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
```

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 500000 data points
# you can change the number to any other number based on your computing power

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

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

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

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

```
Number of data points in our data (525814, 10)
```

Out[2]:

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

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

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
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
```

```
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

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='qui
cksort', na_position='last')
```

In [9]:

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

Out[9]:

```
(364173, 10)
```

In [10]:

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

Out[10]:

```
69.25890143662969
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [12]:

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

In [13]:

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

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

```
(364171, 10)
```

Out[13]:

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

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.

3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [14]:

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

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

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

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

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

In [15]:

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

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [16]:

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

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

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

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

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

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

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [18]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
```

I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they
figured out a way to fix that. I still like it but it could be better.
==

In [19]:

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

this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses: i love all t
he new words this book introduces and the silliness of it all. this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is in college

In [20]:

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

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

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "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%|██████████| 364171/364171 [02:38<00:00, 2300.53it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

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

[3.2] Preprocessing Review Summary

Similartly you can do preprocessing for review summary also.

In [24]:

```
## Summary preprocessing
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].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_summary.append(sentance.strip())
```

```
100%|██████████| 364171/364171 [01:51<00:00, 3280.45it/s]
```

In [23]:

```
final['Cleaned_text'] = preprocessed_reviews
```

In [24]:

```
### Sort data according to time series
final.sort_values('Time',inplace=True)
```

In [25]:

```
### Taking 100k samples
final = final.sample(n=100000)
```

In [26]:

```
x = final['Cleaned_text']
x.size
```

Out[26]:

100000

100000

In [27]:

```
y = final['Score']
y.size
```

Out[27]:

100000

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,random_state=42)
```

[4] Featurization

[4.1] BAG OF WORDS

In [29]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
x_train_bow = count_vect.fit_transform(x_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa', 'aaaaaaaaagghh',
'aaaaaaarrrrrggghhh', 'aaaaaah', 'aaaaaahhh', 'aaaaaahhhh']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 50622)
the number of unique words  50622
```

[4.2] Bi-Grams and n-Grams.

In [31]:

```
#bi-gram, tri-gram and n-gram

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

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (70000, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [30]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
x_train_tfidf = tf_idf_vect.fit_transform(x_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

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

```
some sample features(unique words in the corpus) ['aa', 'abandoned', 'abc', 'abdominal',
'ability', 'able', 'able add', 'able buy', 'able chew', 'able drink']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (30000, 40834)
the number of unique words including both unigrams and bigrams  40834
```

[4.4] Word2Vec

In [31]:

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

In [32]:

```
# Test your own Word2Vec model using your own text corpus
i=0
list_of_sentance_test=[]
for sentance in x_test:
    list_of_sentance_test.append(sentance.split())
```

In [33]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance_train,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=Tr
ue)
```

```
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")
```

```
[('fantastic', 0.8408425450325012), ('awesome', 0.8197475671768188), ('excellent', 0.8184099197387695), ('good', 0.8163270354270935), ('wonderful', 0.8153577446937561), ('terrific', 0.7491371631622314), ('perfect', 0.740469753742218), ('nice', 0.7261111736297607), ('incredible', 0.7181891798973083), ('amazing', 0.6885333061218262)]
==================================================
[('best', 0.7252271175384521), ('nastiest', 0.7250456213951111), ('greatest', 0.6904131174087524), ('disgusting', 0.6440181136131287), ('smoothest', 0.6410361528396606), ('tastiest', 0.6336043477058411), ('closest', 0.629286527633667), ('awful', 0.6092186570167542), ('worse', 0.6057597529220581), ('terrible', 0.602073073387146)]
```

In [34]:

```
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  16081
sample words  ['use', 'cherry', 'juice', 'daily', 'arthritis', 'bought', 'product', 'represents', 'good', 'deal', 'splurged', 'got', 'couple', 'quarts', 'going', 'tough', 'drink', 'nasty', 'taste', 'far', 'tell', 'still', 'effective', 'drinking', 'like', 'bad', 'cough', 'medicine', 'compare', 'dynamic', 'health', 'believe', 'understand', 'long', 'time', 'not', 'used', 'dilute', 'gross', 'tasting', 'also', 'maybe', 'chance', 'amazon', 'shipped', 'date', 'expired', 'tastes', 'terrible', 'whatever']
```

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

[4.4.1.1] Avg W2v

In [35]:

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

```
100%|██████████| 70000/70000 [03:14<00:00, 359.94it/s]
```

```
70000
50
```

In [36]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
```

```
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
100%|██████████| 30000/30000 [01:19<00:00, 378.52it/s]
```

```
30000
50
```

In [37]:

```
x_train_avgw2v=sent_vectors_train
x_test_avgw2v=sent_vectors_test
```

[4.4.1.2] TFIDF weighted W2v

In [38]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_matrix = model.fit_transform(x_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [39]:

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

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

```
100%|██████████| 70000/70000 [04:04<00:00, 286.32it/s]
```

In [40]:

```
tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
```

```
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 30000/30000 [01:28<00:00, 339.22it/s]
```

In [41]:

```
x_train_tfidfw2v = tfidf_sent_vectors_train
x_test_tfidfw2v = tfidf_sent_vectors_test
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression 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. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
   - 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
3. **Pertubation Test**
   - Get the weights W after fit your model with the data X i.e Train data.
   - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
   - Fit the model again on data X' and get the weights W'
   - Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
   - Now find the % change between W and W' (| (W-W') / (W) |)*100)
   - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
   - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
   - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)
4. **Sparsity**
   - Calculate sparsity on weight vector obtained after using L1 regularization

   NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.
5. **Feature importance**
   - Get top 10 important features for both positive and negative classes separately.
6. **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.
7. **Representation of results**

7. **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.

