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

[1]. Reading Data

Applying SVM

[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".

Mounting Google Drive locally

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
....
Mounted at /content/gdrive

In [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [4]:

```
# using SQLite Table to read data.
con = sqlite3.connect("/content/gdrive/My Drive/Dataset/database.sqlite")



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

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

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

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

Out[4]:

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

In [0]:

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

In [6]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[6]:

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

In [7]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[7]:

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

In [8]:
```
display['COUNT(*)'].sum()
```

Out[8]: 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 [9]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
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 [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_po
```

In [11]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[11]: (87775, 10)

In [12]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[12]: 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 [13]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[13]:

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

In [14]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
final.shape
```

Out[14]: (87773, 10)

In [15]:
```
#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[15]:
```
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 [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '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", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [18]:
```
# Combining all the above stundents
from bs4 import BeautifulSoup
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:40<00:00, 2193.43it/s]
```

In [0]:
```
final["CleanText"] = [preprocessed_reviews[i] for i in range(len(final))]
```

In [20]:
```
final.head(2)
```

Out[20]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	

[4] Featurization

In [0]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
import seaborn as sns

from sklearn.metrics import confusion_matrix

# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
```

In [0]:
```
Total_X = final['CleanText'].values
Total_y = final['Score'].values
```

In [0]:
```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(Total_X, Total_y, test_size=0.33)

# split the train data set into cross validation train and cross validation test
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33)
```

In [32]:
```
print(f"Train Data : ({len(X_train)} , {len(y_train)})")
print(f"CV Data : ({len(X_cv)} , {len(y_cv)})")
print(f"Test Data : ({len(X_test)} , {len( y_test)})")
```

```
Train Data : (39400 , 39400)
CV Data : (19407 , 19407)
Test Data : (28966 , 28966)
```

L1 and L2 Regularizer Function

In [0]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
```

In [0]:
```
def SVM_l1(X_train_reg,X_cv_reg, y_train=y_train,  y_cv=y_cv,  y_test=y_test):
    train_auc = []
    cv_auc = []
    max_alpha=0
    max_roc_auc=-1
    all_alpha = [1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]
    for i in tqdm(all_alpha):

        clf =  SGDClassifier(penalty='l1',alpha = i)
        clf.fit(X_train_reg, y_train)
        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
        # not the predicted outputs


        # Calibrated with sigmoid calibration
        clf_sigmoid = CalibratedClassifierCV(clf, method='sigmoid')
        clf_sigmoid.fit(X_train_reg, y_train)
        prob_pos_sigmoid = clf_sigmoid.predict_proba(X_cv_reg)[:, 1]


        y_train_pred = clf_sigmoid.predict_proba(X_train_reg)[:, 1]
        y_cv_pred =  clf_sigmoid.predict_proba(X_cv_reg)[:, 1]
        #proba1 =roc_auc_score(y_train,y_train_pred) * float(100)
        proba2 = roc_auc_score(y_cv, y_cv_pred) * float(100)

        if(max_roc_auc<proba2):
            max_roc_auc=proba2
            max_alpha=i
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

    print(f"\nThe 'alpha' value {max_alpha} with highest roc_auc Score is {proba2} %" )
    plt.plot(all_alpha, train_auc, label='Train AUC')
    plt.plot(all_alpha, cv_auc, label='CV AUC')
    plt.xscale(value = 'log')
    plt.legend()
    plt.xlabel("alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
```

In [0]:
```
def SVM_l2(X_train_reg,X_cv_reg, y_train=y_train,  y_cv=y_cv,  y_test=y_test):
    train_auc = []
    cv_auc = []
    max_alpha=0
    max_roc_auc=-1
    all_alpha = [1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]
    for i in tqdm(all_alpha):

        clf =  SGDClassifier(penalty='l2',alpha = i)
        clf.fit(X_train_reg, y_train)
        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
        # not the predicted outputs


        # Calibrated with sigmoid calibration
        clf_sigmoid = CalibratedClassifierCV(clf, method='sigmoid')
        clf_sigmoid.fit(X_train_reg, y_train)
        prob_pos_sigmoid = clf_sigmoid.predict_proba(X_cv_reg)[:, 1]


        y_train_pred = clf_sigmoid.predict_proba(X_train_reg)[:, 1]
        y_cv_pred =  clf_sigmoid.predict_proba(X_cv_reg)[:, 1]
        #proba1 =roc_auc_score(y_train,y_train_pred) * float(100)
        proba2 = roc_auc_score(y_cv, y_cv_pred) * float(100)

        if(max_roc_auc<proba2):
            max_roc_auc=proba2
            max_alpha=i
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

    print(f"\nThe 'alpha' value {max_alpha} with highest roc_auc Score is {proba2} %" )
    plt.plot(all_alpha, train_auc, label='Train AUC')
    plt.plot(all_alpha, cv_auc, label='CV AUC')
    plt.xscale(value = 'log')
    plt.legend()
    plt.xlabel("alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
```

Testing the best alpha value with Test datapoints and Confusion Matrix

In [0]:
```
def testing_l1(X_train_reg,X_test_reg, max_alpha, y_train=y_train,  y_test=y_test):
    clf =   SGDClassifier(penalty='l1', alpha = max_alpha)
    clf.fit(X_test_reg, y_test)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    # Calibrated with sigmoid calibration
    clf_sigmoid = CalibratedClassifierCV(clf, method='sigmoid')
    clf_sigmoid.fit(X_train_reg, y_train)
    #prob_pos_sigmoid = clf_sigmoid.predict_proba(X_cv_reg)[:, 1]

    train_fpr, train_tpr, thresholds = roc_curve(y_train, clf_sigmoid.predict_proba(X_train_reg)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_sigmoid.predict_proba(X_test_reg)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.xscale(value = 'log')
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC curves")
    plt.show()

    print(f"Precision on test data: {precision_score(y_test, clf.predict(X_test_reg))}")
    print(f"Recall on test data: {recall_score(y_test, clf.predict(X_test_reg))}")
    print(f"F1-Score on test data: {f1_score(y_test, clf.predict(X_test_reg))}")

    print("\nConfusion Matrix of Train and Test set:\n [ [TN  FP]\n   [FN TP] ]\n")
    confusionMatrix_train=confusion_matrix(y_train, clf.predict(X_train_reg))
    confusionMatrix_test=confusion_matrix(y_test, clf.predict(X_test_reg))
    df_cm_tr = pd.DataFrame(confusionMatrix_train, range(2),range(2))
    df_cm_te = pd.DataFrame(confusionMatrix_test, range(2),range(2))
    plt.figure(figsize = (7,5))
    plt.ylabel("Predicted label")
    plt.xlabel("Actual label")
    plt.title("Confusion Matrix of Train Set")
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm_tr, annot=True,annot_kws={"size": 12},fmt="d")

    plt.figure(figsize = (7,6))
    plt.ylabel("Predicted label")
    plt.xlabel("Actual label")
    plt.title("Confusion Matrix of Test Set")
    sns.heatmap(df_cm_te, annot=True,annot_kws={"size": 12},fmt="d")
```

In [0]:
```
def testing_l2(X_train_reg,X_test_reg, max_alpha, y_train=y_train,  y_test=y_test):
    clf =   SGDClassifier(penalty='l2', alpha = max_alpha)
    clf.fit(X_test_reg, y_test)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    # Calibrated with sigmoid calibration
    clf_sigmoid = CalibratedClassifierCV(clf, method='sigmoid')
    clf_sigmoid.fit(X_train_reg, y_train)

    train_fpr, train_tpr, thresholds = roc_curve(y_train, clf_sigmoid.predict_proba(X_train_reg)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_sigmoid.predict_proba(X_test_reg)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.xscale(value = 'log')
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC curves")
    plt.show()



    print(f"Precision on test data: {precision_score(y_test, clf.predict(X_test_reg))}")
    print(f"Recall on test data: {recall_score(y_test, clf.predict(X_test_reg))}")
    print(f"F1-Score on test data: {f1_score(y_test, clf.predict(X_test_reg))}")

    print("\nConfusion Matrix of Train and Test set:\n [ [TN  FP]\n   [FN TP] ]\n")
    confusionMatrix_train=confusion_matrix(y_train, clf.predict(X_train_reg))
    confusionMatrix_test=confusion_matrix(y_test, clf.predict(X_test_reg))
    df_cm_tr = pd.DataFrame(confusionMatrix_train, range(2),range(2))
    df_cm_te = pd.DataFrame(confusionMatrix_test, range(2),range(2))
    plt.figure(figsize = (7,5))
    plt.ylabel("Predicted label")
    plt.xlabel("Actual label")
    plt.title("Confusion Matrix of Train Set")
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm_tr, annot=True,annot_kws={"size": 12},fmt="d")

    plt.figure(figsize = (7,6))
    plt.ylabel("Predicted label")
    plt.xlabel("Actual label")
    plt.title("Confusion Matrix of Test Set")
    sns.heatmap(df_cm_te, annot=True,annot_kws={"size": 12},fmt="d")
```

[4.1] BAG OF WORDS

[5.1] SVM on BOW, SET 1

In [38]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[1000:1010])
print('='*50)

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = count_vect.transform(X_train)
X_cv_bow = count_vect.transform(X_cv)
X_test_bow = count_vect.transform(X_test)


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

```
some feature names  ['amazes', 'amazing', 'amazingclubs', 'amazingg', 'amazinggg', 'amazingi', 'amazingly', 'a
mazn', 'amazom', 'amazon']
==================================================
After vectorizations
(39400, 37181) (39400,)
(19407, 37181) (19407,)
(28966, 37181) (28966,)
========================================================================================
============
```

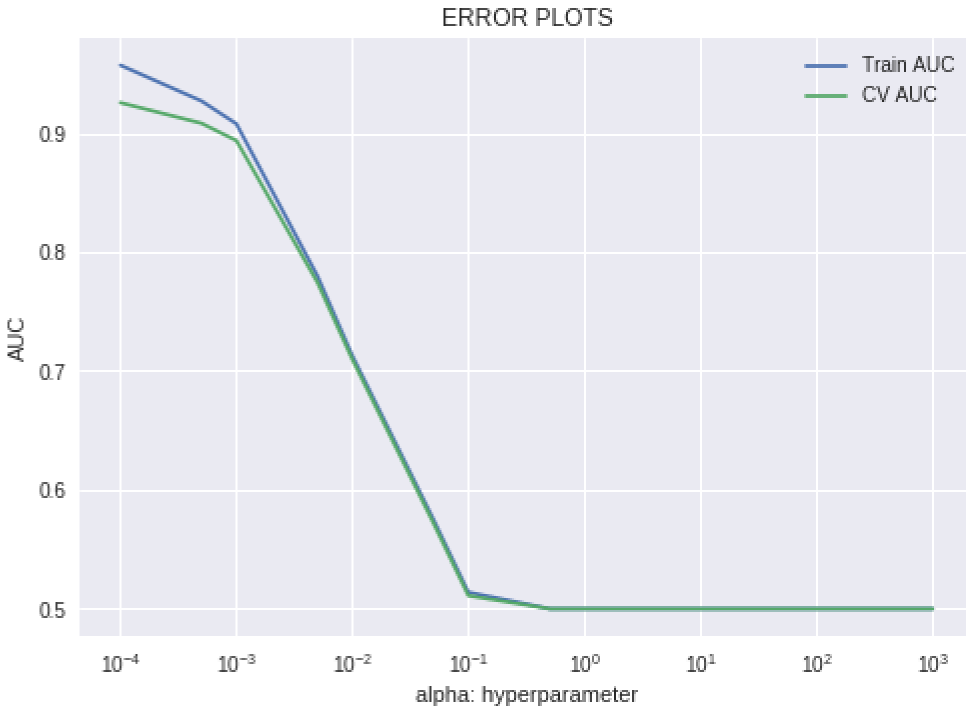
[5.1.1] Applying SVM with L1 regularization on BOW

In [0]:
```
#from tqdm import tqdm_notebook as tqdm
```

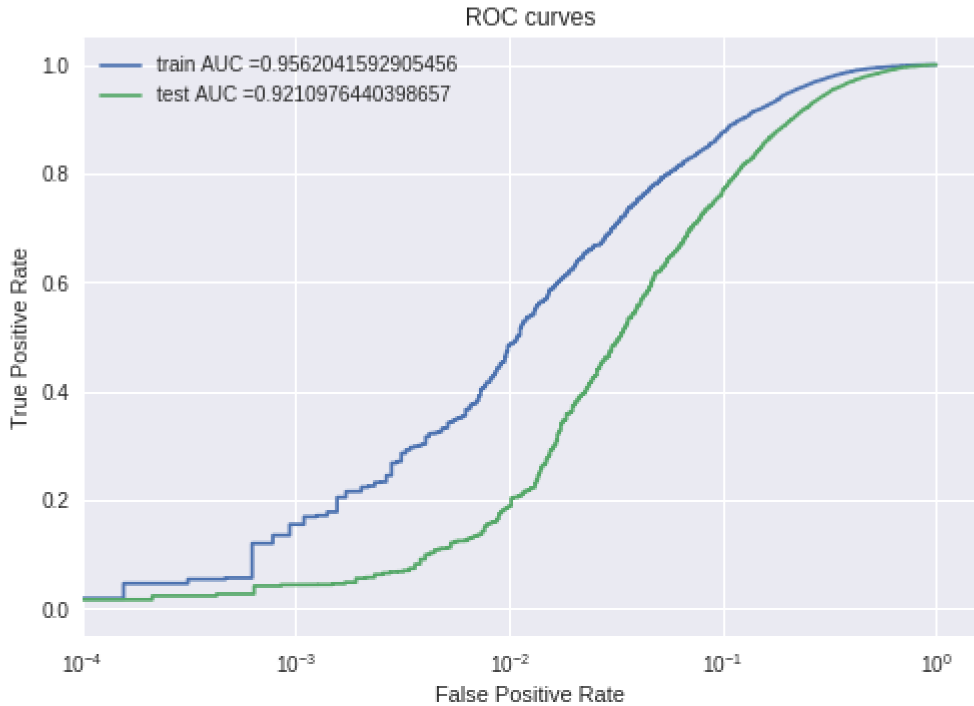
In [39]:
```
SVM_l1(X_train_bow,X_cv_bow)
```

```
100%|██████████| 15/15 [00:11<00:00,  1.04it/s]
```

The 'alpha' value 0.0001 with highest roc_auc Score is 92.55096455164082 %

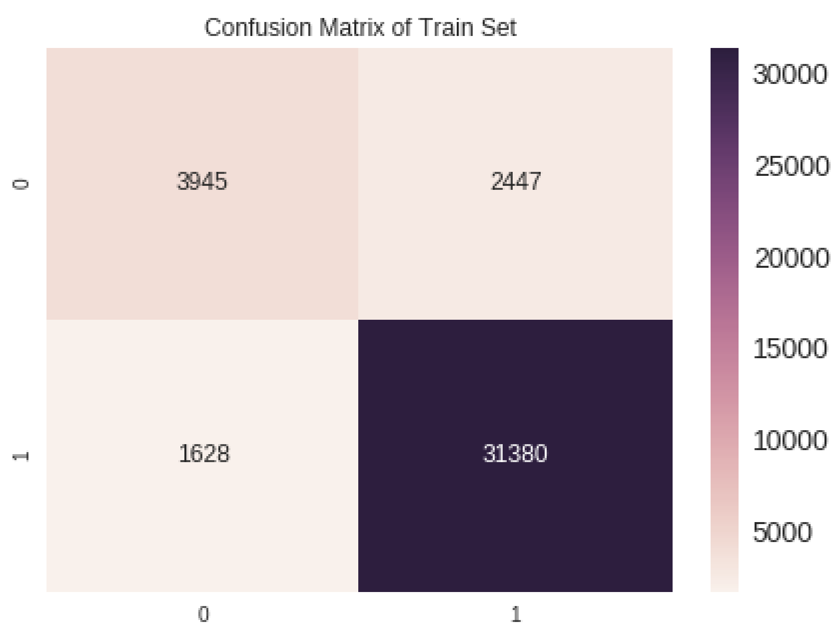


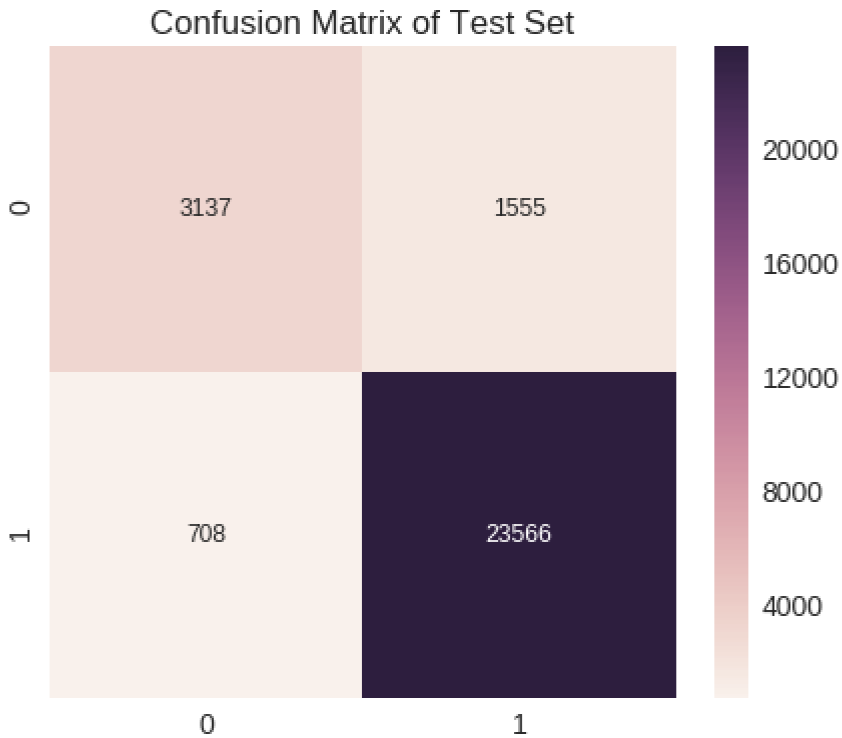
In [40]:
```
testing_l1(X_train_bow,X_test_bow,0.0001)
```



Precision on test data: 0.9380995979459417
Recall on test data: 0.9708329900304853
F1-Score on test data: 0.9541856463204778

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]



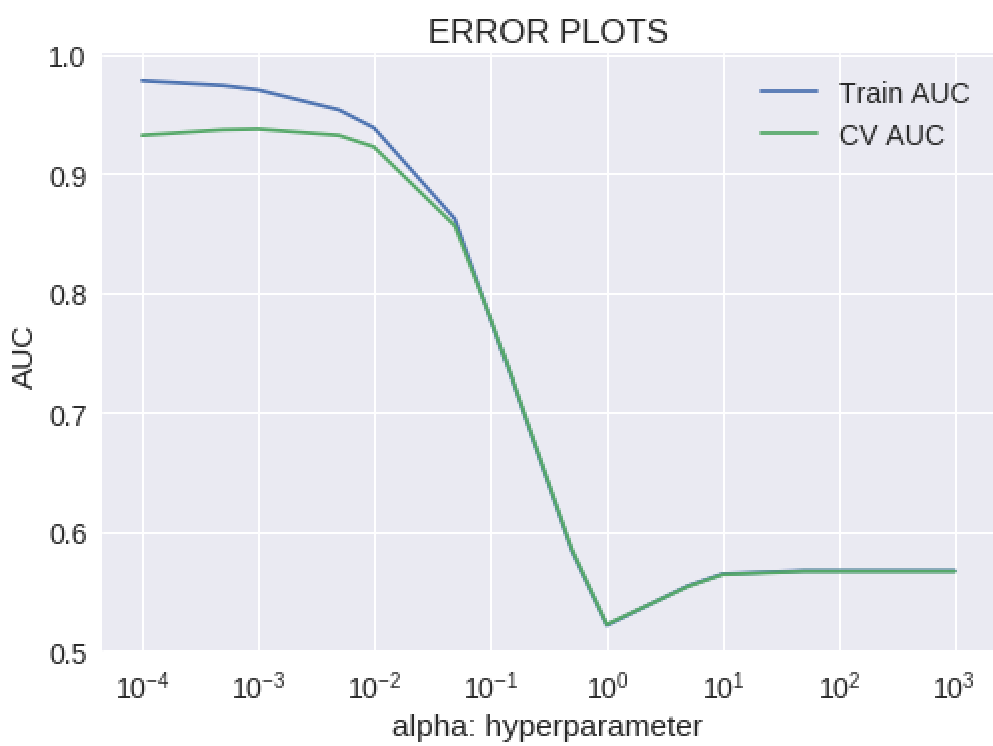


[5.1.2] Applying SVM with L2 regularization on BOW

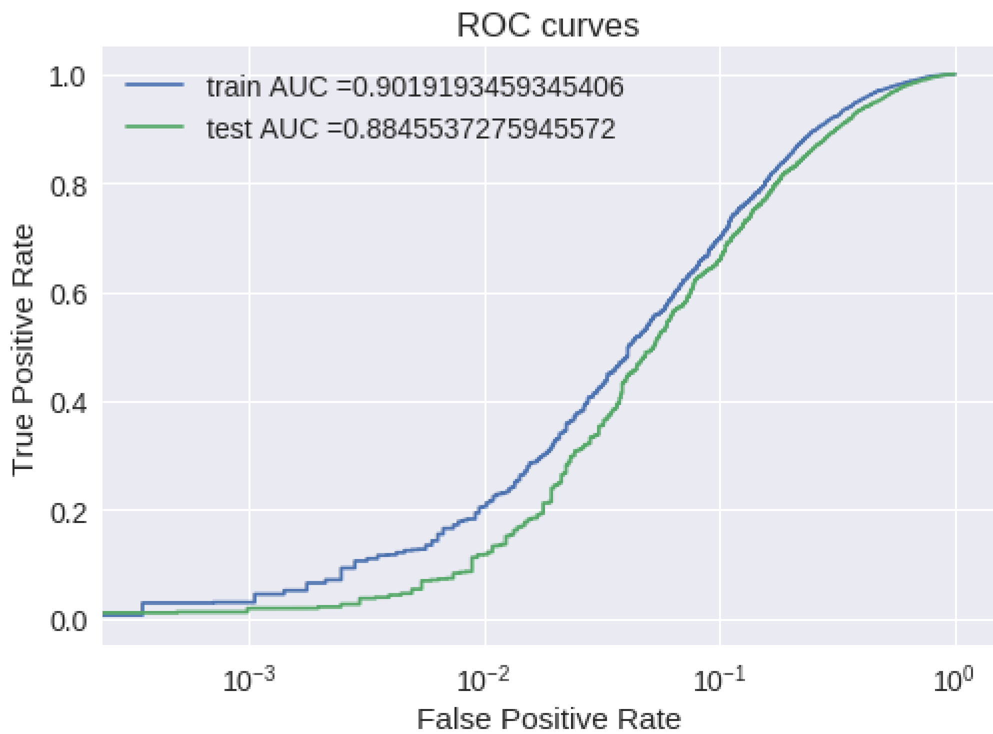
In [41]:
```
SVM_l2(X_train_bow,X_cv_bow)
```

100%|██████████| 15/15 [00:09<00:00, 1.37it/s]

The 'alpha' value 0.001 with highest roc_auc Score is 93.17272287593836 %

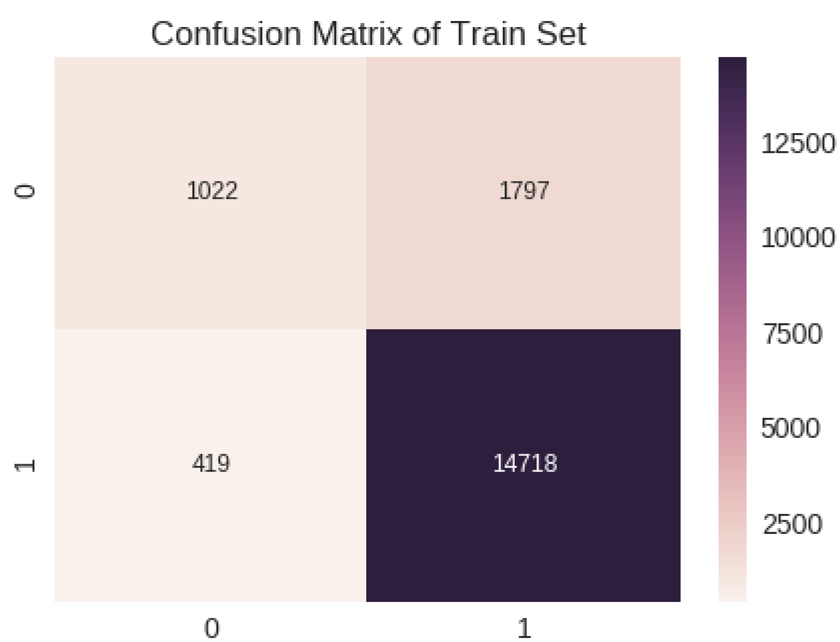


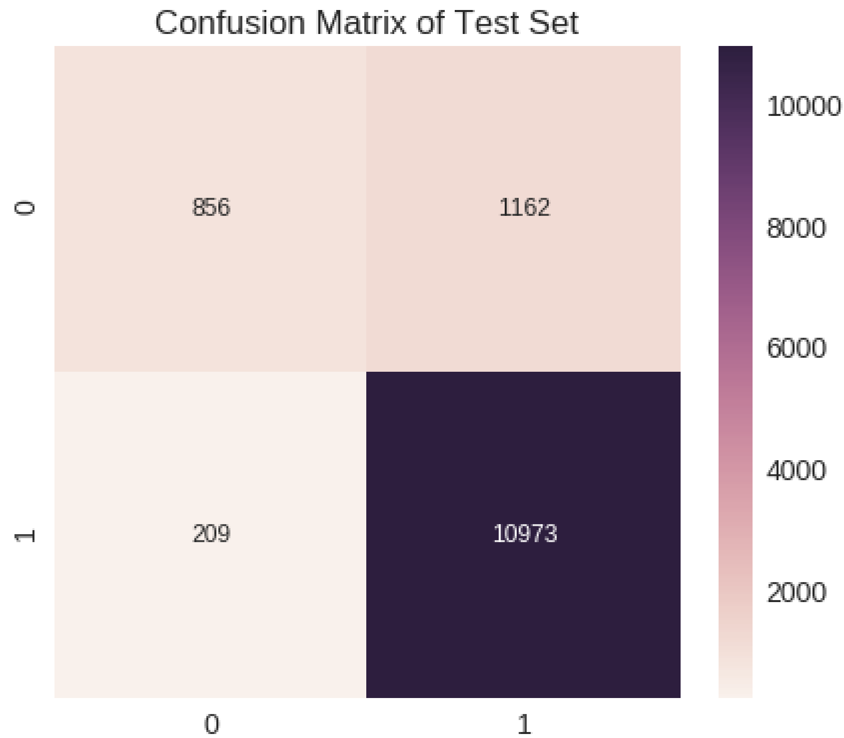
In [106]:
```
testing_l2(X_train_bow,X_test_bow,0.001)
```



Precision on test data: 0.9042439225381129
Recall on test data: 0.9813092470041137
F1-Score on test data: 0.9412016983316892

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





[5.1.1] Top 10 important features of positive and negative class from SET 1

Reference : [https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers (https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers)](https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers)

In [0]:
```
def top_important_features(vectorizer, l, alp, n=10):
    clf =   SGDClassifier(penalty=l, alpha =alp)
    clf.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    # Calibrated with sigmoid calibration
    clf_sigmoid = CalibratedClassifierCV(clf, method='sigmoid')
    clf_sigmoid.fit(X_train_bow, y_train)

    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    print("\t\t\tNegative\t\t\t\t\tPositive")
    print("________________________________________________________________________________________________
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))


top_important_features(count_vect,"l2",0.001)
```

```
            Negative                        Positive
________________________________________________________________________________________________
_________
    -0.9433  disappointing                  0.5934   perfect
    -0.9180  worst                      0.5883   delicious
    -0.8419  terrible                   0.5832   best
    -0.8064  threw                      0.5680   highly
    -0.8013  awful                      0.5629   excellent
    -0.7912  disappointment                 0.5325   great
    -0.7861  horrible                   0.5274   wonderful
    -0.7607  disappointed                   0.5224   amazing
    -0.6542  return                     0.5173   satisfied
    -0.6542  sorry                      0.5173   pleased
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#bi-gram, tri-gram and n-gram

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

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

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

[4.3] TF-IDF

[5.2] SVM on TFIDF, SET 2

[5.2.1] Applying SVM with L1 regularization on TFIDF

In [43]:

```
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)


# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_idf = tf_idf_vect.transform(X_train)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
X_test_tf_idf = tf_idf_vect.transform(X_test)


print("After vectorizations")
print(X_train_tf_idf.shape, y_train.shape)
print(X_cv_tf_idf.shape, y_cv.shape)
print(X_test_tf_idf.shape, y_test.shape)
print("="*100)
```

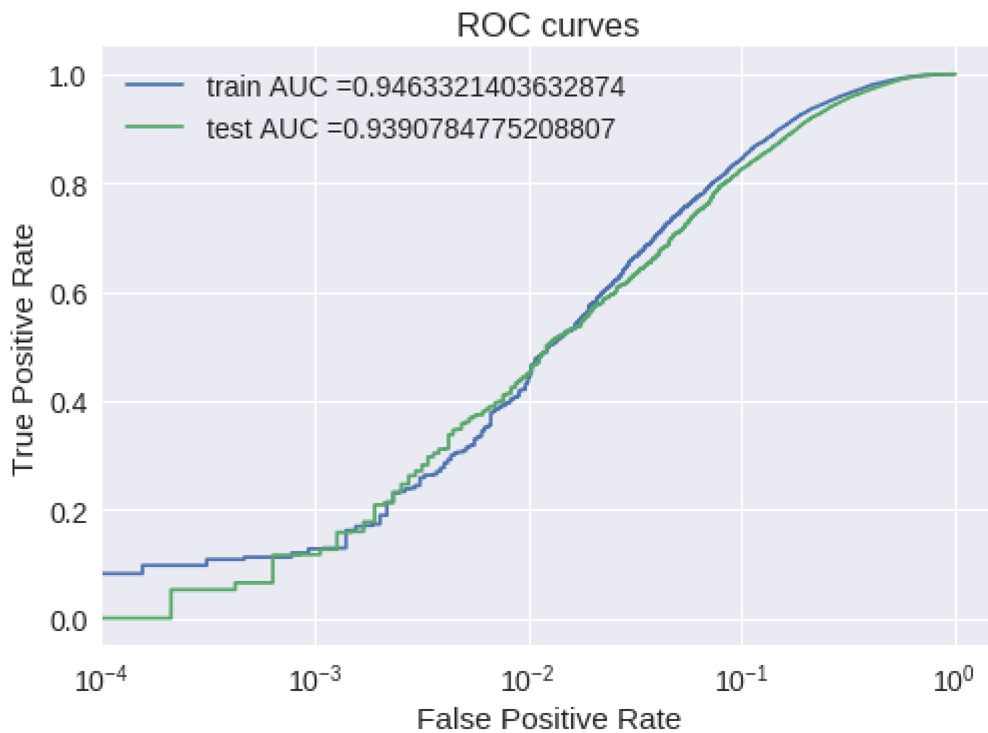
```
some sample features(unique words in the corpus) ['abandon', 'ability', 'able', 'able add', 'able buy', 'able drin
k', 'able eat', 'able enjoy', 'able find', 'able finish']
==================================================
After vectorizations
(39400, 23375) (39400,)
(19407, 23375) (19407,)
(28966, 23375) (28966,)
==========================================================================================
==========
```

In [44]:
```
SVM_l1(X_train_tf_idf, X_cv_tf_idf)
```

100%|██████████| 15/15 [00:09<00:00, 1.48it/s]

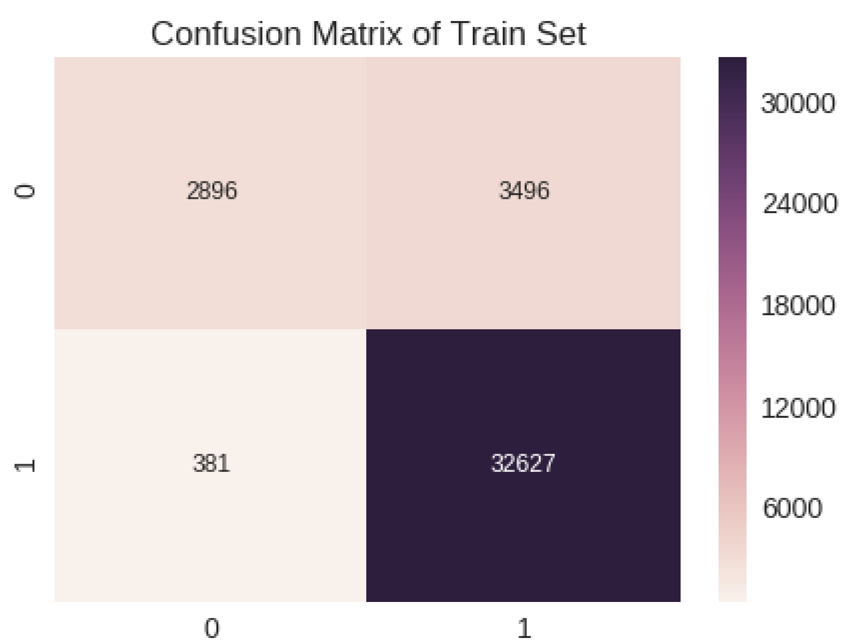
The 'alpha' value 0.0001 with highest roc_auc Score is 93.61096070701517 %

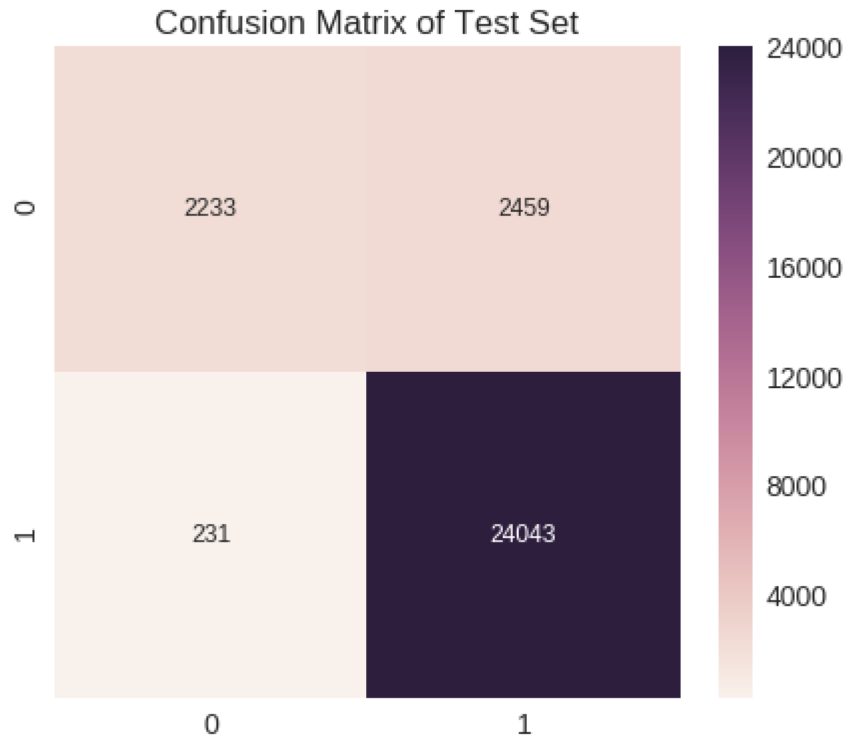
In [45]:
```
testing_l1(X_train_tf_idf, X_test_tf_idf,0.0001)
```



Precision on test data: 0.9072145498452947
Recall on test data: 0.9904836450523193
F1-Score on test data: 0.9470222152197888

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





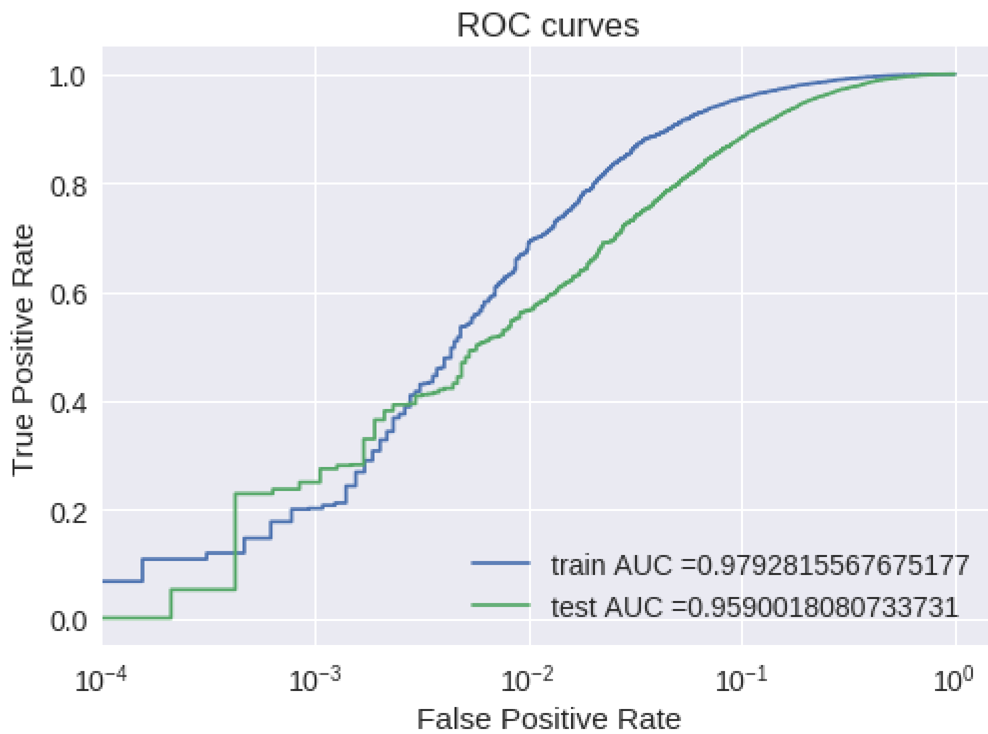
[5.2.2] Applying SVM with L2 regularization on TFIDF,

In [46]:
```
SVM_l2(X_train_tf_idf, X_cv_tf_idf)
```

```
100%|██████████| 15/15 [00:09<00:00,  1.84it/s]
```

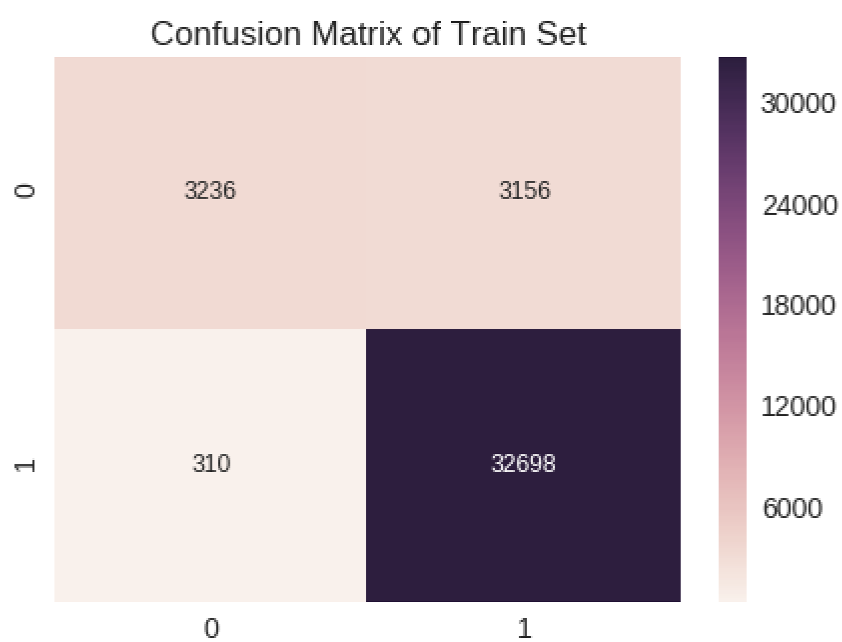
The 'alpha' value 0.0001 with highest roc_auc Score is 95.72366367088104 %

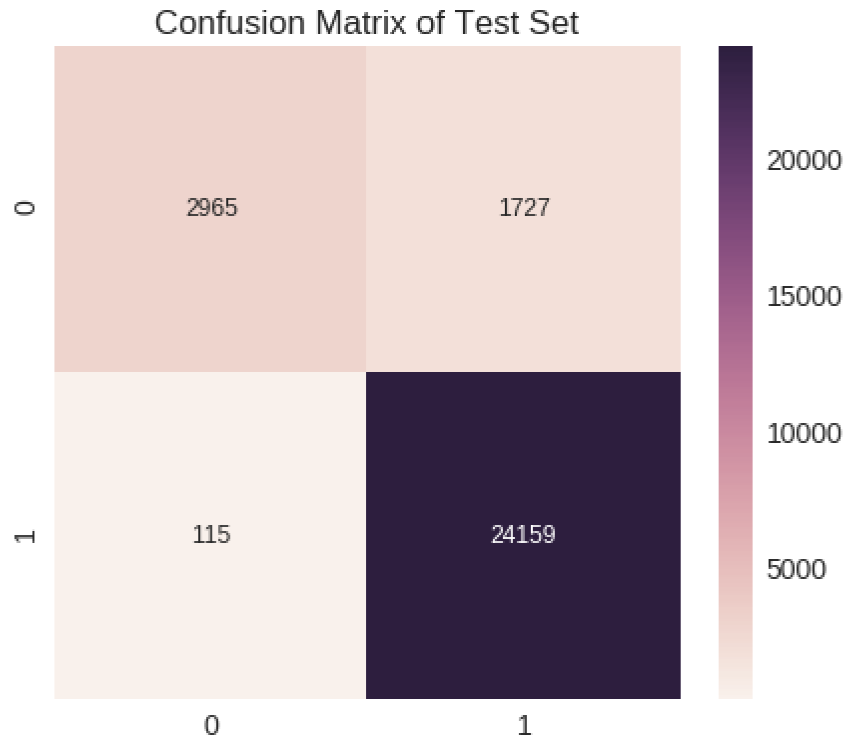
In [47]:
```
testing_l2(X_train_tf_idf, X_test_tf_idf, 0.0001)
```



Precision on test data: 0.9332844008344279
Recall on test data: 0.9952624206970421
F1-Score on test data: 0.9632775119617225

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





[5.2.2] Top 10 important features of Positive and Negative class from SET 2

In [0]:
```
top_important_features(tf_idf_vect, "l2", 0.0001)
```

```
            Negative                        Positive
______________________________________________________________________________________
________
      -2.9798  handle                    2.1212   perfect snack
      -2.6263  handling                  2.0202   cal
      -2.4243  box two                   1.8687   cats get
      -1.9697  pun intended                    1.7677   assuming
      -1.9697  cents                     1.6667   nutty taste
      -1.9192  handfuls                  1.6162   husband said
      -1.9192  little packages                 1.5152   please keep
      -1.8182  favorite dark                   1.5152   keep ordering
      -1.8182  junk                      1.5152   fantastic flavor
      -1.7677  plentiful                 1.4647   use base
```

[4.4] Word2Vec

In [0]:
```
i=0

w2v_train=[]
w2v_cv=[]
w2v_test=[]

for sentance in X_train:
    w2v_train.append(sentance.split())

for sentance in X_cv:
    w2v_cv.append(sentance.split())


for sentance in X_test:
    w2v_test.append(sentance.split())
```

In [49]:
```
want_to_train_w2v = True
if want_to_train_w2v:
# min_count = 5 considers only words that occured atleast 5 times
#w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    w2v_model_train = Word2Vec(w2v_train,min_count=5,size=50, workers=4)
    print(w2v_model_train.wv.most_similar('great'))
    print('='*50)
else:
    pass
```

```
[('awesome', 0.8171955943107605), ('good', 0.7861046195030212), ('fantastic', 0.7795147895812988), ('exc
ellent', 0.759508490562439), ('wonderful', 0.7578698992729187), ('terrific', 0.7423816323280334), ('amazin
g', 0.7391855120658875), ('perfect', 0.7007318139076233), ('nice', 0.6843549013137817), ('decent', 0.6600
550413131714)]
==================================================
```

In [50]:
```
w2v_words_train = list(w2v_model_train.wv.vocab)

print("number of words that occured minimum 5 times ",len(w2v_words_train ))
print("sample words ", w2v_words_train[0:50])
```

```
number of words that occured minimum 5 times  11948
sample words  ['yummy', 'granola', 'bars', 'indeed', 'taste', 'like', 'coconut', 'chocolate', 'macaroon', 'calories',
'grams', 'fat', 'per', 'pack', 'sugar', 'love', 'anything', 'never', 'bar', 'flavor', 'really', 'stand', 'crunchy', 'crumbly',
'not', 'hard', 'way', 'chip', 'tooth', 'arent', 'healthy', 'slightly', 'better', 'choice', 'candy', 'low', 'fiber', 'oh', 'amazo
n', 'great', 'price', 'drank', 'single', 'switch', 'orange', 'tangerine', 'today', 'upon', 'first', 'impression']
```

#Converting text into vectors using Avg W2V, TFIDF-W2V

[5.1.3] Applying SVM on AVG W2V, SET 3

[4.4.1.1] Avg W2v

In [51]:
```
train_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(w2v_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 us
    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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
print()
print(len(train_vectors))
print(len(train_vectors[0]))
```

```
100%|██████████| 39400/39400 [01:16<00:00, 514.54it/s]


39400
50
```

In [52]:
```
# compute average word2vec for each review.
cv_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(w2v_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 us
    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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)
print()
print(len(cv_vectors))
print(len(cv_vectors[0]))
```

```
100%|██████████| 19407/19407 [00:37<00:00, 516.17it/s]


19407
50
```

In [53]:
```
test_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(w2v_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 us
    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_train:
            vec =  w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)
print()
print(len(test_vectors))
print(len(test_vectors[0]))
```

```
100%|██████████| 28966/28966 [00:57<00:00, 507.62it/s]


28966
50
```

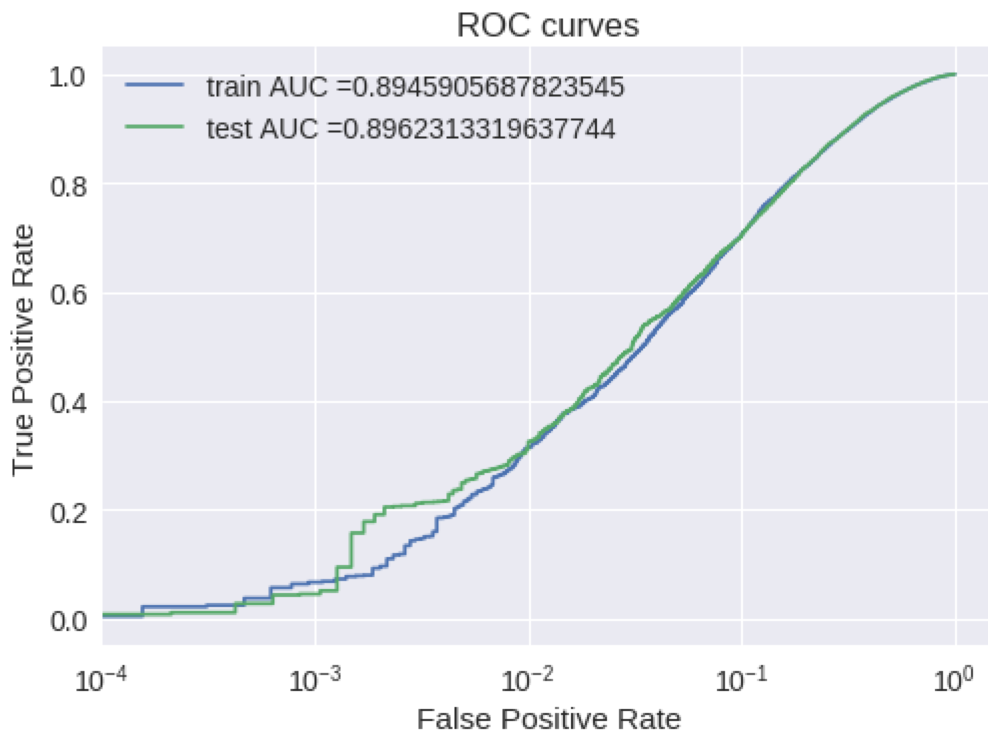
[5.3.1] Applying SVM with L1 regularization on AVG W2V

In [54]:
```
SVM_l1(train_vectors, cv_vectors)
```

```
100%|██████████| 15/15 [00:09<00:00,  1.56it/s]
```

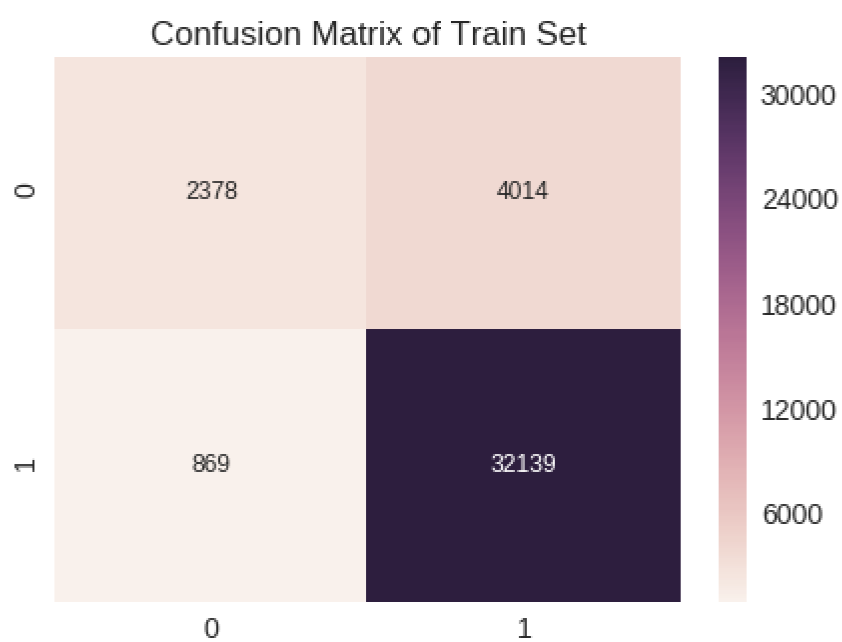
The 'alpha' value 0.0005 with highest roc_auc Score is 89.50262580804944 %

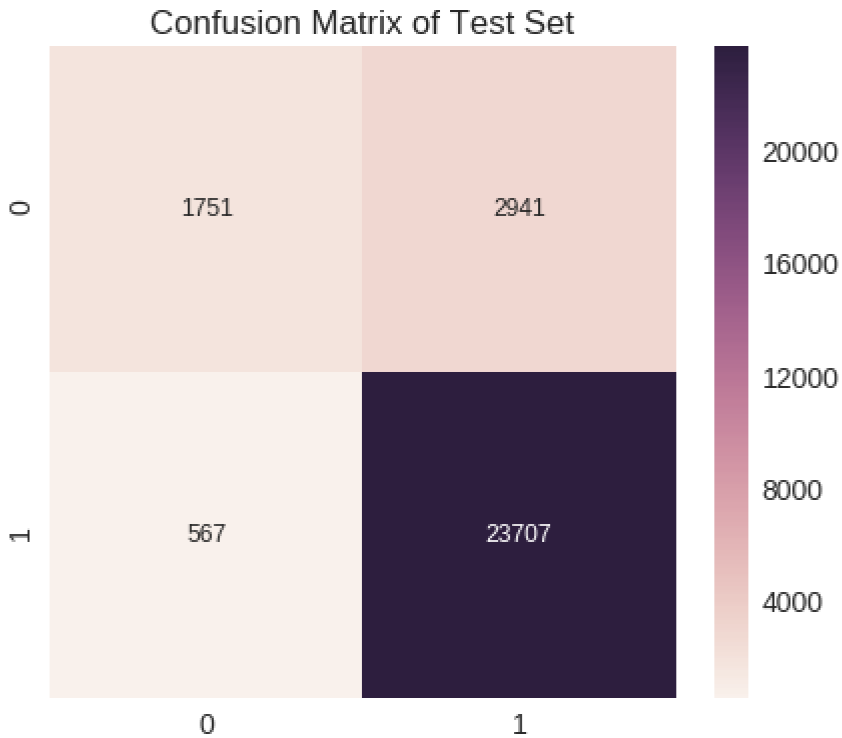
In [55]:
```
testing_l1(train_vectors, test_vectors,0.0005)
```



Precision on test data: 0.8896352446712699
Recall on test data: 0.9766416742193293
F1-Score on test data: 0.9311103255960096

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





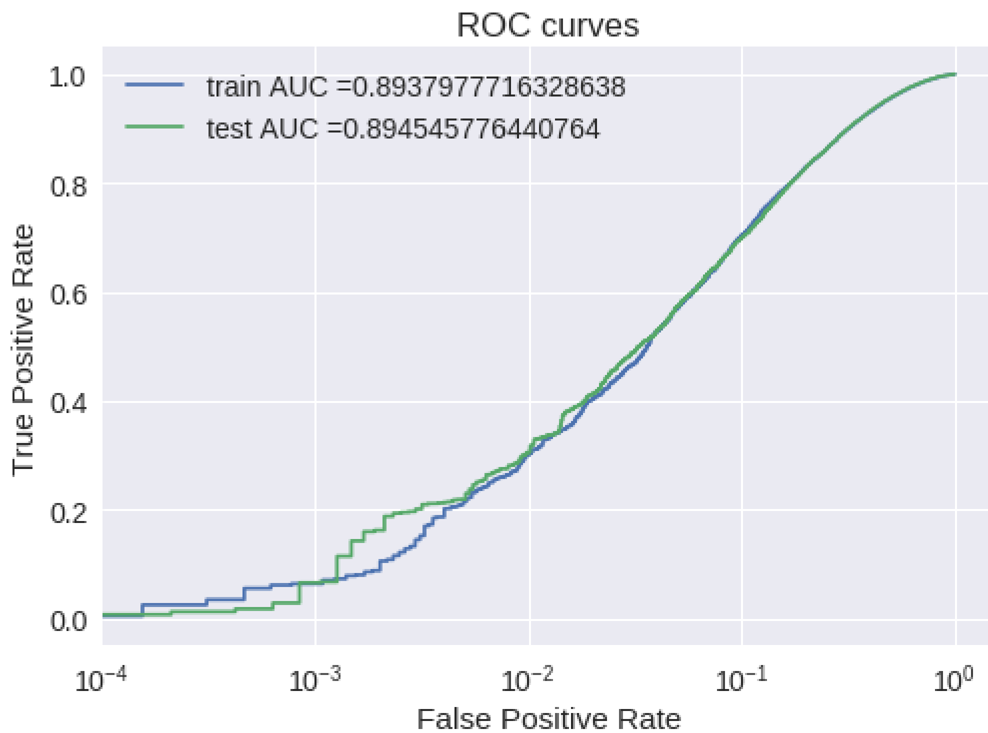
[5.3.2] Applying SVM with L2 regularization on AVG W2V

In [56]:
```
SVM_l2(train_vectors, cv_vectors)
```

100%|██████████| 15/15 [00:07<00:00, 2.10it/s]

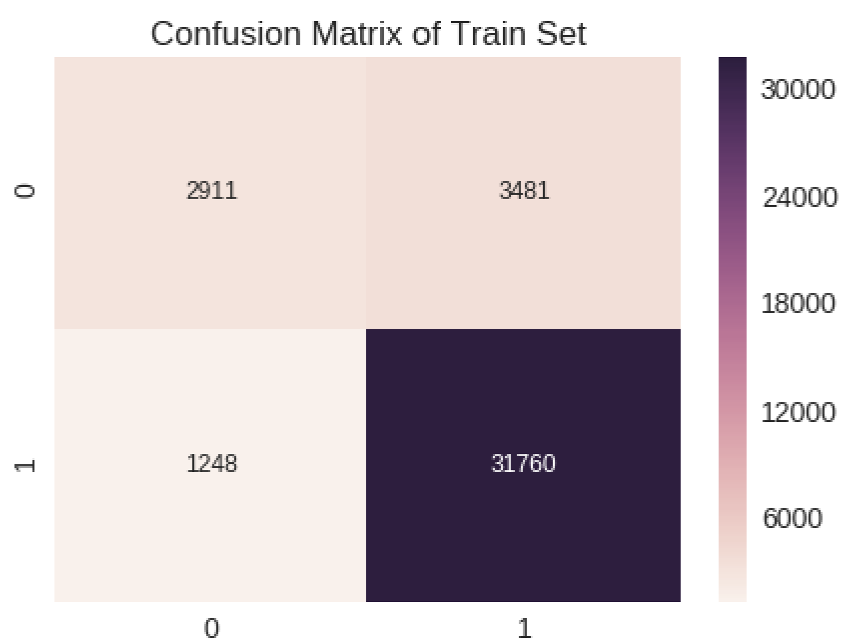
The 'alpha' value 0.005 with highest roc_auc Score is 89.54221634551901 %

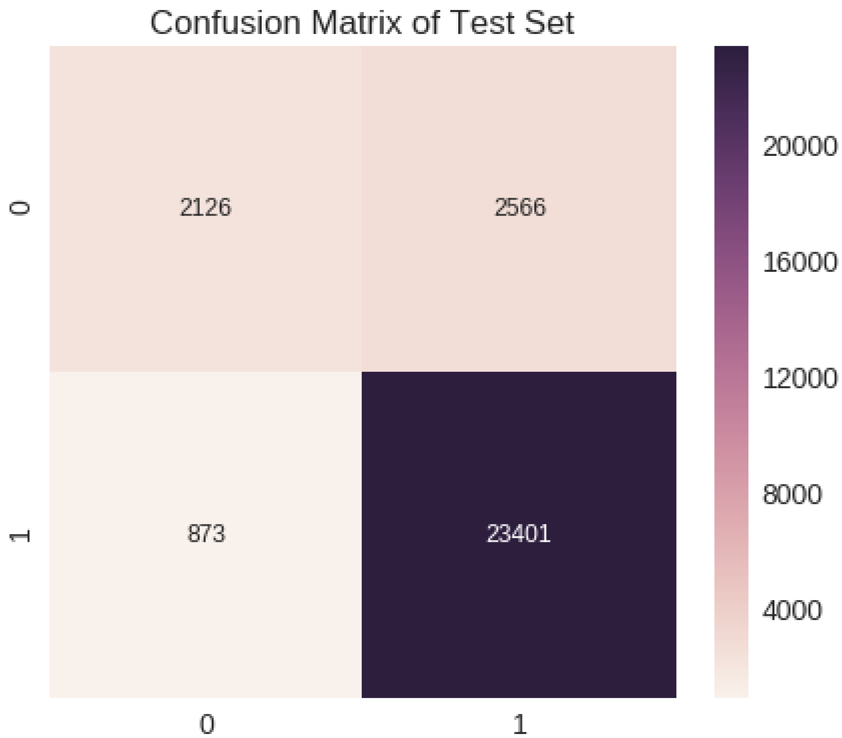
In [57]:
```
testing_l2(train_vectors, test_vectors, 0.001)
```



Precision on test data: 0.901182269803982
Recall on test data: 0.9640355936392848
F1-Score on test data: 0.9315499293405783

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





[5.4] SVM on TFIDF W2V

In [0]:

```
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 [64]:

```
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

train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(w2v_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_train and word in tfidf_feat:
            vec = w2v_model_train.wv[word]
            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
    train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 39400/39400 [12:23<00:00, 53.01it/s]
```

In [60]:
```
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

cv_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(w2v_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_train and word in tfidf_feat:
            vec = w2v_model_train.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
    cv_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 19407/19407 [06:12<00:00, 52.09it/s]
```

In [61]:
```
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

test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(w2v_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_train and word in tfidf_feat:
            vec = w2v_model_train.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
    test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 28966/28966 [09:19<00:00, 51.80it/s]
```

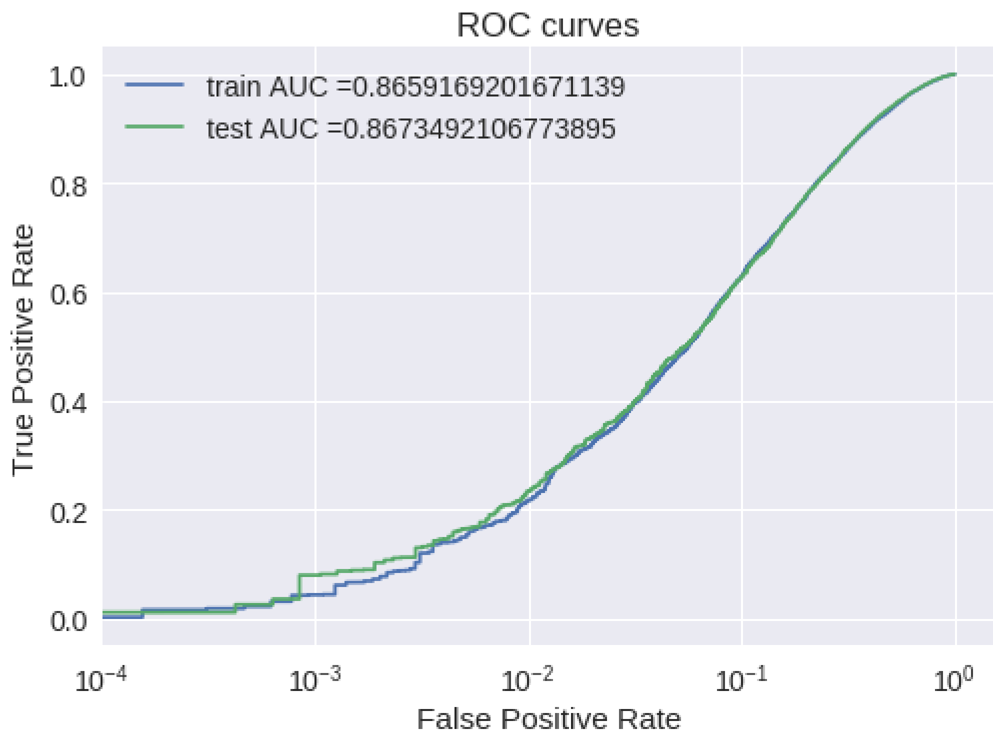
[5.4.1] Applying SVM with L1 regularization on TFIDF W2V

In [62]:
```
SVM_l1(train_tfidf_sent_vectors, cv_tfidf_sent_vectors)
```

100%|██████████| 15/15 [00:09<00:00, 1.56it/s]

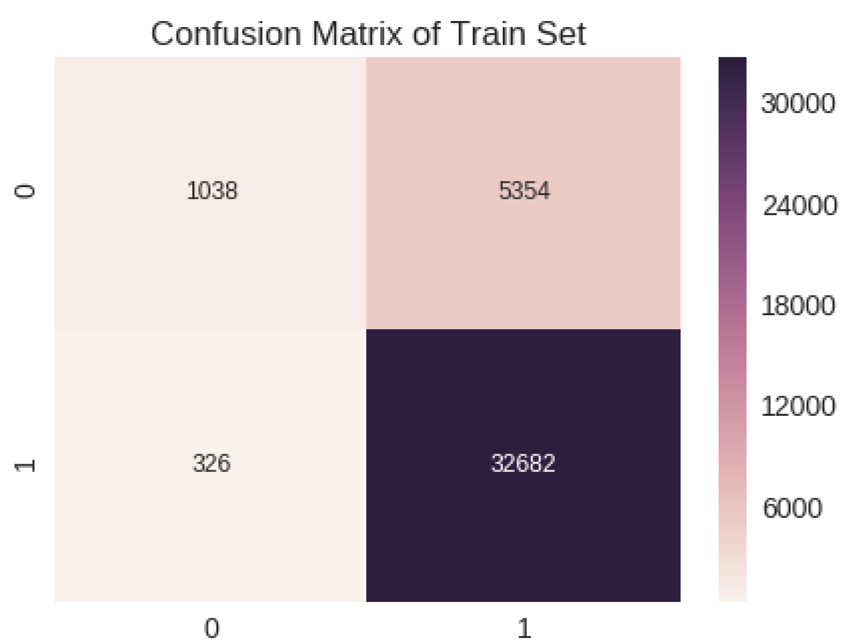
The 'alpha' value 0.0001 with highest roc_auc Score is 86.91512939382608 %

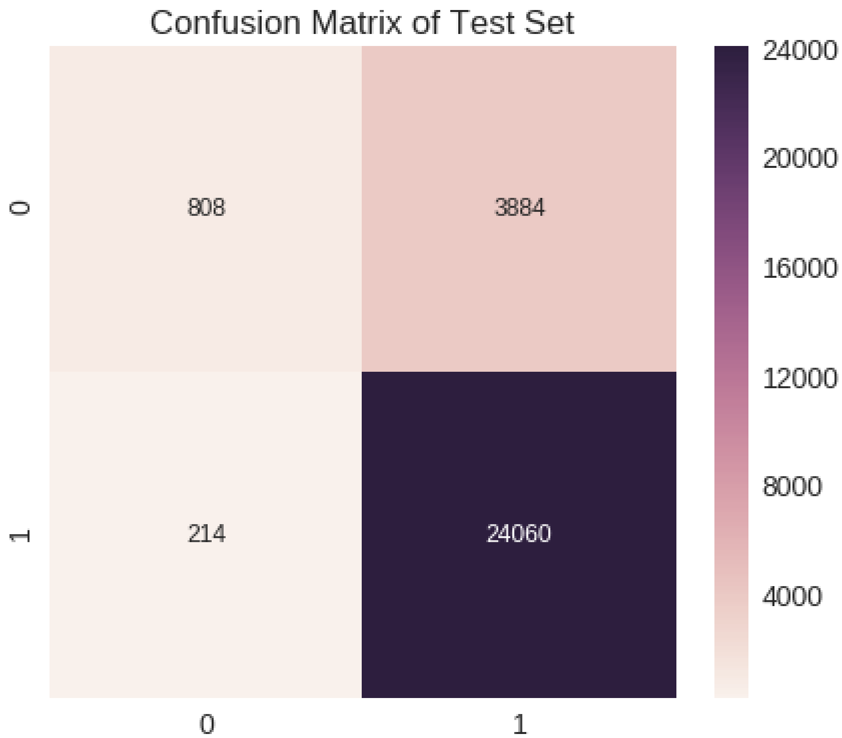
In [63]:
```
testing_l1(train_tfidf_sent_vectors, test_tfidf_sent_vectors,0.001)
```



Precision on test data: 0.8610077297452047
Recall on test data: 0.9911839828623218
F1-Score on test data: 0.9215213144892566

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]



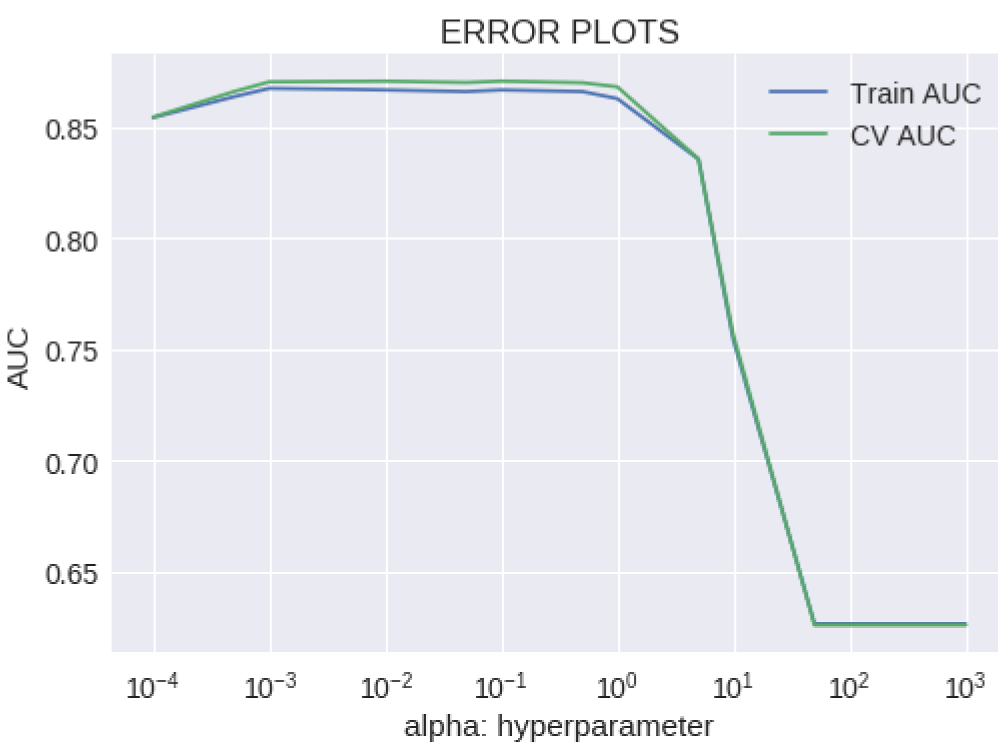


[5.4.2] Applying SVM with L2 regularization on TFIDF W2V,

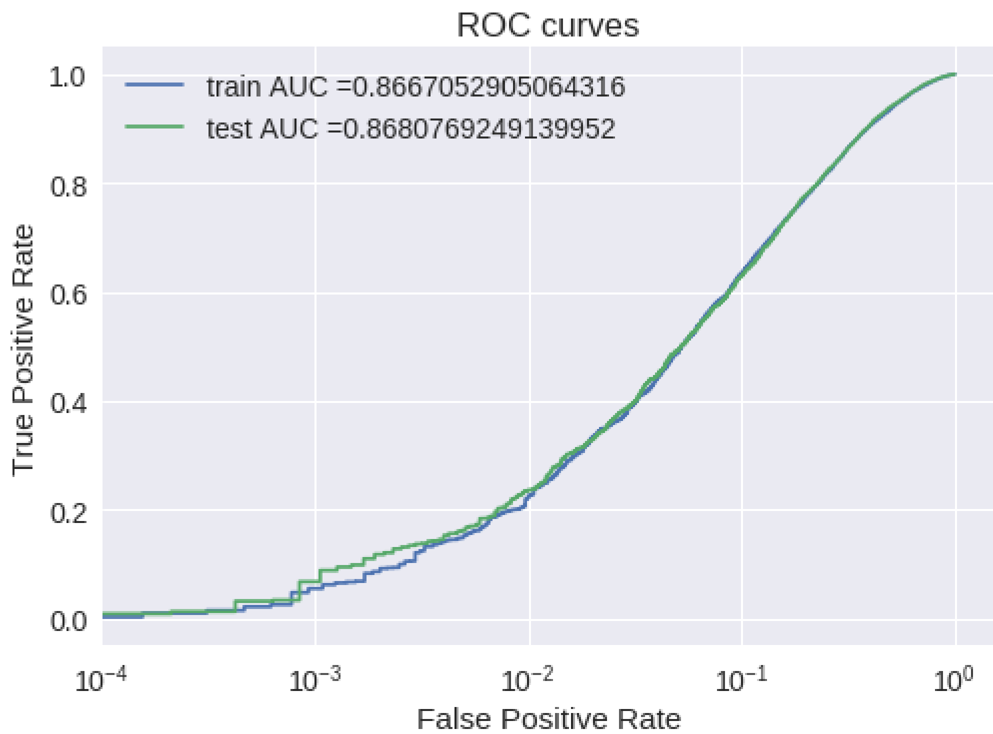
In [65]:
```
SVM_l2(train_tfidf_sent_vectors, cv_tfidf_sent_vectors)
```

```
100%|██████████| 15/15 [00:07<00:00,  2.06it/s]
```

The 'alpha' value 0.1 with highest roc_auc Score is 85.45504074570694 %

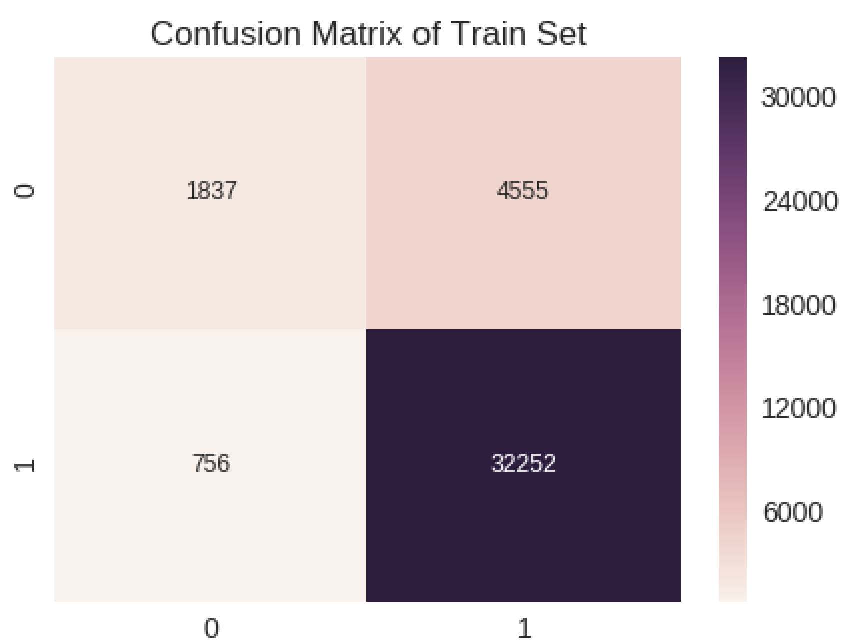


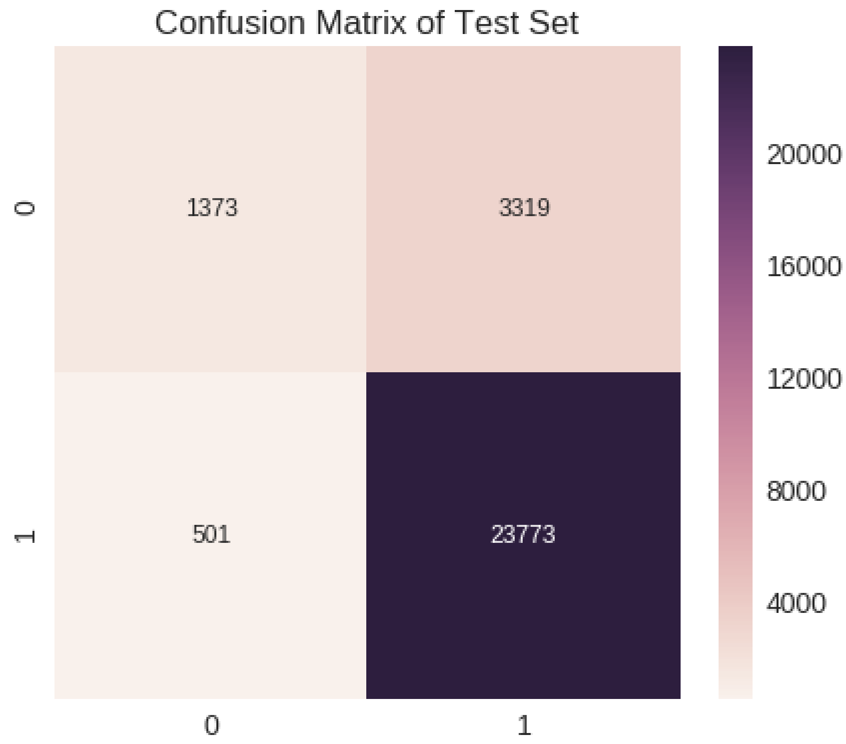
In [66]:
```
testing_l2(train_tfidf_sent_vectors, test_tfidf_sent_vectors,0.001)
```



Precision on test data: 0.8774915104089768
Recall on test data: 0.9793606327758095
F1-Score on test data: 0.9256317408402446

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





In [0]:

[5.2] RBF SVM

In [0]:

```
Total_X_RBF = final['CleanText'].values[:40000]
Total_y_RBF = final['Score'].values[:40000]
```

In [68]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(Total_X_RBF, Total_y_RBF, test_size=0.33)

# split the train data set into cross validation train and cross validation test
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33)


print(f"Train Data : ({len(X_train)} , {len(y_train)})")
print(f"CV Data : ({len(X_cv)} , {len(y_cv)})")
print(f"Test Data : ({len(X_test)} , {len( y_test)})")
```

```
Train Data : (17956 , 17956)
CV Data : (8844 , 8844)
Test Data : (13200 , 13200)
```

In [69]:
```
#BoW
count_vect = CountVectorizer( min_df = 10, max_features = 500)
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[1000:1010])
print('='*50)

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = count_vect.transform(X_train)
X_cv_bow = count_vect.transform(X_cv)
X_test_bow = count_vect.transform(X_test)


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

```
some feature names  []
==================================================
After vectorizations
(17956, 500) (17956,)
(8844, 500) (8844,)
(13200, 500) (13200,)
====================================================================================================
```

Note:

The hyperparameter in SVC with RBF kernel is "C" (Penalty parameter C of the error term). But in this assignment, it was mistakenly replaced with "alpha".

[https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) ([https://scikitlearn.org/stable/modules/generated/sklearn.svm.SVC.html (https://scikitlearn.org/stable/modules/generated/sklearn.svm.SVC.html)](https://scikitlearn.org/stable/modules/generated/sklearn.svm.SVC.html))

In [0]:

```
def SVM_RBF(X_train_reg,X_cv_reg, y_train=y_train,  y_cv=y_cv,  y_test=y_test):
    train_auc = []
    cv_auc = []
    max_alpha=0
    max_roc_auc=-1
    all_alpha = [1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]
    for i in tqdm(all_alpha):

        clf =  SVC(probability=True)
        clf.fit(X_train_reg, y_train)
        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
        # not the predicted outputs

        y_train_pred = clf.predict_proba(X_train_reg)[:,1]
        y_cv_pred = clf.predict_proba(X_cv_reg)[:,1]
        #proba1 =roc_auc_score(y_train,y_train_pred) * float(100)
        proba2 = roc_auc_score(y_cv, y_cv_pred) * float(100)

        if(max_roc_auc<proba2):
            max_roc_auc=proba2
            max_alpha=i
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

    print(f"\nThe 'alpha' value {max_alpha} with highest roc_auc Score is {proba2} %" )
    plt.plot(all_alpha, train_auc, label='Train AUC')
    plt.plot(all_alpha, cv_auc, label='CV AUC')
    plt.xscale(value = 'log')
    plt.legend()
    plt.xlabel("alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
```

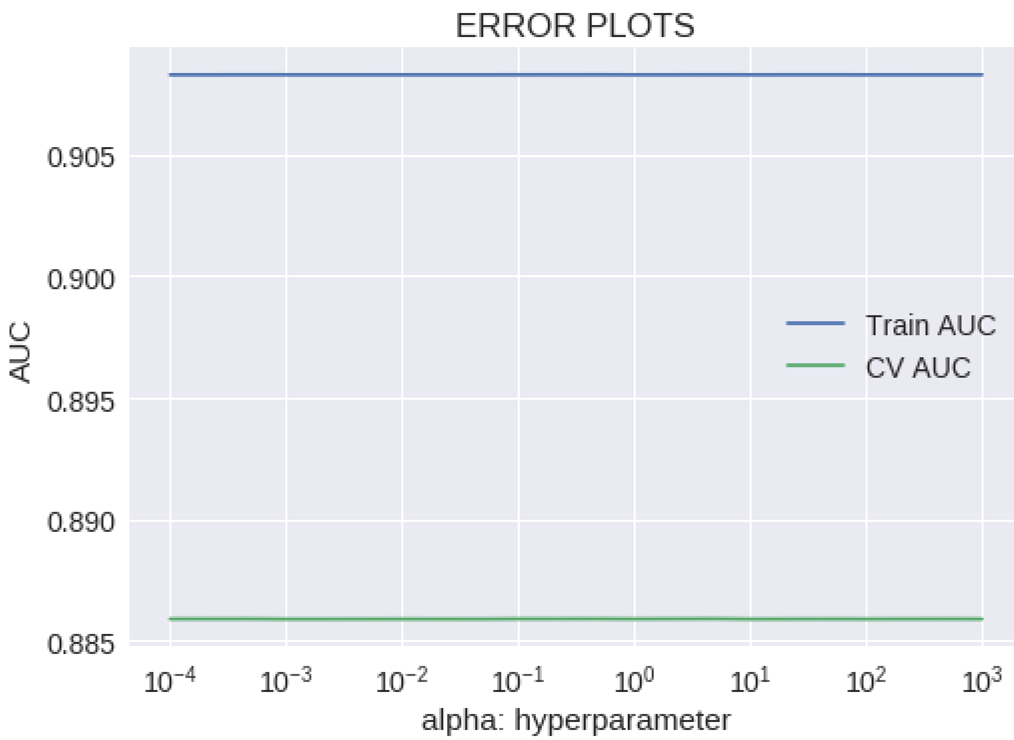
In [0]:
```
def testing(X_train_reg,X_test_reg, max_alpha, y_train=y_train,  y_test=y_test):
    clf =  SVC(C=10,probability=True)
    clf.fit(X_test_reg, y_test)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

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

    print(f"--> {train_fpr}")
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.xscale(value = 'log')
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()

    print(f"Precision on test data: {precision_score(y_test, clf.predict(X_test_reg))}")
    print(f"Recall on test data: {recall_score(y_test, clf.predict(X_test_reg))}")
    print(f"F1-Score on test data: {f1_score(y_test, clf.predict(X_test_reg))}")

    print("\nConfusion Matrix of Train and Test set:\n [ [TN  FP]\n   [FN TP] ]\n")
    confusionMatrix_train=confusion_matrix(y_train, clf.predict(X_train_reg))
    confusionMatrix_test=confusion_matrix(y_test, clf.predict(X_test_reg))
    df_cm_tr = pd.DataFrame(confusionMatrix_train, range(2),range(2))
    df_cm_te = pd.DataFrame(confusionMatrix_test, range(2),range(2))
    plt.figure(figsize = (7,5))
    plt.ylabel("Predicted label")
    plt.xlabel("Actual label")
    plt.title("Confusion Matrix")
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm_tr, annot=True,annot_kws={"size": 12},fmt="d")


    plt.figure(figsize = (7,6))
    plt.ylabel("Predicted label")
    plt.xlabel("Actual label")
    plt.title("Confusion Matrix")
    sns.heatmap(df_cm_te, annot=True,annot_kws={"size": 12},fmt="d")
```

[5.2.1] Applying RBF SVM on BOW, SET 1

In [72]:
```
SVM_RBF(X_train_bow,X_cv_bow )
```

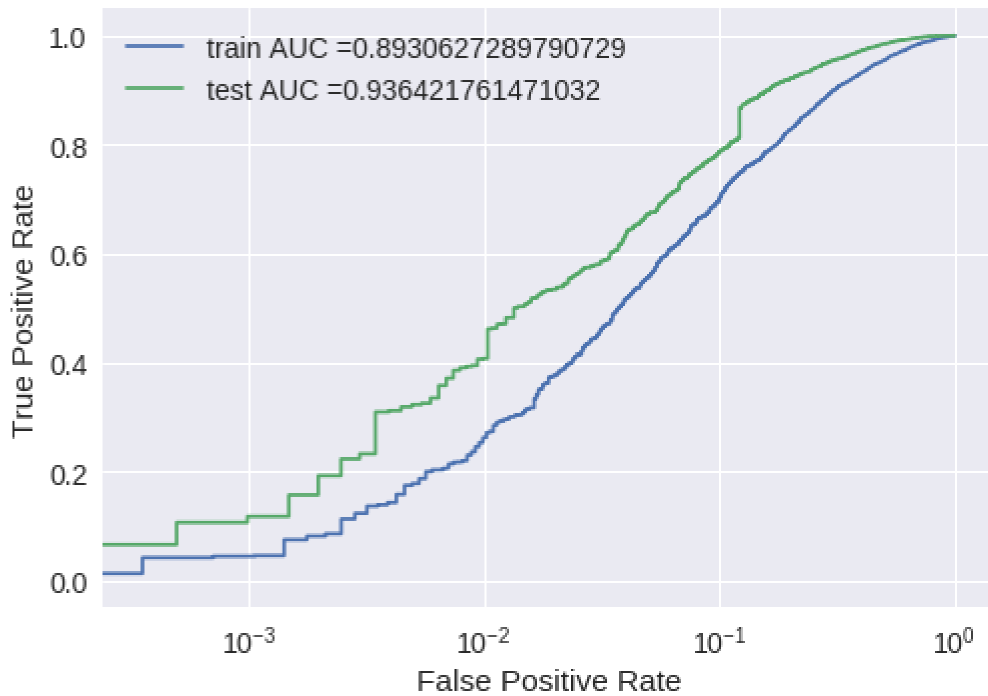
100%|██████████| 15/15 [40:16<00:00, 162.07s/it]

The 'alpha' value 5 with highest roc_auc Score is 88.5902066349429 %



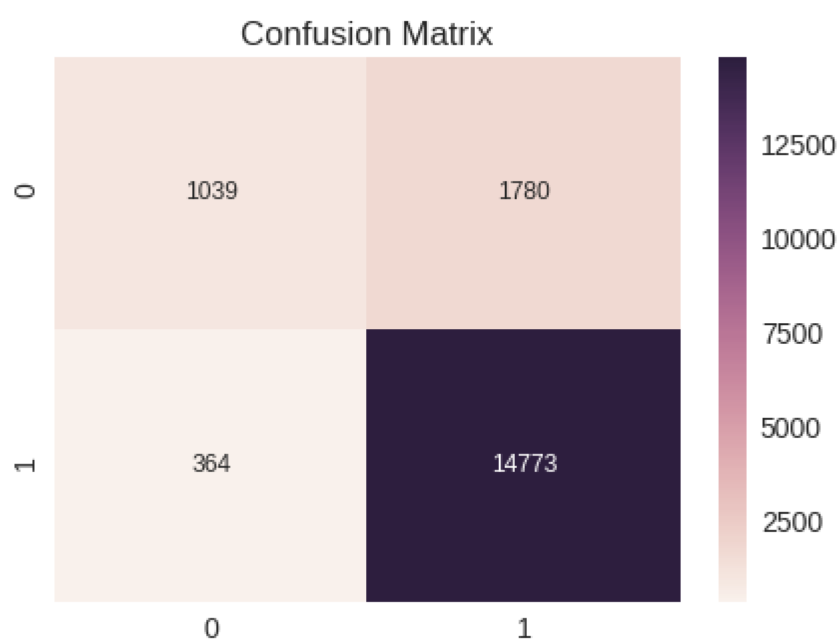
In [97]:
```
testing(X_train_bow,X_test_bow, 5 )
```

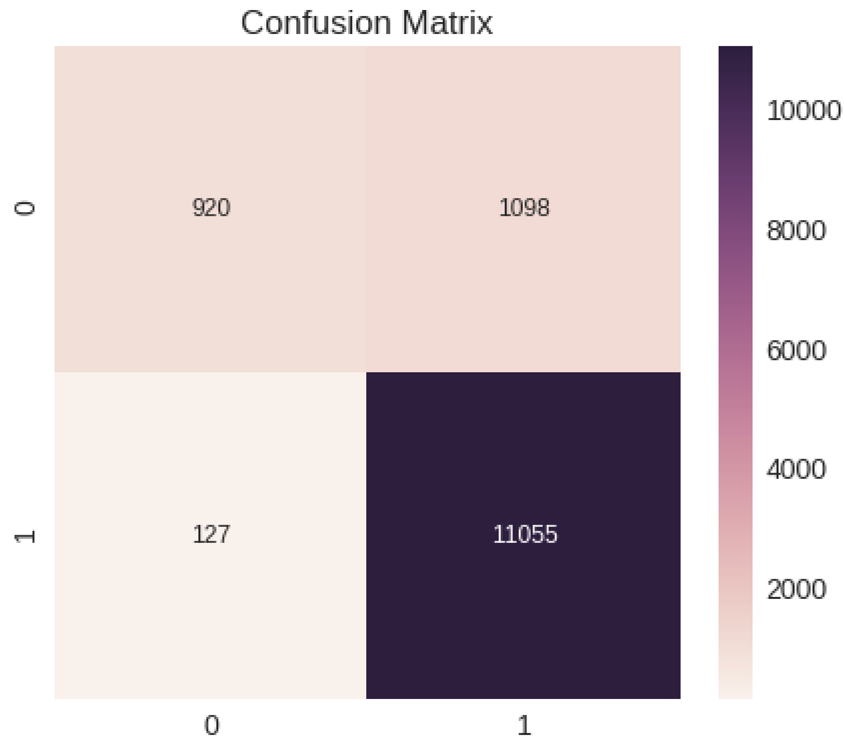
--> [0.   0.   0.   ... 0.99964526 0.99964526 1.   ]



Precision on test data: 0.9096519377931375
Recall on test data: 0.9886424610981935
F1-Score on test data: 0.9475037497321619

Confusion Matrix of Train and Test set:
 [[TN FP]
   [FN TP]]





[5.2.2] Applying RBF SVM on TFIDF, SET 2

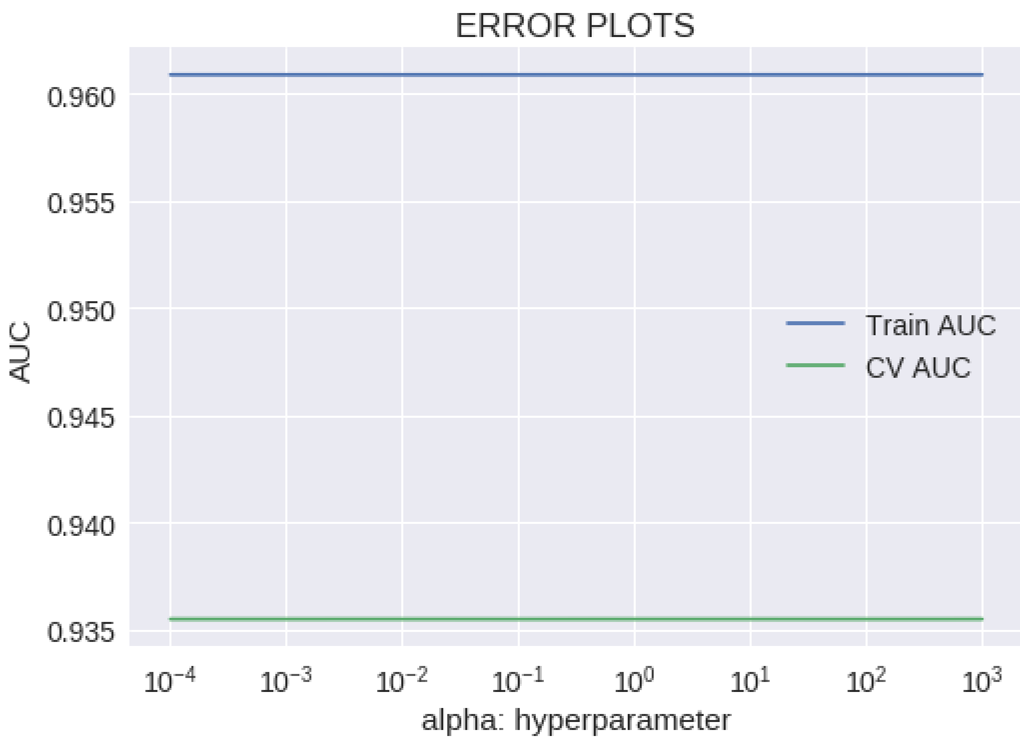
In [93]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)


# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_idf = tf_idf_vect.transform(X_train)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
X_test_tf_idf = tf_idf_vect.transform(X_test)


print("After vectorizations")
print(X_train_tf_idf.shape, y_train.shape)
print(X_cv_tf_idf.shape, y_cv.shape)
print(X_test_tf_idf.shape, y_test.shape)
print("="*100)
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able drink', 'able eat', 'able enj
oy', 'able find', 'able get', 'able make', 'able order']
==================================================
After vectorizations
(17956, 10488) (17956,)
(8844, 10488) (8844,)
(13200, 10488) (13200,)
==========================================================================================
==========
```

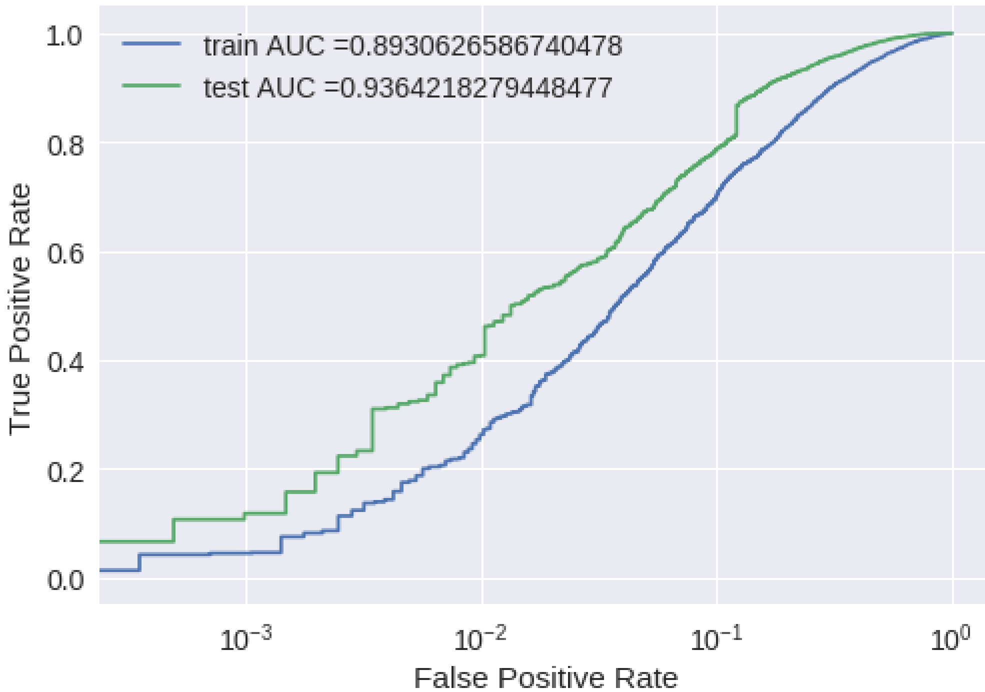
In [94]:
```
SVM_RBF(X_train_tf_idf,X_cv_tf_idf )
```

100%|██████████| 15/15 [1:05:01<00:00, 260.56s/it]

The 'alpha' value 0.005 with highest roc_auc Score is 93.55012237337878 %

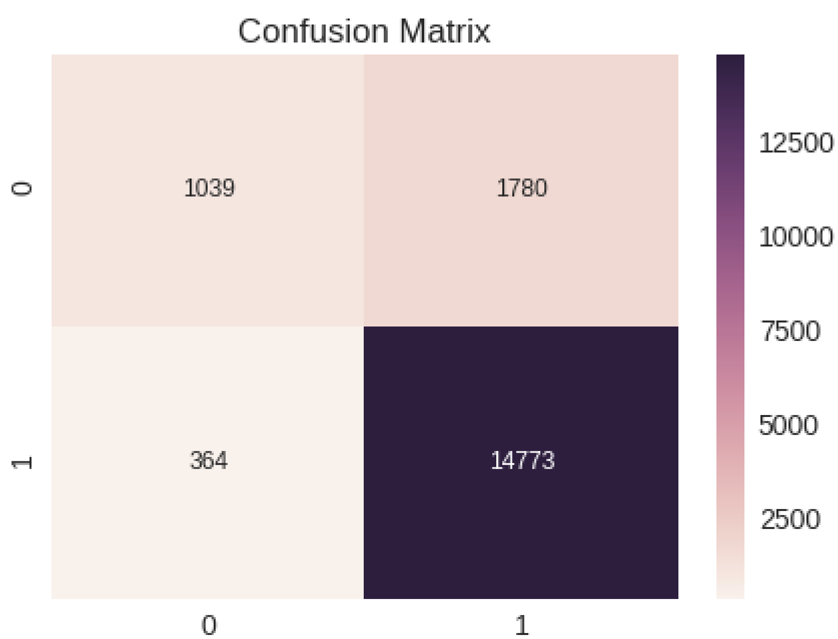


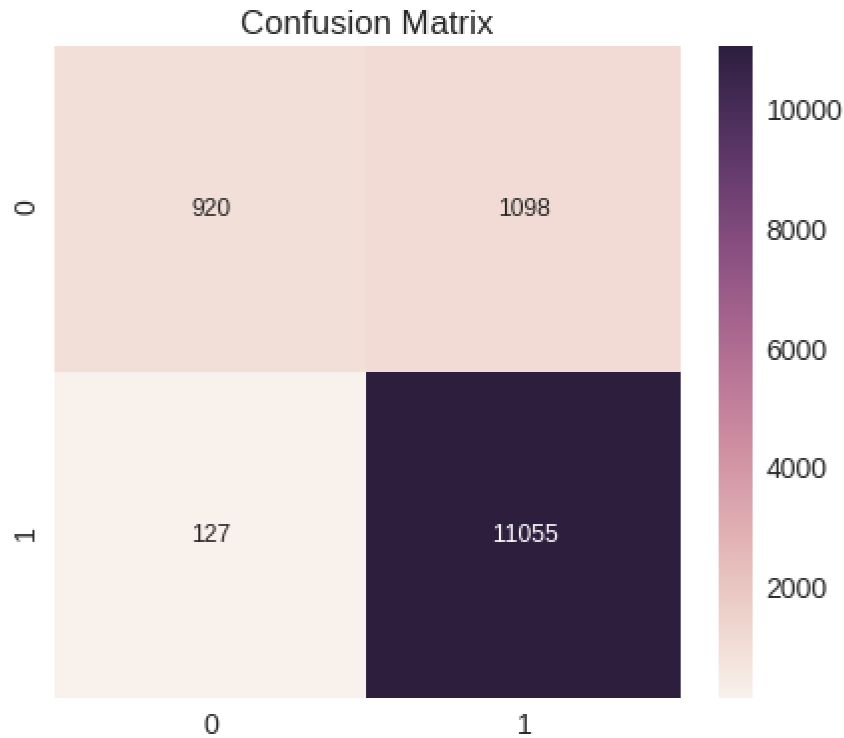
In [95]:
```
testing(X_train_bow,X_test_bow,0.005)
```



Precision on test data: 0.9096519377931375
Recall on test data: 0.9886424610981935
F1-Score on test data: 0.9475037497321619

Confusion Matrix of Train and Test set:
 [[TN FP]
   [FN TP]]





In [0]:

```

```

[4.4] Word2Vec

In [0]:

```
i=0

w2v_train=[]
w2v_cv=[]
w2v_test=[]

for sentance in X_train:
    w2v_train.append(sentance.split())

for sentance in X_cv:
    w2v_cv.append(sentance.split())


for sentance in X_test:
    w2v_test.append(sentance.split())
```

In [78]:
```
want_to_train_w2v = True
if want_to_train_w2v:
# min_count = 5 considers only words that occured atleast 5 times
#w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    w2v_model_train = Word2Vec(w2v_train,min_count=5,size=50, workers=4)
    print(w2v_model_train.wv.most_similar('great'))
    print('='*50)
else:
    pass
```

```
[('good', 0.7806708216667175), ('excellent', 0.7718941569328308), ('perfect', 0.7358829379081726), ('fanta
stic', 0.733282744884491), ('wonderful', 0.7289205193519592), ('amazing', 0.7206151485443115), ('aweso
me', 0.7150614857673645), ('delicious', 0.6816601753234863), ('especially', 0.6756676435470581), ('terrifi
c', 0.6631010174751282)]
==================================================
```

In [79]:
```
w2v_words_train = list(w2v_model_train.wv.vocab)

print("number of words that occured minimum 5 times ",len(w2v_words_train ))
print("sample words ", w2v_words_train[0:50])
```

```
number of words that occured minimum 5 times  8063
sample words  ['wish', 'would', 'use', 'xylitol', 'stevia', 'instead', 'sugar', 'make', 'healthier', 'delicious', 'probabl
y', 'soda', 'though', 'recommend', 'occasional', 'sweet', 'treat', 'addictive', 'earthy', 'buttery', 'fruity', 'peppery',
'grassy', 'looking', 'something', 'bitter', 'might', 'want', 'keep', 'found', 'olive', 'oil', 'overwhelmingly', 'acrid', 'rea
l', 'disappointment', 'understand', 'taste', 'profile', 'good', 'include', 'slight', 'bitterness', 'least', 'fall', 'bottling',
'note', 'ordered', 'family', 'could']
```

#Converting text into vectors using Avg W2V, TFIDF-W2V

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

[4.4.1.1] Avg W2v

In [80]:

```
train_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(w2v_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 us
    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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
print()
print(len(train_vectors))
print(len(train_vectors[0]))
```

```
100%|██████████| 17956/17956 [00:29<00:00, 609.54it/s]


17956
50
```

In [81]:

```
# compute average word2vec for each review.
cv_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(w2v_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 us
    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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)
print()
print(len(cv_vectors))
print(len(cv_vectors[0]))
```

```
100%|██████████| 8844/8844 [00:15<00:00, 572.29it/s]


8844
50
```

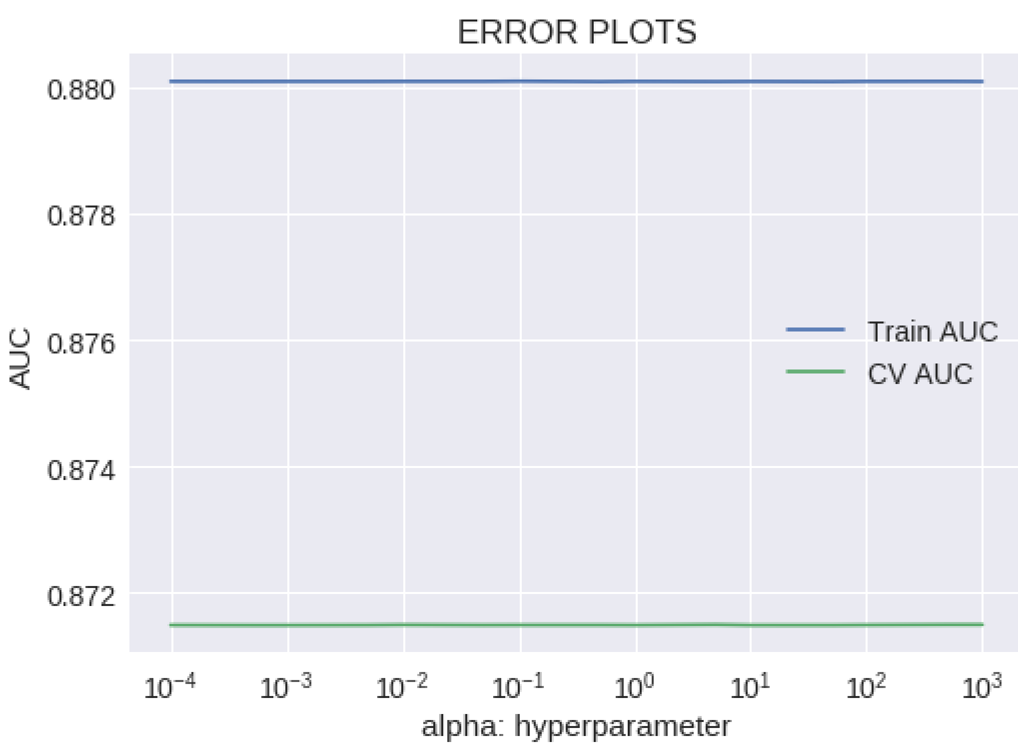
In [82]:
```
test_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(w2v_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you us
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train:
            vec =  w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)
print()
print(len(test_vectors))
print(len(test_vectors[0]))
```

```
100%|██████████| 13200/13200 [00:21<00:00, 609.19it/s]


13200
50
```

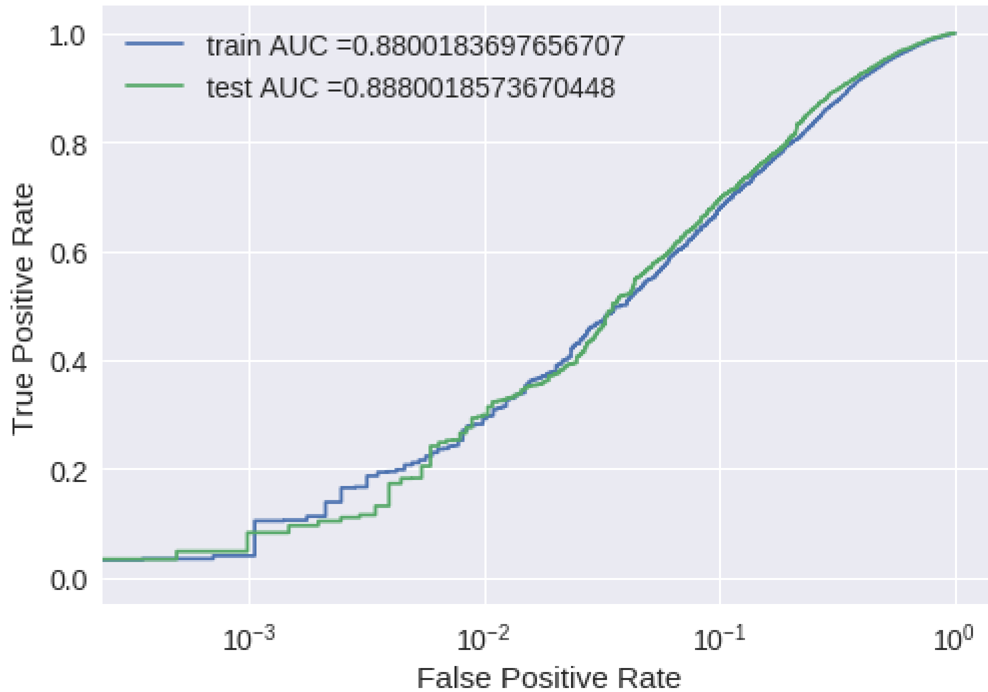
In [83]:
```
SVM_RBF(train_vectors,cv_vectors )
```

```
100%|██████████| 15/15 [19:39<00:00, 78.67s/it]
```

The 'alpha' value 5 with highest roc_auc Score is 87.14798422954154 %

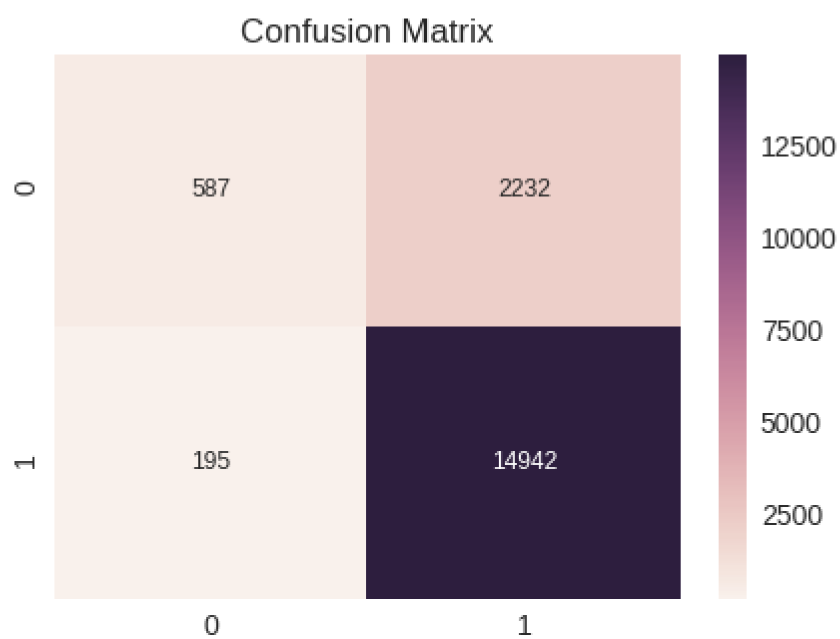


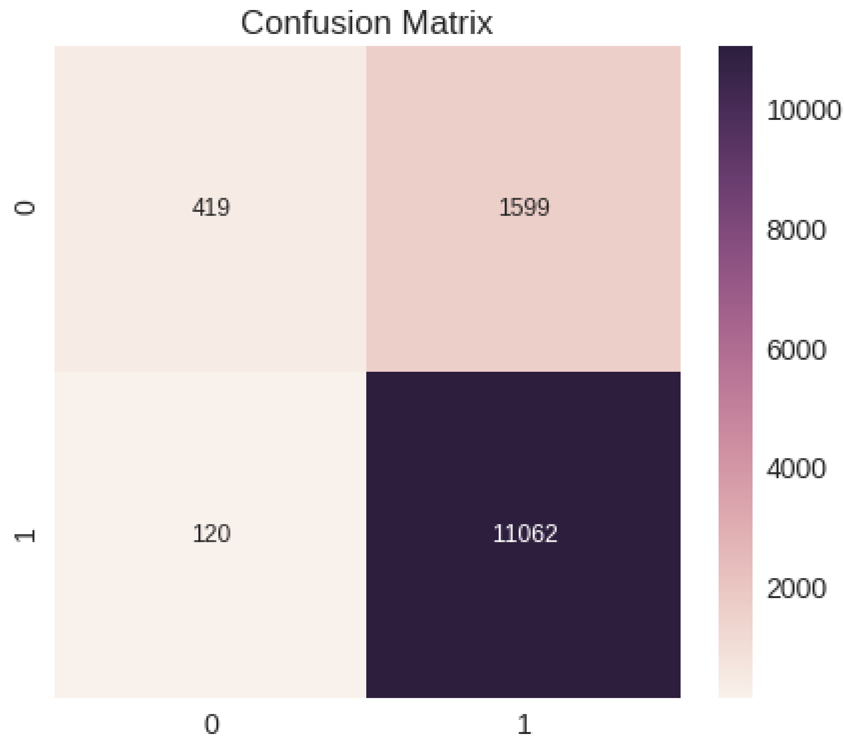
In [84]:
```
testing(train_vectors,test_vectors,5)
```



Precision on test data: 0.873706658241845
Recall on test data: 0.9892684671793954
F1-Score on test data: 0.927903367864782

Confusion Matrix of Train and Test set:
 [[TN FP]
   [FN TP]]





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

In [0]:

```
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 [86]:

```
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

train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(w2v_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_train and word in tfidf_feat:
      vec = w2v_model_train.wv[word]
      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
  train_tfidf_sent_vectors.append(sent_vec)
  row += 1
```

```
100%|██████████| 17956/17956 [04:17<00:00, 69.72it/s]
```

In [87]:
```
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

cv_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(w2v_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_train and word in tfidf_feat:
            vec = w2v_model_train.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
    cv_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 8844/8844 [02:08<00:00, 68.60it/s]
```

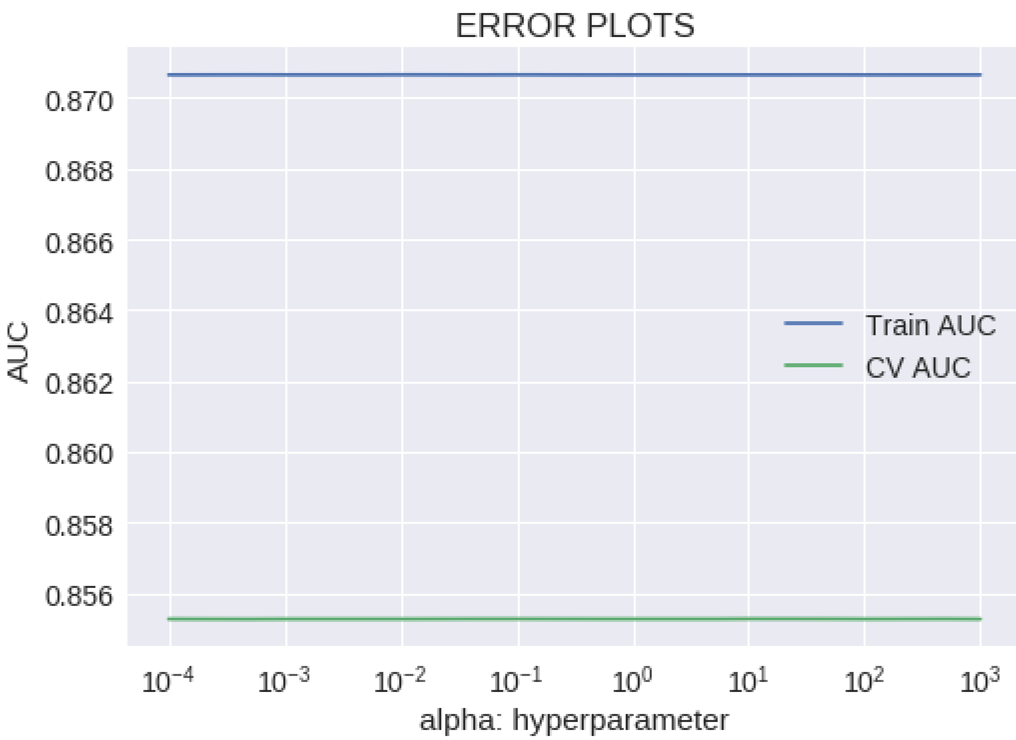
In [88]:
```
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(w2v_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_train and word in tfidf_feat:
            vec = w2v_model_train.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
    test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 13200/13200 [03:07<00:00, 70.24it/s]
```

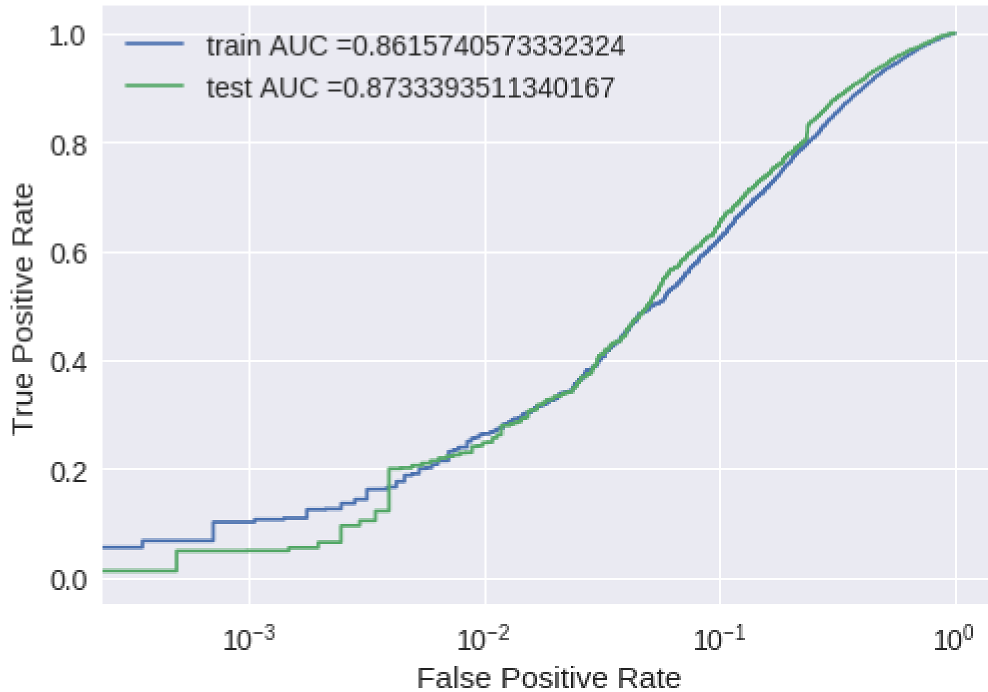
In [89]:
```
SVM_RBF(train_tfidf_sent_vectors,cv_tfidf_sent_vectors )
```

100%|██████████| 15/15 [21:09<00:00, 84.76s/it]

The 'alpha' value 10 with highest roc_auc Score is 85.52656545047952 %

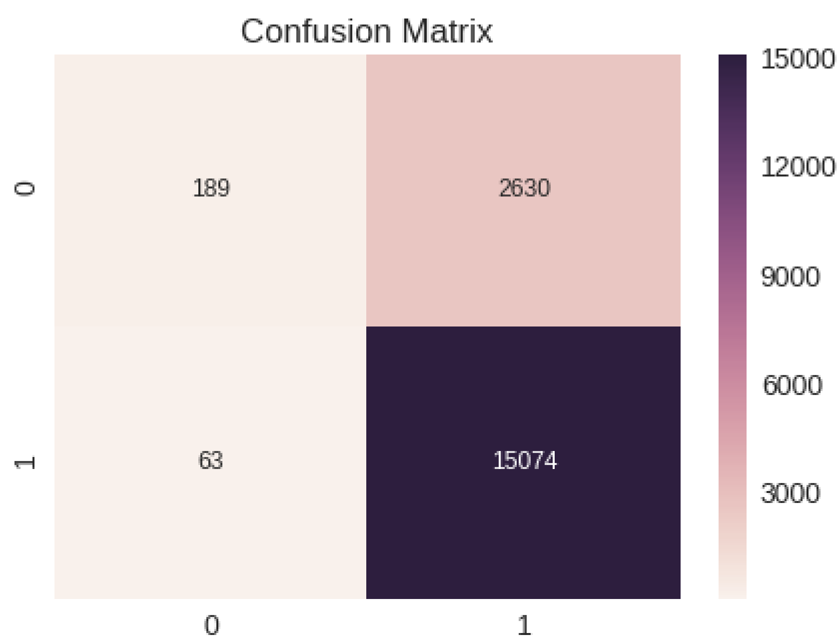


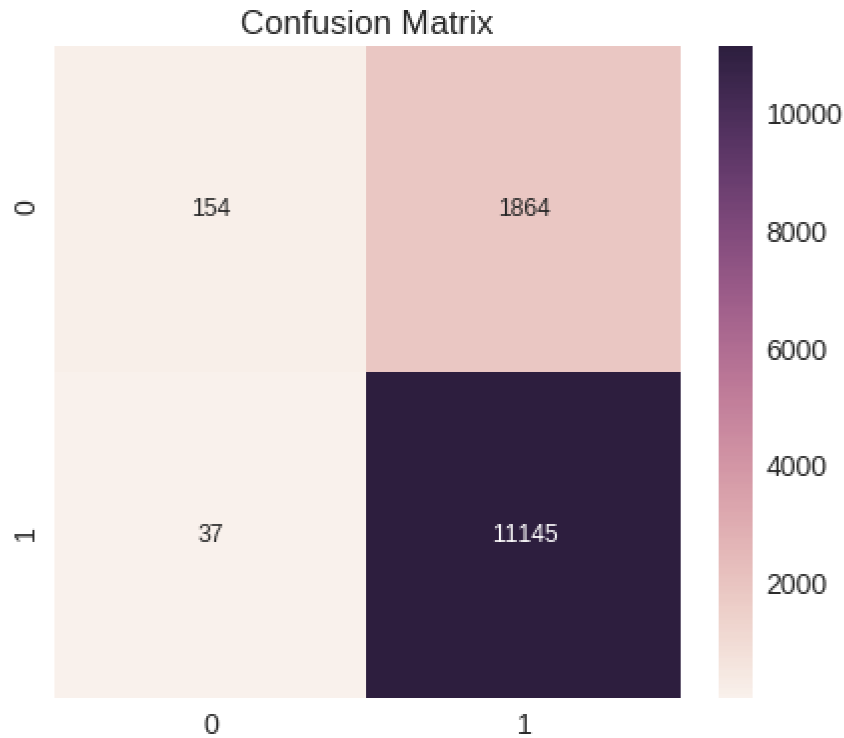
In [90]:
```
testing(train_tfidf_sent_vectors,test_tfidf_sent_vectors,10)
```



Precision on test data: 0.8567145822123146
Recall on test data: 0.9966911107136469
F1-Score on test data: 0.9214170559298912

Confusion Matrix of Train and Test set:
[[TN FP]
[FN TP]]





[6] Conclusions

In [108]:
```
from prettytable import PrettyTable

x = PrettyTable(["Vectorizer", "L1 (alpha)", "Test AUC (L1)", "L2 (alpha)", "Test AUC(L2)"])
y = PrettyTable(["Vectorizer", "alpha", "Test AUC"])

x.title = "Linear Kernel"
x.add_row(["BoW","0.0001" ,"0.92", "0.001", 0.88])
x.add_row(["Tf-Idf","0.0001" ,"0.93",  "0.001", 0.89])
x.add_row(["AVG_W2V","0.0005" ,"0.89", "0.005", 0.89])
x.add_row(["TFIDF_W2V","0.0001" ,"0.86", "0.1", 0.86])

print(x)

y.title = "RBF Kernel"
y.add_row(["BoW","5" ,"0.93"])
y.add_row(["Tf-Idf","0.005" ,"0.93"])
y.add_row(["AVG_W2V","5" ,"0.89"])
y.add_row(["TFIDF_W2V","10" ,"0.87"])

print(y)
```

```
+------------+------------+---------------+------------+--------------+
| Vectorizer | L1 (alpha) | Test AUC (L1) | L2 (alpha) | Test AUC(L2) |
+------------+------------+---------------+------------+--------------+
|    BoW     |   0.0001   |      0.92     |   0.001    |     0.88     |
|   Tf-Idf   |   0.0001   |      0.93     |   0.001    |     0.89     |
|  AVG_W2V   |   0.0005   |      0.89     |   0.005    |     0.89     |
| TFIDF_W2V  |   0.0001   |      0.86     |    0.1     |     0.86     |
+------------+------------+---------------+------------+--------------+
+------------+-------+----------+
| Vectorizer | alpha | Test AUC |
+------------+-------+----------+
|    BoW     |   5   |   0.93   |
|   Tf-Idf   | 0.005 |   0.93   |
|  AVG_W2V   |   5   |   0.89   |
| TFIDF_W2V  |   10  |   0.87   |
+------------+-------+----------+
```

Test Prob.(unseen data) using:L1 and L2 regularization

Linear Kernel: Tf-idf has predicted 93% accurate on test data using L1 regularization and 89% on L2 regularization.

RBF Kernel: Both BoW and Tf-idf has predicted the highest accurate on test data i.e 93%

In [0]: