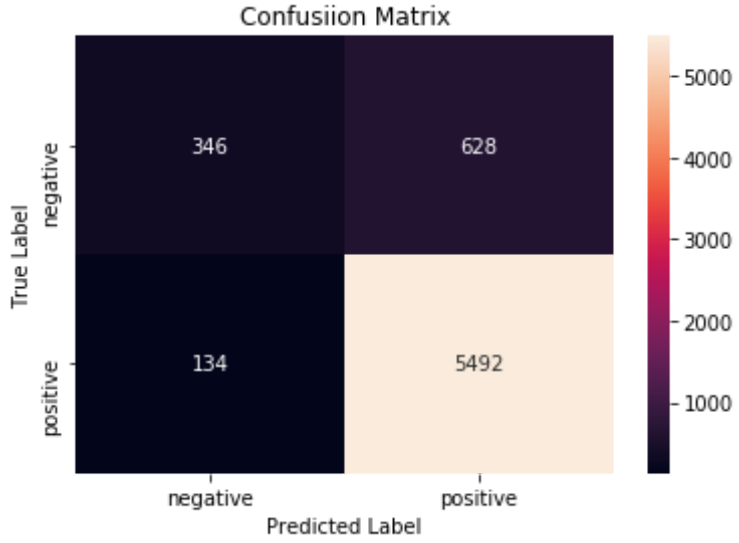
```
train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
(X_train_bo)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, svc.predict_proba(X_
test_bo)[:,1])

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 [97]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, svc.predict(X_test_bo))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [3]:
```
# Please write all the code with proper documentation
```

In [98]:
```
X = final["CleanedText"]
print("shape of X:", X.shape)
```

```
shape of X: (20000,)
```

In [99]:
```
y = final["Score"]
print("shape of y:", y.shape)
```

```
shape of y: (20000,)
```

In [100]:

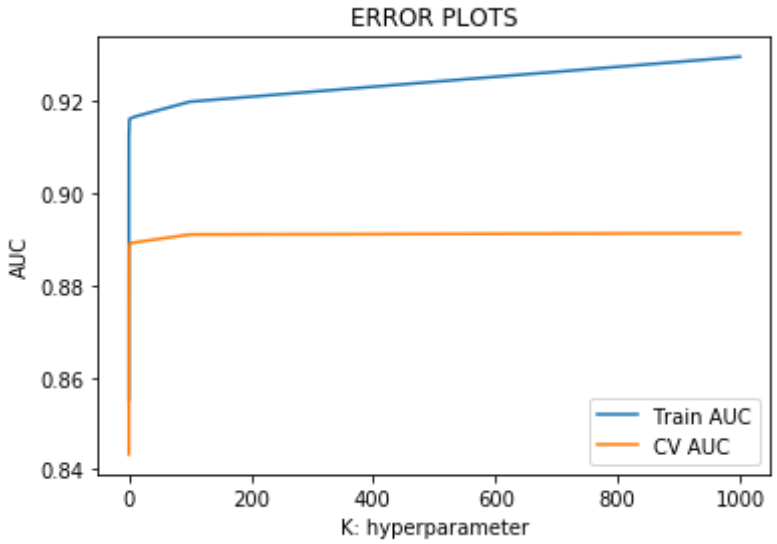
```
from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting


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

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features=
500)


# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = vectorizer.fit_transform(X_train).toarray()
X_cv_tfidf = vectorizer.transform(X_cv).toarray()
X_test_tfidf = vectorizer.transform(X_test).toarray()

print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
print("="*100)
```

```
(8978,) (8978,)
(4422,) (4422,)
(6600,) (6600,)
====================================================================
============================
After vectorizations
```

```
(8978, 500) (8978,)
(4422, 500) (4422,)
(6600, 500) (6600,)
==========================================================================
============================
```

In [101]:

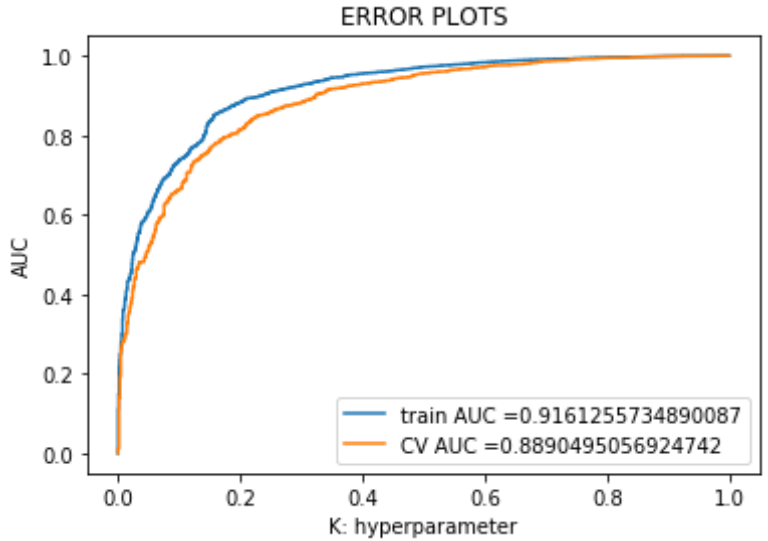
```
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [0.001,.01,.1,1,10,100,1000]
for i in K :
    svc = SVC(C=i,kernel='rbf',probability=True)
    svc.fit(X_train_tfidf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  svc.predict_proba(X_train_tfidf)[:,1]
    y_cv_pred =  svc.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(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 [102]:
```
best_c=1
```

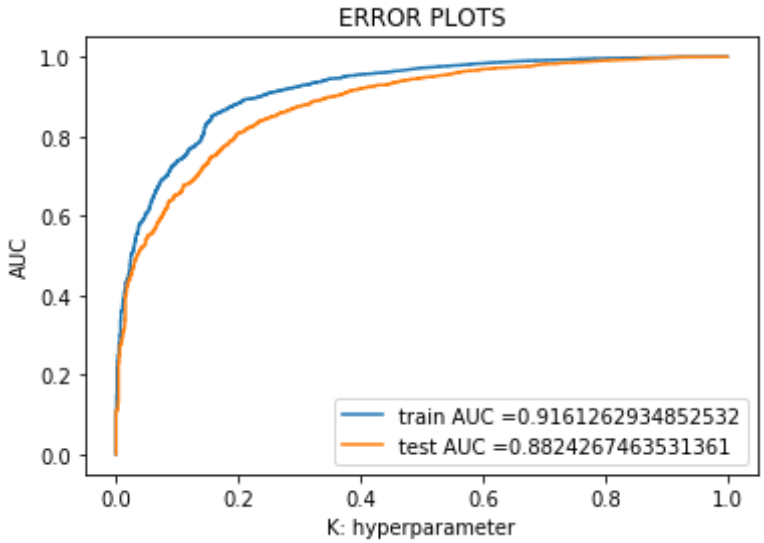
In [103]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.fit(X_train_tfidf, 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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, svc.predict_proba(X_cv
_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="CV AUC ="+str(auc(test_fpr, test_tp
r)))
plt.legend()
```

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



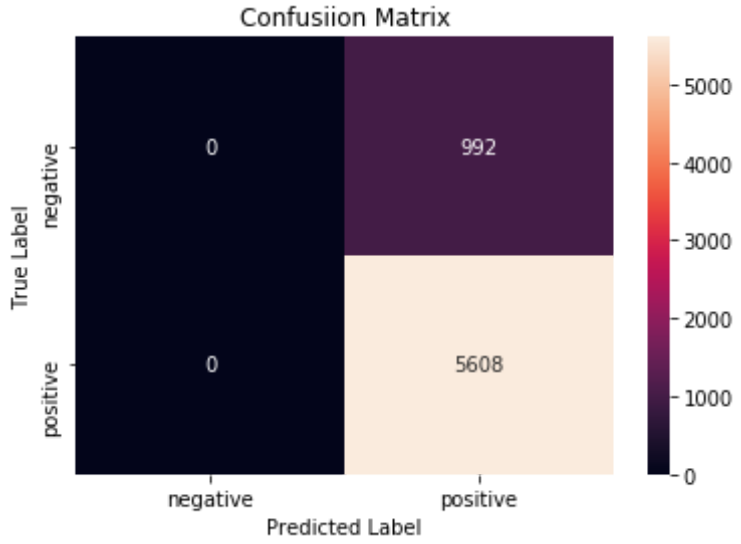
In [104]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.fit(X_train_tfidf, 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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, svc.predict_proba(X_
test_tfidf)[:,1])

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
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, svc.predict(X_test_tfidf))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[5.2.3] Applying RBF SVM on AVG W2V, SET 3

In [3]:
```
# Please write all the code with proper documentation
```

In [106]:
```
i=0
list_of_sentance=[]
for sentance in final['CleanedText']:
    list_of_sentance.append(sentance.split())
```

In [107]:
```
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'))
```

```
[('good', 0.8084676265716553), ('amazing', 0.786135196685791), ('wonder
ful', 0.7835612893104553), ('fantastic', 0.7823663949966431), ('excelle
nt', 0.7751423716545105), ('awesome', 0.7662904858589172), ('perfect',
0.7582064867019653), ('delicious', 0.7171443104743958), ('fabulous', 0.
6927160024642944), ('nice', 0.6804248094558716)]
==================================================
[('compares', 0.9203212261199951), ('ive', 0.9142062664031982), ('softe
st', 0.9127780199050903), ('bahlsen', 0.9018284678459167), ('baklava',
0.8899897933006287), ('closest', 0.8843798637390137), ('premade', 0.880
3465962409973), ('tastiest', 0.8768871426582336), ('world', 0.869809448
7190247), ('stephens', 0.869228720664978)]
```

In [108]:
```
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  8443
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont',
'buying', 'anymore', 'hard', 'find', 'products', 'made', 'usa', 'one',
'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'imp
orts', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding',
'satisfied', 'safe', 'infestation', 'literally', 'everywhere', 'flyin
g', 'around', 'kitchen', 'bought', 'hoping', 'least', 'get', 'rid', 'we
eks', 'fly', 'stuck', 'buggers', 'success', 'rate', 'day']
```

In [109]:
```
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, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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]))
from sklearn.model_selection import train_test_split
```

```
100%|██████████| 20000/20000 [01:02<00:00, 320.26it/s]
```

```
20000
50
```

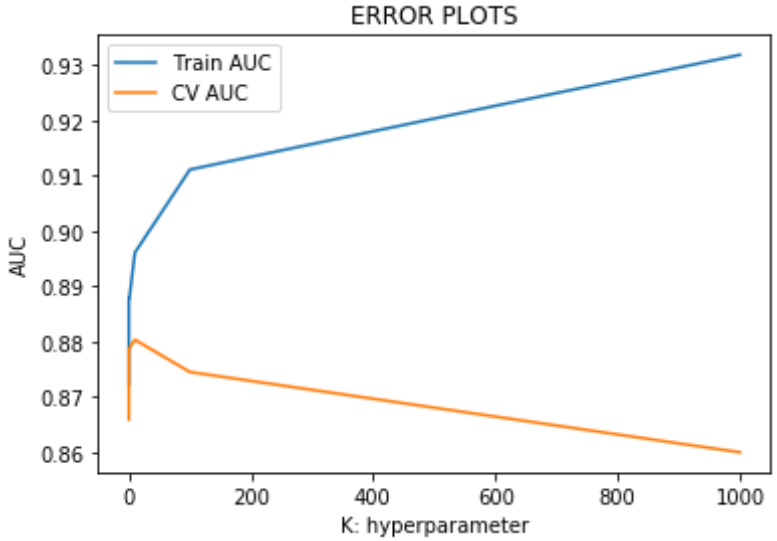
In [110]:

```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(sent_vectors, final
['Score'], test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

In [111]:

```
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
K = [0.001,.01,.1,1,10,100,1000]
for i in K :
    svc = SVC(C=i,kernel='rbf',probability=True)
    svc.fit(X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  svc.predict_proba(X_train)[:,1]
    y_cv_pred =  svc.predict_proba(X_cv)[:,1]

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

plt.plot(K, train_auc, label='Train AUC')
```

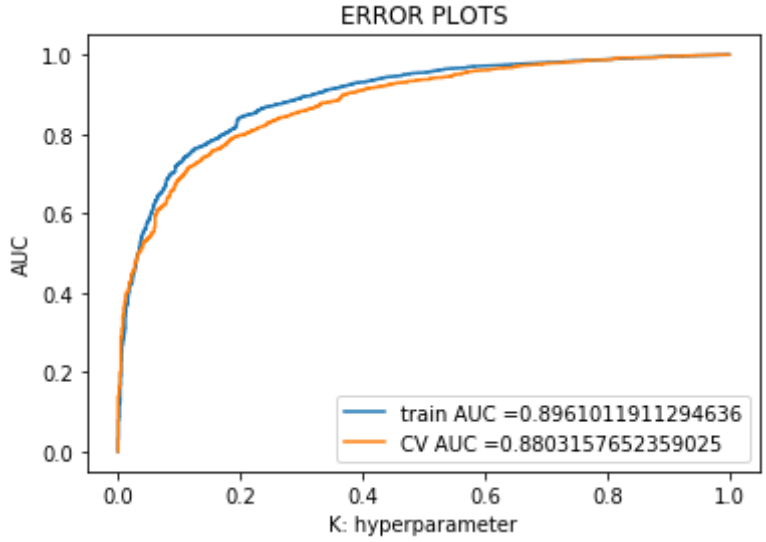
```
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [112]:
```
best_c=10
```

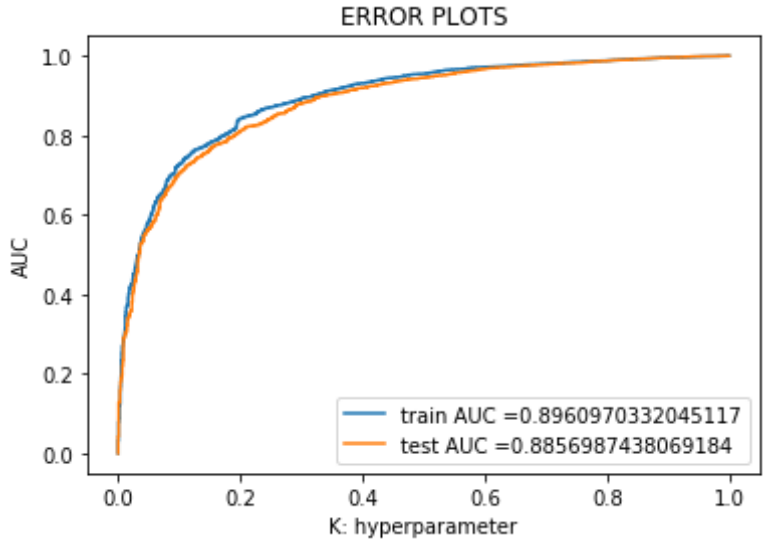
In [113]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, svc.predict_proba(X_cv
```

```
)[:,1])

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



In [114]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
```

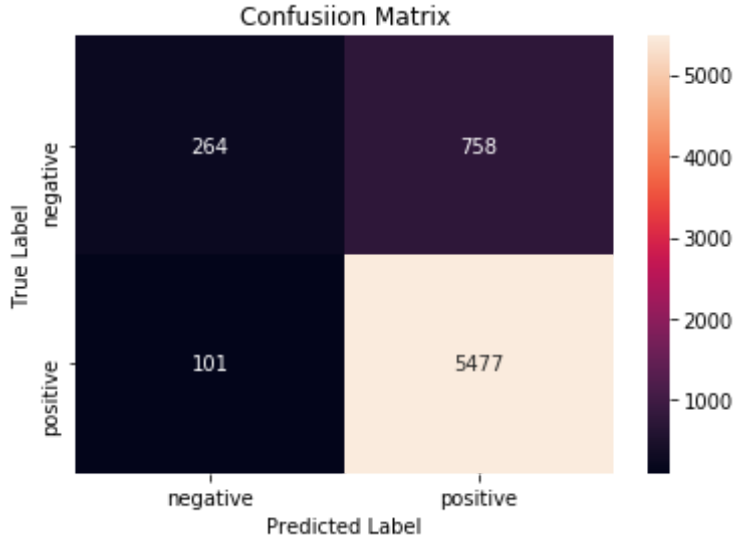
```
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, svc.predict_proba(X_
test)[:,1])

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 [115]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, svc.predict(X_test))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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



[5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

In [3]:
```
# Please write all the code with proper documentation
```

In [116]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'])
# 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 [117]:
```
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
```

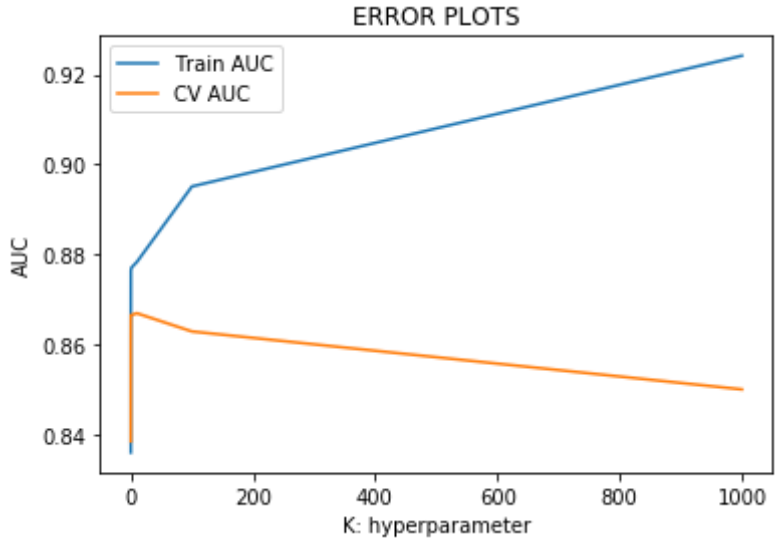
```
tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
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_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 20000/20000 [10:33<00:00, 31.56it/s]
```

In [118]:
```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(tfidf_sent_vectors,
 final['Score'], test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

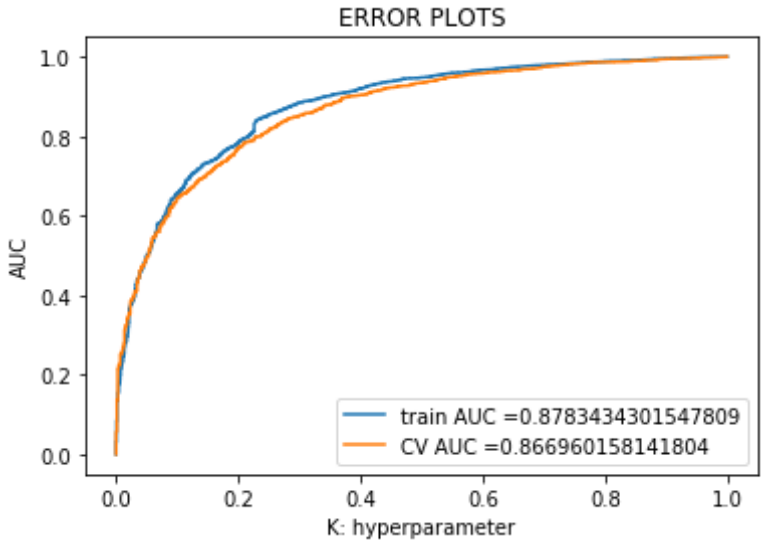
In [119]:
```
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
```

```
K = [0.001,.01,.1,1,10,100,1000]
for i in K :
    svc = SVC(C=i,kernel='rbf',probability=True)
    svc.fit(X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  svc.predict_proba(X_train)[:,1]
    y_cv_pred =  svc.predict_proba(X_cv)[:,1]

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

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



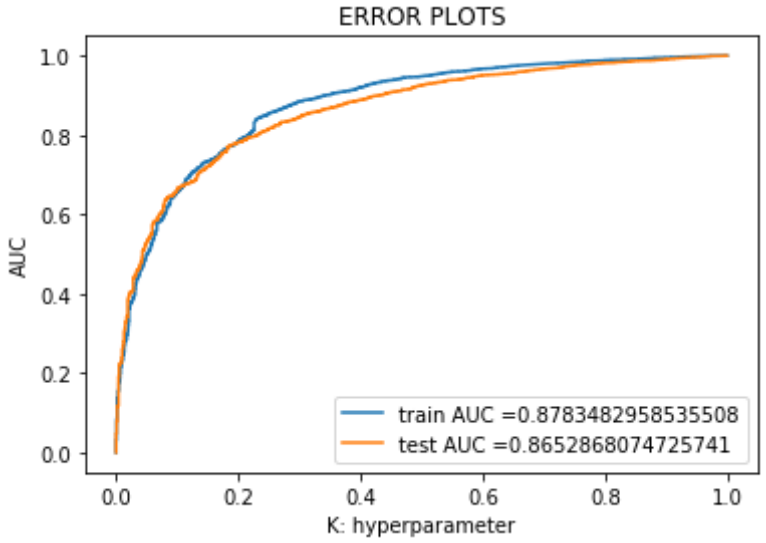
In [120]:
```
best_c=10
```

In [122]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_cv, svc.predict_proba(X_cv
)[:,1])

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



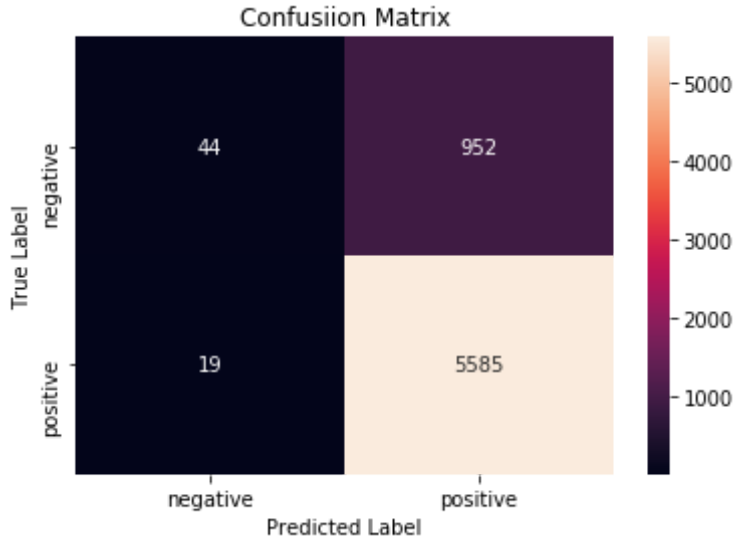
In [123]:
```
from sklearn.metrics import roc_curve, auc


svc =SVC(C=best_c,kernel='rbf',probability=True)
svc.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

train_fpr, train_tpr, thresholds = roc_curve(y_train, svc.predict_proba
(X_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, svc.predict_proba(X_
test)[:,1])

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 [124]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print("Test confusion matrix")
cm=confusion_matrix(y_test, svc.predict(X_test))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, 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
```



[6] Conclusions

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

In [35]:
```
models = pd.DataFrame({'vectorizer': ['SVM with Bow', "SVM with TFIDF",
 "SVM with Avg_w2v", "SVM with tfidf_w2v"], 'Model' : ["Linear kernel",
"Linear kernel","Linear kernel","Linear kernel"],'Hyper Parameter(alph
a)': [10,0.1,1,9], 'AUC':[.56,.59,.90,.82]}, columns = ["vectorizer","M
odel", "Hyper Parameter(alpha)","AUC"])
models
```

Out[35]:

	vectorizer	Model	Hyper Parameter(alpha)	AUC
0	SVM with Bow	Linear kernel	10.0	0.56
1	SVM with TFIDF	Linear kernel	0.1	0.59
2	SVM with Avg_w2v	Linear kernel	1.0	0.90
3	SVM with tfidf_w2v	Linear kernel	9.0	0.82

In [127]:
```
models = pd.DataFrame({'vectorizer': ['SVM with Bow', "SVM with TFIDF",
 "SVM with Avg_w2v", "SVM with tfidf_w2v"], 'Model' : ["RBF kernel","RB
F kernel","RBF kernel","RBF kernel"],'Hyper Parameter(C)': [10,1,10,10
], 'AUC':[.88,.88,.88,.86]}, columns = ["vectorizer","Model", "Hyper Pa
rameter(C)","AUC"])
models
```

Out[127]:

	vectorizer	Model	Hyper Parameter(C)	AUC
0	SVM with Bow	RBF kernel	10	0.88
1	SVM with TFIDF	RBF kernel	1	0.88
2	SVM with Avg_w2v	RBF kernel	10	0.88
3	SVM with tfidf_w2v	RBF kernel	10	0.86

In [ ]:
```

```