8. **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 Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [42]:

```
from sklearn import preprocessing
x_train_bow = preprocessing.normalize(x_train_bow)
x_test_bow = preprocessing.normalize(x_test_bow)
```

In [43]:

```
from math import log
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]
x =[log(y,10) for y in c_values]
x
```

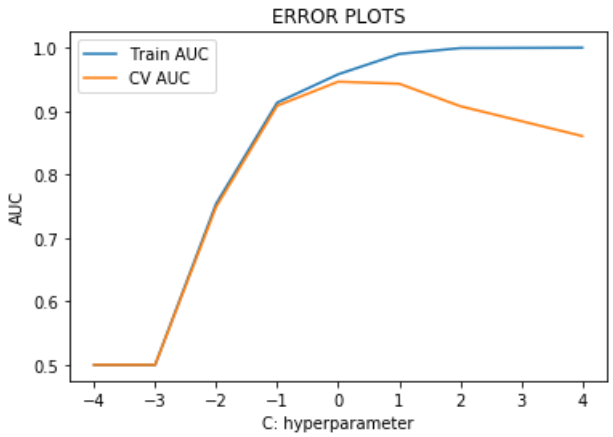
Out[43]:

```
[-3.999999999999999,
 -2.9999999999999996,
 -1.9999999999999996,
 -0.9999999999999998,
 0.0,
 1.0,
 2.0,
 4.0]
```

In [44]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l1')
```

```
    clf = LogisticRegression(C=c,penalty='l1')
    clf.fit(x_train_bow, y_train)
    y_train_pred =  clf.predict_proba(x_train_bow)[:,1]
    y_test_pred =  clf.predict_proba(x_test_bow)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Observation: 1) BY observing AUC curve the hyperparameter will be 0 to 1. 2) as lambda increases model is overfitting .

In [45]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
```

Using GridsearchCV

In [56]:

```
## find hyperparameter alpha using GridserachCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_bow,y_train)
print(model.best_estimator_)
print(model.score(x_test_bow,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

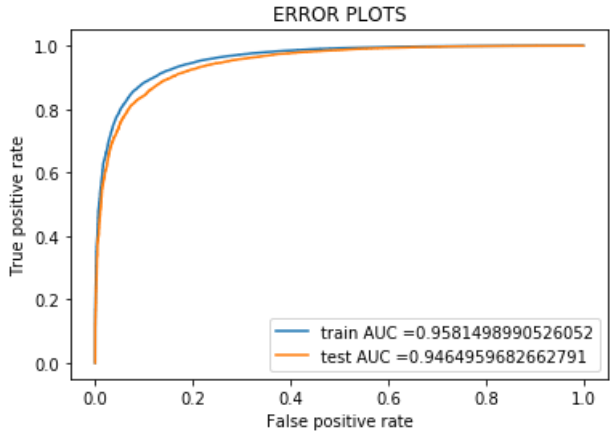
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.947334213745972
Best HyperParameter:  {'C': 1}
Best Accuracy: 93.80%
```

In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [46]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l1')
clf.fit(x_train_bow,y_train)
y_pred = clf.predict(x_test_bow)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, 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, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_bow)))
```




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

Train confusion matrix
[[ 7319  3714]
 [ 1294 57673]]
Test confusion matrix
[[ 2970  1828]
 [  647 24555]]
```

In [58]:

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

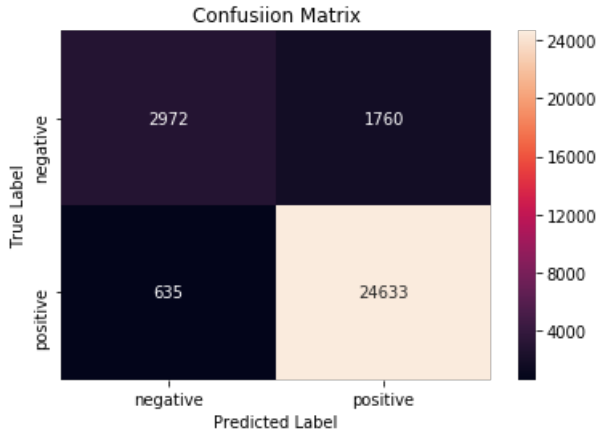
Out[58]:

```
array([[ 2972,  1760],
       [  635, 24633]])
```

In [59]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

In [60]:

```
# Please write all the code with proper documentation
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,precision_score,
f1_score
```

In [61]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=1000,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 13397
```

In [62]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=100,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 11740
```

In [63]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=10,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 6579
```

In [64]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=1,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
```

```
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 1230
```

In [65]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=0.1,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 194
```

In [66]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=0.01,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 9
```

In [67]:

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C=0.001,penalty='l1')
clf.fit(x_train_bow,y_train)
clf.predict(x_test_bow)
print('Number of nonzero weights:',np.count_nonzero(clf.coef_))
```

```
Number of nonzero weights: 0
```

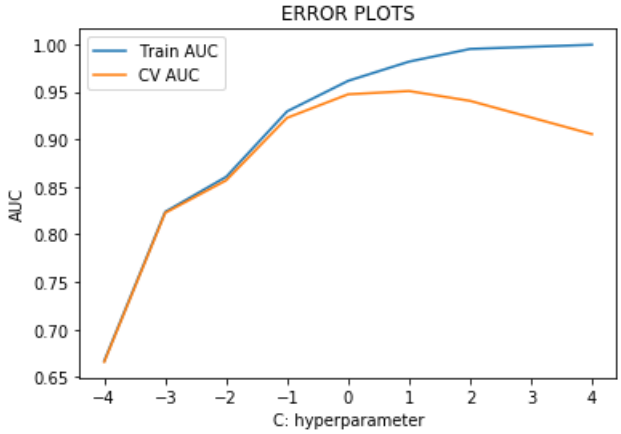
Observation: Sparsity increases from 13397 non-zero weights to only 0 non-zero weights as c value decreases from C=1000 to C = 0.01 when we use L1 Regularization

[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [68]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l2')
    clf.fit(x_train_bow, y_train)
    y_train_pred =  clf.predict_proba(x_train_bow)[:,1]
    y_test_pred =  clf.predict_proba(x_test_bow)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

```
plt.show()
```



Observation: 1) BY observing AUC curve the hyperparameter will be 0 to 1.

In [59]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_bow,y_train)
print(model.best_estimator_)
print(model.score(x_test_bow,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9495144576289032
Best HyperParameter:  {'C': 1}
Best Accuracy: 93.91%
```

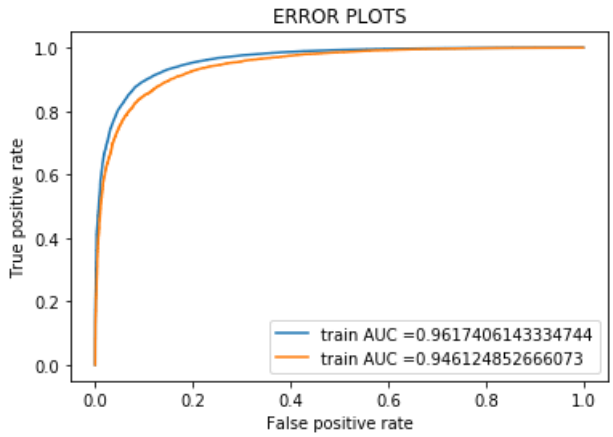
In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [47]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l2')
clf.fit(x_train_bow,y_train)
y_pred = clf.predict(x_test_bow)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, 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, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
```

```
print(confusion_matrix(y_train, clf.predict(x_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_bow)))
```




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

Train confusion matrix
[[ 7005  4028]
 [  951 58016]]
Test confusion matrix
[[ 2747  2051]
 [  525 24677]]
```

In [70]:

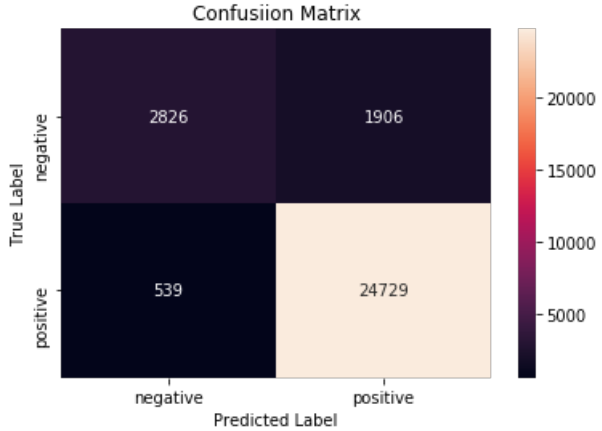
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[70]:

```
array([[ 2826,  1906],
       [  539, 24729]])
```

In [71]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [ ]:

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

In [72]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C=1 , penalty= 'l2')
clf.fit(x_train_bow,y_train)
y_pred = clf.predict(x_test_bow)
```

##Get the weights W after fit your model

In [73]:

```
#Weights before adding random noise
from scipy.sparse import find
weights1 = find(clf.coef_[0])[2]
print(weights1[:50])
```

```
[-2.22066913e-02  1.17462108e-01  7.95481024e-04  8.49135658e-03
  8.87037425e-04  1.18545617e-03  1.30873851e-03  3.42695058e-03
 -1.05032699e-01  1.18258569e-02  1.38415158e-02  3.40067151e-02
  1.67535667e-03  3.58060961e-04 -5.49735064e-02  4.97923918e-03
  8.77145297e-03  3.15094664e-03  2.68082311e-04  6.41081079e-03
  1.51334993e-02  8.57861690e-03  1.34249485e-02  1.95955582e-03
  2.51794767e-03  1.19795543e-02  9.98614649e-02 -1.40141111e-02
 -1.87552079e-01  1.94950161e-03  8.82486487e-03  1.28121973e-03
  8.30218220e-02 -3.99778692e-01  4.00049094e-02  1.94950161e-03
 -6.94668456e-03  4.59689006e-03  1.31223329e-02 -8.46680481e-03
  9.07928827e-03  1.39075825e-03  9.81748272e-02 -5.59193095e-02
 -1.97302092e-03  1.31533938e-02 -3.13416937e-01  8.62637113e-03
  8.62637113e-03 -4.52072490e-02]
```

Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)

In [74]:

```
x_train_t = x_train_bow
#Random noise
epsilon = np.random.uniform(low=-0.0001, high=0.0001, size=(find(x_train_t)[0].size,))
#Getting the postions(row and column) and value of non-zero datapoints
a,b,c = find(x_train_t)

#Introducing random noise to non-zero datapoints
x_train_t[a,b] = epsilon + x_train_t[a,b]
```

We fit the model again on data X' and get the weights W'

In [75]:

```
#Training on train data having random noise
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1, penalty= 'l2')
clf.fit(x_train_t,y_train)
y_pred = clf.predict(x_test_bow)
```

Add the small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6

In [76]:

```
epsilon = np.random.uniform(low=-0.0001, high=0.0001, size=(find(weights1)[0].size,))
#Getting the postions(row and column) and value of non-zero datapoints
a,b,c = find(weights1)

#Introducing random noise to non-zero datapoints
weights2 = epsilon + weights1
```

In [77]:

In [77]:

```
#Weights after adding random noise
from scipy.sparse import find
weights2 = find(clf.coef_[0])[2]
print(weights2[:50])
```

```
[-2.21149795e-02  1.17440165e-01  7.95214539e-04  8.48992022e-03
  8.87343081e-04  1.18513658e-03  1.30824465e-03  3.42558668e-03
 -1.05089939e-01  1.18387340e-02  1.38401902e-02  3.39962667e-02
  1.67320227e-03  3.57832877e-04 -5.50545913e-02  4.97576526e-03
  8.77137335e-03  3.14975312e-03  2.68084027e-04  6.41588996e-03
  1.51482384e-02  8.57512105e-03  1.34126826e-02  1.95786753e-03
  2.51786976e-03  1.19898151e-02  9.98548422e-02 -1.39971057e-02
 -1.87572294e-01  1.94676759e-03  8.80592619e-03  1.28390110e-03
  8.29657635e-02 -3.99774749e-01  3.99512588e-02  1.94980558e-03
 -6.95337318e-03  4.59431355e-03  1.31267480e-02 -8.46375400e-03
  9.07331014e-03  1.39137042e-03  9.81753047e-02 -5.58877698e-02
 -1.97077904e-03  1.31669104e-02 -3.13505994e-01  8.63414721e-03
  8.61238361e-03 -4.51936290e-02]
```

In [78]:

```
print(weights2.size)
## size of weights
```

```
51199
```

In [79]:

```
### find the % change between W and W', percentage_change_vector = (| (W-W') / (W) |)*100)
percentage_change_vector = abs((weights1-weights2) / (weights1))*100
```

calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and ### observe any sudden rise in the values of percentage_change_vector

In [80]:

```
import numpy as np

arr = percentage_change_vector

print("0th percentile of arr : ",  np.percentile(arr, 0))
print("10th percentile of arr : ", np.percentile(arr, 10))
print("20th percentile of arr : ", np.percentile(arr, 20))
print("30th percentile of arr : ",  np.percentile(arr, 30))
print("40th percentile of arr : ", np.percentile(arr, 40))
print("50th percentile of arr : ", np.percentile(arr, 50))
print("60th percentile of arr : ",  np.percentile(arr, 60))
print("70th percentile of arr : ", np.percentile(arr, 70))
print("80th percentile of arr : ", np.percentile(arr, 80))
print("90th percentile of arr : ",  np.percentile(arr,90))
print("100th percentile of arr : ", np.percentile(arr, 100))
```

```
0th percentile of arr :  3.0890766641114905e-06
10th percentile of arr :  0.00726098604414246
20th percentile of arr :  0.014981421551493065
30th percentile of arr :  0.023586560515140145
40th percentile of arr :  0.03315298713428301
50th percentile of arr :  0.04443706948306634
60th percentile of arr :  0.058212906607170635
70th percentile of arr :  0.07626509278742113
80th percentile of arr :  0.10338054109126482
90th percentile of arr :  0.15837987705834558
100th percentile of arr :  17215.770529548878
```

In [81]:

```
### print the feature names whose % change is more than a threshold x(in our example it's 2.5)
print(arr[np.where(arr > 2.5)].size)
```

```
180
```

Observation: 1)There is sudden rise after 90th percentile , at 90th percentile getting 0.16 and at 519.79 th percentile getting 292.7. 2) 180 features have weight changes greater than 2.5. Hence the features are multicollinear

In [82]:

```
 ### print the top 10 colinear features.
feature_names = count_vect.get_feature_names()
coefs_with_fns = sorted(zip(clf.coef_[0], feature_names),reverse=True)[:10]
coefs_with_fns
```

Out[82]:

```
[(6.574196119302709, 'starting'),
 (5.264351753368807, 'fat content'),
 (4.60744418952237, 'occasional'),
 (3.442659755225565, 'routine'),
 (2.9779692755269473, 'thankful'),
 (2.5739759257368084, 'expense'),
 (2.332093160985883, 'bread mix'),
 (1.9707837599411486, 'not lot'),
 (1.9258535290660646, 'afford'),
 (1.8739176254639751, 'hi')]
```

[5.1.3] Feature Importance on BOW, SET 1

[5.1.3.1] Top 10 important features of positive class from SET 1

In [72]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names),reverse=True)[:n]
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("Positive")

print("_____________________________________________________________________________________________
")
    for (coef_1, fn_1),(coef_2, fn_2) in top:
        print("\t%.4f\t%-15s" % (coef_1, fn_1))

show_most_informative_features(count_vect,clf)
```

```
Positive
_____________________________________________________________________________________________
 0.1931 sound
 0.0709 either way
 0.0675 ever used
 0.0613 boxed
 0.0523 tastes pretty
 0.0395 failed
 0.0363 extremely
 0.0301 not expected
 0.0298 not understand
 0.0257 advertising
```

[5.1.3.2] Top 10 important features of negative class from SET 1

In [ ]:

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

In [75]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
```

```
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))[:n]
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("Negative")

print("_____________________________________________________________________________________________
")
    for (coef_1, fn_1),(coef_2, fn_2) in top:
        print("\t%.4f\t%-15s" % (coef_2, fn_2))

show_most_informative_features(count_vect,clf)
```

```
Negative

_____________________________________________________________________________________________
 -0.0163 quit
 -0.0170 got one
 -0.0172 started
 -0.0182 ones
 -0.0186 coffee flavor
 -0.0203 tea drinker
 -0.0212 thought would
 -0.0492 not want
 -0.0538 not take
 -0.1035 overly
```

[5.2] Logistic Regression on TFIDF, SET 2

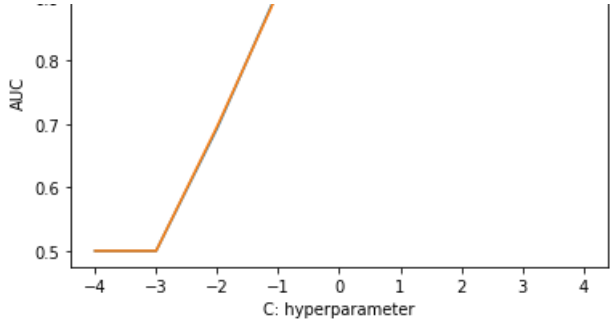
[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [48]:

```
from sklearn import preprocessing
x_train_tfidf = preprocessing.normalize(x_train_tfidf)
x_test_tfidf = preprocessing.normalize(x_test_tfidf)
```

In [84]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l1')
    clf.fit(x_train_tfidf, y_train)
    y_train_pred =  clf.predict_proba(x_train_tfidf)[:,1]
    y_test_pred =  clf.predict_proba(x_test_tfidf)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Using GridsearchCV

In [78]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_tfidf,y_train)
print(model.best_estimator_)
print(model.score(x_test_tfidf,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9631367327138368
Best HyperParameter:  {'C': 1}
Best Accuracy: 95.34%
```

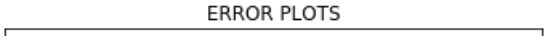
Observation: 1) BY observing AUC curve the hyperparameter will be 1.

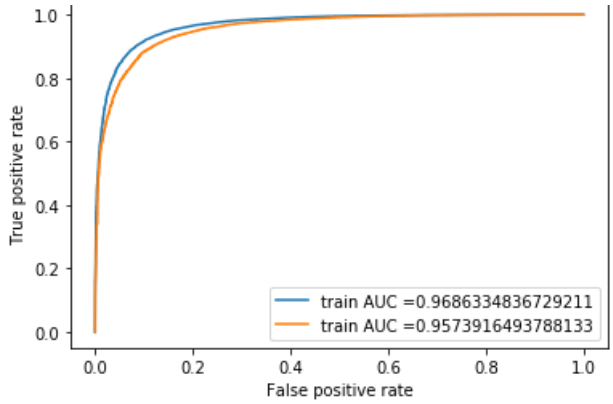
In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [49]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l1')
clf.fit(x_train_tfidf,y_train)
y_pred = clf.predict(x_test_tfidf)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, 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, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_tfidf)))
```





==========

```
Train confusion matrix
[[ 7724  3309]
 [ 1000 57967]]
Test confusion matrix
[[ 3105  1693]
 [  506 24696]]
```

In [86]:

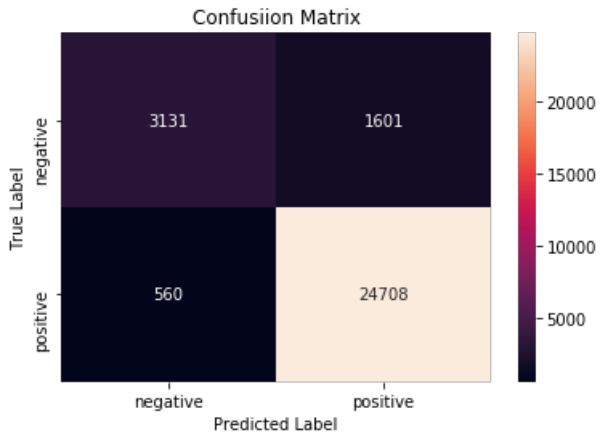
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[86]:

```
array([[ 3131,  1601],
       [  560, 24708]])
```

In [87]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



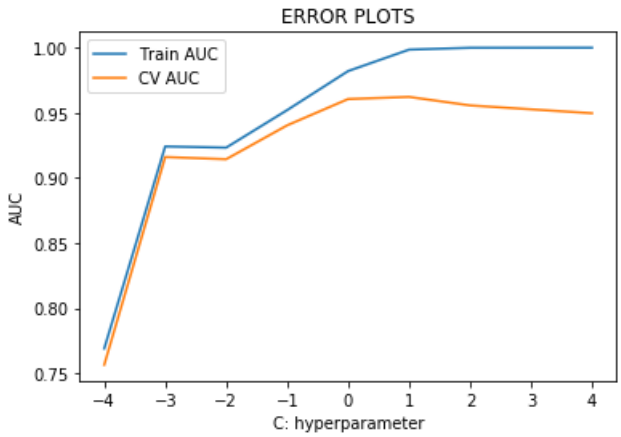
[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [ ]:

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

In [88]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l2')
    clf.fit(x_train_tfidf, y_train)
    y_train_pred =  clf.predict_proba(x_train_tfidf)[:,1]
    y_test_pred =  clf.predict_proba(x_test_tfidf)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [83]:

```
## Using GridsearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_tfidf,y_train)
print(model.best_estimator_)
print(model.score(x_test_tfidf,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9631367327138368
Best HyperParameter:  {'C': 1}
Best Accuracy: 95.34%
```

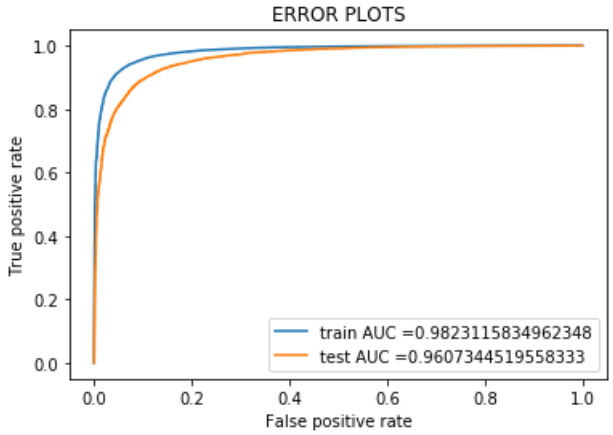
Observation: 1) BY observing AUC curve the hyperparameter will be 0 to 1.

In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [50]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l2')
clf.fit(x_train_tfidf,y_train)
y_pred = clf.predict(x_test_tfidf)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, 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, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_tfidf)))
```




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

Train confusion matrix
[[ 7596  3437]
 [  472 58495]]
Test confusion matrix
[[ 2817  1981]
 [  344 24858]]
```

In [90]:

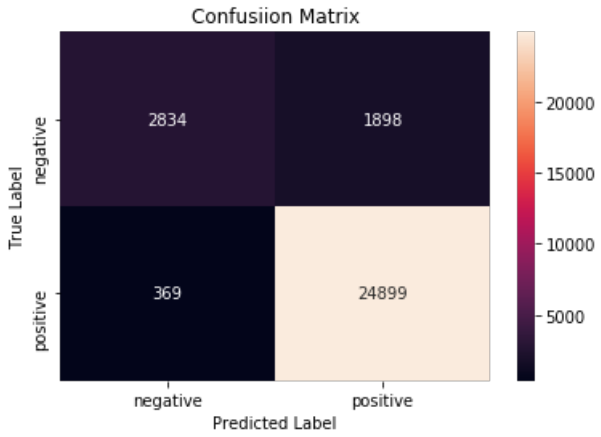
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[90]:

```
array([[ 2834,  1898],
       [  369, 24899]])
```

In [91]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



[5.2.3] Feature Importance on TFIDF, SET 2

[5.2.3.1] Top 10 important features of positive class from SET 2

In [ ]:

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

In [87]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names),reverse=True)[:n]
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("Positive")

print("_____________________________________________________________________________________
")
    for (coef_1, fn_1),(coef_2, fn_2) in top:
        print("\t%.4f\t%-15s" % (coef_1, fn_1))

show_most_informative_features(tf_idf_vect,clf)
```

```
Positive
_____________________________________________________________________________________
 12.0046 great
 8.8474 delicious
 8.4005 best
 7.5891 good
 7.4020 perfect
 7.3351 love
 6.0345 excellent
 5.7824 loves
 5.4981 not disappointed
 5.4709 wonderful
```

[5.2.3.2] Top 10 important features of negative class from SET 2

In [ ]:

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

In [88]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))[:n]
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("Negative")

print("_____________________________________________________________________________________________
")
    for (coef_1, fn_1),(coef_2, fn_2) in top:
        print("\t%.4f\t%-15s" % (coef_2, fn_2))

show_most_informative_features(tf_idf_vect,clf)
```

```
Negative
_____________________________________________________________________________________________
 -5.5265 not buy
 -5.7485 not worth
 -6.0263 horrible
 -6.5391 terrible
 -6.6606 not good
 -6.7342 awful
 -6.9716 disappointing
 -7.3731 worst
 -8.0015 not
 -8.3134 disappointed
```

[5.3] Logistic Regression on AVG W2V, SET 3

[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

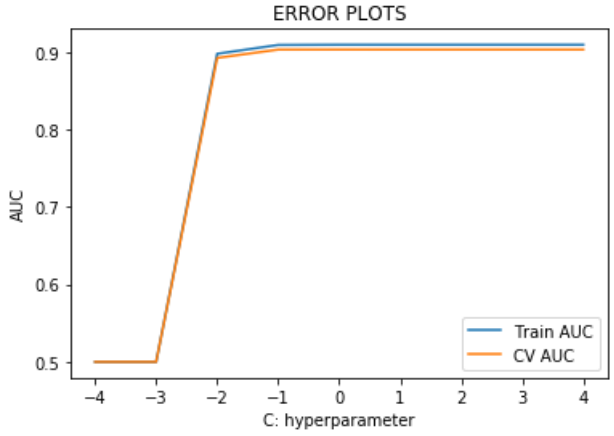
In [51]:

```
from sklearn import preprocessing
x_train_avgw2v = preprocessing.normalize(x_train_avgw2v)
x_test_avgw2v = preprocessing.normalize(x_test_avgw2v)
```

In [52]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l1')
    clf.fit(x_train_avgw2v, y_train)
    y_train_pred =  clf.predict_proba(x_train_avgw2v)[:,1]
    y_test_pred =  clf.predict_proba(x_test_avgw2v)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
```

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



In [53]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_avgw2v,y_train)
print(model.best_estimator_)
print(model.score(x_test_avgw2v,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9030742687042015
Best HyperParameter:  {'C': 1}
Best Accuracy: 90.83%
```

Observation: 1) BY observing the hyperparameter will be 1.

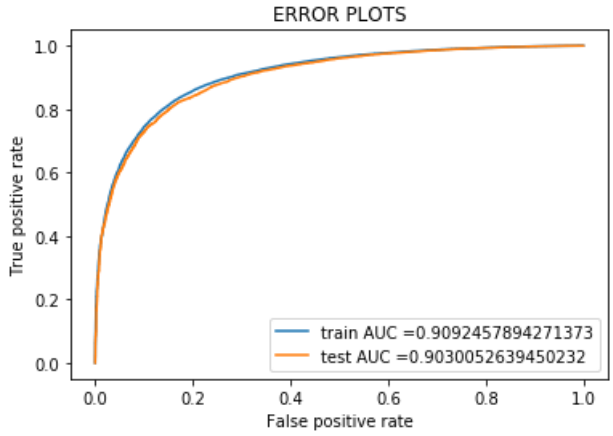
In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [54]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l1')
clf.fit(x_train_avgw2v,y_train)
y_pred = clf.predict(x_test_avgw2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_avgw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_avgw2v)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)
```

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_avgw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_avgw2v)))
```



==========

```
Train confusion matrix
[[ 5507  5526]
 [ 2115 56852]]
Test confusion matrix
[[ 2271  2527]
 [  884 24318]]
```

In [55]:

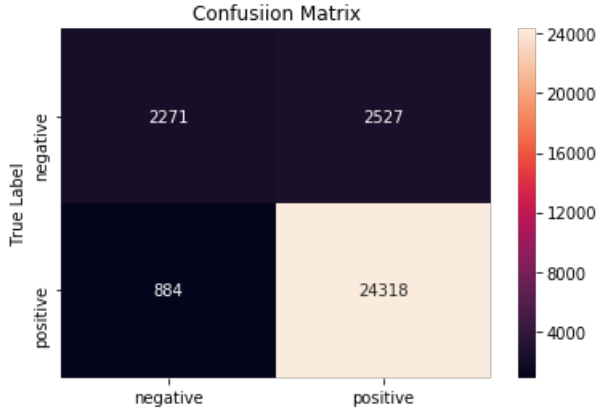
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[55]:

```
array([[ 2271,  2527],
       [  884, 24318]])
```

In [56]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



Predicted Label

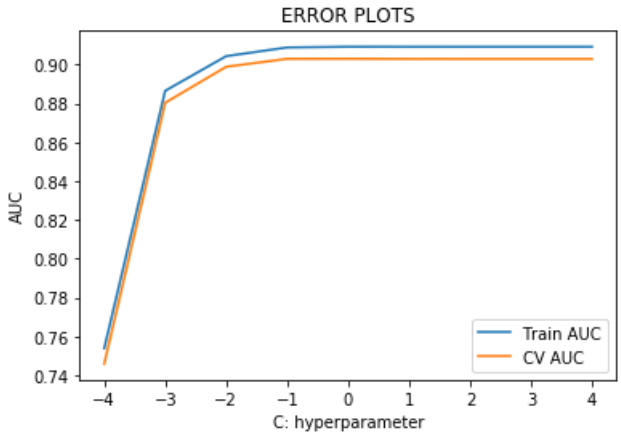
[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [ ]:

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

In [57]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l2')
    clf.fit(x_train_avgw2v, y_train)
    y_train_pred =  clf.predict_proba(x_train_avgw2v)[:,1]
    y_test_pred =  clf.predict_proba(x_test_avgw2v)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [58]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_avgw2v,y_train)
print(model.best_estimator_)
print(model.score(x_test_avgw2v,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
```

```
          tol=0.0001, verbose=0, warm_start=False)
0.9030742687042015
Best HyperParameter:  {'C': 1}
Best Accuracy: 90.83%
```

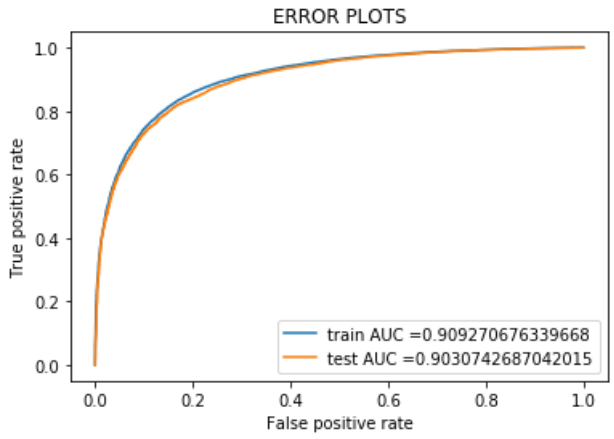
Observation: 1) BY observing AUC curve the hyperparameter will be 1.

In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [59]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l2')
clf.fit(x_train_avgw2v,y_train)
y_pred = clf.predict(x_test_avgw2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_avgw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_avgw2v)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_avgw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_avgw2v)))
```




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

Train confusion matrix
[[ 5466  5567]
 [ 2083 56884]]
Test confusion matrix
[[ 2251  2547]
 [  863 24339]]
```

In [60]:

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
```

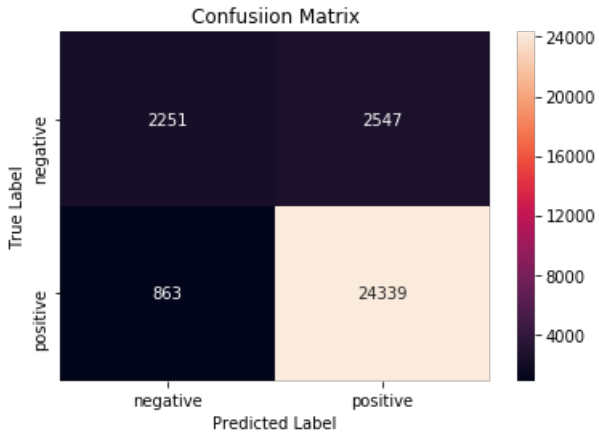
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[60]:

```
array([[ 2251,  2547],
       [  863, 24339]])
```

In [61]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



[5.4] Logistic Regression on TFIDF W2V, SET 4

[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

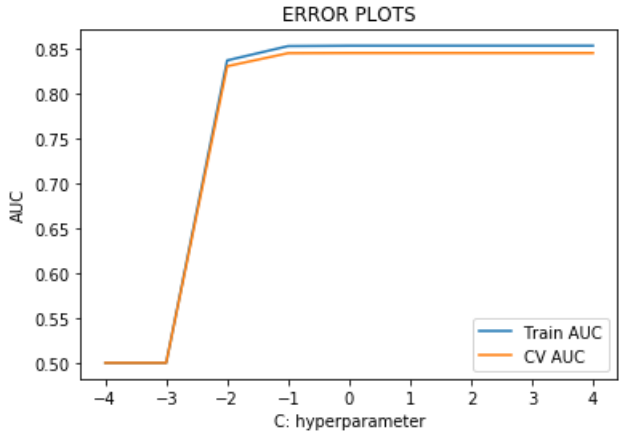
In [62]:

```
from sklearn import preprocessing
x_train_tfidfw2v = preprocessing.normalize(x_train_tfidfw2v)
x_test_tfidfw2v = preprocessing.normalize(x_test_tfidfw2v)
```

In [63]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l1')
    clf.fit(x_train_tfidfw2v, y_train)
    y_train_pred =  clf.predict_proba(x_train_tfidfw2v)[:,1]
    y_test_pred =  clf.predict_proba(x_test_tfidfw2v)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
```

```
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [64]:

```
### Using GridsearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_tfidfw2v,y_train)
print(model.best_estimator_)
print(model.score(x_test_tfidfw2v,y_test))
print("Best HyperParameter: ",model.best_params_)
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.8447809849810778
Best HyperParameter:  {'C': 1}
Best Accuracy: 85.23%
```

Observation: 1) BY observing AUC curve the hyperparameter will be 1.

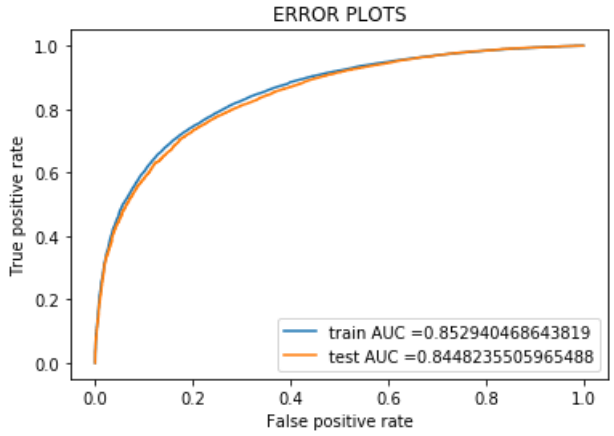
In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [65]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l1')
clf.fit(x_train_tfidfw2v,y_train)
y_pred = clf.predict(x_test_tfidfw2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tfidfw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_tfidfw2v)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
```

```
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_tfidfw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_tfidfw2v)))
```




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

Train confusion matrix
[[ 3358  7675]
 [ 1811 57156]]
Test confusion matrix
[[ 1409  3389]
 [  736 24466]]
```

In [66]:

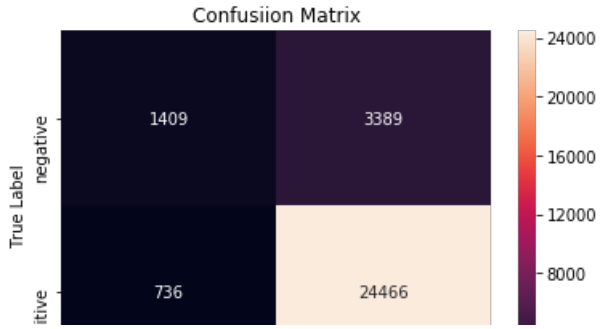
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

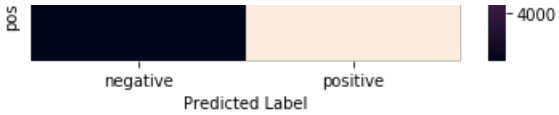
Out[66]:

```
array([[ 1409,  3389],
       [  736, 24466]])
```

In [67]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



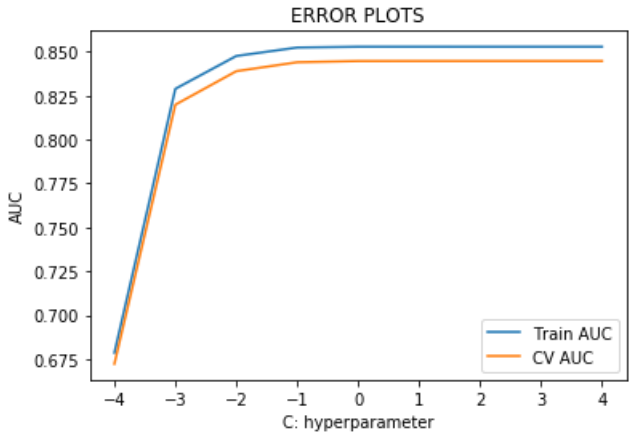


[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

Please write all the code with proper documentation

In [68]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
c_values = [10**-4,10**-3, 10**-2,10**-1, 10**0,10, 10**2, 10**4]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for c in c_values:
    clf = LogisticRegression(C=c,penalty='l2')
    clf.fit(x_train_tfidfw2v, y_train)
    y_train_pred =  clf.predict_proba(x_train_tfidfw2v)[:,1]
    y_test_pred =  clf.predict_proba(x_test_tfidfw2v)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))


#plot cross-validated score, training score vs alpha
plt.plot(x, training_scores, label='Train AUC')
plt.plot(x, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [69]:

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
```

In [70]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model = GridSearchCV(LogisticRegression(),tuned_parameters,cv=tscv,scoring='roc_auc')
model.fit(x_train_tfidfw2v,y_train)
print(model.best_estimator_)
print(model.score(x_test_tfidfw2v,y_test))
print("Best HyperParameter: ",model.best_params_)
```

```
print("Best Accuracy: %.2f%%"%(model.best_score_*100))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.8447809849810778
Best HyperParameter:  {'C': 1}
Best Accuracy: 85.23%
```

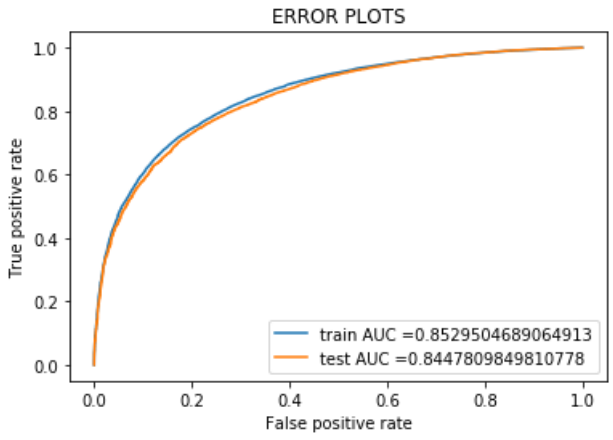
Observation: 1) BY observing AUC curve the hyperparameter will be 1.

In [ ]:

```
### ROC Curve using false positive rate versus true positive rate
```

In [71]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C=1,penalty='l2')
clf.fit(x_train_tfidfw2v,y_train)
y_pred = clf.predict(x_test_tfidfw2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tfidfw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_tfidfw2v)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_tfidfw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_tfidfw2v)))
```




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

Train confusion matrix
[[ 3335  7698]
 [ 1791 57176]]
Test confusion matrix
[[ 1393  3405]
 [  726 24476]]
```

In [72]:

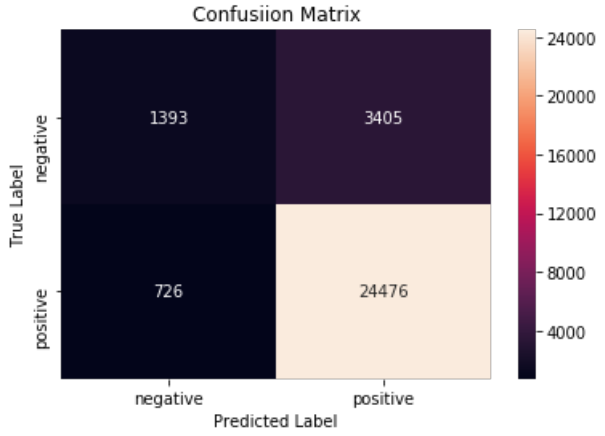
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[72]:

```
array([[ 1393,  3405],
       [  726, 24476]])
```

In [73]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
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()
```



[6] Conclusions

In [ ]:

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

In [74]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Regularization","Hyperparameter","AUC"]

x.add_row(["BOW", "l1","1","94.6%"])
x.add_row(["TFIDF","l1","1","95.7%"])
x.add_row(["AvgW2V","l1","1","90%"])
x.add_row(["TFIDF W2V","l1","1","84.4%"])
x.add_row(["BOW","l2","1","94.6%"])
x.add_row(["TFIDF","l2","1","96%"])
x.add_row(["AvgW2V","l2","1","90%"])
x.add_row(["TFIDF W2V","l2","1","84.4%"])

print(x)
```

```
+------------+----------------+----------------+-------+
| Vectorizer | Regularization | Hyperparameter |  AUC  |
+------------+----------------+----------------+-------+
```

```
|     BOW     |       l1       |       1        | 94.6% |
|    TFIDF    |       l1       |       1        | 95.7% |
|    AvgW2V   |       l1       |       1        |  90%  |
|  TFIDF W2V  |       l1       |       1        | 84.4% |
|     BOW     |       l2       |       1        | 94.6% |
|    TFIDF    |       l2       |       1        |  96%  |
|    AvgW2V   |       l2       |       1        |  90%  |
|  TFIDF W2V  |       l2       |       1        | 84.4% |
+-------------+----------------+----------------+-------+
```

1) Using 'l1' regularization, as 'C' value decreases number of nonzero weights decreases. 2) TFIDF with l2 regularization give more accuracy 96%. 3) Logistic regressin has takes less computation time.

In [ ]: