Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[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
#print(os.getcwd())
```

In [137]:

```
# 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 50
0000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1000
00""", con)

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

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

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

Out[137]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [138]:

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

In [139]:

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

```
(80668, 7)
```

Out[139]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [140]:

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

Out[140]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [141]:

```
display['COUNT(*)'].sum()
```

Out[141]:

```
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 [142]:

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

Out[142]:

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

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

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

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

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

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

In [143]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
e, kind='quicksort', na_position='last')
```

In [144]:

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

Out[144]:

```
(87775, 10)
```

In [145]:

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

Out[145]:

```
87.775
```

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

In [146]:

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

display.head()
```

Out[146]:

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

In [147]:

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

In [148]:

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

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

```
(87773, 10)
```

Out[148]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [149]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buy
ing it anymore.  Its very hard to find any chicken products made in the US
A but they are out there, but this one isnt.  Its too bad too because its
a good product but I wont take any chances till they know what is going on
with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the ca
ndy has little taste to it.  Very little of the 2 lbs that I bought were e
aten and I threw the rest away.  I would not buy the candy again.
==================================================
```

In [150]:

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


print(sent_0)
```

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

In [151]:

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

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

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

In [152]:

```
# 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 [153]:

```
sent_0 = decontracted(sent_0)
print(sent_0)
print("="*50)
```

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

In [154]:

```
# 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', 'ourselve
s', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [155]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
```

In [156]:

```
Y =final['Score'].values
X=final['Text'].values
print(X.shape)
print(Y.shape)
```

```
(87773,)
(87773,)
```

In [157]:

```
print(type(Y))
```

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

In [158]:

```
# Combining all the above stundents
from bs4 import BeautifulSoup
from tqdm import tqdm
X = []
# 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 stopwor
ds)
    X.append(sentance.strip())
```

```
100%|█████████████████████████████████████████████████████████████████████
█████| 87773/87773 [01:20<00:00, 1085.63it/s]
```

In [159]:

```
print(type(X))
X[25000]
```

```
<class 'list'>
```

Out[159]:

```
'great product delicious sour tea potent wonderful quality dried hibiscus
petals hot mug water honey makes great tea good lowers blood pressure woul
d business vendor'
```

In [160]:

```
# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Fla
se)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=
1) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, rando
m_state=2) # this is random splitting


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

print("="*100)
```

```
39400 (39400,)
19407 (19407,)
28966 (28966,)
==========================================================================
==========================
```

In [161]:

```
import collections, numpy
print(collections.Counter(y_train))
print(collections.Counter(y_cv))
print(collections.Counter(y_test))
```

```
Counter({1: 33075, 0: 6325})
Counter({1: 16225, 0: 3182})
Counter({1: 24292, 0: 4674})
```

[4] Featurization

[4.1] BAG OF WORDS

In [162]:

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

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaah', 'aaaand', 'aaahs', 'aa
chen', 'aaf', 'aafco', 'aamazon']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (39400, 37328)
the number of unique words  37328
```

In [163]:

```
# we use the fitted CountVectorizer to convert the text to vector
#.transform to apply on train, cv and test which will give set 1 of vectorized data
set1_train = count_vect.transform(X_train)
set1_cv = count_vect.transform(X_cv)
set1_test = count_vect.transform(X_test)

print("After vectorizations")
print(set1_train.shape, y_train.shape)
print(set1_cv.shape, y_cv.shape)
print(set1_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(39400, 37328) (39400,)
(19407, 37328) (19407,)
(28966, 37328) (28966,)
===========================================================================
=========================
```

[4.2] Bi-Grams and n-Grams.

In [29]:

```
#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/modul
es/generated/sklearn.feature_extraction.text.CountVectorizer.html

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

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-29-5f930c5a2a76> in <module>
      7 # you can choose these numebrs min_df=10, max_features=5000, of yo
ur choice
      8 count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_fea
tures=5000)
----> 9 final_bigram_counts = count_vect.fit_transform(preprocessed_review
s)
     10 print("the type of count vectorizer ",type(final_bigram_counts))
     11 print("the shape of out text BOW vectorizer ",final_bigram_counts.
get_shape())

NameError: name 'preprocessed_reviews' is not defined
```

[4.3] TF-IDF

In [164]:

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

set2_train= tf_idf_vect.transform(X_train)
set2_cv = tf_idf_vect.transform(X_cv)
set2_test = tf_idf_vect.transform(X_test)
print("the type of count vectorizer ",type(set2_train))
print("the shape of out text TFIDF vectorizer ",set2_train.get_shape())

print(set2_cv.shape, y_cv.shape)
print(set2_test.shape, y_test.shape)
print("the number of unique words including both unigrams and bigrams ", set2_train.get
_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able
buy', 'able drink', 'able eat', 'able enjoy', 'able find', 'able finish',
'able get', 'able make']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (39400, 23410)
(19407, 23410) (19407,)
(28966, 23410) (28966,)
the number of unique words including both unigrams and bigrams  23410
```

In [165]:

```
# we use the fitted CountVectorizer to convert the text to vector
#.transform to apply on train, cv and test which will give set 1 of vectorized data
set2_train = tf_idf_vect.transform(X_train)
set2_cv = tf_idf_vect.transform(X_cv)
set2_test = tf_idf_vect.transform(X_test)

print("After vectorizations")
print(set2_train.shape, y_train.shape)
print(set2_cv.shape, y_cv.shape)
print(set2_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(39400, 23410) (39400,)
(19407, 23410) (19407,)
(28966, 23410) (28966,)
==========================================================================
==========================
```

[4.4] Word2Vec

In [166]:

```
# 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 [167]:

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


# 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'))

# this line of code trains your w2v model on the give list of sentances
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])
```

```
[('fantastic', 0.8205050826072693), ('terrific', 0.8114068508148193), ('aw
esome', 0.8072959184646606), ('wonderful', 0.7883588075637817), ('good',
0.7850236892700195), ('excellent', 0.7846232652664185), ('amazing', 0.7379
043102264404), ('perfect', 0.7345172166824341), ('fabulous', 0.69020152091
97998), ('nice', 0.6776461005210876)]
==================================================
[('best', 0.7172499299049377), ('greatest', 0.682648241519928), ('hottes
t', 0.6544857025146484), ('ive', 0.6471073627471924), ('tastiest', 0.64595
80659866333), ('toughest', 0.638321042060852), ('experienced', 0.635458409
7862244), ('smoothest', 0.6257159113883972), ('awful', 0.611542046070098
9), ('hardly', 0.6045807600021362)]
number of words that occured minimum 5 times  11996
sample words  ['syrup', 'help', 'eliminate', 'sugar', 'morning', 'coffee',
'pack', 'last', 'months', 'using', 'every', 'starbucks', 'charges', 'almos
t', 'couple', 'squirts', 'order', 'dispenser', 'use', 'measuring', 'cap',
'one', 'best', 'flavored', 'coffees', 'tried', 'usually', 'not', 'like',
'great', 'serve', 'company', 'love', 'discovered', 'product', 'make', 'fee
l', 'someone', 'belongs', 'way', 'italian', 'wafer', 'melts', 'mouth', 'cr
eates', 'sensation', 'difficult', 'describe', 'far', 'think']
```

In [168]:

```
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  11996
sample words  ['syrup', 'help', 'eliminate', 'sugar', 'morning', 'coffee',
'pack', 'last', 'months', 'using', 'every', 'starbucks', 'charges', 'almos
t', 'couple', 'squirts', 'order', 'dispenser', 'use', 'measuring', 'cap',
'one', 'best', 'flavored', 'coffees', 'tried', 'usually', 'not', 'like',
'great', 'serve', 'company', 'love', 'discovered', 'product', 'make', 'fee
l', 'someone', 'belongs', 'way', 'italian', 'wafer', 'melts', 'mouth', 'cr
eates', 'sensation', 'difficult', 'describe', 'far', 'think']
```

In [169]:

```
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  11996
sample words  ['syrup', 'help', 'eliminate', 'sugar', 'morning', 'coffee',
'pack', 'last', 'months', 'using', 'every', 'starbucks', 'charges', 'almos
t', 'couple', 'squirts', 'order', 'dispenser', 'use', 'measuring', 'cap',
'one', 'best', 'flavored', 'coffees', 'tried', 'usually', 'not', 'like',
'great', 'serve', 'company', 'love', 'discovered', 'product', 'make', 'fee
l', 'someone', 'belongs', 'way', 'italian', 'wafer', 'melts', 'mouth', 'cr
eates', 'sensation', 'difficult', 'describe', 'far', 'think']
```

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

[4.4.1.1] Avg W2v

In [170]:

```
from tqdm import tqdm
import numpy as np
```

In [171]:

```
# average Word2Vec
# compute average word2vec for each review.
set3_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
    set3_train.append(sent_vec)
set3_train = np.array(set3_train)
print(set3_train.shape)
print(set3_train[0])
```

```
100%|█████████████████████████████████████████████████████████████████████
██████| 39400/39400 [01:58<00:00, 331.74it/s]

(39400, 50)
[-0.65954047 -0.02134442  0.37980434 -0.67014372 -0.04420673 -1.02085792
 -0.61764176 -0.47749436 -0.21310703 -0.25350874 -0.9720033  -0.53598936
 -0.90283787 -0.13977595  0.3592658   0.15279117 -0.35706392 -0.33193417
  0.40275934 -0.28535814 -0.67419392 -0.72001847 -1.07610162  0.30793584
 -0.52388009 -0.37583736 -0.3900439   0.28207502 -0.64110006  0.31354867
 -0.03339415 -0.27896628  0.54609524  0.02299254  0.00750757 -0.25573534
  1.07466302 -0.80808846 -0.14821317  1.03607471 -0.20921346 -0.26414978
  0.3438609   0.25077736 -0.55760604  0.33899882 -0.02551107 -0.31703716
  0.15369314  0.38094912]
```

In [172]:

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

In [173]:

```
# average Word2Vec
# compute average word2vec for each review.
set3_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
 change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    set3_cv.append(sent_vec)
set3_cv = np.array(set3_cv)
print(set3_cv.shape)
print(set3_cv[0])
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 19407/19407 [00:59<00:00, 325.69it/s]

(19407, 50)
[-3.18625496e-01  3.10046264e-01  1.33429504e-01  3.74176838e-02
 -9.78291666e-01 -3.70962601e-01 -1.65990875e-01  3.14016267e-01
  5.83470189e-01 -1.08292106e+00 -3.95399066e-01  9.27181545e-02
 -6.80040980e-01 -3.30301377e-01  1.31544450e-01 -3.04389573e-01
  4.20207213e-01  6.35197497e-01  5.43654151e-02  5.35812055e-01
 -1.08372369e+00 -4.54652707e-01 -6.83708588e-01  4.51795138e-01
  8.47596675e-04  1.90110753e-02  8.88738954e-01 -1.62895019e-01
 -2.48367649e-01 -7.76213648e-02  1.31445459e-01  1.78353139e-01
 -1.22539138e-01 -4.61541886e-02 -1.51050511e-01 -2.77722424e-01
 -1.38536963e-01  1.27238097e-02 -3.53009999e-01  5.03266513e-01
 -4.00182268e-01 -6.32098946e-02  3.33838806e-01  5.31893004e-01
  3.30832482e-01  2.06092222e-02 -5.02309730e-01  8.33472810e-01
  6.88289073e-01  3.12697995e-01]
```

In [174]:

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

In [175]:

```
# average Word2Vec
# compute average word2vec for each review.
set3_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
    set3_test.append(sent_vec)
set3_test = np.array(set3_test)
print(set3_test.shape)
print(set3_test[0])
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 28966/28966 [01:29<00:00, 323.05it/s]

(28966, 50)
[-0.22907626 -0.44096636 -0.08889248  0.15834944 -0.47234945 -0.18999585
 -0.61963913 -0.66474266  0.14854186 -0.52536492 -0.12396148 -0.74350531
  0.1222549   0.0366992   0.18760397  0.28978676  0.10805948 -0.19825645
 -0.18295148  0.39653832 -0.47777517 -0.48142658 -0.77874688  0.6314125
  0.14976739  0.16498533 -0.04610641 -0.38402684 -0.71615476  0.05537617
 -0.05769803 -0.74908292  0.71500615  0.27700968 -0.52671422 -0.13119344
  0.52477308  0.69780508 -0.61727612  0.39042168 -0.247436   -0.30384037
  0.13016302  0.12668282 -0.28177586  0.28068253 -0.20246389  0.24785945
  0.1610972   0.15985047]
```

[4.4.1.2] TFIDF weighted W2v

In [194]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
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 [195]:

```
# 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

set4_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
    set4_train.append(sent_vec)
    row += 1
print(len(set4_train))
print(len(set4_train[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
███████| 39400/39400 [20:38<00:00, 31.80it/s]

39400
50
```

In [196]:

```
set4_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    set4_cv.append(sent_vec)
    row += 1
print(len(set4_cv))
print(len(set4_cv[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 19407/19407 [10:03<00:00, 32.15it/s]

19407
50
```

In [197]:

```
set4_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
    set4_test.append(sent_vec)
    row += 1
print(len(set4_test))
print(len(set4_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 28966/28966 [15:24<00:00, 31.34it/s]

28966
50
```

In [43]:

```
def cv_svm(x_train,y_train,x_cv,y_cv):
    alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
    auc_train=[]
    auc_cv=[]
    for a in alpha:
        model=SGDClassifier(alpha=a) #loss default hinge
        svm=CalibratedClassifierCV(model, cv=3) #calibrated classifier cv for calculati
on of predic_proba
        svm.fit(x_train,y_train)
        probcv=svm.predict_proba(x_cv)[:,1]
        auc_cv.append(roc_auc_score(y_cv,probcv))
        probtr=svm.predict_proba(x_train)[:,1]
        auc_train.append(roc_auc_score(y_train,probtr))
    optimal_alpha= alpha[auc_cv.index(max(auc_cv))]
    alpha=[math.log(x) for x in alpha]#converting values of alpha into logarithm
    fig = plt.figure()
    ax = plt.subplot(111)
    ax.plot(alpha, auc_train, label='AUC train')
    ax.plot(alpha, auc_cv, label='AUC CV')
    plt.title('AUC vs hyperparameter')
    plt.xlabel('alpha')
    plt.ylabel('AUC')
    ax.legend()
    plt.show()
    print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

[5] Assignment 7: SVM

1. **Apply SVM on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Procedure**

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

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

    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

4. **Feature importance**

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

5. **Feature engineering**

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

6. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-

fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

7. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   - 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

Applying SVM

[5.1] Linear SVM

In [198]:

```
def GSCV_svml1(x_train,y_train,x_cv,y_cv):
    from sklearn.model_selection import GridSearchCV

    from sklearn.linear_model import SGDClassifier
    from sklearn.metrics import roc_auc_score
    import plotly.offline as offline
    import plotly.graph_objs as go
    offline.init_notebook_mode()
    from mpl_toolkits.mplot3d import axes3d, Axes3D
    %matplotlib notebook
    import numpy as np
    from math import log
    %matplotlib inline
    import matplotlib.pyplot as plt

    a_list=[]
    p_list=[]
    clf = SGDClassifier(loss='hinge')
    #[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]
    para_grid = { 'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3,
10**4] ,
                  'penalty':['l1','l2']}
    model_grid = GridSearchCV(clf, para_grid, cv = 3,  scoring = 'roc_auc')
    model_grid.fit(x_train, y_train)
    best_para=model_grid.best_params_
    print(best_para)
    param_alpha=para_grid.get('alpha')
    param_alpha=np.log(param_alpha)
    param_penalty=para_grid.get('penalty')
    print(param_alpha)
    print(para_grid.get('alpha'))
    print(para_grid.get('penalty'))

    train_auc= model_grid.cv_results_['mean_train_score']
    cv_auc = model_grid.cv_results_['mean_test_score']

    para_list=model_grid.cv_results_['params']
    print(para_list)

    print(model_grid.cv_results_['mean_train_score'])
    print(model_grid.cv_results_['mean_test_score'])

    print("roc_auc_train",model_grid.score(x_train, y_train))
    print("roc_auc_cv",model_grid.score(x_cv, y_cv))
    b_a=best_para.get('alpha')
    b_p=best_para.get('penalty')
    print(b_a)
    print(b_p)

    for d in para_list:
        a_list.append(np.log(d.get('alpha')))
        if(d.get('penalty')=='l1'):
            p_list.append(1)
        else:
            p_list.append(2)
    print(a_list)
    print(p_list)
    print("Best HyperParameters using Grid SearchCV & roc_auc metric are: ",best_para)
    fig = plt.figure()
    #ax = plt.axes(projection='3d')
    ax = Axes3D(fig)
    my_yticks = ['l1','l2','l1','l2','l1','l2']
    plt.yticks(p_list, my_yticks)
    # Data for a three-dimensional line
    zline = train_auc
    xline = a_list
    yline = p_list
    ax.set_xlabel('alpha')
    ax.set_ylabel('penalty')
    ax.set_zlabel('auc')
    ax.plot3D(xline, yline, zline, label='Train AUC', color='Red')
    ax.legend()
    # rotate the axes and update
    zline = cv_auc
    xline = a_list
    yline = p_list
    ax.set_xlabel('alpha')
    ax.set_ylabel('penalty')
    ax.plot3D(xline, yline, zline, label='cv AUC', color='Blue')
    ax.legend()
    plt.show()
# Data for three-dimensional scattered points
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    #ax = Axes3D(fig)
    my_yticks = ['l1','l2','l1','l2','l1','l2']
    plt.yticks(p_list, my_yticks)
    zdata = train_auc
    xdata = a_list
    ydata = p_list
    ax.set_xlabel('alpha')
    ax.set_ylabel('penalty')
    ax.set_zlabel('auc')
    ax.scatter3D(xdata, ydata, zdata, label='Train AUC', c='r', marker='o')
    ax.legend()
    zdata = cv_auc
    ax.scatter3D(xdata, ydata, zdata, label='CV AUC',c='b', marker='^')
    ax.legend()
    plt.show()
    return(best_para)
```

In [210]:

```
def GSCV_svml2(x_train,y_train,x_cv,y_cv):
    from sklearn.model_selection import GridSearchCV
    from sklearn.calibration import CalibratedClassifierCV
    from sklearn.linear_model import SGDClassifier
    from sklearn.metrics import roc_auc_score
    import plotly.offline as offline
    import plotly.graph_objs as go
    offline.init_notebook_mode()
    from mpl_toolkits.mplot3d import axes3d, Axes3D
    %matplotlib notebook
    import numpy as np
    from math import log
    %matplotlib inline
    import matplotlib.pyplot as plt

    a_list=[]
    p_list=[]
    clf = SGDClassifier(loss='hinge')
    #[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]
    para_grid = { 'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3,
10**4] ,
                  'penalty':['l1','l2']}
    model_grid = GridSearchCV(clf, para_grid, cv = 3,  scoring = 'roc_auc')
    model_grid.fit(x_train, y_train)
    best_para=model_grid.best_params_
    print(best_para)
    b_a=best_para.get('alpha')#best Alpha
    b_p=best_para.get('penalty')# Best Penalty as per Grid SearchCV
    print(b_a)
    print(b_p)

    param_alpha=para_grid.get('alpha')
    param_alpha1=np.log(param_alpha)
    param_penalty=para_grid.get('penalty')
    print(param_alpha)
    print(type(param_alpha))
    print(param_alpha1)
    print(para_grid.get('alpha'))
    print(para_grid.get('penalty'))

    ###################################################
    #This part is added for plotting Train AUC and cv_Auc
    ###################################################

    #clf1 = SGDClassifier(loss='hinge',  penalty='l2')
    train_auc1 = []
    cv_auc1 = []
    for i in param_alpha:
        clf1 = SGDClassifier(loss='hinge',  alpha=i, penalty='l2')
        model_svm=CalibratedClassifierCV(clf1, cv=3)#as probability estimates are not a
vailable for Hinge Loss Calibration is used
        model_svm.fit(x_train, y_train)

        y_train_pred =  model_svm.predict_proba(x_train)[:,1]
        y_cv_pred =  model_svm.predict_proba(x_cv)[:,1]

        train_auc1.append(roc_auc_score(y_train,y_train_pred))
        cv_auc1.append(roc_auc_score(y_cv, y_cv_pred))
####################################################################
```

```
    #clf1 = SGDClassifier(loss='hinge',  penalty='l2')
    train_auc2 = []
    cv_auc2 = []
    for p in param_penalty:
        clf2 = SGDClassifier(loss='hinge',  alpha=b_a, penalty=p)
        model_svm2=CalibratedClassifierCV(clf2, cv=3)#as probability estimates are not
 available for Hinge Loss Calibration is used
        model_svm2.fit(x_train, y_train)

        y_train_pred =  model_svm2.predict_proba(x_train)[:,1]
        y_cv_pred =  model_svm2.predict_proba(x_cv)[:,1]

        train_auc2.append(roc_auc_score(y_train,y_train_pred))
        cv_auc2.append(roc_auc_score(y_cv, y_cv_pred))
#########################################################################

    plt.plot(param_alpha1, train_auc1, label='Train AUC')

    plt.plot(param_alpha1, cv_auc1, label='CV AUC')

    plt.legend()
    plt.xlabel(" hyperparameter alpha")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    print('-'*100)
    plt.plot(param_penalty, train_auc2, label='Train AUC')
    plt.plot(param_penalty, cv_auc2, label='CV AUC')
    plt.legend()
    plt.xlabel(" hyperparameter penalty")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    print('-'*100)
    return(best_para)
```

In [200]:

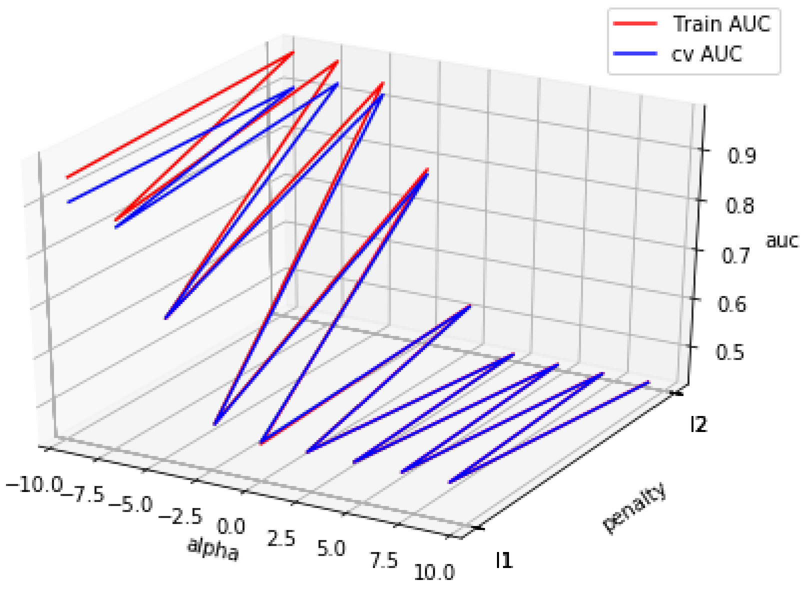
```
def test_svml(x_train,y_train,x_test,y_test,a,p):
    from sklearn.linear_model import SGDClassifier
    from sklearn.calibration import CalibratedClassifierCV
    from sklearn.metrics import roc_auc_score
    svm=SGDClassifier(alpha=a, penalty=p)
    model=CalibratedClassifierCV(svm, cv=3)
    model.fit(x_train,y_train)

    train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)
[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,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)))
    auc_test=auc(test_fpr, test_tpr)
    print('auc for Test data is::',auc_test)

    plt.legend()
    plt.xlabel("fpr")
    plt.ylabel("tpr")
    plt.title("AUC PLOTS")
    plt.show()

    print("="*100)


    yhat_train=model.predict(x_train)
    yhat_test=model.predict(x_test)

    con_mat_train = confusion_matrix(y_train, yhat_train)
    con_mat_test = confusion_matrix(y_test, yhat_test)

    plt.figure()
    class_label = ["negative", "positive"]
    df_con_mat_train = pd.DataFrame(con_mat_train, index = class_label, columns = class
_label)
    sns.heatmap(df_con_mat_train , annot = True, fmt = "d")
    plt.title("Confusiion Matrix for Train data")
    plt.xlabel("Actual Label")
    plt.ylabel("Predicted Label")
    print("Train confusion matrix")
    print(con_mat_train)

    plt.figure()
    class_label = ["negative", "positive"]
    df_con_mat_test = pd.DataFrame(con_mat_test, index = class_label, columns = class_l
abel)
    sns.heatmap(df_con_mat_test , annot = True, fmt = "d")
    plt.title("Confusiion Matrix for Test data")
    plt.xlabel("Actual Label")
    plt.ylabel("Predicted Label")
    plt.show()
    print("Test confusion matrix")
    print(con_mat_test)
    return(auc_test)
```

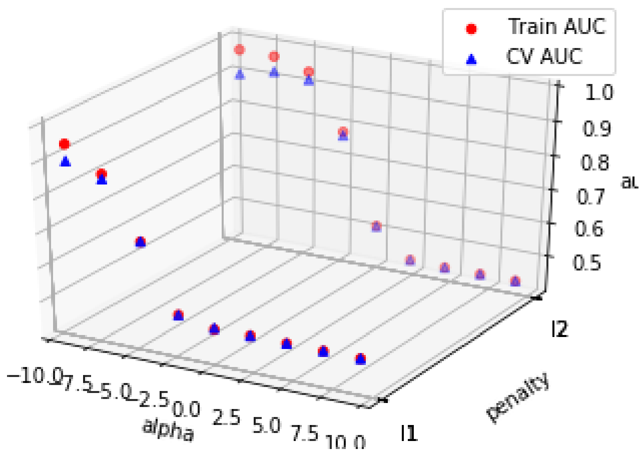
[5.1.1] Applying Linear SVM on BOW, SET 1

In [201]:

```
best_para_set1=GSCV_svml1(set1_train,y_train,set1_cv,y_cv)
```

```
{'alpha': 0.001, 'penalty': 'l2'}
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
[{'alpha': 0.0001, 'penalty': 'l1'}, {'alpha': 0.0001, 'penalty': 'l2'},
{'alpha': 0.001, 'penalty': 'l1'}, {'alpha': 0.001, 'penalty': 'l2'}, {'al
pha': 0.01, 'penalty': 'l1'}, {'alpha': 0.01, 'penalty': 'l2'}, {'alpha':
0.1, 'penalty': 'l1'}, {'alpha': 0.1, 'penalty': 'l2'}, {'alpha': 1, 'pena
lty': 'l1'}, {'alpha': 1, 'penalty': 'l2'}, {'alpha': 10, 'penalty': 'l
1'}, {'alpha': 10, 'penalty': 'l2'}, {'alpha': 100, 'penalty': 'l1'}, {'al
pha': 100, 'penalty': 'l2'}, {'alpha': 1000, 'penalty': 'l1'}, {'alpha': 1
000, 'penalty': 'l2'}, {'alpha': 10000, 'penalty': 'l1'}, {'alpha': 10000,
'penalty': 'l2'}]
[0.94927576 0.97458487 0.88378661 0.97172904 0.71051367 0.94381803
 0.5175924  0.78620689 0.49630163 0.52041568 0.5        0.4348991
 0.5        0.43185552 0.5        0.43185438 0.5        0.43185425]
[0.90181184 0.90380874 0.8700748  0.92746551 0.71130623 0.92062349
 0.51677975 0.77585932 0.50025127 0.51848856 0.5        0.43412473
 0.5        0.43112314 0.5        0.43112237 0.5        0.43112207]
roc_auc_train 0.9647259940427643
roc_auc_cv 0.9262123132915407
0.001
l2
[-9.210340371976182, -9.210340371976182, -6.907755278982137, -6.9077552789
82137, -4.605170185988091, -4.605170185988091, -2.3025850929940455, -2.302
5850929940455, 0.0, 0.0, 2.302585092994046, 2.302585092994046, 4.605170185
988092, 4.605170185988092, 6.907755278982137, 6.907755278982137, 9.2103403
71976184, 9.210340371976184]
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'alpha':
0.001, 'penalty': 'l2'}
```



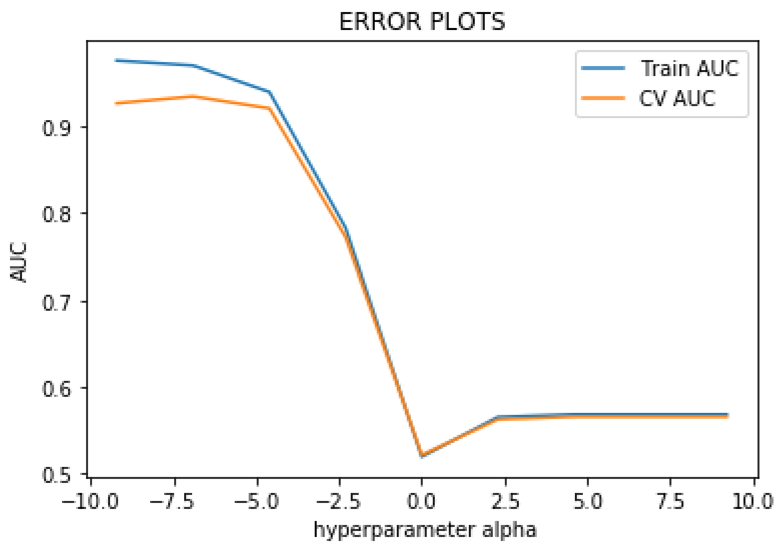


Train AUC
CV AUC
1.0
0.9
0.8
0.7
0.6
0.5
au
l2
penalty
l1
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
alpha

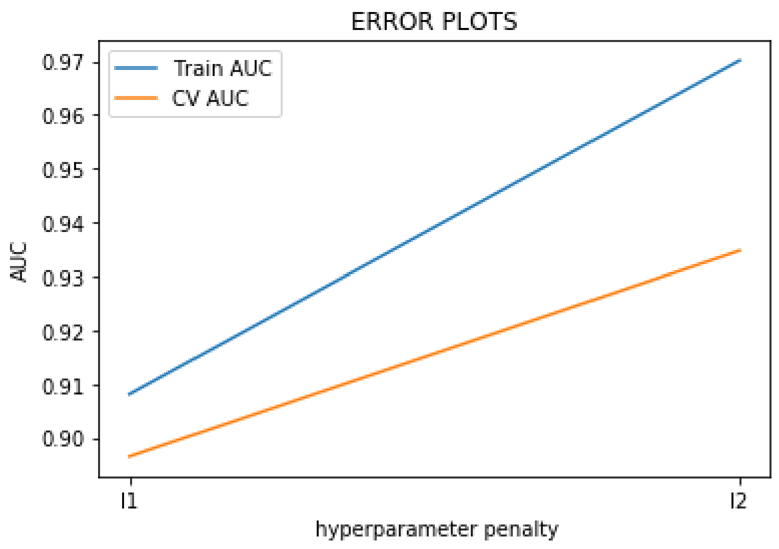
In [202]:

```
best_para_set1=GSCV_svml2(set1_train,y_train,set1_cv,y_cv)
```

```
{'alpha': 0.001, 'penalty': 'l2'}
0.001
l2
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
<class 'list'>
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
```



```
----------------------------------------------------------------------------
--------------------------
```



```
----------------------------------------------------------------------------
--------------------------
```

In [ ]:

```
For plotting train and cv auc plot I used 3 different ways.
1) Use of 3 d plot of Hyperparameter1 (alpha) Hyperparameter2(penalty)and AUC
2) Use of 3 d scatter plot of Hyperparameter1 (alpha) Hyperparameter2(penalty)and AUC
3) After finding best parameters using GridSearchCV, one parameter kept constant to bes
t obtianed value and auc Vs other
hyperparameter is plotted (this is done after finding best values to observe overfittin
g/underfitting only.....Please comment)
```

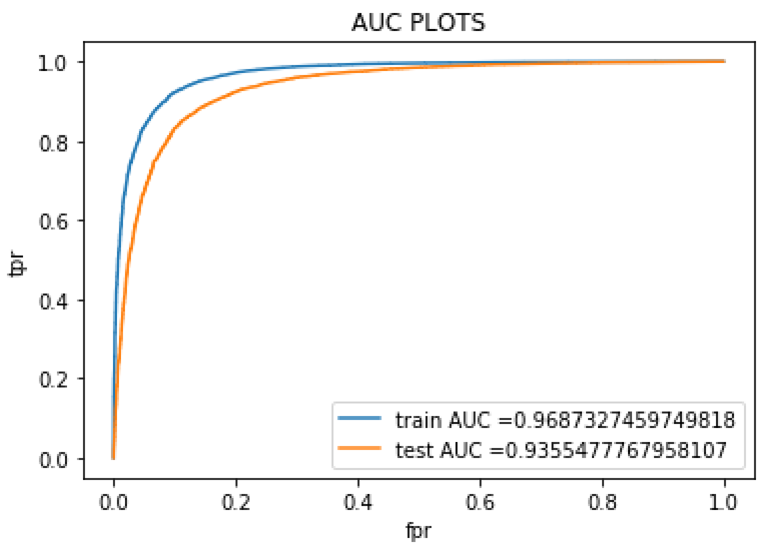
In [203]:

```
opt_a=best_para_set1.get('alpha')
opt_p=best_para_set1.get('penalty')
```

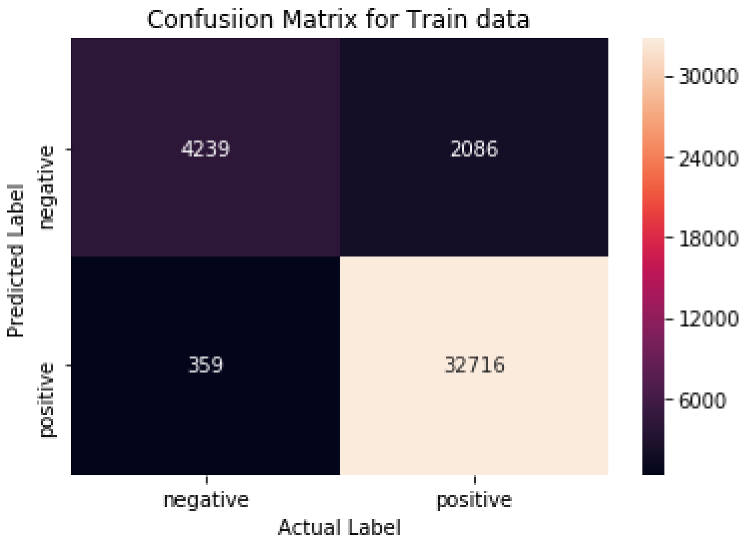
In [204]:

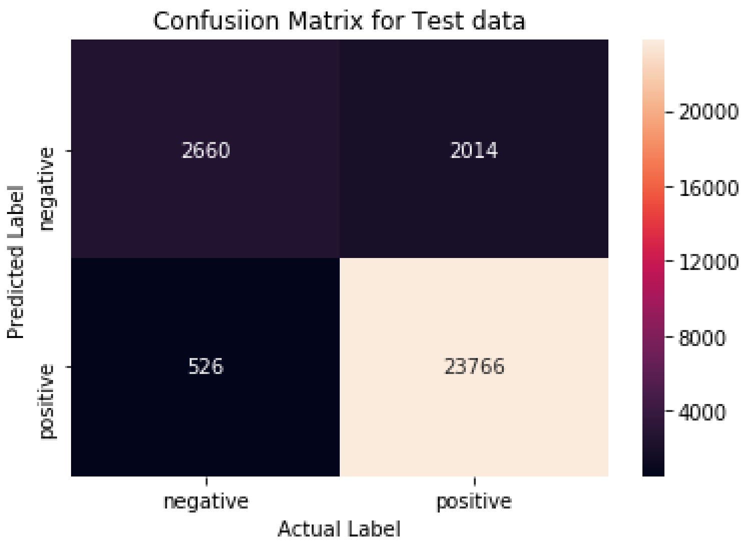
```
auc_set1=test_svml(set1_train,y_train,set1_test,y_test,opt_a,opt_p)
```

```
auc for Test data is:: 0.9355477767958107
```



```
================================================================================
============================
Train confusion matrix
[[ 4239  2086]
 [  359 32716]]
```





```
Test confusion matrix
[[ 2660  2014]
 [  526 23766]]
```

In [205]:

```
#top 10 positive features
from sklearn.linear_model import SGDClassifier
all_features = count_vect.get_feature_names()
model=SGDClassifier(alpha=opt_a, penalty=opt_p)
model.fit(set1_train,y_train)
f_imp=model.coef_
pos_ind=np.argsort(f_imp)[:,::-1]

neg_ind=np.argsort(f_imp)

print('Top 10 positive features :')
for i in list(pos_ind[0][0:10]):
    print(all_features[i])
print('-----------------------------------------------')
print('Top 10 Negative features :')
for i in list(neg_ind[0][0:10]):
    print(all_features[i])
```

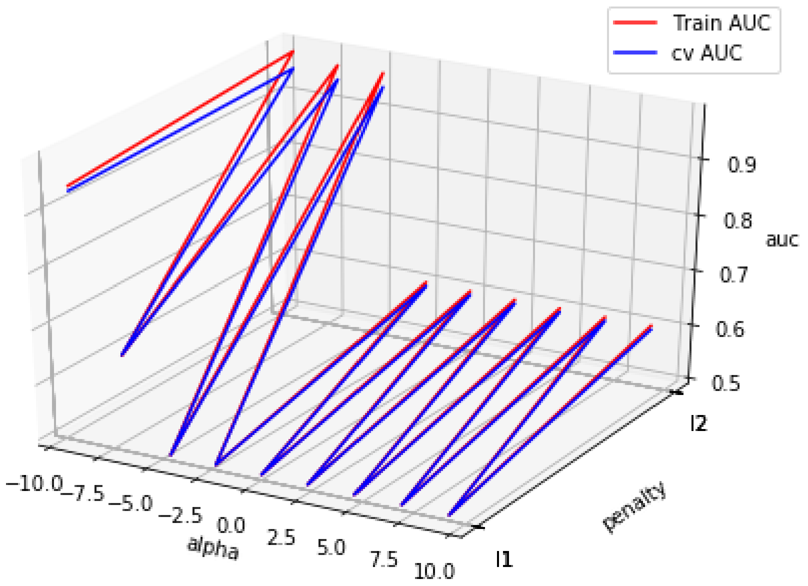
```
Top 10 positive features :
delicious
perfect
wonderful
amazing
smooth
great
loves
yummy
best
nice
-----------------------------------------------
Top 10 Negative features :
worst
horrible
disappointing
terrible
awful
disappointed
threw
disappointment
return
stale
```

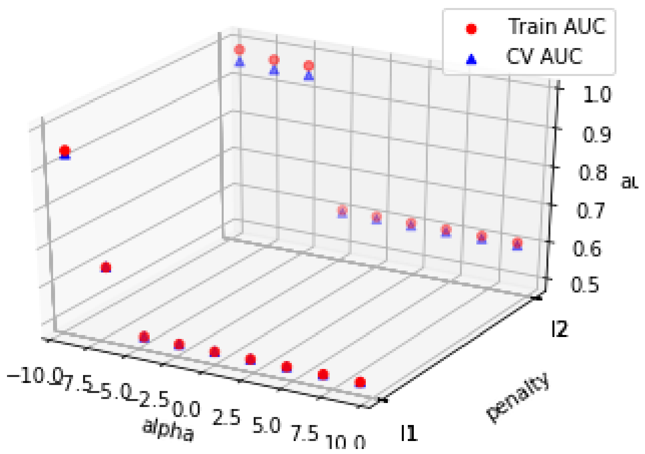
[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [211]:

```
# Please write all the code with proper documentation
best_para_set2=GSCV_svml1(set2_train,y_train,set2_cv,y_cv)
```

```
{'alpha': 0.0001, 'penalty': 'l2'}
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
[{'alpha': 0.0001, 'penalty': 'l1'}, {'alpha': 0.0001, 'penalty': 'l2'},
{'alpha': 0.001, 'penalty': 'l1'}, {'alpha': 0.001, 'penalty': 'l2'}, {'al
pha': 0.01, 'penalty': 'l1'}, {'alpha': 0.01, 'penalty': 'l2'}, {'alpha':
0.1, 'penalty': 'l1'}, {'alpha': 0.1, 'penalty': 'l2'}, {'alpha': 1, 'pena
lty': 'l1'}, {'alpha': 1, 'penalty': 'l2'}, {'alpha': 10, 'penalty': 'l
1'}, {'alpha': 10, 'penalty': 'l2'}, {'alpha': 100, 'penalty': 'l1'}, {'al
pha': 100, 'penalty': 'l2'}, {'alpha': 1000, 'penalty': 'l1'}, {'alpha': 1
000, 'penalty': 'l2'}, {'alpha': 10000, 'penalty': 'l1'}, {'alpha': 10000,
'penalty': 'l2'}]
[0.94550136 0.98413835 0.66452058 0.97270383 0.5        0.9726281
 0.5        0.60512374 0.5        0.60511832 0.5        0.60511832
 0.5        0.60511832 0.5        0.60511832 0.5        0.60511832]
[0.93677599 0.95370843 0.66554601 0.947547   0.5        0.94791472
 0.5        0.59798412 0.5        0.59797877 0.5        0.59797877
 0.5        0.59797877 0.5        0.59797877 0.5        0.59797877]
roc_auc_train 0.9784459083589517
roc_auc_cv 0.9549686749134915
0.0001
l2
[-9.210340371976182, -9.210340371976182, -6.907755278982137, -6.9077552789
82137, -4.605170185988091, -4.605170185988091, -2.3025850929940455, -2.302
5850929940455, 0.0, 0.0, 2.302585092994046, 2.302585092994046, 4.605170185
988092, 4.605170185988092, 6.907755278982137, 6.907755278982137, 9.2103403
71976184, 9.210340371976184]
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'alpha':
0.0001, 'penalty': 'l2'}
```

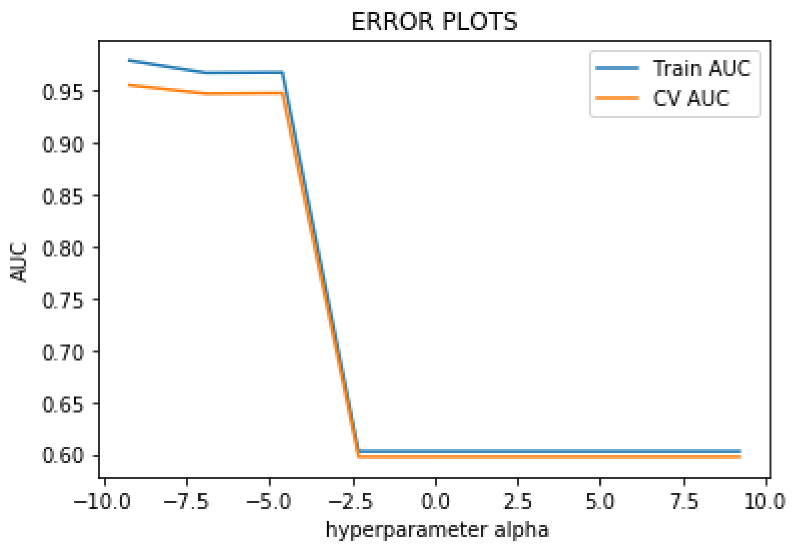



Train AUC
CV AUC
alpha
penalty
au
l1
l2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0.5
0.6
0.7
0.8
0.9
1.0

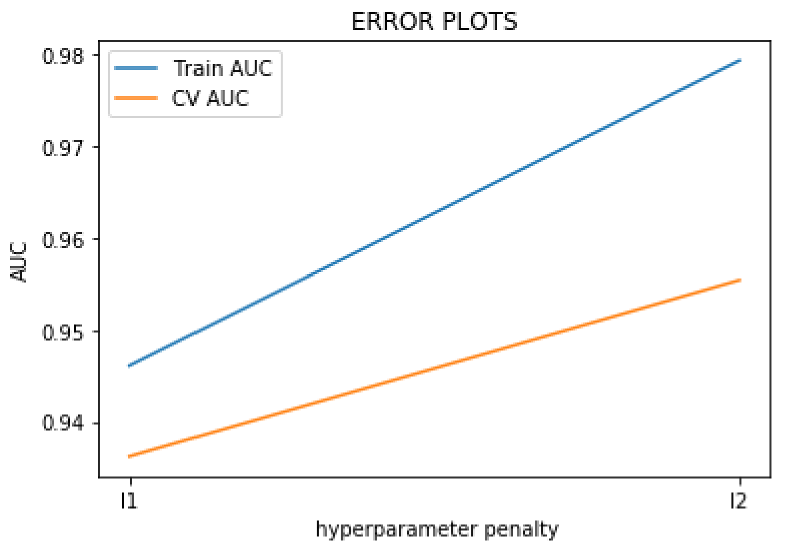
In [212]:

```
best_para_set2=GSCV_svml2(set2_train,y_train,set2_cv,y_cv)
```

```
{'alpha': 0.0001, 'penalty': 'l2'}
0.0001
l2
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
<class 'list'>
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
```



```
--------------------------------------------------------------------------
--------------------------
```

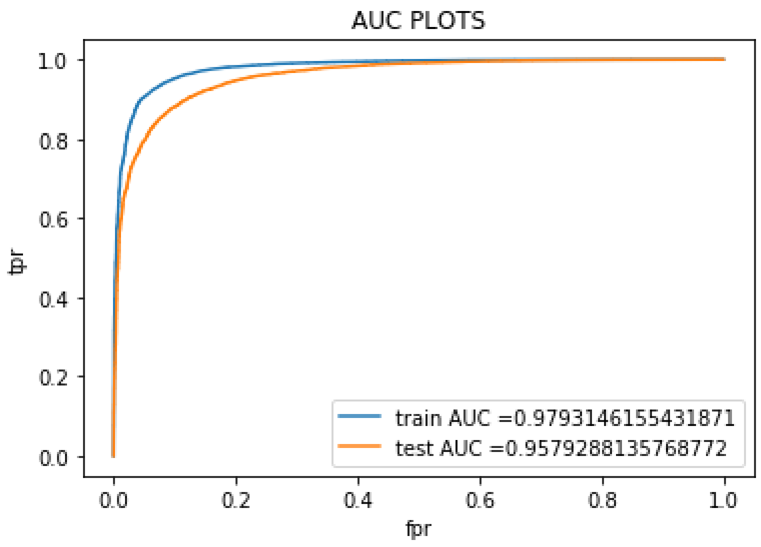


```
--------------------------------------------------------------------------
--------------------------
```

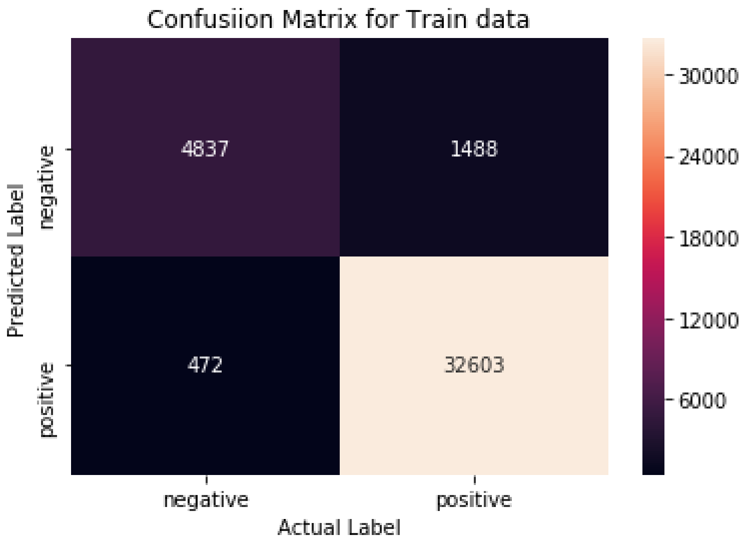
In [213]:

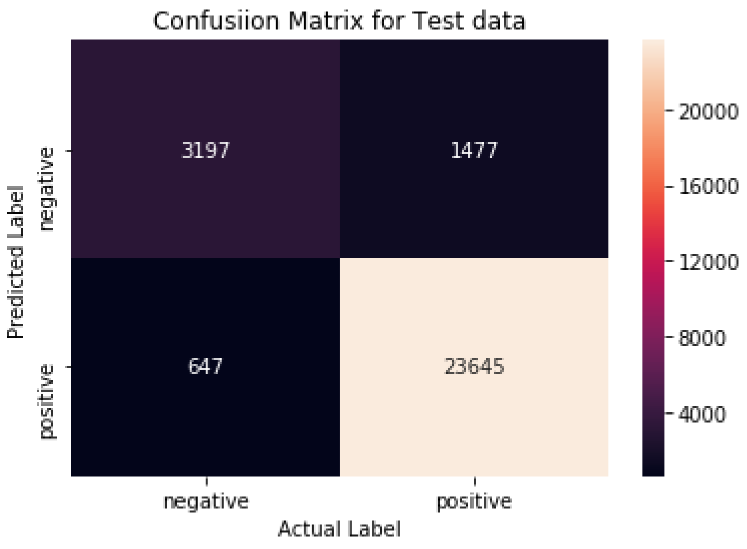
```
opt_a2=best_para_set2.get('alpha')
opt_p2=best_para_set2.get('penalty')
auc_set2=test_svml(set2_train,y_train,set2_test,y_test,opt_a2,opt_p2)
```

```
auc for Test data is:: 0.9579288135768772
```



```
==========================================================================
==========================
Train confusion matrix
[[ 4837  1488]
 [  472 32603]]
```





```
Test confusion matrix
[[ 3197  1477]
 [  647 23645]]
```

```
In [214]:
```

```python
#top 10 positive features
from sklearn.linear_model import SGDClassifier
all_features = tf_idf_vect.get_feature_names()
model=SGDClassifier(alpha=opt_a2, penalty=opt_p2)
model.fit(set2_train,y_train)
f_imp=model.coef_
pos_ind=np.argsort(f_imp)[:,::-1]

neg_ind=np.argsort(f_imp)

print('Top 10 positive features :')
for i in list(pos_ind[0][0:10]):
    print(all_features[i])
print('-----------------------------------------------')
print('Top 10 Negative features :')
for i in list(neg_ind[0][0:10]):
    print(all_features[i])
```

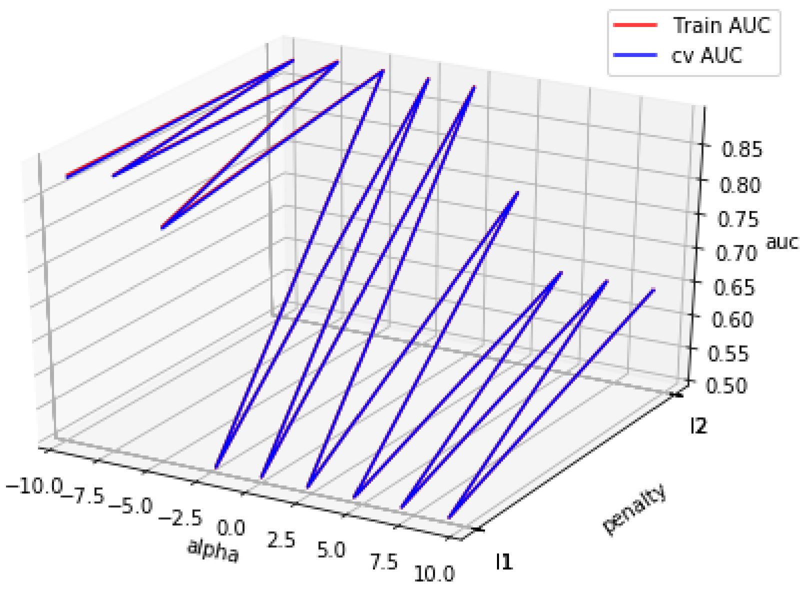
```
Top 10 positive features :
great
best
good
delicious
not disappointed
loves
perfect
nice
wonderful
amazing
-----------------------------------------------
Top 10 Negative features :
disappointed
worst
terrible
horrible
not
not worth
return
awful
not buy
disappointing
```

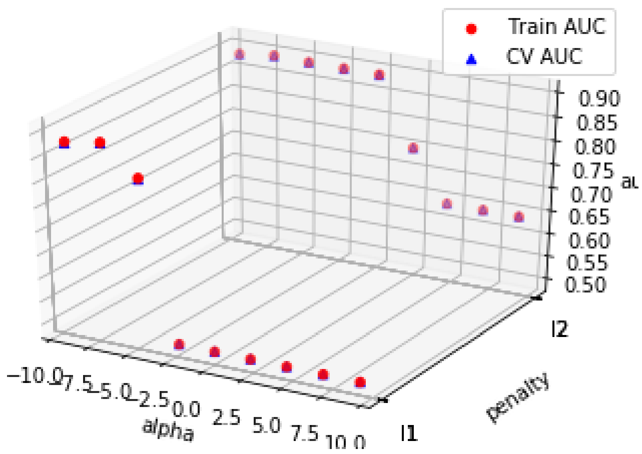
[5.1.3] Applying Linear SVM on AVG W2V, SET 3

In [215]:

```
best_para_set3=GSCV_svml1(set3_train,y_train,set3_cv,y_cv)
```

```
{'alpha': 0.001, 'penalty': 'l2'}
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
[{'alpha': 0.0001, 'penalty': 'l1'}, {'alpha': 0.0001, 'penalty': 'l2'},
{'alpha': 0.001, 'penalty': 'l1'}, {'alpha': 0.001, 'penalty': 'l2'}, {'al
pha': 0.01, 'penalty': 'l1'}, {'alpha': 0.01, 'penalty': 'l2'}, {'alpha':
0.1, 'penalty': 'l1'}, {'alpha': 0.1, 'penalty': 'l2'}, {'alpha': 1, 'pena
lty': 'l1'}, {'alpha': 1, 'penalty': 'l2'}, {'alpha': 10, 'penalty': 'l
1'}, {'alpha': 10, 'penalty': 'l2'}, {'alpha': 100, 'penalty': 'l1'}, {'al
pha': 100, 'penalty': 'l2'}, {'alpha': 1000, 'penalty': 'l1'}, {'alpha': 1
000, 'penalty': 'l2'}, {'alpha': 10000, 'penalty': 'l1'}, {'alpha': 10000,
'penalty': 'l2'}]
[0.88135631 0.886024   0.89308867 0.8948348  0.83424614 0.89281302
 0.5        0.89208093 0.5        0.89167259 0.5        0.75010413
 0.5        0.64296295 0.5        0.64296337 0.5        0.64296316]
[0.87831285 0.88508572 0.89195373 0.89333614 0.83179296 0.89202199
 0.5        0.89122996 0.5        0.89059966 0.5        0.74957782
 0.5        0.642805   0.5        0.64280611 0.5        0.64280589]
roc_auc_train 0.8928098064346511
roc_auc_cv 0.8924004536302528
0.001
l2
[-9.210340371976182, -9.210340371976182, -6.907755278982137, -6.9077552789
82137, -4.605170185988091, -4.605170185988091, -2.3025850929940455, -2.302
5850929940455, 0.0, 0.0, 2.302585092994046, 2.302585092994046, 4.605170185
988092, 4.605170185988092, 6.907755278982137, 6.907755278982137, 9.2103403
71976184, 9.210340371976184]
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'alpha':
0.001, 'penalty': 'l2'}
```

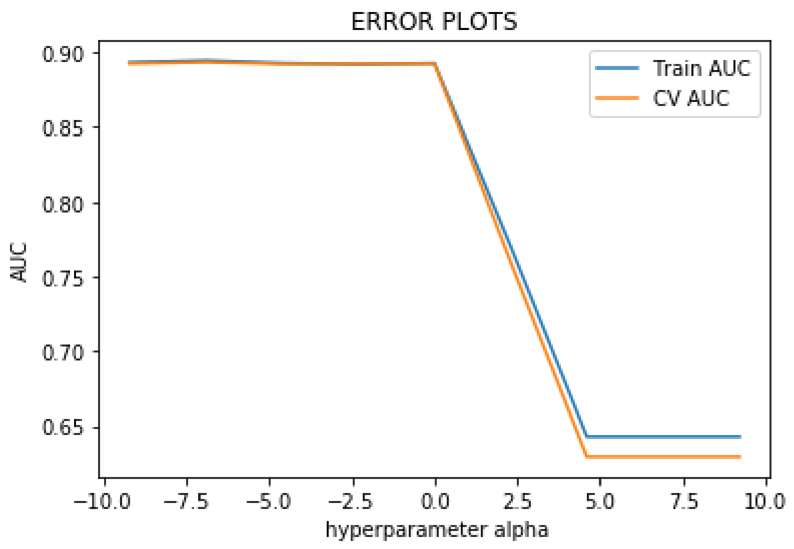



Train AUC
CV AUC
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
au
l2
l1
penalty
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
alpha

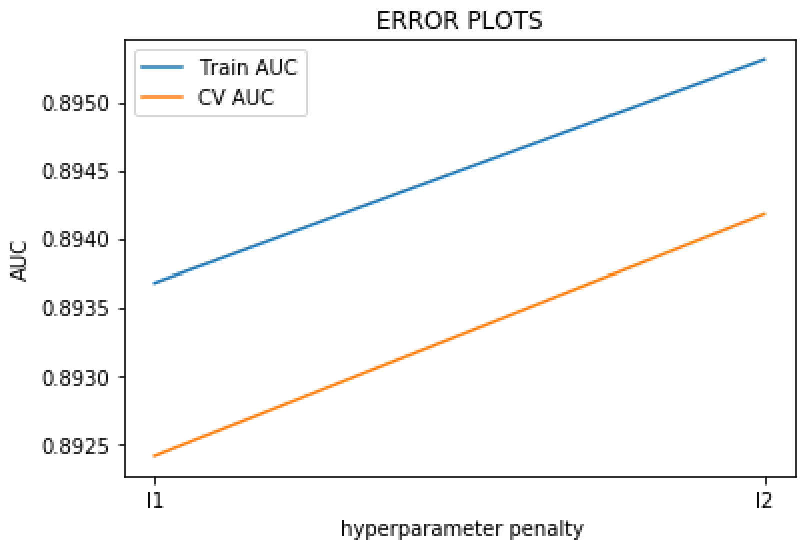
In [216]:

```
# Please write all the code with proper documentation
best_para_set3=GSCV_svml2(set3_train,y_train,set3_cv,y_cv)
```

```
{'alpha': 0.001, 'penalty': 'l1'}
0.001
l1
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
<class 'list'>
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
```



```
--------------------------------------------------------------------------
--------------------------
```

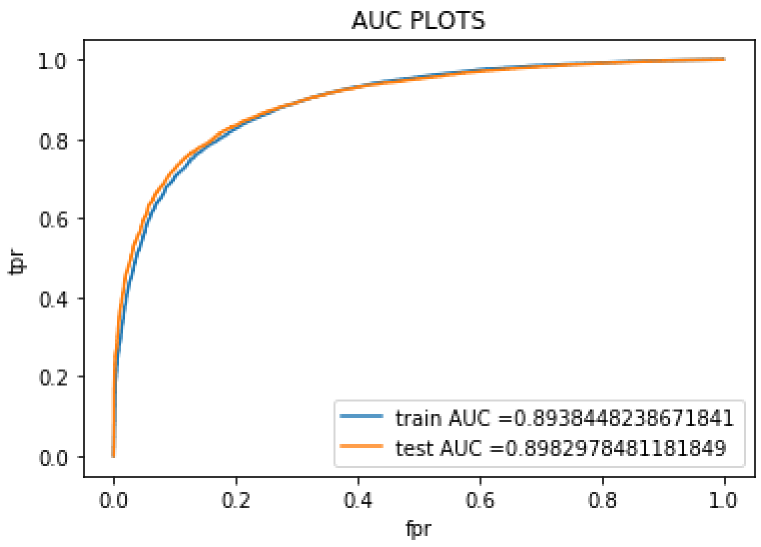


```
--------------------------------------------------------------------------
--------------------------
```

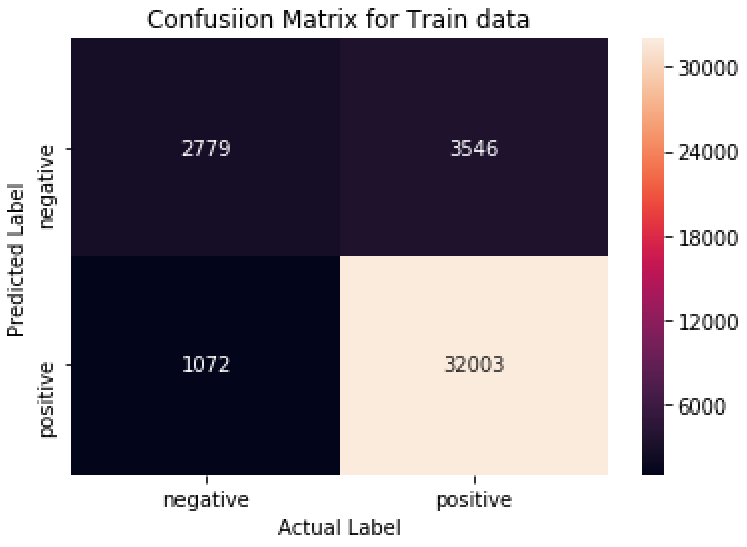
In [217]:

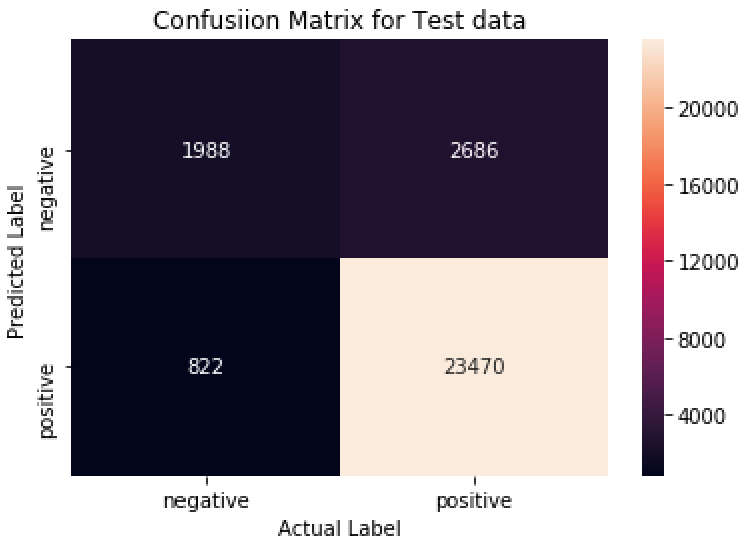
```
opt_a3=best_para_set3.get('alpha')
opt_p3=best_para_set3.get('penalty')
auc_set3=test_svml(set3_train,y_train,set3_test,y_test,opt_a3,opt_p3)
```

```
auc for Test data is:: 0.8982978481181849
```



```
==========================================================================
==========================
Train confusion matrix
[[ 2779  3546]
 [ 1072 32003]]
```





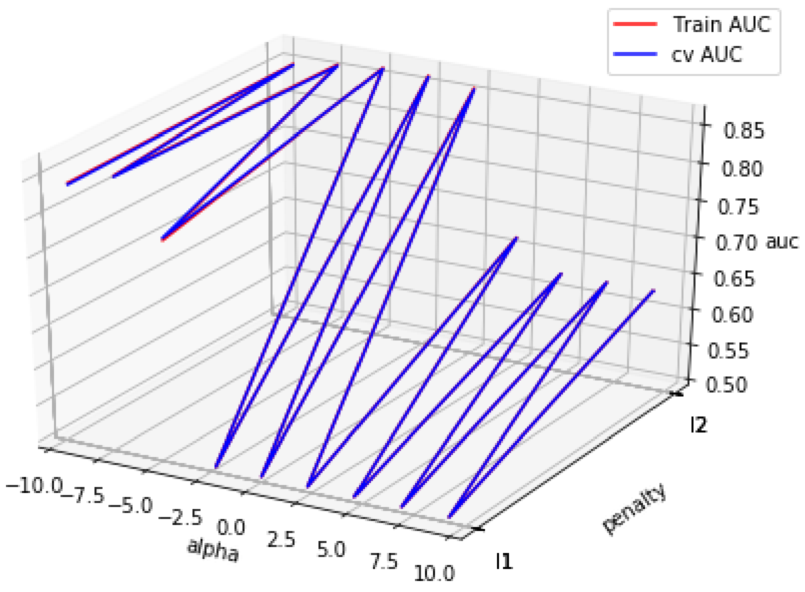
```
Test confusion matrix
[[ 1988  2686]
 [  822 23470]]
```

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

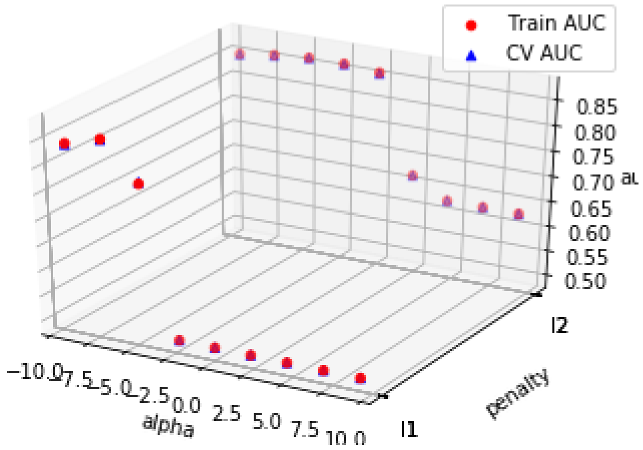
In [218]:

```
# Please write all the code with proper documentation
best_para_set4=GSCV_svml1(set4_train,y_train,set4_cv,y_cv)
```

```
{'alpha': 0.01, 'penalty': 'l2'}
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
[{'alpha': 0.0001, 'penalty': 'l1'}, {'alpha': 0.0001, 'penalty': 'l2'},
{'alpha': 0.001, 'penalty': 'l1'}, {'alpha': 0.001, 'penalty': 'l2'}, {'al
pha': 0.01, 'penalty': 'l1'}, {'alpha': 0.01, 'penalty': 'l2'}, {'alpha':
0.1, 'penalty': 'l1'}, {'alpha': 0.1, 'penalty': 'l2'}, {'alpha': 1, 'pena
lty': 'l1'}, {'alpha': 1, 'penalty': 'l2'}, {'alpha': 10, 'penalty': 'l
1'}, {'alpha': 10, 'penalty': 'l2'}, {'alpha': 100, 'penalty': 'l1'}, {'al
pha': 100, 'penalty': 'l2'}, {'alpha': 1000, 'penalty': 'l1'}, {'alpha': 1
000, 'penalty': 'l2'}, {'alpha': 10000, 'penalty': 'l1'}, {'alpha': 10000,
'penalty': 'l2'}]
[0.84352861 0.85146519 0.86373285 0.86082076 0.79214864 0.8666844
 0.5        0.86644381 0.5        0.86045811 0.5        0.669291
 0.5        0.63053066 0.5        0.63053136 0.5        0.63053136]
[0.841571   0.85024667 0.86254496 0.8593116  0.79517204 0.86563924
 0.5        0.86508287 0.5        0.85858489 0.5        0.66846977
 0.5        0.6303501  0.5        0.63035097 0.5        0.63035097]
roc_auc_train 0.8668174391056379
roc_auc_cv 0.867939226717311
0.01
l2
[-9.210340371976182, -9.210340371976182, -6.907755278982137, -6.9077552789
82137, -4.605170185988091, -4.605170185988091, -2.3025850929940455, -2.302
5850929940455, 0.0, 0.0, 2.302585092994046, 2.302585092994046, 4.605170185
988092, 4.605170185988092, 6.907755278982137, 6.907755278982137, 9.2103403
71976184, 9.210340371976184]
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'alpha':
0.01, 'penalty': 'l2'}
```



Train AUC
cv AUC
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
auc
l2
penalty
l1
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
alpha

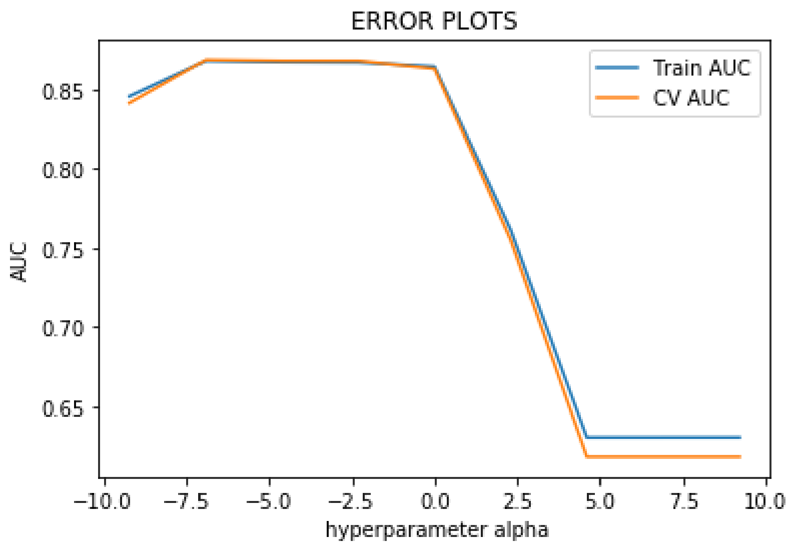


Train AUC
CV AUC
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
auc
l2
penalty
l1
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
alpha

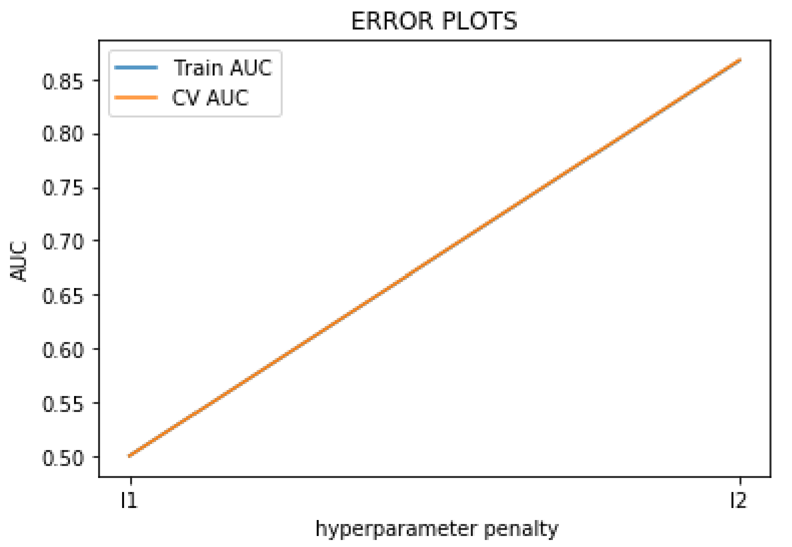
In [220]:

```
best_para_set4=GSCV_svml2(set4_train,y_train,set4_cv,y_cv)
```

```
{'alpha': 0.1, 'penalty': 'l2'}
0.1
l2
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
<class 'list'>
[-9.21034037 -6.90775528 -4.60517019 -2.30258509  0.          2.30258509
  4.60517019  6.90775528  9.21034037]
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
['l1', 'l2']
```



```
--------------------------------------------------------------------------
--------------------------
```

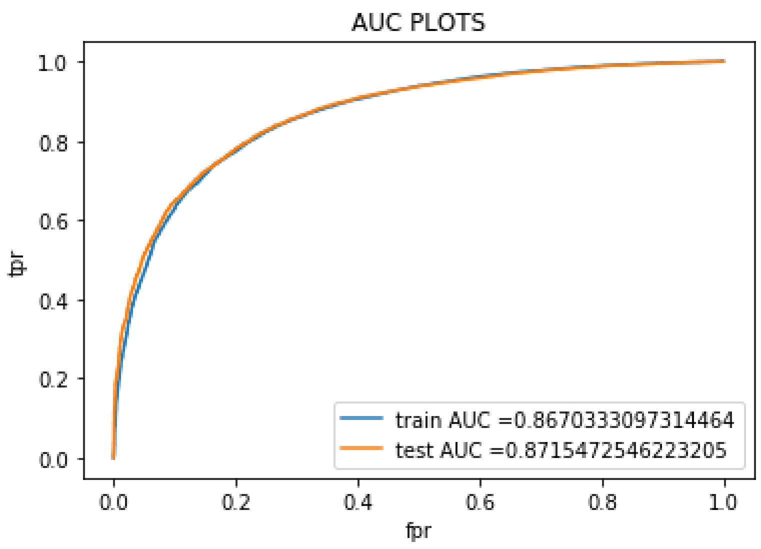


```
--------------------------------------------------------------------------
--------------------------
```

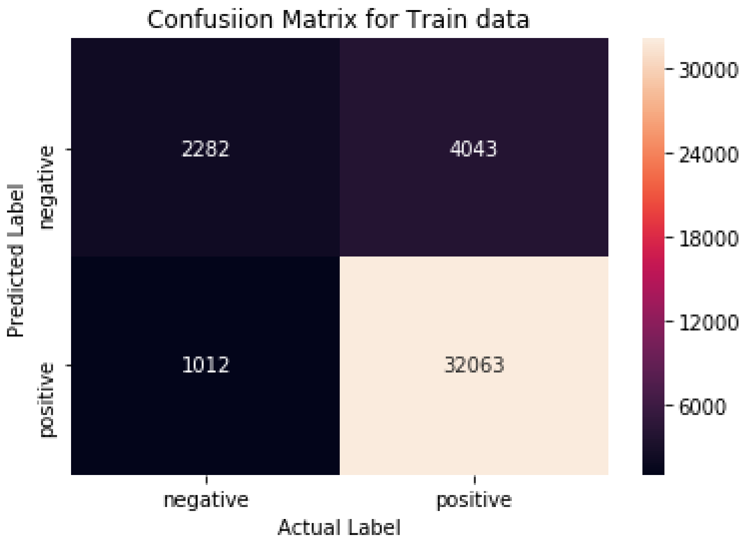
In [221]:

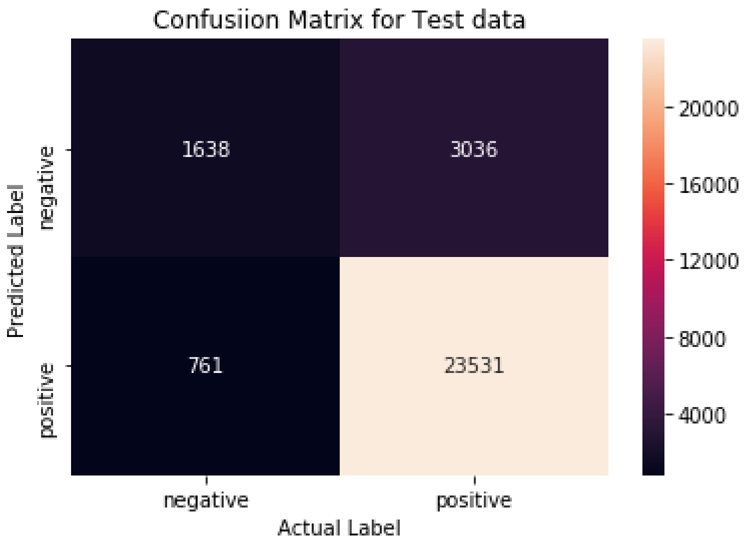
```
opt_a4=best_para_set4.get('alpha')
opt_p4=best_para_set4.get('penalty')
auc_set4=test_svml(set4_train,y_train,set4_test,y_test,opt_a4,opt_p4)
```

```
auc for Test data is:: 0.8715472546223205
```



```
============================================================================
==========================
Train confusion matrix
[[ 2282  4043]
 [ 1012 32063]]
```





```
Test confusion matrix
[[ 1638  3036]
 [  761 23531]]
```

[5.2] RBF SVM

In [266]:

```
import collections, numpy
X1=X[:40000]
Y1=Y[:40000]

X_1, X_test, y_1, y_test = train_test_split(X1, Y1, test_size=0.3, random_state=5)
X_train, X_cv, y_train, y_cv = train_test_split(X_1, y_1, test_size=0.3)
print(len(X_train), y_train.shape)
print(len(X_cv), y_cv.shape)
print(len(X_test), y_test.shape)

print("="*100)
print(collections.Counter(y_train))
print(collections.Counter(y_cv))
print(collections.Counter(y_test))
```

```
19600 (19600,)
8400 (8400,)
12000 (12000,)
==========================================================================
==========================
Counter({1: 16504, 0: 3096})
Counter({1: 7107, 0: 1293})
Counter({1: 10134, 0: 1866})
```

In [267]:

```
count_vect1 = CountVectorizer(min_df=10, max_features=500)
count_vect1.fit(X_train)
print("some feature names ", count_vect1.get_feature_names()[:10])
print('='*50)

set1a_train = count_vect1.transform(X_train)
set1a_cv = count_vect1.transform(X_cv)
set1a_test = count_vect1.transform(X_test)

print("the type of count vectorizer ",type(set1a_train))
print("="*100)
print("After vectorizations")
print(set1a_train.shape, y_train.shape)
print(set1a_cv.shape, y_cv.shape)
print(set1a_test.shape, y_test.shape)
print("="*100)
print("the number of unique words ", set1a_train.get_shape()[1])
```

```
some feature names  ['able', 'absolutely', 'actually', 'add', 'added', 'ad
ding', 'ago', 'almost', 'already', 'also']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
==========================================================================
==========================
After vectorizations
(19600, 500) (19600,)
(8400, 500) (8400,)
(12000, 500) (12000,)
==========================================================================
==========================
the number of unique words  500
```

In [268]:

```
tf_idf_vect1 = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
tf_idf_vect1.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect1.get_feature_names
()[0:10])
print('='*50)

set2a_train = tf_idf_vect1.transform(X_train)
set2a_cv = tf_idf_vect1.transform(X_cv)
set2a_test = tf_idf_vect1.transform(X_test)
print("the type of count vectorizer ",type(set2a_train))
print("="*100)
print("After vectorizations")
print(set2a_train.shape, y_train.shape)
print(set2a_cv.shape, y_cv.shape)
print(set2a_test.shape, y_test.shape)
print("="*100)
print("the number of unique words ", set2a_train.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['able', 'absolutely', 'a
ctually', 'add', 'added', 'adding', 'ago', 'almost', 'also', 'although']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
==========================================================================
==========================
After vectorizations
(19600, 500) (19600,)
(8400, 500) (8400,)
(12000, 500) (12000,)
==========================================================================
==========================
the number of unique words  500
```

In [269]:

```
# 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 [270]:

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

# 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'))

# this line of code trains your w2v model on the give list of sentances
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])
```

```
[('good', 0.8241574764251709), ('excellent', 0.7833988070487976), ('wonder
ful', 0.7799737453460693), ('awesome', 0.7756421566009521), ('perfect', 0.
7486790418624878), ('fantastic', 0.7454037666320801), ('amazing', 0.744664
0729904175), ('delicious', 0.6907408237457275), ('decent', 0.6506381034851
074), ('nice', 0.620872974395752)]
==================================================
[('bahlsen', 0.8489474058151245), ('ive', 0.8362393379211426), ('world',
0.8226278424263), ('hooked', 0.8197627067565918), ('tastiest', 0.815722703
9337158), ('lindt', 0.8119768500328064), ('imagined', 0.7970116138458252),
('eaten', 0.7949073314666748), ('cousin', 0.7933571338653564), ('disgustin
g', 0.7912352681159973)]
number of words that occured minimum 5 times  8557
sample words  ['love', 'simply', 'organic', 'great', 'item', 'sprinkle',
'meal', 'salad', 'highly', 'recommend', 'bottles', 'really', 'packed', 'n
o', 'room', 'settling', 'get', 'worth', 'needed', 'iron', 'diet', 'tryin
g', 'watch', 'weight', 'chips', 'wonderful', 'healthy', 'elements', 'nee
d', 'satisfying', 'crunch', 'crave', 'unbelievable', 'taste', 'none', 'no
t', 'like', 'doubled', 'original', 'recipe', 'seemed', 'toy', 'dogs', 'sec
onds', 'end', 'treats', 'come', 'rubber', 'parts', 'smell']
```

In [271]:

```
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  8557
sample words  ['love', 'simply', 'organic', 'great', 'item', 'sprinkle',
'meal', 'salad', 'highly', 'recommend', 'bottles', 'really', 'packed', 'n
o', 'room', 'settling', 'get', 'worth', 'needed', 'iron', 'diet', 'tryin
g', 'watch', 'weight', 'chips', 'wonderful', 'healthy', 'elements', 'nee
d', 'satisfying', 'crunch', 'crave', 'unbelievable', 'taste', 'none', 'no
t', 'like', 'doubled', 'original', 'recipe', 'seemed', 'toy', 'dogs', 'sec
onds', 'end', 'treats', 'come', 'rubber', 'parts', 'smell']
```

In [272]:

```python
# average Word2Vec
# compute average word2vec for each review.
from tqdm import tqdm
import numpy as np
set3_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
    set3_train.append(sent_vec)
set3_train = np.array(set3_train)
print(set3_train.shape)
print(set3_train[0])
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 19600/19600 [00:50<00:00, 387.02it/s]

(19600, 50)
[-0.24357575 -0.51415604  0.12926536  0.49926308  0.03929917 -0.0584722
 -0.04631096  0.1374217   0.05246143 -1.30401233 -0.80541825 -0.17127845
  0.28409517  0.03771534  0.16053836  0.23512588  0.29260886 -0.41503319
 -0.31850139 -0.35681772 -0.49004349 -0.63579348 -0.94008153  0.25019078
  0.28385286 -0.02820736 -0.04700775 -0.24322881 -0.65527597 -0.2045527
  0.08599633 -0.11064697 -0.56631361 -0.3287897  -0.35747542 -0.30056179
  0.07986318  0.07333909  0.10519682  0.24946708  0.1050817  -0.21071752
 -0.47980046  0.41494353  0.10008555  0.15527705 -0.14277568  0.49826323
 -0.07254878  0.76739164]
```

```
In [273]:
```

```
# average Word2Vec
# compute average word2vec for each review.
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
set3_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
 change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    set3_cv.append(sent_vec)
set3_cv = np.array(set3_cv)
print(set3_cv.shape)
print(set3_cv[0])
```

```
100%|█████████████████████████████████████████████████████████████████████
████████| 8400/8400 [00:21<00:00, 394.68it/s]

(8400, 50)
[-0.21678633 -0.42360365 -0.28023138  0.13289692  0.06753448 -0.38157268
 -0.13148985  0.11437895  0.25465758 -0.85255222 -0.94833812 -0.23838909
 -0.53324523  0.1183474   0.48999075 -0.4791902   0.13390045 -0.3168153
  0.10031197 -0.46289476 -0.16711561  0.05236046 -0.58942427  0.1839379
 -0.03849137 -0.03827133  0.07272224  0.27057874 -0.57920504 -0.11286451
 -0.4788958  -0.04077119 -0.20781047 -0.11924318 -0.44816623 -0.10837571
 -0.15519613  0.53485106 -0.03838059  0.35062205 -0.37174339 -0.45864264
 -0.47926637  0.26339506  0.51708553  0.6158336  -0.21565718  0.23736093
  0.2027038   0.36690616]
```

In [274]:

```
# average Word2Vec
# compute average word2vec for each review.
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
set3_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
    set3_test.append(sent_vec)
set3_test = np.array(set3_test)
print(set3_test.shape)
print(set3_test[0])
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 12000/12000 [00:31<00:00, 379.16it/s]

(12000, 50)
[-0.01512706 -0.45123298  0.03103947  0.4272552   0.18865477  0.05049198
 -0.08099436 -0.04541388 -0.03498139 -0.90443352 -0.90354189 -0.13739192
 -0.4558059   0.1090901   0.52738514 -0.24170219  0.29599752 -0.14445761
  0.20798977 -0.52010077 -0.15414883  0.06406627 -0.27021116  0.00292665
 -0.23031705 -0.00711621 -0.26736135 -0.08986736 -0.53743726 -0.40715913
 -0.01274318  0.14795034 -0.51770213 -0.13680157 -0.82037728 -0.03207461
 -0.38369648  0.30564065 -0.03401878 -0.15579279 -0.22962034 -0.48512356
 -0.6547738   0.40584713  0.15457135  0.46764877 -0.08785652  0.17362586
  0.1467768   0.59825796]
```

In [275]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
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 [276]:

```
# 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

set4_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
    set4_train.append(sent_vec)
    row += 1
print(len(set4_train))
print(len(set4_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
███████| 19600/19600 [07:02<00:00, 46.40it/s]

19600
50
```

In [277]:

```
set4_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    set4_cv.append(sent_vec)
    row += 1
print(len(set4_cv))
print(len(set4_cv[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█████████| 8400/8400 [02:59<00:00, 46.88it/s]

8400
50
```

In [278]:

```
set4_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
    set4_test.append(sent_vec)
    row += 1
print(len(set4_test))
print(len(set4_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
███████| 12000/12000 [04:13<00:00, 47.35it/s]

12000
50
```

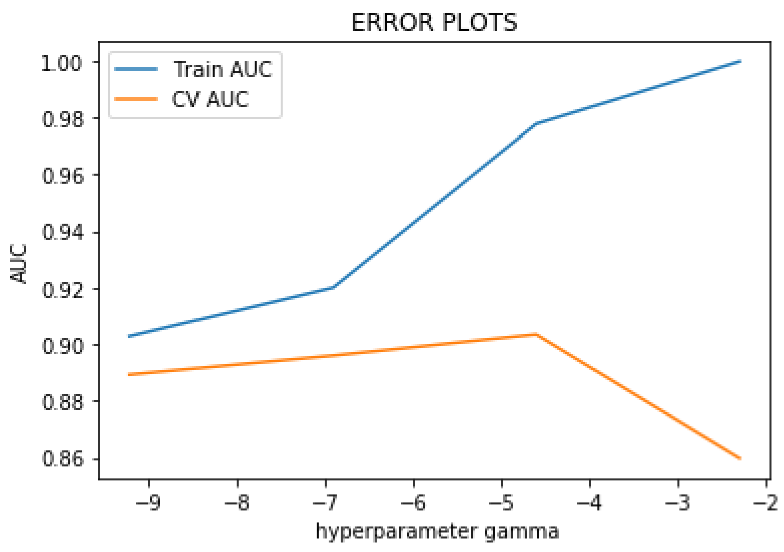
In [279]:

```python
def GSCV_svmrbf(x_train,y_train,x_cv,y_cv):
    from sklearn.model_selection import GridSearchCV
    from sklearn.svm import SVC
    from sklearn.metrics import roc_auc_score
    #clf = SVC(kernel='rbf', probability=True)
    #'C': [0.1, 1, 10, 100, 1000]
    #'gamma':[0.1,0.01,0.001, 0.0001]
    para_grid = { 'C': [0.1, 1, 10, 100, 1000],
                  'gamma':[0.1,0.01,0.001, 0.0001]}
    clf = SVC(kernel='rbf', probability=True)
    model_grid = GridSearchCV(clf, para_grid, cv = 3,  scoring = 'roc_auc')
    model_grid.fit(x_train, y_train)
    best_para=model_grid.best_params_
    print(best_para)
    print("roc_auc_train",model_grid.score(x_train, y_train))
    print("roc_auc_cv",model_grid.score(x_cv, y_cv))
    print("Best HyperParameters using Grid SearchCV & roc_auc metric are: ",best_para)

    b_C=best_para.get('C')#best C
    b_gamma=best_para.get('gamma')# Best gamma as per Grid SearchCV
    print(b_C)
    print(b_gamma)

    param_gamma=para_grid.get('gamma')
    param_gamma1=np.log(param_gamma)
    param_C=para_grid.get('C')
    print(param_gamma)
    print(param_gamma1)
    ###################################################
    #This part is added for plotting Train AUC and cv_Auc
    ###################################################
    train_auc1 = []
    cv_auc1 = []
    for c in param_C:
        clf1 = SVC(C=c, kernel='rbf', probability=True, gamma=b_gamma)
        clf1.fit(x_train, y_train)

        y_train_pred =  clf1.predict_proba(x_train)[:,1]
        y_cv_pred =  clf1.predict_proba(x_cv)[:,1]

        train_auc1.append(roc_auc_score(y_train,y_train_pred))
        cv_auc1.append(roc_auc_score(y_cv, y_cv_pred))
####################################################################

    train_auc2 = []
    cv_auc2 = []
    for g in param_gamma:
        clf2 = SVC(C=b_C, kernel='rbf', probability=True, gamma=g)
        clf2.fit(x_train, y_train)

        y_train_pred =  clf2.predict_proba(x_train)[:,1]
        y_cv_pred =  clf2.predict_proba(x_cv)[:,1]

        train_auc2.append(roc_auc_score(y_train,y_train_pred))
        cv_auc2.append(roc_auc_score(y_cv, y_cv_pred))
########################################################################
    print(param_gamma1)
    print(train_auc1)
    print(cv_auc2)
    print(param_C)
    plt.plot(param_gamma1, train_auc2, label='Train AUC')

    plt.plot(param_gamma1, cv_auc2, label='CV AUC')

    plt.legend()
    plt.xlabel(" hyperparameter gamma")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    print('-'*100)
    plt.plot(param_C, train_auc1, label='Train AUC')
    plt.plot(param_C, cv_auc1, label='CV AUC')
    plt.legend()
    plt.xlabel(" hyperparameter C")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    print('-'*100)
    return(best_para)
```

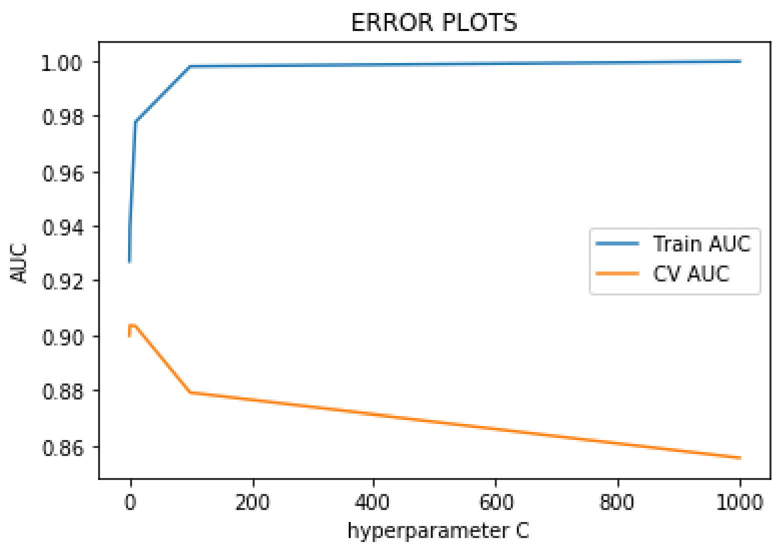
In [280]:

```
best_p1=GSCV_svmrbf(set1a_train,y_train,set1a_cv,y_cv)
```

```
{'C': 10, 'gamma': 0.01}
roc_auc_train 0.9776832642403814
roc_auc_cv 0.903388280630482
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'C': 10,
'gamma': 0.01}
10
0.01
[0.1, 0.01, 0.001, 0.0001]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[0.9270474012407609, 0.9397246192607289, 0.9776835773740857, 0.99800448697
11329, 0.9997678602853775]
[0.8597233906942938, 0.9033869203603171, 0.8959287766894528, 0.88921100086
39348]
[0.1, 1, 10, 100, 1000]
```



```
--------------------------------------------------------------------------
--------------------------
```



```
--------------------------------------------------------------------------
--------------------------
```

In [282]:

```
print(best_p1)
```

```
{'C': 10, 'gamma': 0.01}
```

[5.2.1] Applying RBF SVM on BOW, SET 1

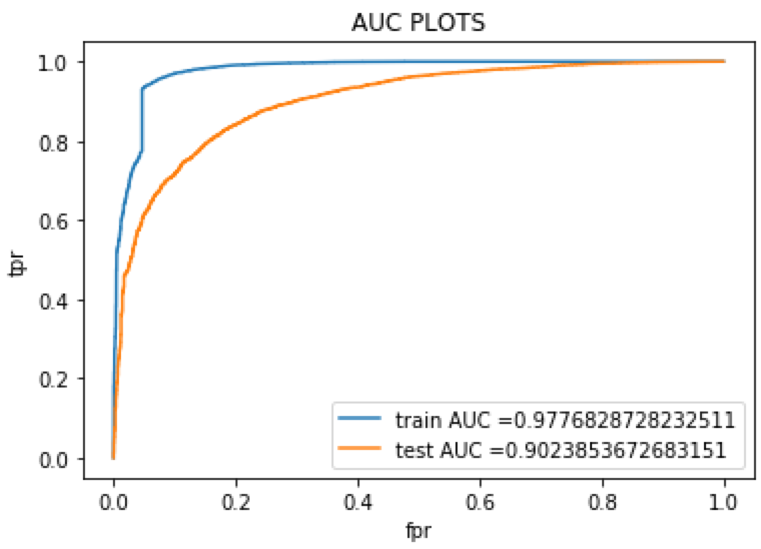
In [283]:

```
def test_svmrbf(x_train,y_train,x_test,y_test,c,g):
    from sklearn.svm import SVC
    from sklearn.calibration import CalibratedClassifierCV
    from sklearn.metrics import roc_auc_score

    model=SVC(C=c, gamma=g, kernel='rbf',probability=True)
    model.fit(x_train,y_train)

    train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)
[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,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)))
    auc_test=auc(test_fpr, test_tpr)
    print('auc for Test data is::',auc_test)

    plt.legend()
    plt.xlabel("fpr")
    plt.ylabel("tpr")
    plt.title("AUC PLOTS")
    plt.show()

    print("="*100)


    yhat_train=model.predict(x_train)
    yhat_test=model.predict(x_test)

    con_mat_train = confusion_matrix(y_train, yhat_train)
    con_mat_test = confusion_matrix(y_test, yhat_test)

    plt.figure()
    class_label = ["negative", "positive"]
    df_con_mat_train = pd.DataFrame(con_mat_train, index = class_label, columns = class
_label)
    sns.heatmap(df_con_mat_train , annot = True, fmt = "d")
    plt.title("Confusiion Matrix for Train data")
    plt.xlabel("Actual Label")
    plt.ylabel("Predicted Label")
    print("Train confusion matrix")
    print(con_mat_train)

    plt.figure()
    class_label = ["negative", "positive"]
    df_con_mat_test = pd.DataFrame(con_mat_test, index = class_label, columns = class_l
abel)
    sns.heatmap(df_con_mat_test , annot = True, fmt = "d")
    plt.title("Confusiion Matrix for Test data")
    plt.xlabel("Actual Label")
    plt.ylabel("Predicted Label")
    plt.show()
    print("Test confusion matrix")
    print(con_mat_test)
    return(auc_test)
```

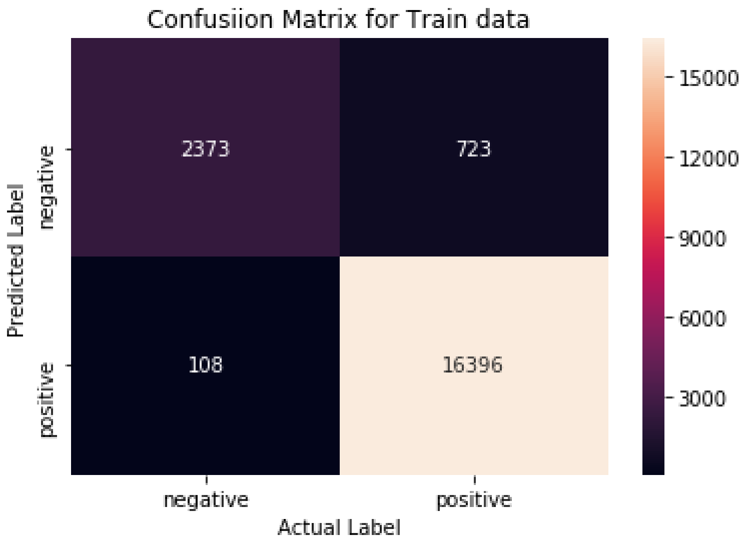
In [285]:

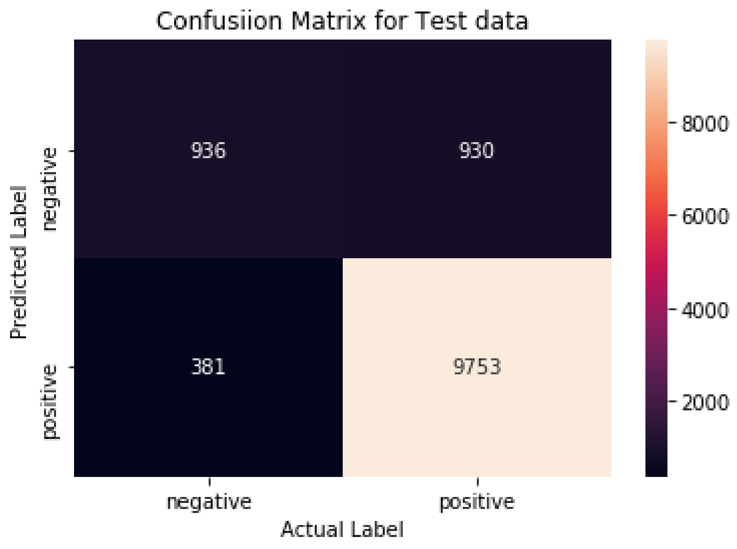
```
opt_c1=best_p1.get('C')
opt_g1=best_p1.get('gamma')
#test_svmrbf(x_train,y_train,x_test,y_test,c,g)
auc_set1a=test_svmrbf(set1a_train,y_train,set1a_test,y_test,opt_c1,opt_g1)
```

```
auc for Test data is:: 0.9023853672683151
```



```
=============================================================================
==========================
Train confusion matrix
[[ 2373   723]
 [  108 16396]]
```





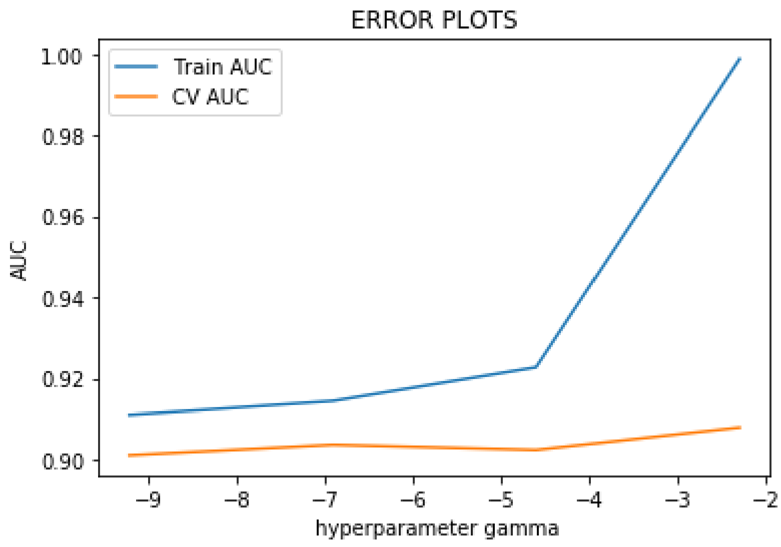
```
Test confusion matrix
[[ 936  930]
 [ 381 9753]]
```

[5.2.2] Applying RBF SVM on TFIDF, SET 2

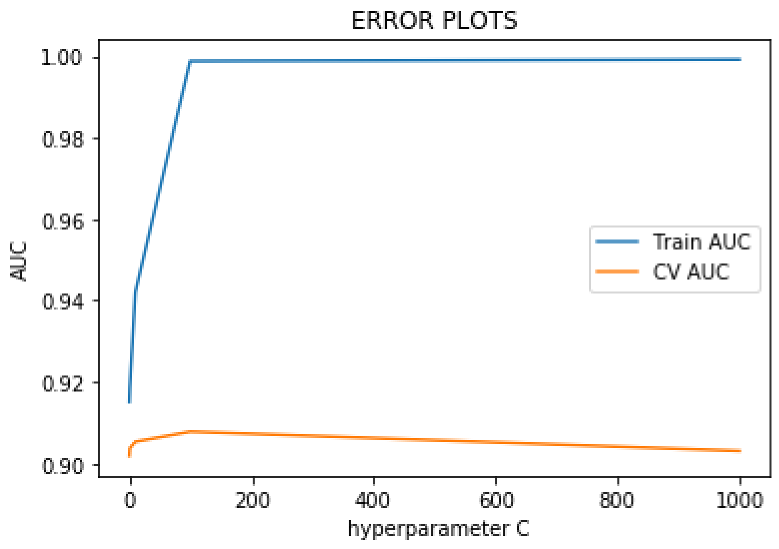
In [286]:

```
# Please write all the code with proper documentation
best_p2=GSCV_svmrbf(set2a_train,y_train,set2a_cv,y_cv)
```

```
{'C': 100, 'gamma': 0.1}
roc_auc_train 0.9987721146764514
roc_auc_cv 0.9078619371487714
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'C': 100,
'gamma': 0.1}
100
0.1
[0.1, 0.01, 0.001, 0.0001]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[0.915040807975766, 0.9186882676472762, 0.9420550287863814, 0.998772007036
7405, 0.9991589717972997]
[0.9078599783597339, 0.9023465313274027, 0.9036065767865434, 0.90102636192
69739]
[0.1, 1, 10, 100, 1000]
```



```
--------------------------------------------------------------------------
--------------------------
```

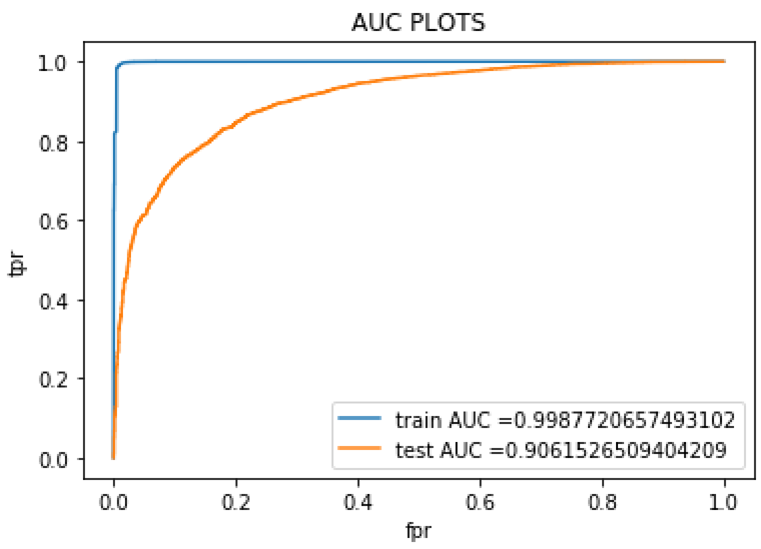


```
--------------------------------------------------------------------------
--------------------------
```

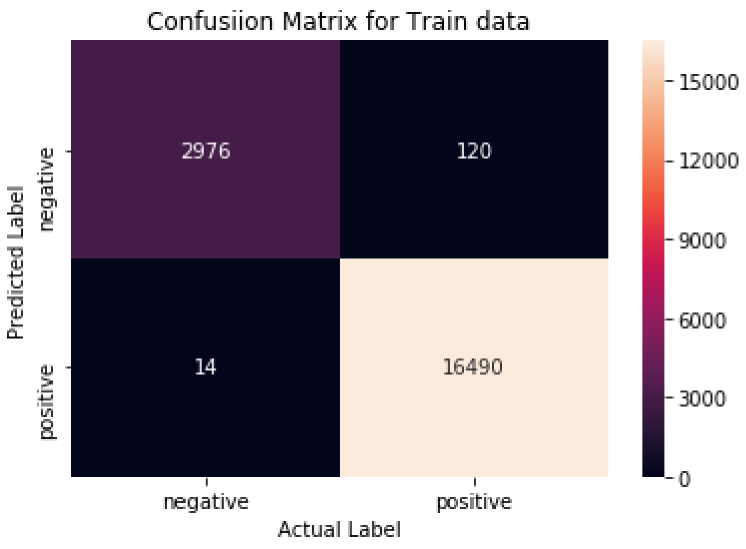
In [287]:

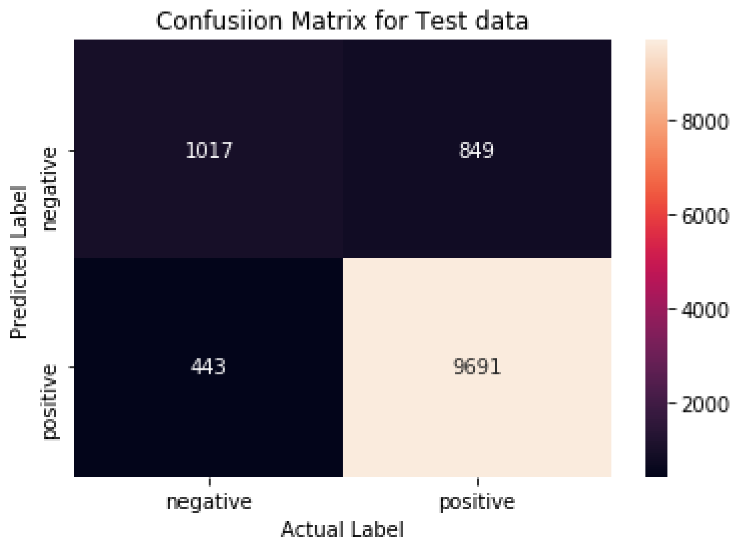
```
opt_c2=best_p2.get('C')
opt_g2=best_p2.get('gamma')
#test_svmrbf(x_train,y_train,x_test,y_test,c,g)
auc_set2a=test_svmrbf(set2a_train,y_train,set2a_test,y_test,opt_c2,opt_g2)
```

```
auc for Test data is:: 0.9061526509404209
```



```
================================================================================
==========================
Train confusion matrix
[[ 2976   120]
 [   14 16490]]
```





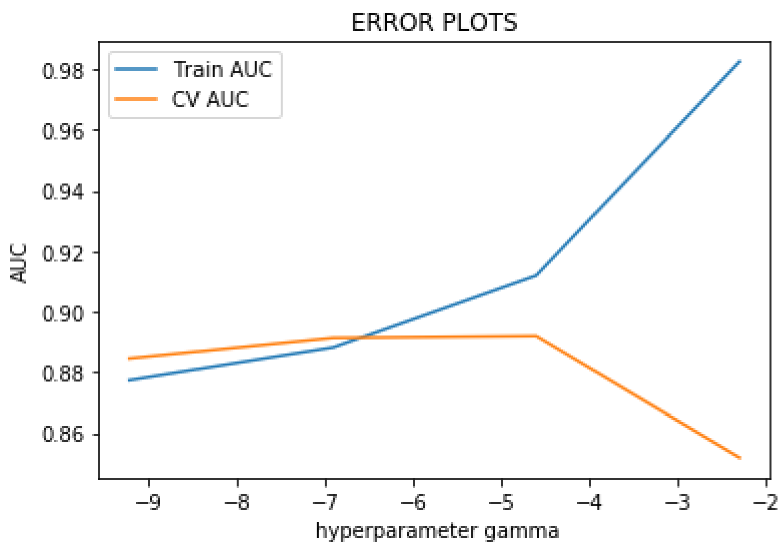
```
Test confusion matrix
[[1017  849]
 [ 443 9691]]
```

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

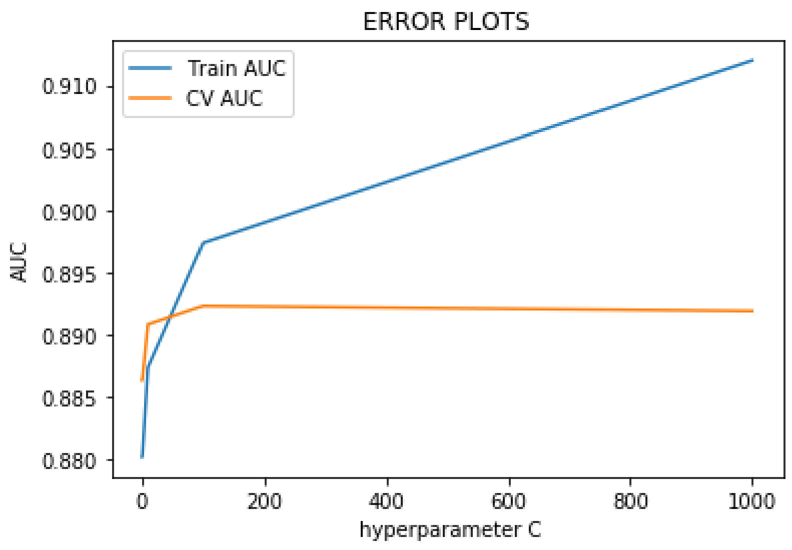
In [288]:

```python
# Please write all the code with proper documentation
best_p3=GSCV_svmrbf(set3_train,y_train,set3_cv,y_cv)
```

```
{'C': 1000, 'gamma': 0.01}
roc_auc_train 0.9119765304722933
roc_auc_cv 0.8919187002433577
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'C': 100
0, 'gamma': 0.01}
1000
0.01
[0.1, 0.01, 0.001, 0.0001]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[0.8802518197765227, 0.8803868391156604, 0.8873834594635894, 0.89738005726
58917, 0.9119765500431499]
[0.8518136917394927, 0.89192457661047, 0.8913133256091753, 0.8844854223111
077]
[0.1, 1, 10, 100, 1000]
```



```
--------------------------------------------------------------------------
--------------------------
```



```
--------------------------------------------------------------------------
--------------------------
```

In [289]:

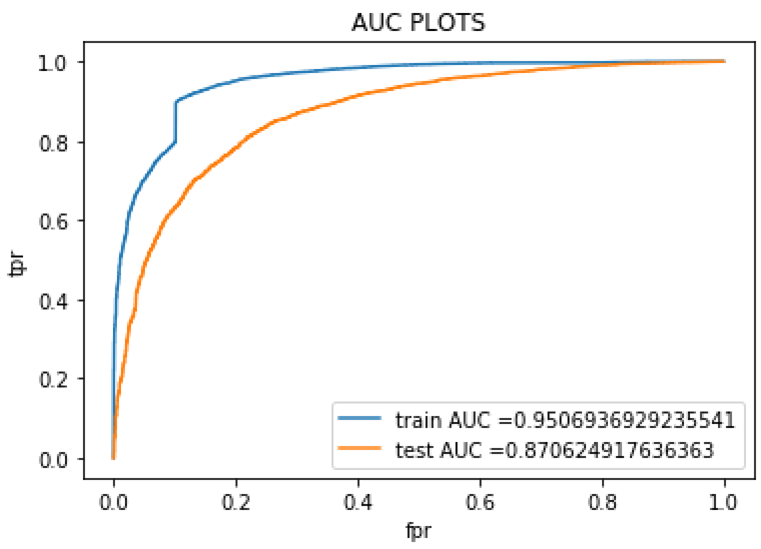
```
print(best_p3)
```

```
{'C': 1000, 'gamma': 0.01}
```

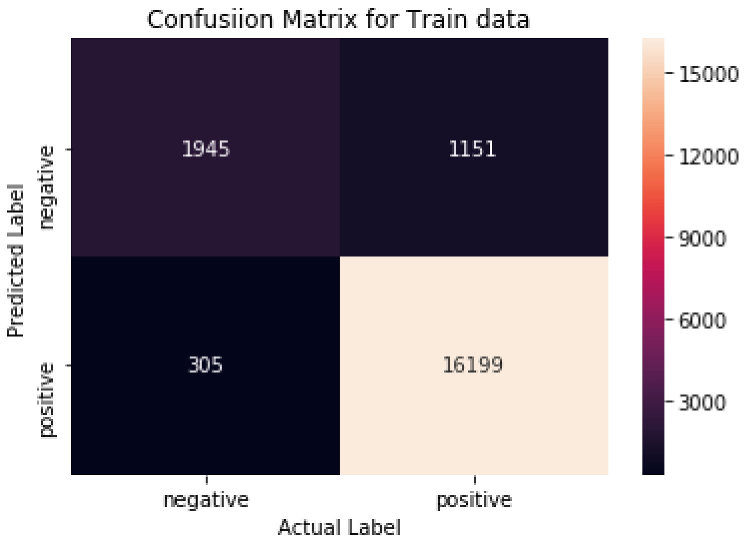
In [290]:

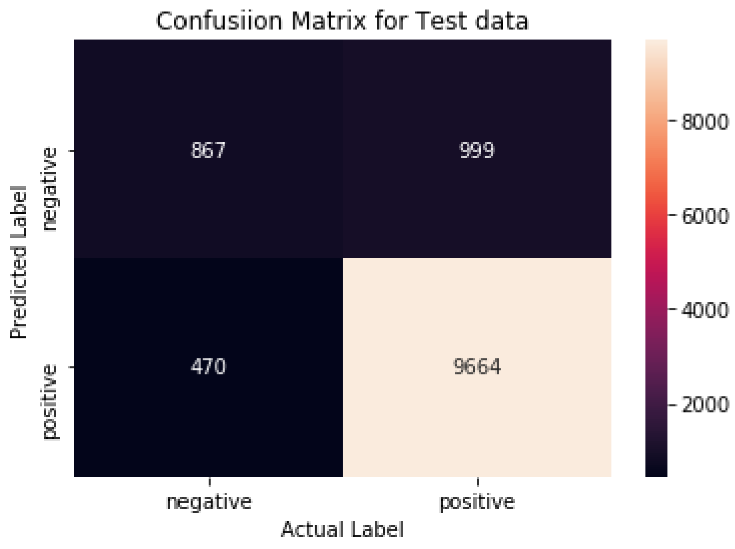
```
opt_c3=best_p2.get('C')
opt_g3=best_p2.get('gamma')
#test_svmrbf(x_train,y_train,x_test,y_test,c,g)
auc_set3rbf=test_svmrbf(set3_train,y_train,set3_test,y_test,opt_c3,opt_g3)
```

```
auc for Test data is:: 0.870624917636363
```



```
===========================================================================
==========================
Train confusion matrix
[[ 1945  1151]
 [  305 16199]]
```





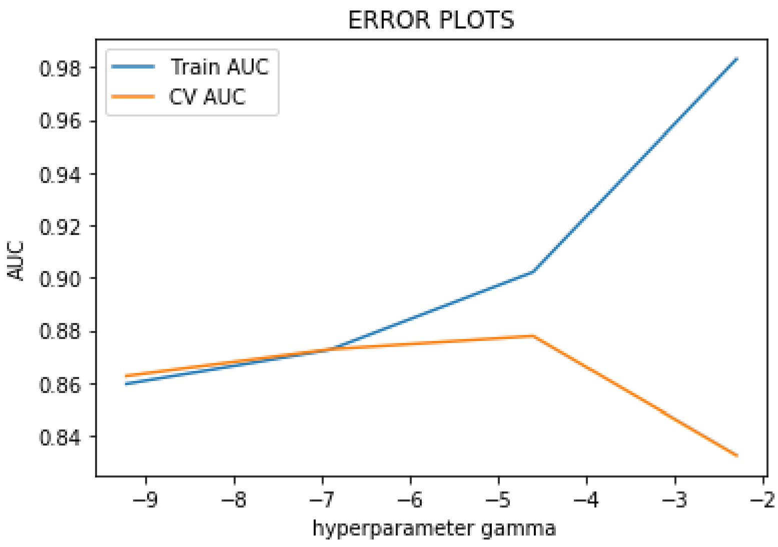
```
Test confusion matrix
[[ 867  999]
 [ 470 9664]]
```

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

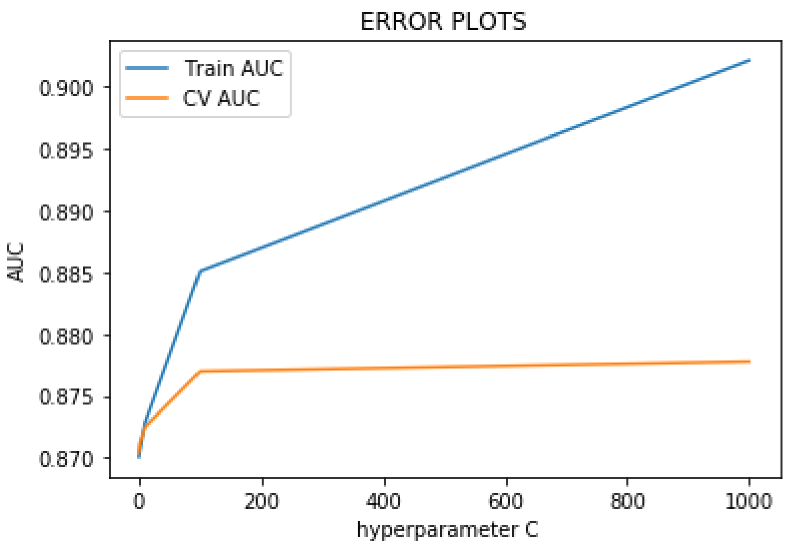
```
In [291]:
```

```
# Please write all the code with proper documentation
best_p4=GSCV_svmrbf(set4_train,y_train,set4_cv,y_cv)
# Please write all the code with proper documentation
```

```
{'C': 1000, 'gamma': 0.01}
roc_auc_train 0.9021288864589714
roc_auc_cv 0.8777611171887982
Best HyperParameters using Grid SearchCV & roc_auc metric are:  {'C': 100
0, 'gamma': 0.01}
1000
0.01
[0.1, 0.01, 0.001, 0.0001]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[-2.30258509 -4.60517019 -6.90775528 -9.21034037]
[0.8700897503823363, 0.8703347383642646, 0.8728151878614345, 0.88509507835
23155, 0.9021281134101388]
[0.8324520959097111, 0.8777642730155808, 0.8726781140474447, 0.86254317633
53038]
[0.1, 1, 10, 100, 1000]
```



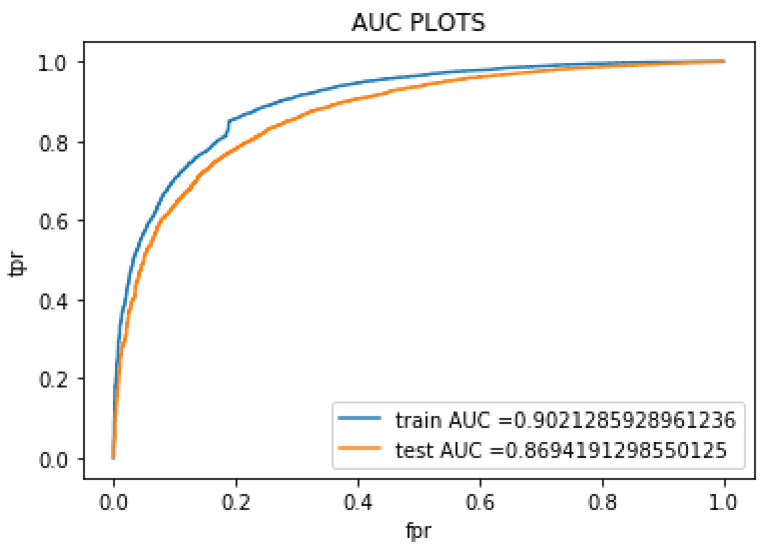
```
-----------------------------------------------------------------------------
--------------------------
```



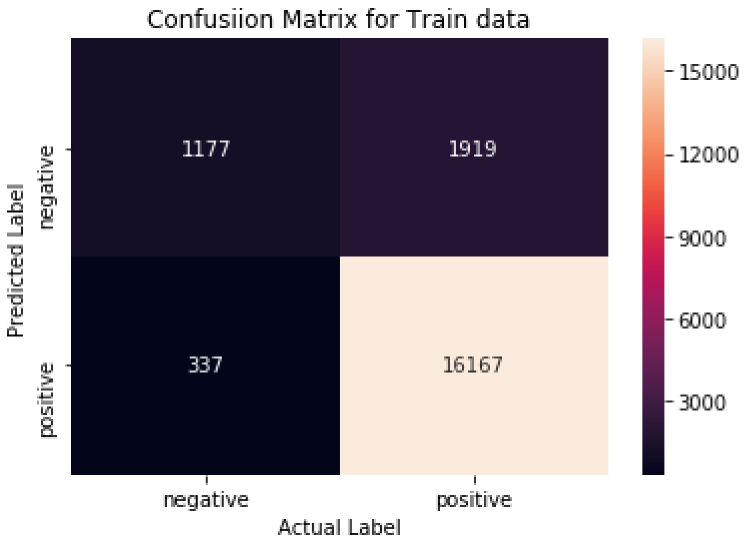
In [292]:

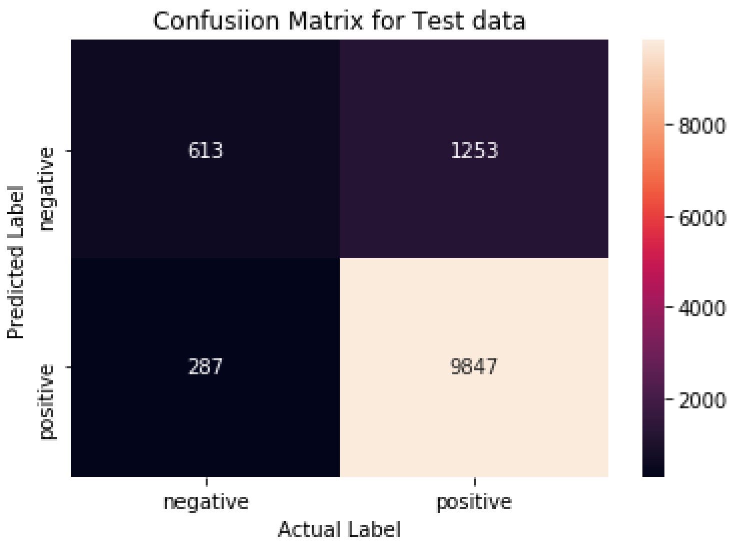
```
opt_c4=best_p4.get('C')
opt_g4=best_p4.get('gamma')
#test_svmrbf(x_train,y_train,x_test,y_test,c,g)
auc_set4a=test_svmrbf(set4_train,y_train,set4_test,y_test,opt_c4,opt_g4)
```

```
auc for Test data is:: 0.8694191298550125
```



```
==========================================================================
==========================
Train confusion matrix
[[ 1177  1919]
 [  337 16167]]
```





```
Test confusion matrix
[[ 613 1253]
 [ 287 9847]]
```

[6] Conclusions

In [294]:

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

from prettytable import PrettyTable
result=PrettyTable()
result.field_names=["Vectorizer","dataset","Hyperparameter1","Hyperparameter2","AUC"]
result.add_row(["BoW","set1",('a',opt_a),('l',opt_p), auc_set1])
result.add_row(["tfidf","set2",('a',opt_a2),('l',opt_p2), auc_set2])
result.add_row(["AvgW2v","set3",('a',opt_a3),('l',opt_p3), auc_set3])
result.add_row(["TFIDF_weighted_W2v","set4",('a',opt_a4),('l',opt_p4), auc_set4])

result.add_row(["BoW","set1",('c',opt_c1),('g',opt_g1), auc_set1a])
result.add_row(["tfidf","set2",('c',opt_c2),('g',opt_g2), auc_set2a])
result.add_row(["AvgW2v","set3",('c',opt_c3),('g',opt_g3), auc_set3rbf])
result.add_row(["TFIDF_weighted_W2v","set4",('c',opt_c4),('g',opt_g4), auc_set4a])



print(result)
```

```
+--------------------+---------+-----------------+-----------------+------
--------------+
|     Vectorizer     | dataset | Hyperparameter1 | Hyperparameter2 |
AUC         |
+--------------------+---------+-----------------+-----------------+------
--------------+
|        BoW         |   set1  |   ('a', 0.001)  |    ('l', 'l2')  | 0.935
5477767958107 |
|       tfidf        |   set2  |  ('a', 0.0001)  |    ('l', 'l2')  | 0.957
9288135768772 |
|       AvgW2v       |   set3  |   ('a', 0.001)  |    ('l', 'l1')  | 0.898
2978481181849 |
| TFIDF_weighted_W2v |   set4  |    ('a', 0.1)   |    ('l', 'l2')  | 0.871
5472546223205 |
|        BoW         |   set1  |    ('c', 10)    |   ('g', 0.01)   | 0.902
3853672683151 |
|       tfidf        |   set2  |    ('c', 100)   |    ('g', 0.1)   | 0.906
1526509404209 |
|       AvgW2v       |   set3  |    ('c', 100)   |    ('g', 0.1)   | 0.870
624917636363  |
| TFIDF_weighted_W2v |   set4  |   ('c', 1000)   |   ('g', 0.01)   | 0.869
4191298550125 |
+--------------------+---------+-----------------+-----------------+------
--------------+
```

Observations:

1.In case of linear SVM for finding probabilities we used Calibrated Classifier cv.

2.For RBF SVM we used SVC and considered only 40k datapoints as it is more expensive than linear SVM.

3 In all cases Best performance is shown by linear svm with tfidf vectorizer.

4 For Linear svm on AvgW2v vectorizered data l1 shows better performance than l2

5 svm with rbf kernel on tfidf vectorized data shows overfitting