07 Amazon Fine Food Reviews Analysis_Support Vector Machines

March 18, 2019

1 Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

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

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

2 [1]. Reading Data

2.1 [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
import sys

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
```

In [2]:
```
# using SQLite Table to read data.
```

```
con = sqlite3.connect(os.path.join( os.getcwd(), '..', '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
# for tsne assignment you can take 5k data points

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

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

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

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

Out[2]:

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

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

In [3]:

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

```
        HAVING COUNT(*)>1
        """, con)
```

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

```
(80668, 7)
```

```
Out[4]:                 UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

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

```
Out[5]:               UserId   ProductId                       ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

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

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

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

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

```
Out[7]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        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

           HelpfulnessDenominator  Score        Time  \
        0                       2      5  1199577600
        1                       2      5  1199577600
        2                       2      5  1199577600
        3                       2      5  1199577600
        4                       2      5  1199577600

                                     Summary  \
        0  LOACKER QUADRATINI VANILLA WAFERS
        1  LOACKER QUADRATINI VANILLA WAFERS
        2  LOACKER QUADRATINI VANILLA WAFERS
        3  LOACKER QUADRATINI VANILLA WAFERS
        4  LOACKER QUADRATINI VANILLA WAFERS

                                                        Text
        0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

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

```
In [8]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

Out[9]: (364173, 10)

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

Out[10]: 69.25890143662969

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

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

display.head()
```

Out[11]:
```
      Id   ProductId          UserId             ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                     Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                    Summary  \
0             Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                          Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

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

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

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

```
(364171, 10)
```

Out[13]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

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

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

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

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

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

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         from bs4 import BeautifulSoup

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

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
```

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

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

             # general
             phrase = re.sub(r"n\'t", " not", phrase)
```

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

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

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only tl
==================================================
```

```
In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
```

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

```
Great ingredients although chicken should have been 1st rather than chicken broth the only thi
```

```
In [21]: # https://gist.github.com/sebleier/554280
         # we are removing the words from the stop words list: 'no', 'nor', 'not'
         # <br /><br /> ==> after the above steps, we are getting "br br"
         # we are including them into stop words list
         # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

         stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
                     "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
                     'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
                     'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'l
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
                     'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
                     's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
                     've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
```

```
                "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                'won', "won't", 'wouldn', "wouldn't"])
```

5 Applying SVM

[3.2] Preprocessing Review Text and Summary

```
In [22]: preprocessed_reviews = []
         preprocessed_summary = []
         # Sampling the data and preprocessing
         def data_sampling_preprocessing(final, no_of_samples):

             final = final.sample(n=no_of_samples)

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

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

                 preprocessed_summary.append(summary.strip())


             final['CleanedText'] = preprocessed_reviews #adding a column of CleanedText which
             final['CleanedText'] = final['CleanedText'].astype('str')
```

```
    final['CleanedSummary'] = preprocessed_summary #adding a column of CleanedSummary
    final['CleanedSummary'] = final['CleanedSummary'].astype('str')

    final['Text_Summary'] = final['CleanedSummary'] + final['CleanedText']

    # # store final table into an SQlLite table for future.
    # conn = sqlite3.connect('final.sqlite')
    # c=conn.cursor()
    # conn.text_factory = str
    # final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
    #              index=True, index_label=None, chunksize=None, dtype=None)
    # conn.close()

    return final['Text_Summary'], final['Score']
```

6 [4] Featurization

6.1 [4.1] BAG OF WORDS

In [23]:
```
# #BoW
# count_vect = CountVectorizer() #in scikit-learn
# count_vect.fit(preprocessed_reviews)
# print("some feature names ", count_vect.get_feature_names()[:10])
# print('='*50)

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

6.2 [4.2] Bi-Grams and n-Grams.

In [24]:
```
# #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/m

# # 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_bigr
```

6.3 [4.3] TF-IDF

In [25]:
```
# tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
# tf_idf_vect.fit(preprocessed_reviews)
# print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_na
# print('='*50)

# final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
# print("the type of count vectorizer ",type(final_tf_idf))
# print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
# print("the number of unique words including both unigrams and bigrams ", final_tf_i
```

6.4 [4.4] Word2Vec

In [26]:
```
# # Train your own Word2Vec model using your own text corpus
# i=0
# list_of_sentance=[]
# for sentance in preprocessed_reviews:
#     list_of_sentance.append(sentance.split())
```

In [27]:
```
# # Using Google News Word2Vectors

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


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

# is_your_ram_gt_16g=False
# want_to_use_google_w2v = False
# want_to_train_w2v = True

# if want_to_train_w2v:
#     # min_count = 5 considers only words that occured atleast 5 times
#     w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
#     print(w2v_model.wv.most_similar('great'))
#     print('='*50)
#     print(w2v_model.wv.most_similar('worst'))

# elif want_to_use_google_w2v and is_your_ram_gt_16g:
#     if os.path.isfile('GoogleNews-vectors-negative300.bin'):
```

```
#         w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300
#         print(w2v_model.wv.most_similar('great'))
#         print(w2v_model.wv.most_similar('worst'))
#     else:
#         print("you don't have gogole's word2vec file, keep want_to_train_w2v = True
```

In [28]:
```
# 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])
```

6.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [29]:
```
# # average Word2Vec
# # compute average word2vec for each review.
# sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
# for sent in tqdm(list_of_sentance): # for each review/sentence
#     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need
#     cnt_words =0; # num of words with a valid vector in the sentence/review
#     for word in sent: # for each word in a review/sentence
#         if word in w2v_words:
#             vec = w2v_model.wv[word]
#             sent_vec += vec
#             cnt_words += 1
#     if cnt_words != 0:
#         sent_vec /= cnt_words
#     sent_vectors.append(sent_vec)
# print(len(sent_vectors))
# print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

In [30]:
```
# # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
# model = TfidfVectorizer()
# tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# # we are converting a dictionary with word as a key, and the idf as a value
# dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [31]:
```
# # 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 = tfi

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

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

7 [5] Assignment 7: SVM

```
<li><strong>Apply SVM on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Procedure</strong>
    <ul>
<li>You need to work with 2 versions of SVM
    <ul><li>Linear kernel</li>
        <li>RBF kernel</li></ul>
<li>When you are working with linear kernel, use SGDClassifier with hinge loss because it is co
<li>When you are working with SGDClassifier with hinge loss and trying to find the AUC
    score, you would have to use <a href='https://scikit-learn.org/stable/modules/generated/skl
<li>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.

```
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penal
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
```

```
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>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.

```
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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.

8 Applying SVM

In [32]:
```
# Source: https://docs.python.org/3/library/pickle.html

# Saving data to pickle file
def topicklefile(obj, file_name):
    pickle.dump(obj,open(file_name+'.pkl', 'wb'))
```

In [33]:
```
# Data from pickle file
def frompicklefile(file_name):
    data = pickle.load(open(file_name+'.pkl', 'rb'))
    return data
```

In [34]:
```
# Sort 'Time' column
final = final.sort_values(by='Time', ascending=True)
```

In [35]:
```
# Source: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.t

# Train Test split for train and test data
def data_split(final, no_of_samples):
    X, y = data_sampling_preprocessing(final, no_of_samples)
    # split the data set into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
    topicklefile(X_train, 'X_train')
    topicklefile(X_test, 'X_test')
    topicklefile(y_train, 'y_train')
    topicklefile(y_test, 'y_test')

    return X_train, X_test, y_train, y_test
```

In [36]:
```
def apply_avgw2v_train_test(X_train, X_test):

    # Training own Word2Vec model using your own text corpus
    list_of_sent_train = []
    for sent in X_train:#final['Text_Summary'].values:
        list_of_sent_train.append(sent.split())
    list_of_sent_test = []
    for sent in X_test:#final['Text_Summary'].values:
        list_of_sent_test.append(sent.split())

    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=8)

    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])

    # compute average word2vec for each review for train data
    avgw2v_train = []; # the avg-w2v for each sentence/review is stored in this list
```

```
    for sent in tqdm(list_of_sent_train): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        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
        avgw2v_train.append(sent_vec)
#     print(len(avgw2v_train))
#     print(len(avgw2v_train[0]))

    # compute average word2vec for each review for test data
    avgw2v_test = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent_test): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        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
        avgw2v_test.append(sent_vec)
#     print(len(avgw2v_test))
#     print(len(avgw2v_test[0]))

    return avgw2v_train, avgw2v_test
```

In [37]:
```
def apply_tfidfw2v_train_test(X_train, X_test):

    # Training own Word2Vec model using your own text corpus
    list_of_sent_train = []
    for sent in X_train:#final['Text_Summary'].values:
        list_of_sent_train.append(sent.split())
    list_of_sent_test = []
    for sent in X_test:#final['Text_Summary'].values:
        list_of_sent_test.append(sent.split())

    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=8)

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

```
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_)))

# 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 = t

tfidfw2v_train = []; # the tfidf-w2v for each sentence/review is stored in this l
row=0;
for sent in tqdm(list_of_sent_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
    tfidfw2v_train.append(sent_vec)
    row += 1


tf_idf_matrix = model.transform(X_test)
# 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_)))

# 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 = t

tfidfw2v_test = []; # the tfidf-w2v for each sentence/review is stored in this li
row=0;
for sent in tqdm(list_of_sent_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
        tfidfw2v_test.append(sent_vec)
        row += 1


    return tfidfw2v_train, tfidfw2v_test
```

In [38]:
```
# Applying BOW on train and test data and creating the
from sklearn.preprocessing import StandardScaler
from scipy.sparse import hstack

#Standardize 'bow_train' data features by removing the mean and scaling to unit varia
std_scalar1 = StandardScaler(copy=True, with_mean=False, with_std=True)
std_scalar2 = StandardScaler(copy=True, with_mean=True, with_std=True)

count = 0

def apply_vectorizers_train_test(final, algo, model_name):

    global count
    if count == 0 or count == 4:
        if algo == 'LinearSVM':
            train_data, test_data, y_train, y_test = data_split(final, 100000)
        elif algo == 'RBFKernel':
            train_data, test_data, y_train, y_test = data_split(final, 40000)
        count += 1
        print("count: ", count)
        topicklefile(train_data,'train_data')
        topicklefile(test_data,'test_data')
        topicklefile(y_train,'y_train')
        topicklefile(y_test,'y_test')
    else:
        train_data = frompicklefile('train_data')
        test_data = frompicklefile('test_data')
        y_train = frompicklefile('y_train')
        y_test = frompicklefile('y_test')
        count += 1
        print("count: ", count)

    if model_name == 'BOW':
        #Applying BoW on Train data
```

```
        if algo == 'LinearSVM':
            count_vect = CountVectorizer()
        elif algo == 'RBFKernel':
            count_vect = CountVectorizer(min_df = 10, max_features = 500)

        #Applying BoW on Test data
        bow_train_vect = count_vect.fit_transform(train_data)

        #Applying BoW on Test data similar to the bow_train data
        bow_test_vect = count_vect.transform(test_data)

        # Standardise train data
        bow_train_vect = std_scalar1.fit_transform(bow_train_vect)
        # Standardize the unseen bow_test data
        bow_test_vect = std_scalar1.transform(bow_test_vect)

        topicklefile(bow_train_vect, 'bow_train_vect')
        topicklefile(bow_test_vect, 'bow_test_vect')

        print("'bow_train_vect' and 'bow_test_vect' are the pickle files.")
        return count_vect

    elif model_name == 'TF-IDF':
        #Applying TF-IDF on Train data
        if algo == 'LinearSVM':
            count_vect = TfidfVectorizer(ngram_range=(1,2))
        elif algo == 'RBFKernel':
            count_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features =

        #Applying BoW on Test data
        tfidf_train_vect = count_vect.fit_transform(train_data)

        #Applying BoW on Test data similar to the bow_train data
        tfidf_test_vect = count_vect.transform(test_data)

        # Standardise train data
        tfidf_train_vect = std_scalar1.fit_transform(tfidf_train_vect)
        # Standardize the unseen bow_test data
        tfidf_test_vect = std_scalar1.transform(tfidf_test_vect)

        topicklefile(tfidf_train_vect, 'tfidf_train_vect')
        topicklefile(tfidf_test_vect, 'tfidf_test_vect')

        print("'tfidf_train_vect' and 'tfidf_test_vect' are the pickle files.")
        return count_vect

    elif model_name == 'AvgW2V':
        avgw2v_train_vect, avgw2v_test_vect = apply_avgw2v_train_test(train_data, tes
```

```
            # Standardise train data
            avgw2v_train_vect = std_scalar2.fit_transform(avgw2v_train_vect)
            # Standardize the unseen bow_test data
            avgw2v_test_vect = std_scalar2.transform(avgw2v_test_vect)

            topicklefile(train_vect, 'avgw2v_train_vect')
            topicklefile(test_vect, 'avgw2v_test_vect')
            print("'avgw2v_train_vect' and 'avgw2v_test_vect' are the pickle files.")

        elif model_name == 'TF-IDF W2V':
            tfidfw2v_train_vect, tfidfw2v_test_vect = apply_tfidfw2v_train_test(train_data

            # Standardise train data
            tfidfw2v_train_vect = std_scalar2.fit_transform(tfidfw2v_train_vect)
            # Standardize the unseen bow_test data
            tfidfw2v_test_vect = std_scalar2.transform(tfidfw2v_test_vect)

            topicklefile(train_vect, 'tfidfw2v_train_vect')
            topicklefile(test_vect, 'tfidfw2v_test_vect')
            print("'tfidfw2v_train_vect' and 'tfidfw2v_test_vect' are the pickle files.")

        else:
            #Error Message
            print('Model specified is not valid! Please check.')
```

```
In [39]: def apply_svm(algo, parameters, train_data, y_train):

        if algo == 'LinearSVM':
 #           parameters = {'alpha':alpha_values}
            svm_clf = SGDClassifier(class_weight='balanced')

            clf = GridSearchCV(svm_clf, parameters, cv=10, scoring= 'roc_auc', n_jobs=-1,
            clf.fit(train_data, y_train)
            alpha_optimal = clf.best_params_.get('alpha')

            optimal_hyperprameter = alpha_optimal
            optimal_penalty= clf.best_params_.get('penalty')

            #Getting the Train and CV AUC score values for only 'optimal_penalty' for the
            clf_cv_results = pd.DataFrame(clf.cv_results_)
            clf_cv_results = clf_cv_results[clf_cv_results['param_penalty']== optimal_pena

            train_auc= clf_cv_results['mean_train_score']
            train_auc_std= clf_cv_results['std_train_score']
            cv_auc = clf_cv_results['mean_test_score']
            cv_auc_std= clf_cv_results['std_test_score']
```

```
            return clf, optimal_penalty, optimal_hyperprameter, train_auc, train_auc_std,

        elif algo == 'RBFKernel':
#             parameters = {'C':alpha_values}
            svm_clf = SVC(kernel='rbf', probability = True, class_weight='balanced', cache

            clf = GridSearchCV(svm_clf, parameters, cv=10, scoring= 'roc_auc', n_jobs=-1,
            clf.fit(train_data, y_train)
            c_optimal = clf.best_params_.get('C')

            optimal_hyperprameter = c_optimal

#             print(clf.cv_results_)

            train_auc= clf.cv_results_['mean_train_score']
            train_auc_std= clf.cv_results_['std_train_score']
            cv_auc = clf.cv_results_['mean_test_score']
            cv_auc_std= clf.cv_results_['std_test_score']

            return clf, optimal_hyperprameter, train_auc, train_auc_std, cv_auc, cv_auc_s
```

In [40]:
```
def train_cv_error_plot(alpha_values, train_auc, train_auc_std, cv_auc, cv_auc_std):
#     alpha_values = np.log10(alpha_values)
    plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')

    # Source: https://stackoverflow.com/a/48803361/4084039
#     plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_

    plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')
    # Source: https://stackoverflow.com/a/48803361/4084039
#     plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alph
    plt.legend()
    plt.xlabel("Hyperparameter-log Values")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
```

In [41]:
```
def svm_optimal(algo, optimal_hyperparameter, optimal_penalty, train_vect, y_train):
    if algo == 'LinearSVM':
        optimal_svm = SGDClassifier(loss="hinge",penalty = optimal_penalty, alpha = op
        optimal_svm.fit(train_vect, y_train)
    elif algo == 'RBFKernel':
        optimal_svm = SVC(kernel='rbf', C = optimal_hyperparameter, probability = True
        optimal_svm.fit(train_vect, y_train)
    return optimal_svm
```

In [42]:
```
def retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test):
```

```
        # fitting the model with optimal K for training data
        optimal_svm.fit(train_vect, y_train)

        # predict the response for the unseen bow_test data
        y_pred = optimal_svm.predict(test_vect)
```

In [43]:
```
# Confusion Matrix
def cm_fig(optimal_svm, y_test, test_vect):
    cm = pd.DataFrame(confusion_matrix(y_test, optimal_svm.predict(test_vect)))
    print(confusion_matrix(y_test, optimal_svm.predict(test_vect)))

    plt.figure(1, figsize=(18,5))
    plt.subplot(121)
    plt.title("Confusion Matrix")
    sns.set(font_scale=1.4)
    sns.heatmap(cm, cmap= 'gist_earth', annot=True, annot_kws={'size':15}, fmt='g')
```

In [44]:
```
def svm_calibratedclassifierCV(algo, optimal_svm, penalty_given, train_data, y_train)
    if algo == 'LinearSVM':
        svm_calib = CalibratedClassifierCV(optimal_svm, method = 'sigmoid', cv='prefi
        svm_calib.fit(train_data, y_train)
    elif algo == 'RBFKernel':
        print("No need of Calibrated Classifier CV for SVC")
        svm_calib = optimal_svm
#          svm_calib = CalibratedClassifierCV(optimal_svm, method='sigmoid', cv=5)
#          svm_calib.fit(train_data, y_train)
    return svm_calib
```

In [45]:
```
#Reference: https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-
def error_plot(svm_obj, train_vec, y_train, test_vec, y_test):
    train_fpr, train_tpr, thresholds = roc_curve(y_train, svm_obj.predict_proba(train
    test_fpr, test_tpr, thresholds = roc_curve(y_test, svm_obj.predict_proba(test_vec

    plt.plot(train_fpr, train_tpr, label="train AUC = %0.3f" %auc(train_fpr, train_tp
    plt.plot(test_fpr, test_tpr, label="train AUC = %0.3f" %auc(test_fpr, test_tpr))
    plt.plot([0.0, 1.0], [0.0, 1.0],'k--')
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.show()

    return auc(test_fpr, test_tpr)
```

In [46]:
```
def get_features_top(count_vect, optimal_svm):
    features=count_vect.get_feature_names()
    feature_prob=optimal_svm.coef_.ravel()
    print(len(features))
    print('='*100)
```

```
            print(feature_prob.shape)
            df_feature_proba = pd.DataFrame({'features':features, 'probabilities':feature_prol
            df_feature_proba = df_feature_proba.sort_values(by=['probabilities'],ascending=Fal
        #     print(df_feature_proba)
            return df_feature_proba[:11], df_feature_proba[-11:]
```

8.1 [5.1] Linear SVM

8.1.1 [5.1.1] Applying Linear SVM on BOW, SET 1

```
In [47]: # Please write all the code with proper documentation
```

```
In [48]: bow_count_vect = apply_vectorizers_train_test(final, 'LinearSVM', 'BOW')
```

```
100%|| 100000/100000 [00:41<00:00, 2395.96it/s]
100%|| 100000/100000 [00:26<00:00, 3724.77it/s]


count:  1
'bow_train_vect' and 'bow_test_vect' are the pickle files.
```

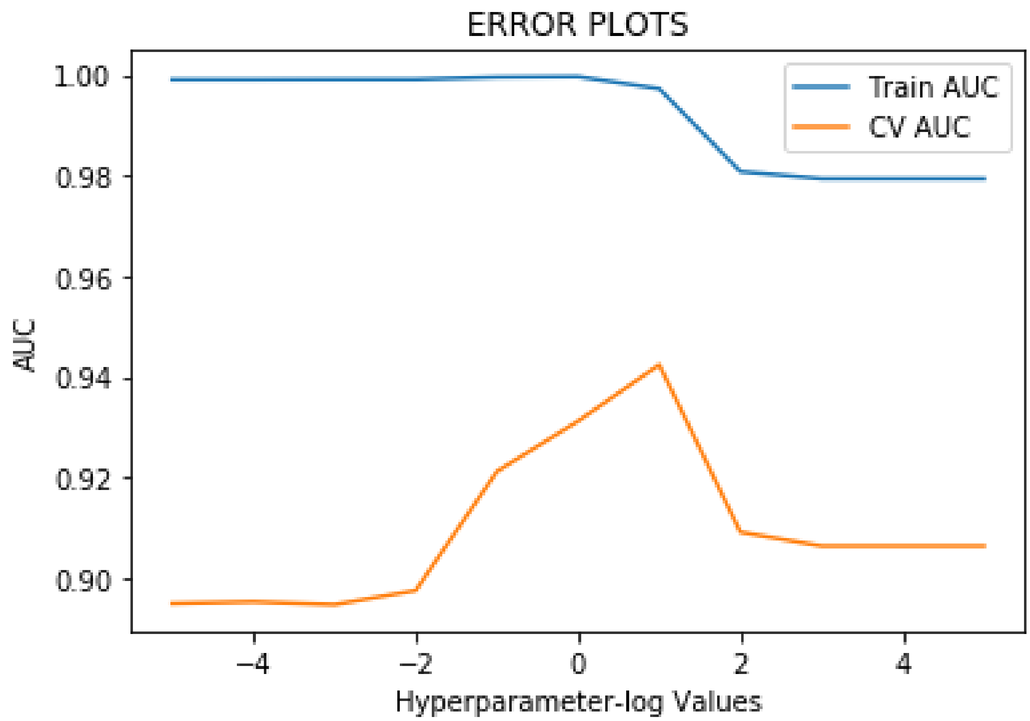
```
In [49]: train_vect = frompicklefile('bow_train_vect')
         test_vect = frompicklefile('bow_test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')

         print(train_vect.shape)
         print(len(y_train))
```

```
(70000, 100497)
70000
```

```
In [50]: alpha_values = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4
         penalties = ['l1', 'l2']
         # penalties = ['l1']
         hyper_parameters = {'alpha':alpha_values, 'penalty':penalties}
         clf, optimal_penalty, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_auc

         print('The optimal penalty is {}' .format(optimal_penalty))
         print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

         optimal_hyperparameter_bow1 = optimal_hyperparameter
```

```
The optimal penalty is l2
The optimal hyperparameter is 10
```

```
In [51]: train_cv_error_plot(alpha_values, train_auc, train_auc_std, cv_auc, cv_auc_std)
```

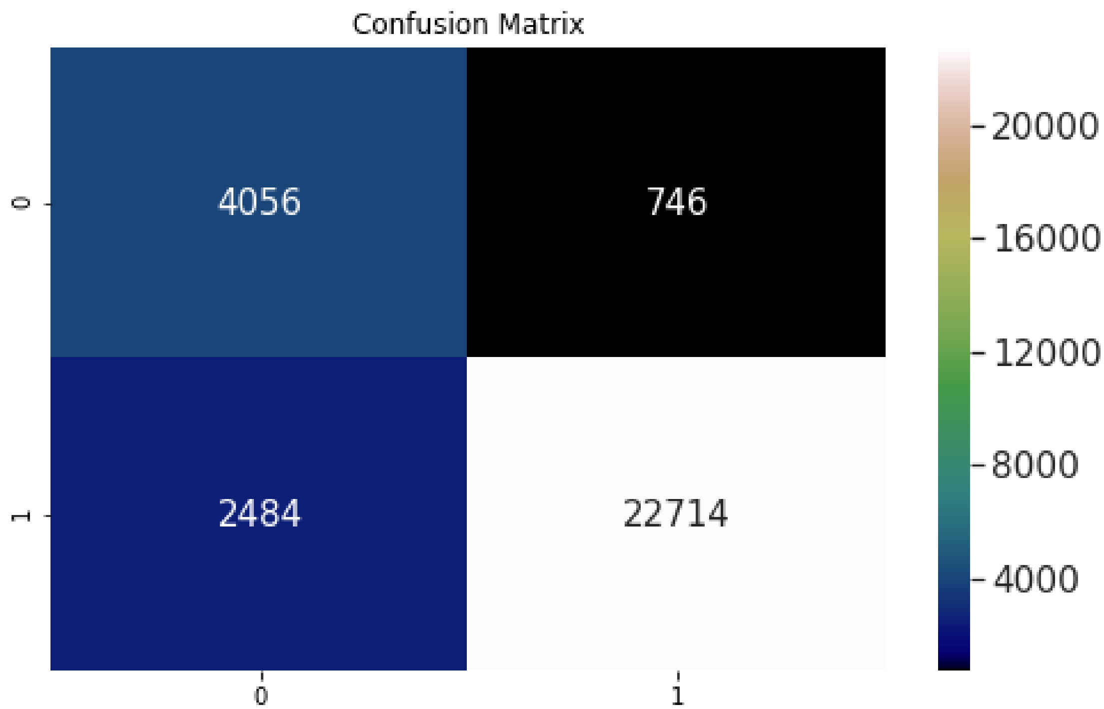



```
In [52]: bow_optimal_svm = svm_optimal('LinearSVM', optimal_hyperparameter_bow1, optimal_penal
```

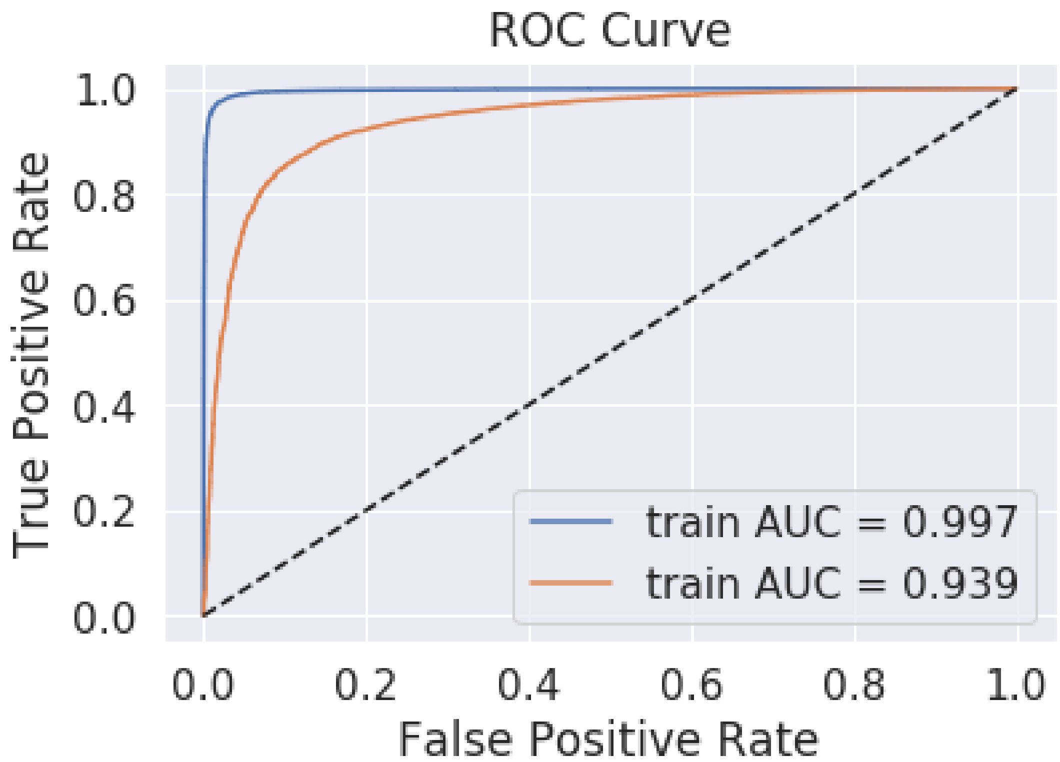
```
In [53]: # retrain_svm(bow_optimal_svm, train_vect, y_train, test_vect, y_test)
```

```
In [54]: cm_fig(bow_optimal_svm, y_test, test_vect)
```

```
[[ 4056   746]
 [ 2484 22714]]
```




```
In [55]: svm_calib = svm_calibratedclassifierCV('LinearSVM', bow_optimal_svm, optimal_penalty,

In [56]: bow_auc1 = error_plot(svm_calib, train_vect, y_train, test_vect, y_test)
         bow_auc1
```



Out[56]: 0.9386344698096035

In [57]:
```
positive_features, negative_features = get_features_top(bow_count_vect, bow_optimal_sv
positive_features
```

```
100497
==========================================================================================
(100497,)
```

Out[57]:

	features	probabilities
40073	great	0.023859
7471	best	0.015814
52053	love	0.013308
23230	delicious	0.013134
64990	perfect	0.011331
38723	good	0.010757
30778	excellent	0.010381
32646	favorite	0.009903
43189	highly	0.009863
52616	loves	0.009572
97611	wonderful	0.009149

In [58]: negative_features

Out[58]:

	features	probabilities
25521	disappointing	-0.012450
89812	threw	-0.012903
5092	awful	-0.013096
43949	horrible	-0.013649
57159	money	-0.013701
88823	terrible	-0.013801
5450	bad	-0.014145
98236	worst	-0.014452
95998	waste	-0.014561
25409	disappointed	-0.015801
59781	not	-0.019110

8.1.2 [5.1.2] Applying Linear SVM on TFIDF, SET 2

In [59]:
```
# Please write all the code with proper documentation
```

In [60]:
```
tfidf_count_vect = apply_vectorizers_train_test(final, 'LinearSVM', 'TF-IDF')
```

```
count:  2
'tfidf_train_vect' and 'tfidf_test_vect' are the pickle files.
```

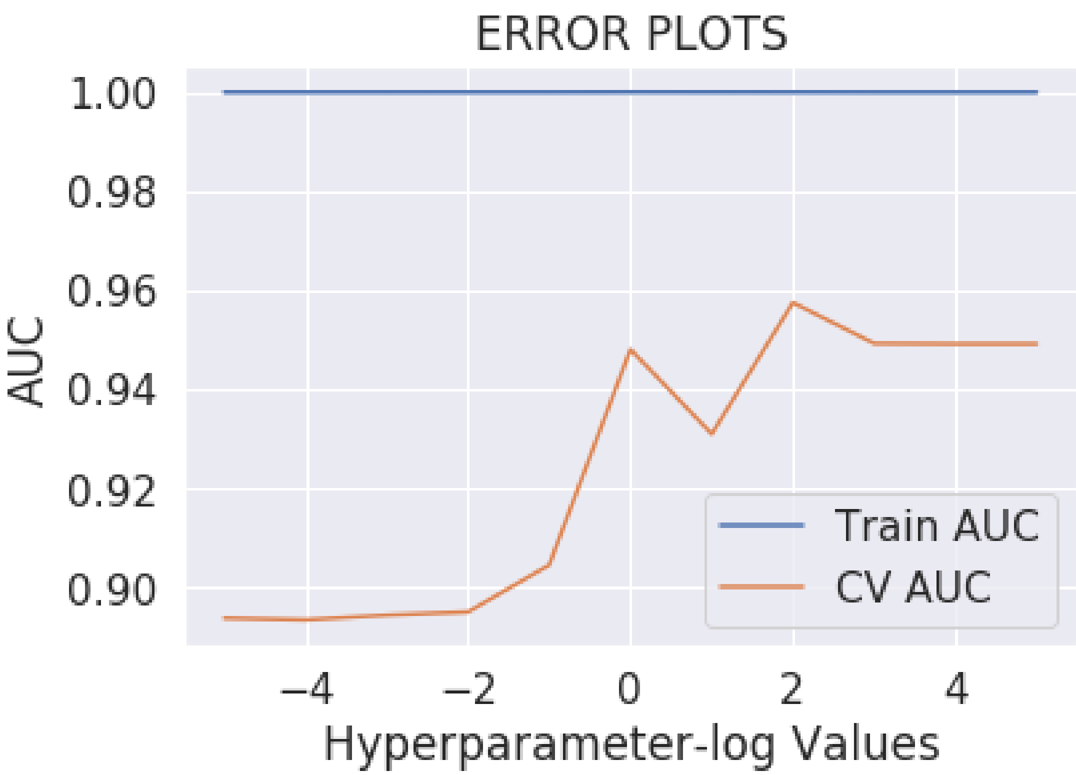
In [61]:
```
train_vect = frompicklefile('tfidf_train_vect')
test_vect = frompicklefile('tfidf_test_vect')
y_train = frompicklefile('y_train')
y_test = frompicklefile('y_test')

print(train_vect.shape)
print(len(y_train))
```

```
(70000, 1384302)
70000
```

In [62]:
```
alpha_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,
penalties = ['l1', 'l2']
hyper_parameters = {'alpha':alpha_values, 'penalty':penalties}
clf, optimal_penalty, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_au

print('The optimal penalty is {}' .format(optimal_penalty))
print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

optimal_hyperparameter_tfidf1 = optimal_hyperparameter
```

```
The optimal penalty is l2
The optimal hyperparameter is 100
```

In [63]: train_cv_error_plot(alpha_values,train_auc, train_auc_std, cv_auc, cv_auc_std)



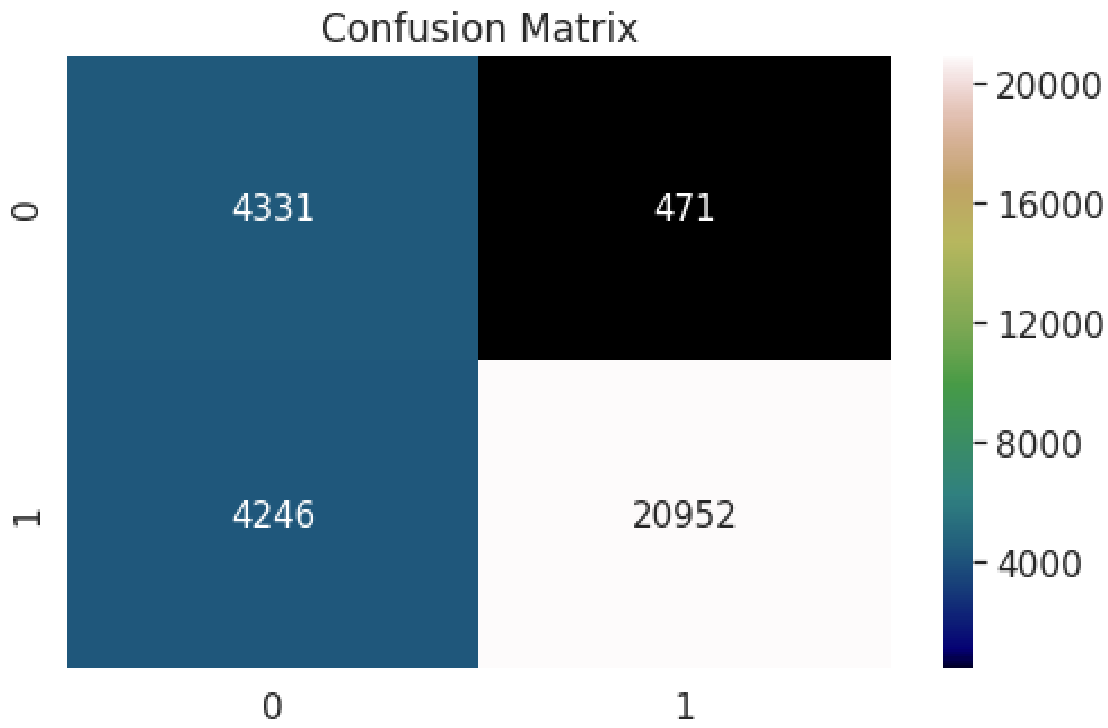
In [64]:
```
tfidf_optimal_svm = svm_optimal('LinearSVM', optimal_hyperparameter_tfidf1, optimal_pe
print(train_vect.shape)
print(len(y_train))
```

```
(70000, 1384302)
70000
```

In [65]:
```
# retrain_svm(tfidf_optimal_svm, train_vect, y_train, test_vect, y_test)
```

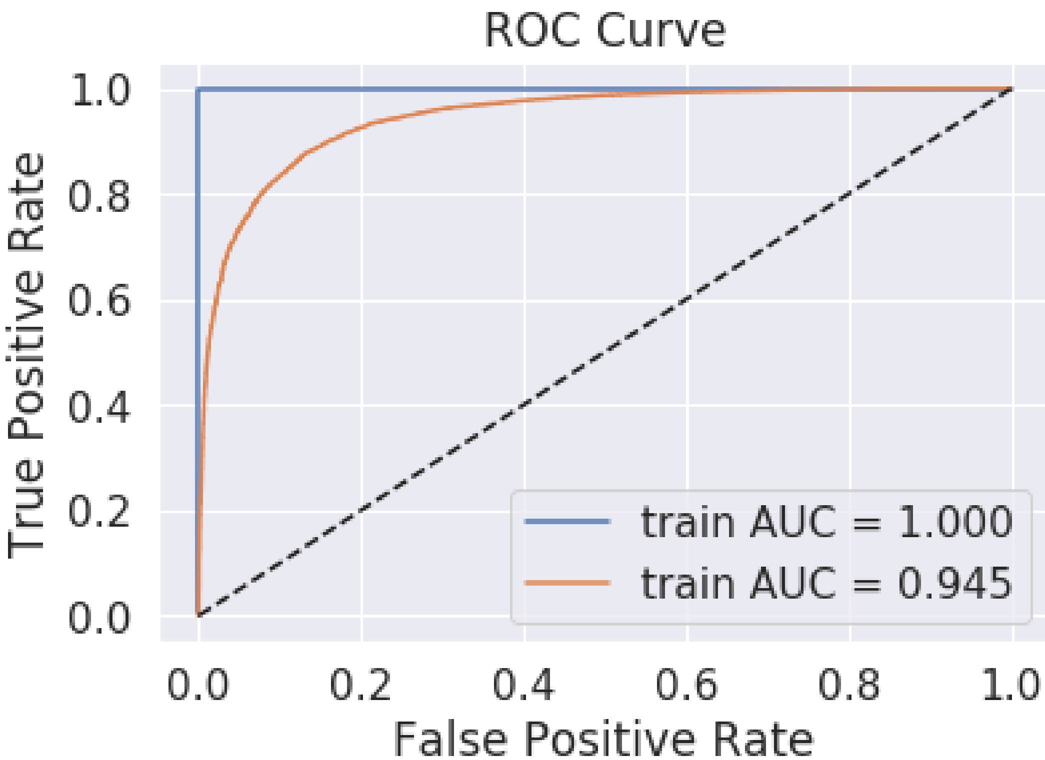
In [66]:
```
cm_fig(tfidf_optimal_svm, y_test, test_vect)
```

```
[[ 4331   471]
 [ 4246 20952]]
```




```
In [67]: svm_calib = svm_calibratedclassifierCV('LinearSVM', tfidf_optimal_svm, optimal_penalty
```

```
In [68]: tfidf_auc1 = error_plot(svm_calib, train_vect, y_train, test_vect, y_test)
         tfidf_auc1
```



Out[68]: 0.9448584206008033

In [69]:
```
positive_features, negative_features = get_features_top(tfidf_count_vect, tfidf_optima
positive_features
```

```
1384302
==========================================================================================
(1384302,)
```

Out[69]:

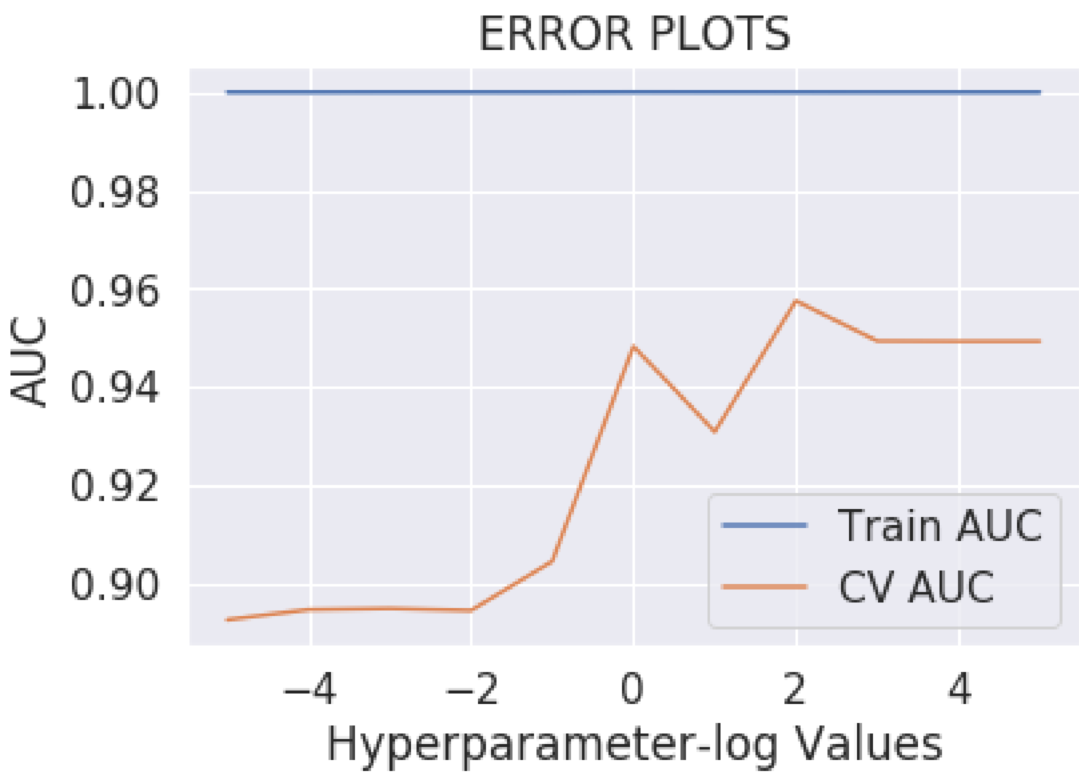
	features	probabilities
538649	great	0.002450
100832	best	0.001634
706396	love	0.001458
313573	delicious	0.001358
521922	good	0.001228
893684	perfect	0.001173
431808	favorite	0.001054
406735	excellent	0.001017
713427	loves	0.001005
581282	highly	0.000997
367772	easy	0.000983

```
In [70]: negative_features

Out[70]:             features  probabilities
         1242115        threw      -0.001668
         590461      horrible      -0.001733
         1225716     terrible      -0.001737
         773720         money      -0.001743
         336662  disappointed      -0.001748
         72909            bad      -0.001844
         1329463  waste money      -0.001876
         1364483        worst      -0.001899
         1329302        waste      -0.001947
         813738       not buy      -0.002005
         812910           not      -0.003051
```

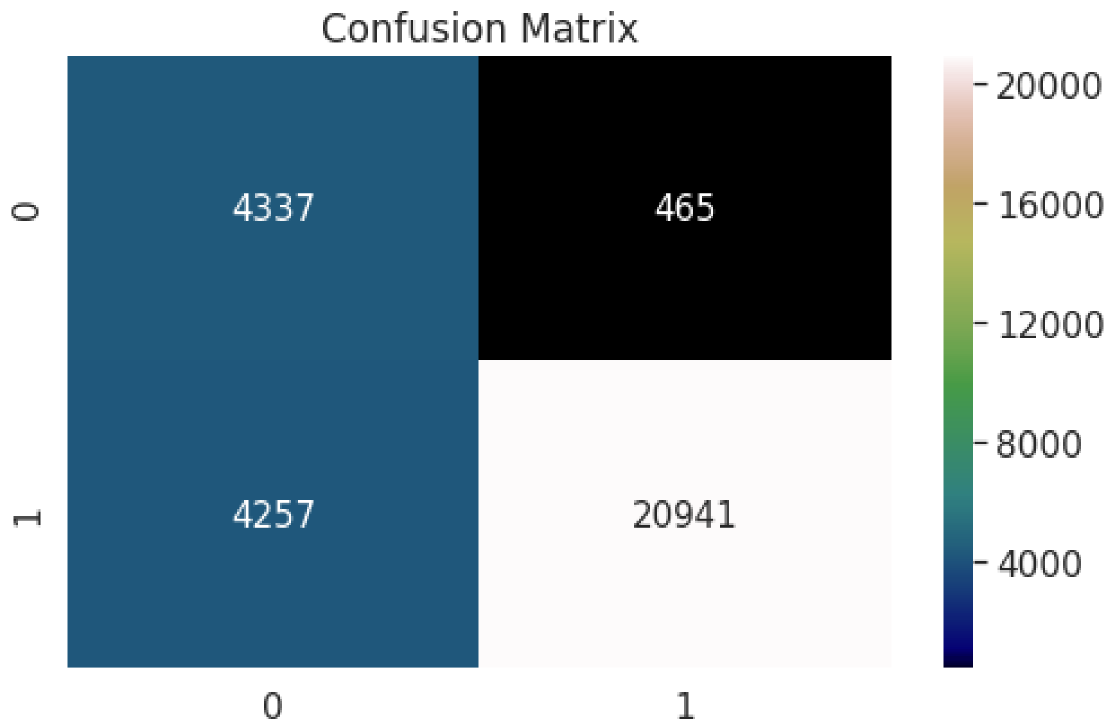
8.1.3 [5.1.3] Applying Linear SVM on AVG W2V, SET 3

```
In [71]: # Please write all the code with proper documentation

In [72]: count_vect = apply_vectorizers_train_test(final, 'LinearSVM', 'AvgW2V')

count:  3


100%|| 70000/70000 [20:10<00:00, 57.81it/s]
100%|| 30000/30000 [08:19<00:00, 60.09it/s]


'avgw2v_train_vect' and 'avgw2v_test_vect' are the pickle files.


In [73]: train_vect = frompicklefile('avgw2v_train_vect')
         test_vect = frompicklefile('avgw2v_test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')

In [74]: alpha_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,1
         penalties = ['l1', 'l2']
         hyper_parameters = {'alpha':alpha_values, 'penalty':penalties}
         clf, optimal_penalty, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_auc

         print('The optimal penalty is {}' .format(optimal_penalty))
         print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

         optimal_hyperparameter_avgw2v1 = optimal_hyperparameter

The optimal penalty is l2
The optimal hyperparameter is 100
```

```
In [75]: train_cv_error_plot(alpha_values,train_auc, train_auc_std, cv_auc, cv_auc_std)
```

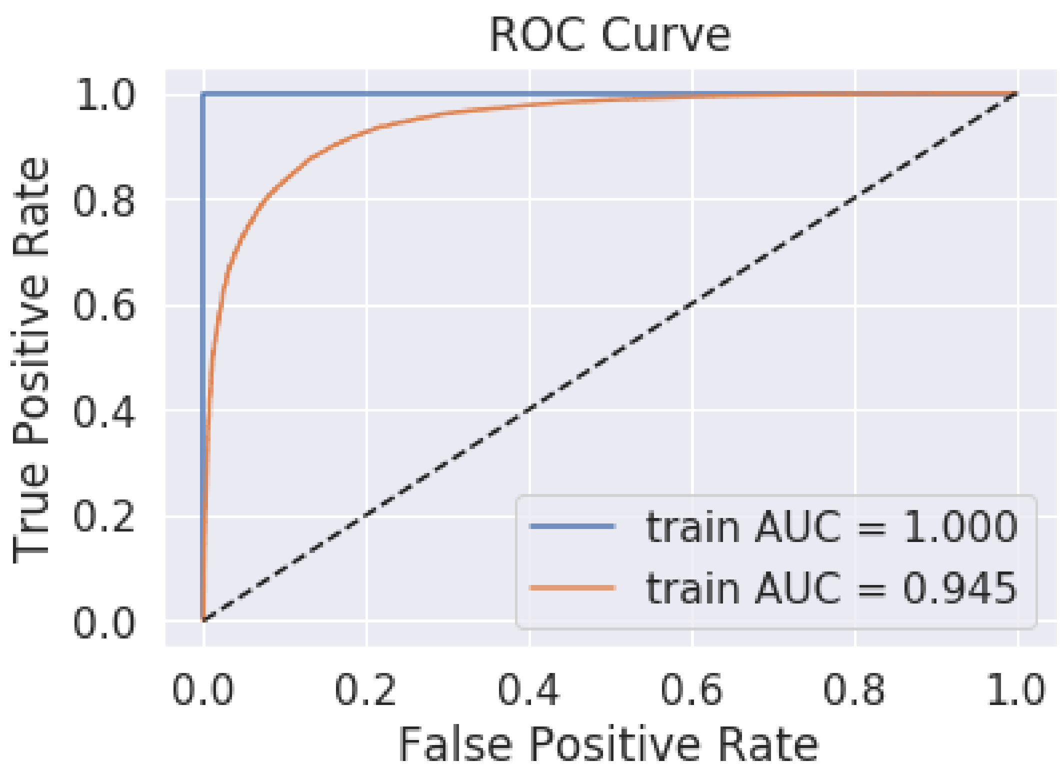



```
In [76]: optimal_svm = svm_optimal('LinearSVM', optimal_hyperparameter, optimal_penalty, train_

In [77]: # retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test)

In [78]: cm_fig(optimal_svm, y_test, test_vect)

[[ 4337   465]
 [ 4257 20941]]
```




```
In [79]: svm_calib = svm_calibratedclassifierCV('LinearSVM', optimal_svm, optimal_penalty, trai
```

```
In [80]: avgw2v_auc1 = error_plot(svm_calib, train_vect, y_train, test_vect, y_test)
         avgw2v_auc1
```




```
Out[80]: 0.9450313781406859
```

8.1.4 [5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

```
In [81]: # Please write all the code with proper documentation
```

```
In [82]: count_vect = apply_vectorizers_train_test(final, 'LinearSVM', 'TF-IDF W2V')
```

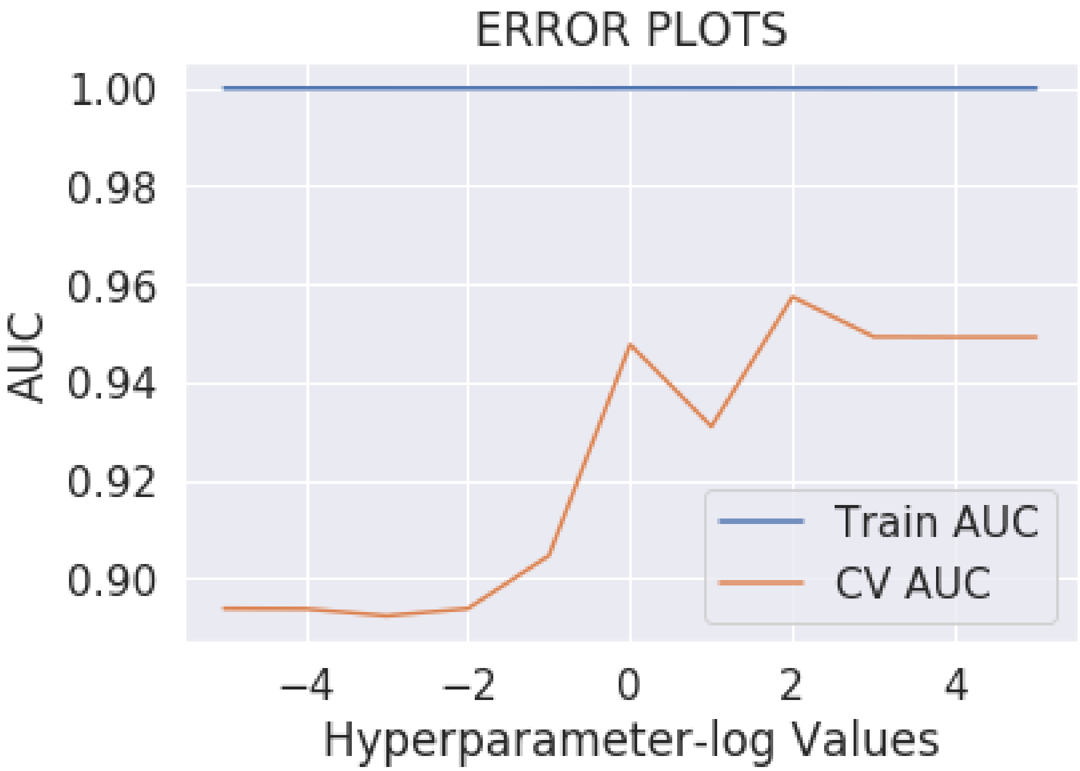
```
count:  4


100%|| 70000/70000 [1:39:08<00:00, 11.77it/s]
100%|| 30000/30000 [46:50<00:00, 10.68it/s]


'tfidfw2v_train_vect' and 'tfidfw2v_test_vect' are the pickle files.
```

```
In [83]: train_vect = frompicklefile('tfidf_train_vect')
         test_vect = frompicklefile('tfidf_test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
```

In [84]:
```
alpha_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,
penalties = ['l1', 'l2']
hyper_parameters = {'alpha':alpha_values, 'penalty':penalties}
clf, optimal_penalty, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_au

print('The optimal penalty is {}' .format(optimal_penalty))
print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

optimal_hyperparameter_tfidfw2v1 = optimal_hyperparameter
```

```
The optimal penalty is l2
The optimal hyperparameter is 100
```

In [85]:
```
train_cv_error_plot(alpha_values,train_auc, train_auc_std, cv_auc, cv_auc_std)
```

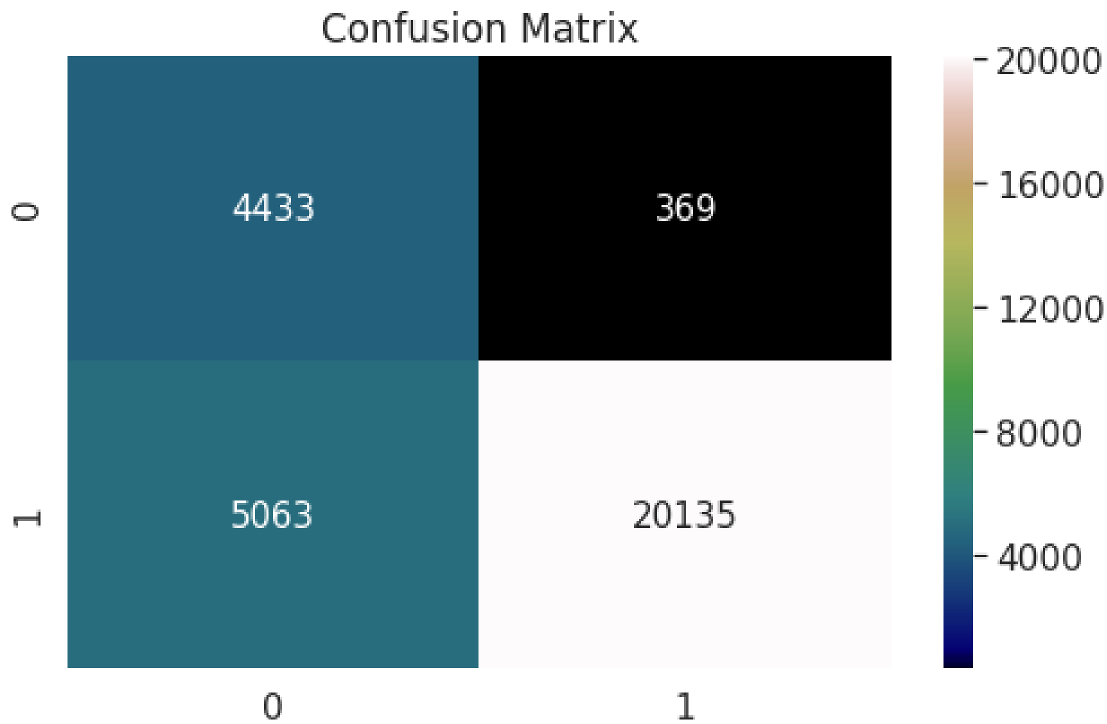


In [86]:
```
optimal_svm = svm_optimal('LinearSVM', optimal_hyperparameter, optimal_penalty,train_
```

In [87]:
```
# retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test)
```

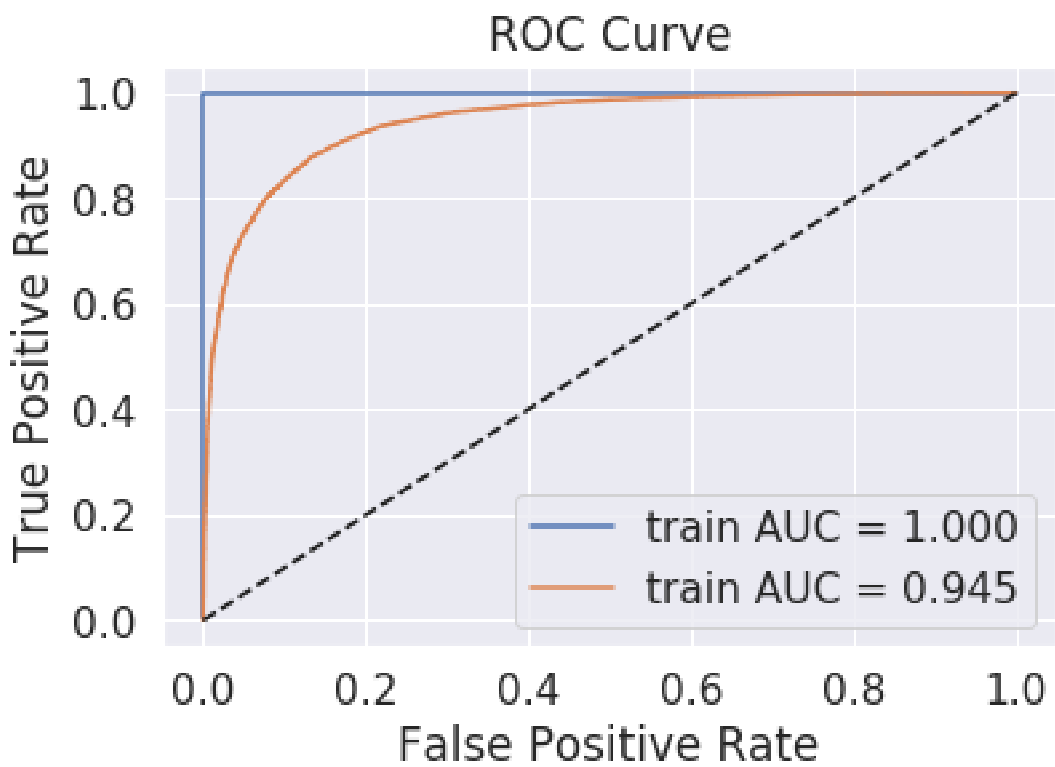
In [88]:
```
cm_fig(optimal_svm, y_test, test_vect)
```

```
[[ 4433   369]
 [ 5063 20135]]
```




```
In [89]: svm_calib = svm_calibratedclassifierCV('LinearSVM', optimal_svm, optimal_penalty, trai
```

```
In [90]: tfidfw2v_auc1 = error_plot(svm_calib, train_vect, y_train, test_vect, y_test)
         tfidfw2v_auc1
```




```
Out[90]: 0.9450462375470654
```

8.2 [5.2] RBF SVM

8.2.1 [5.2.1] Applying RBF SVM on BOW, SET 1

```
In [91]: # Please write all the code with proper documentation
```

```
In [92]: count_vect = apply_vectorizers_train_test(final, 'RBFKernel', 'BOW')
```

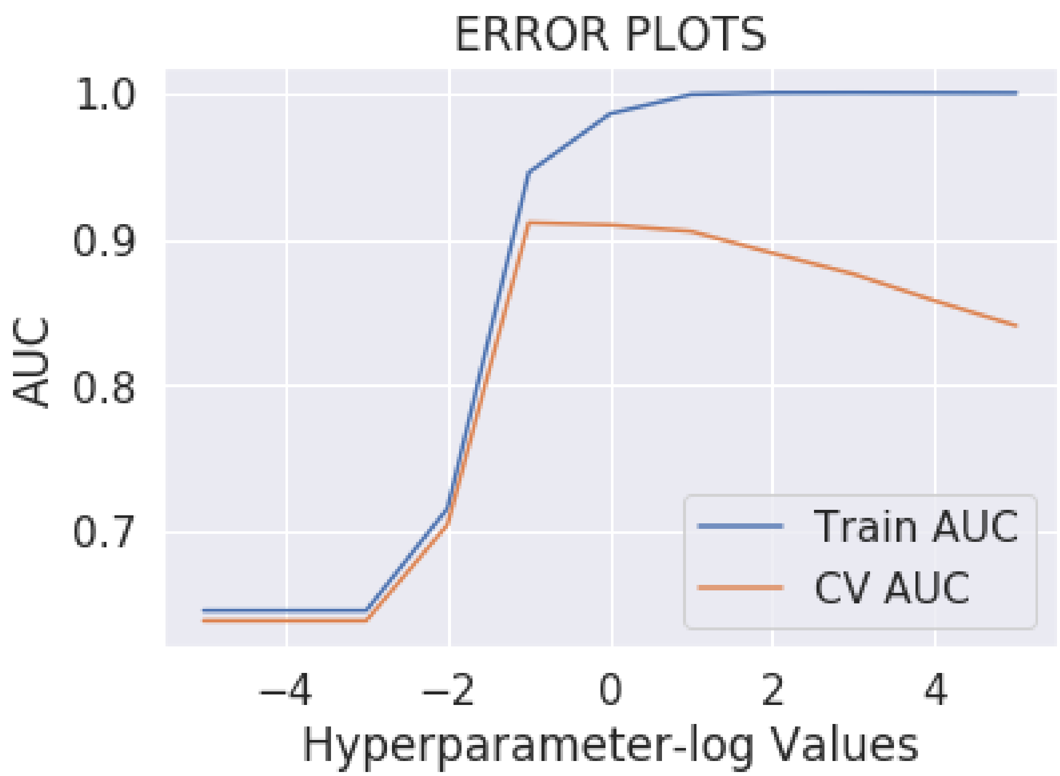
```
100%|| 40000/40000 [00:16<00:00, 2406.48it/s]
100%|| 40000/40000 [00:10<00:00, 3707.13it/s]


count:  5
'bow_train_vect' and 'bow_test_vect' are the pickle files.
```

```
In [93]: train_vect = frompicklefile('bow_train_vect')
         test_vect = frompicklefile('bow_test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
```

```
In [94]: C_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,10**5
         hyper_parameters = {'C':C_values}
         clf, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_auc_std = apply_svm

         print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

         optimal_hyperparameter_bow2 = optimal_hyperparameter
```

```
The optimal hyperparameter is 0.1
```

```
In [95]: train_cv_error_plot(C_values,train_auc, train_auc_std, cv_auc, cv_auc_std)
```

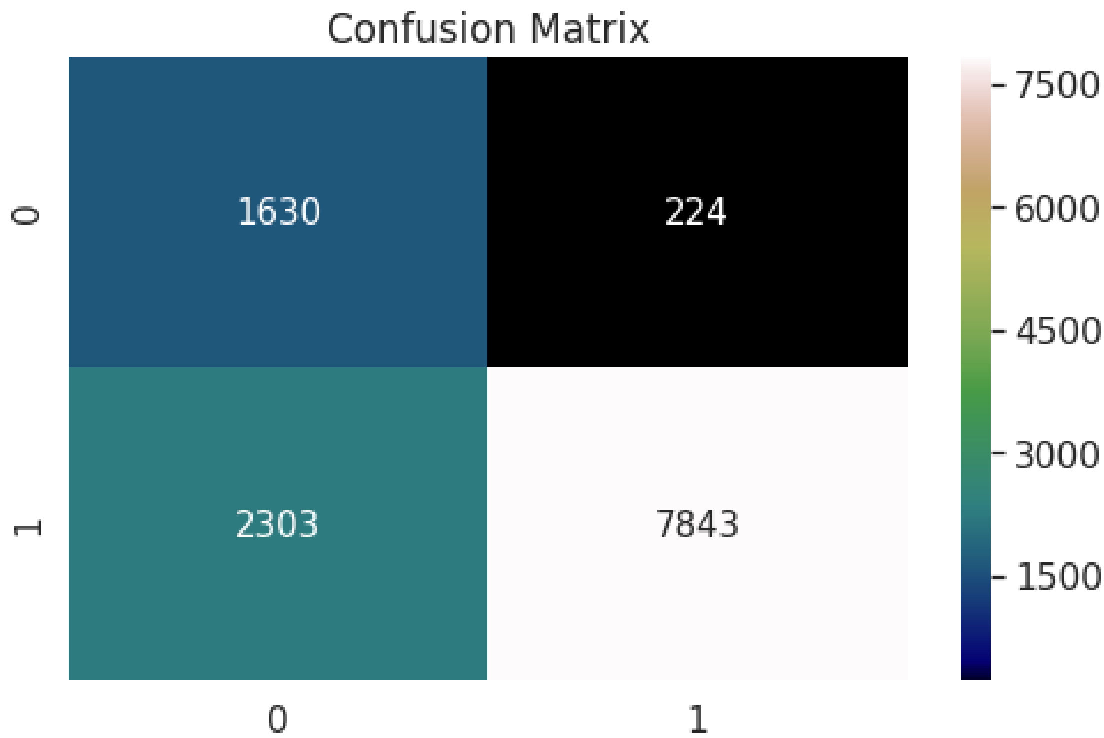



```
In [96]: optimal_svm = svm_optimal('RBFKernel', optimal_hyperparameter, 'l2', train_vect, y_tra
```

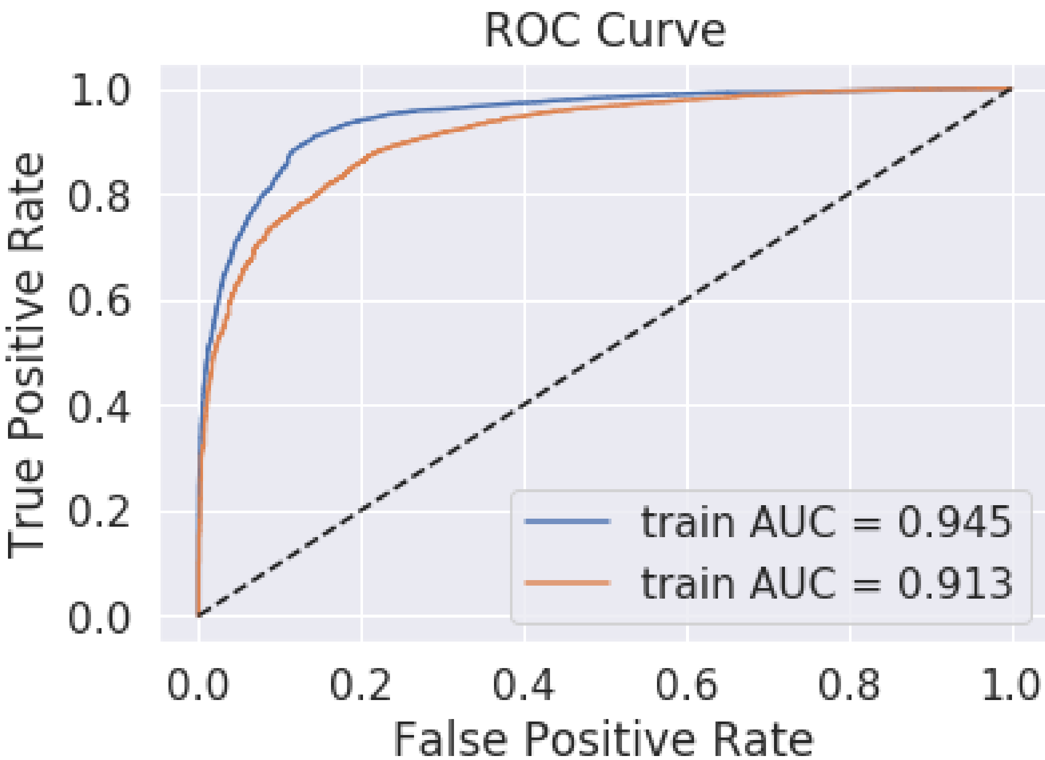
```
In [97]: # retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test)
```

```
In [98]: cm_fig(optimal_svm, y_test, test_vect)
```

```
[[1630  224]
 [2303 7843]]
```



In [99]:
```
bow_auc2 = error_plot(optimal_svm, train_vect, y_train, test_vect, y_test)
bow_auc2
```




```
Out[99]: 0.9131805892863865
```

8.2.2 [5.2.2] Applying RBF SVM on TFIDF, SET 2

```
In [100]: # Please write all the code with proper documentation
```

```
In [101]: count_vect = apply_vectorizers_train_test(final, 'RBFKernel', 'TF-IDF')
```

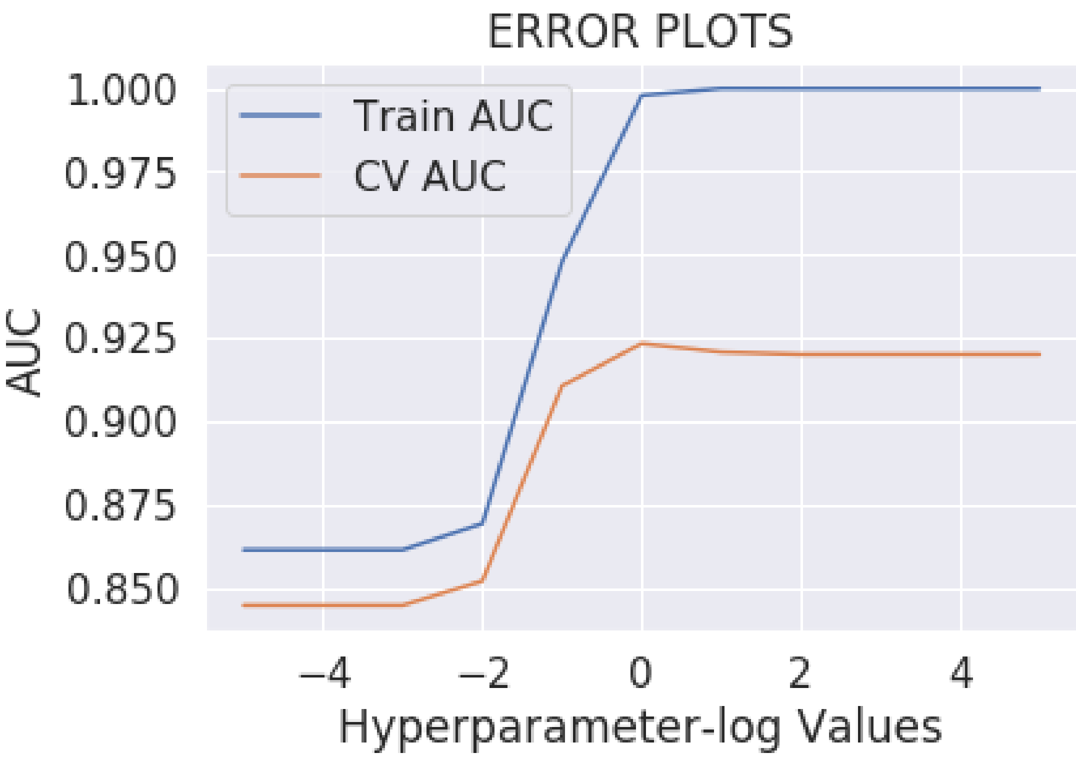
```
count:  6
'tfidf_train_vect' and 'tfidf_test_vect' are the pickle files.
```

```
In [102]: train_vect = frompicklefile('tfidf_train_vect')
          test_vect = frompicklefile('tfidf_test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

```
In [103]: C_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,10*
          hyper_parameters = {'C':C_values}
          clf, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_auc_std = apply_svm

          print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

          optimal_hyperparameter_tfidf2 = optimal_hyperparameter
```

```
The optimal hyperparameter is 1
```

```
In [104]: train_cv_error_plot(C_values,train_auc, train_auc_std, cv_auc, cv_auc_std)
```

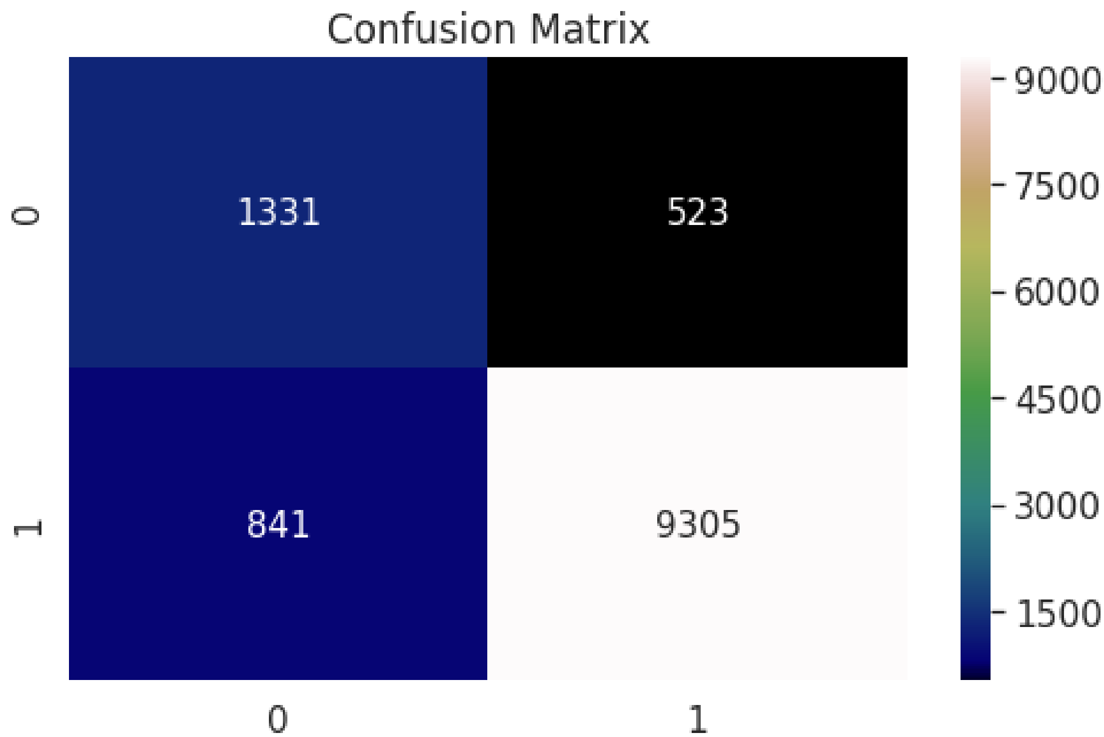



```
In [105]: optimal_svm = svm_optimal('RBFKernel', optimal_hyperparameter, 'l2', train_vect, y_t
```

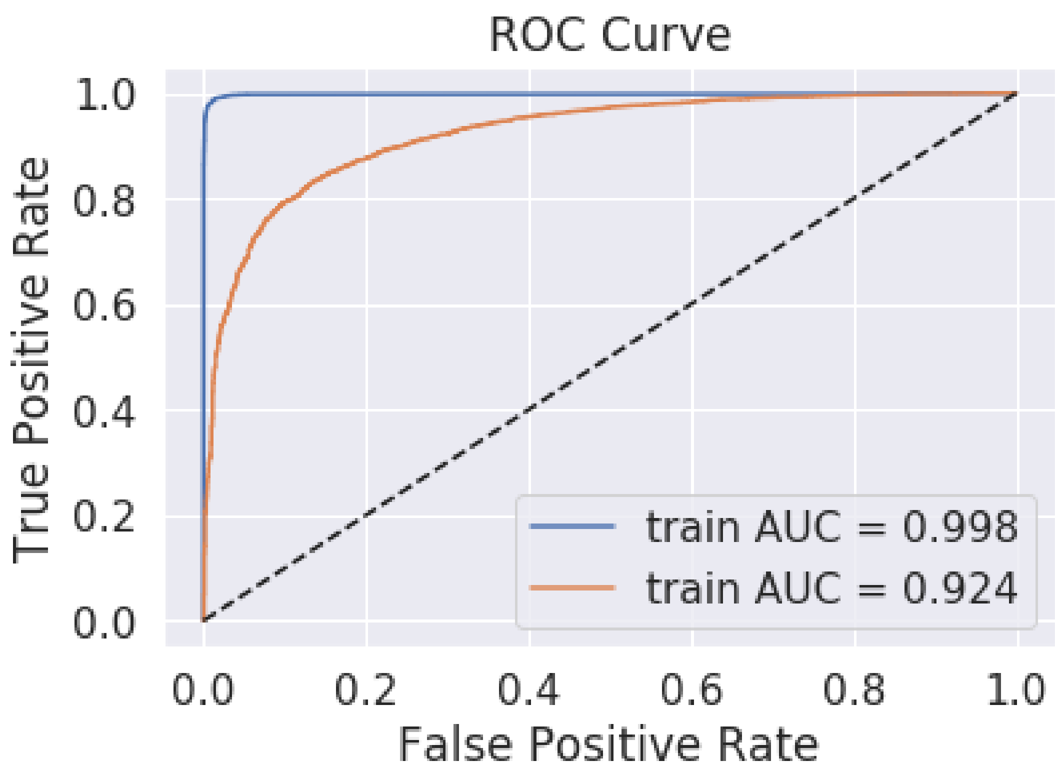
```
In [106]: # retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test)
```

```
In [107]: cm_fig(optimal_svm, y_test, test_vect)
```

```
[[1331  523]
 [ 841 9305]]
```



In [108]:
```
tfidf_auc2 = error_plot(optimal_svm, train_vect, y_train, test_vect, y_test)
tfidf_auc2
```




```
Out[108]: 0.9235218931964408
```

8.2.3 [5.2.3] Applying RBF SVM on AVG W2V, SET 3

```
In [109]: # Please write all the code with proper documentation
```

```
In [110]: count_vect = apply_vectorizers_train_test(final, 'RBFKernel', 'AvgW2V')
```

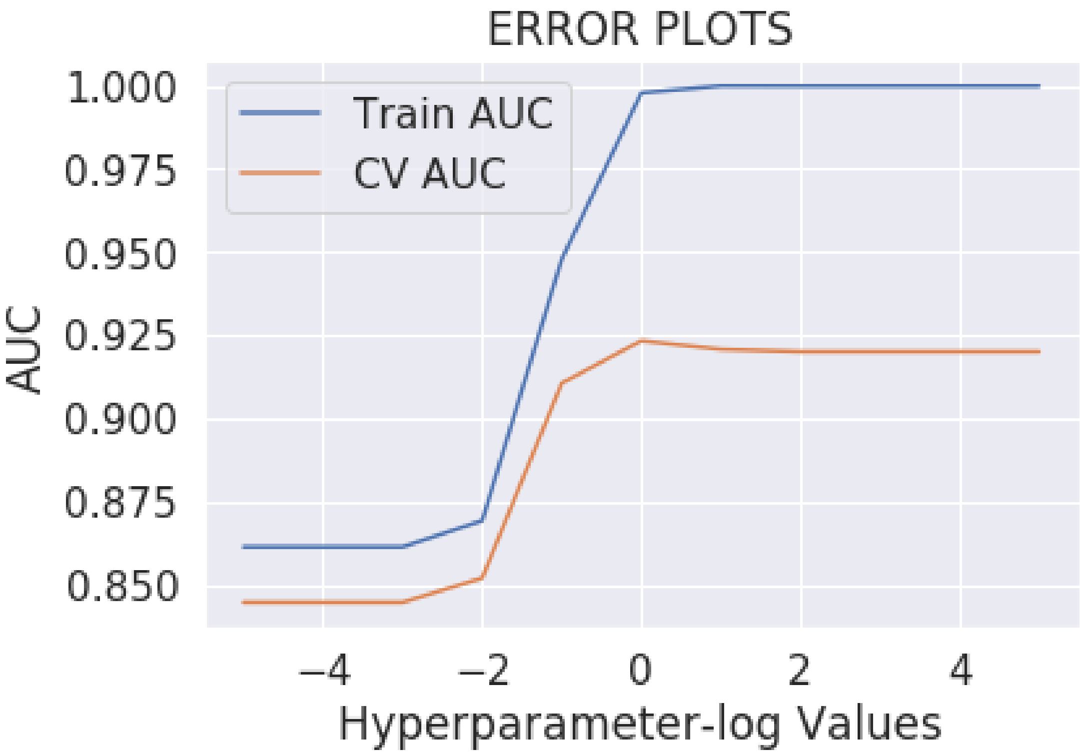
```
count:  7


100%|| 28000/28000 [04:18<00:00, 108.38it/s]
100%|| 12000/12000 [01:53<00:00, 105.48it/s]


'avgw2v_train_vect' and 'avgw2v_test_vect' are the pickle files.
```

```
In [111]: train_vect = frompicklefile('avgw2v_train_vect')
          test_vect = frompicklefile('avgw2v_test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

In [112]:
```
C_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,10**
hyper_parameters = {'C':C_values}
clf, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_auc_std = apply_svm

print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

optimal_hyperparameter_avgw2v2 = optimal_hyperparameter
```

```
The optimal hyperparameter is 1
```

In [113]:
```
train_cv_error_plot(C_values,train_auc, train_auc_std, cv_auc, cv_auc_std)
```

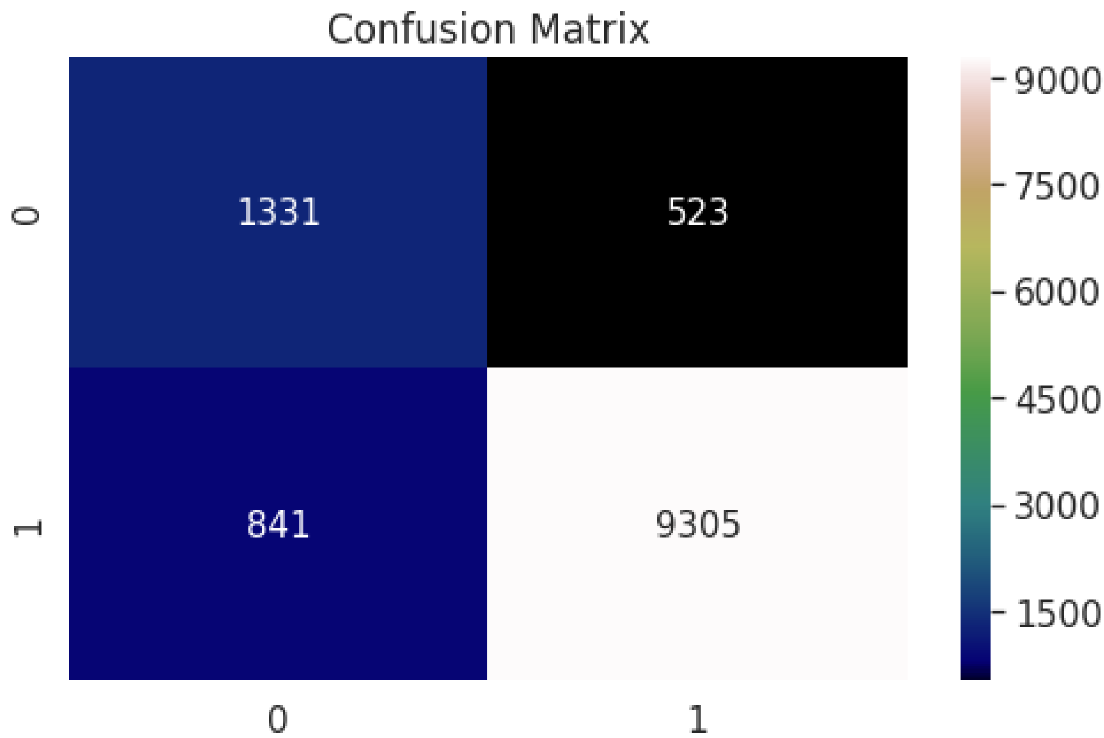


In [114]:
```
optimal_svm = svm_optimal('RBFKernel', optimal_hyperparameter, 'l2', train_vect, y_tr
```

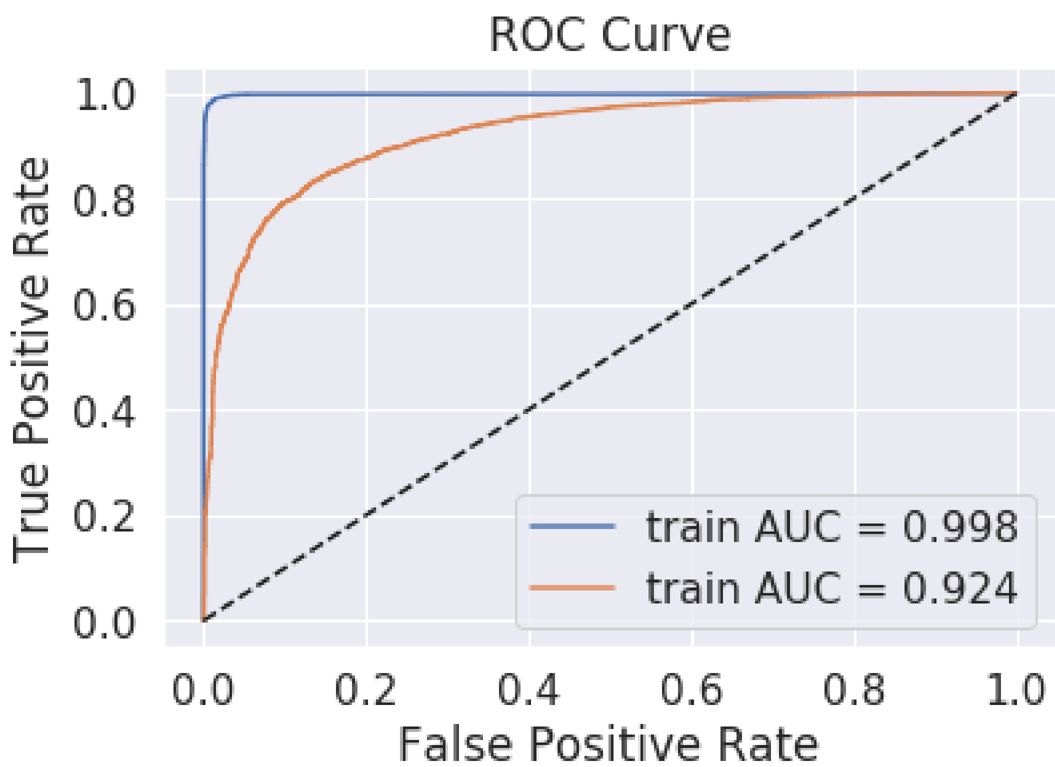
In [115]:
```
# retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test)
```

In [116]:
```
cm_fig(optimal_svm, y_test, test_vect)
```

```
[[1331  523]
 [ 841 9305]]
```




```
In [117]: avgw2v_auc2 = error_plot(optimal_svm, train_vect, y_train, test_vect, y_test)
          avgw2v_auc2
```




```
Out[117]: 0.9235226374543317
```

8.2.4 [5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

```
In [118]: # Please write all the code with proper documentation
```

```
In [119]: count_vect = apply_vectorizers_train_test(final, 'RBFKernel', 'TF-IDF W2V')
```

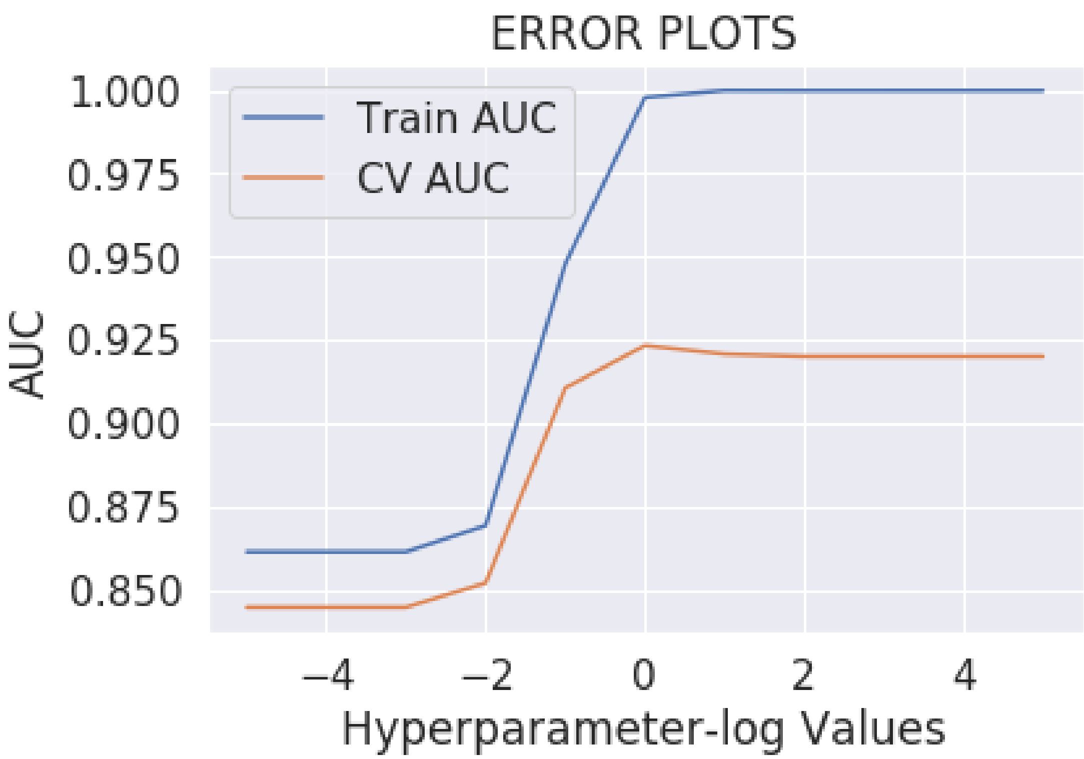
```
count:  8


100%|| 28000/28000 [17:43<00:00, 26.33it/s]
100%|| 12000/12000 [07:42<00:00, 25.93it/s]


'tfidfw2v_train_vect' and 'tfidfw2v_test_vect' are the pickle files.
```

```
In [120]: train_vect = frompicklefile('tfidfw2v_train_vect')
          test_vect = frompicklefile('tfidfw2v_test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

```
In [121]: C_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4,10**
          hyper_parameters = {'C':C_values}
          clf, optimal_hyperparameter, train_auc, train_auc_std, cv_auc, cv_auc_std = apply_svm

          print('The optimal hyperparameter is {}' .format(optimal_hyperparameter))

          optimal_hyperparameter_tfidfw2v2 = optimal_hyperparameter
```

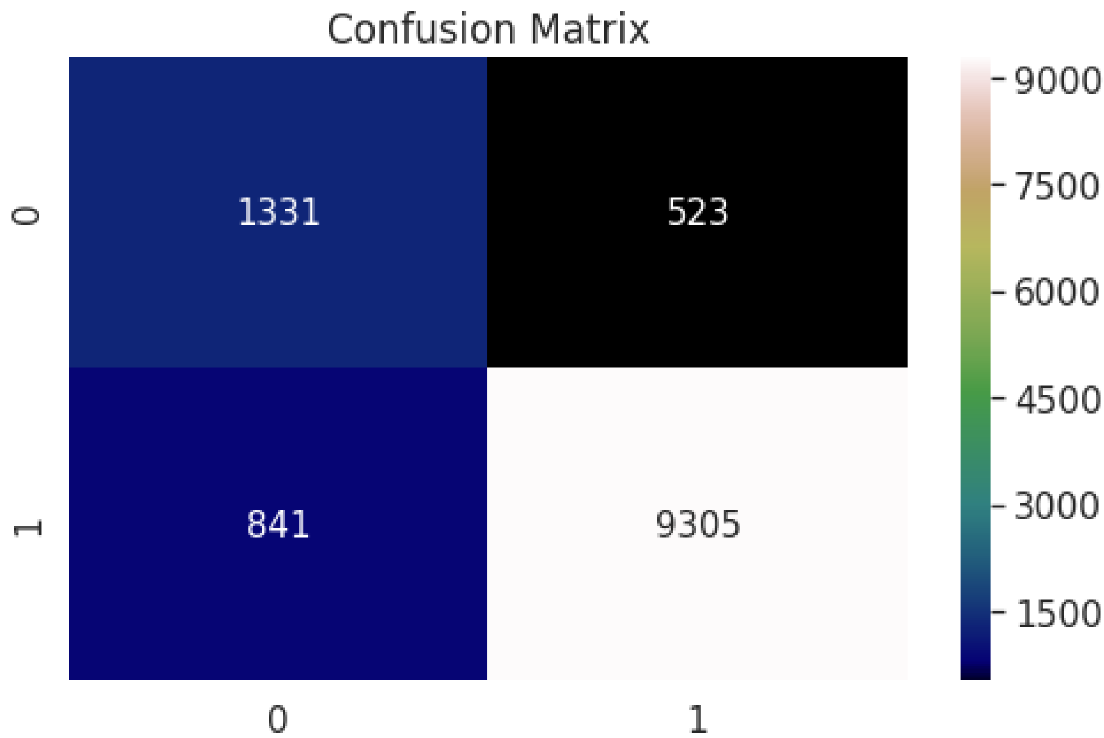
```
The optimal hyperparameter is 1
```

```
In [122]: train_cv_error_plot(C_values,train_auc, train_auc_std, cv_auc, cv_auc_std)
```

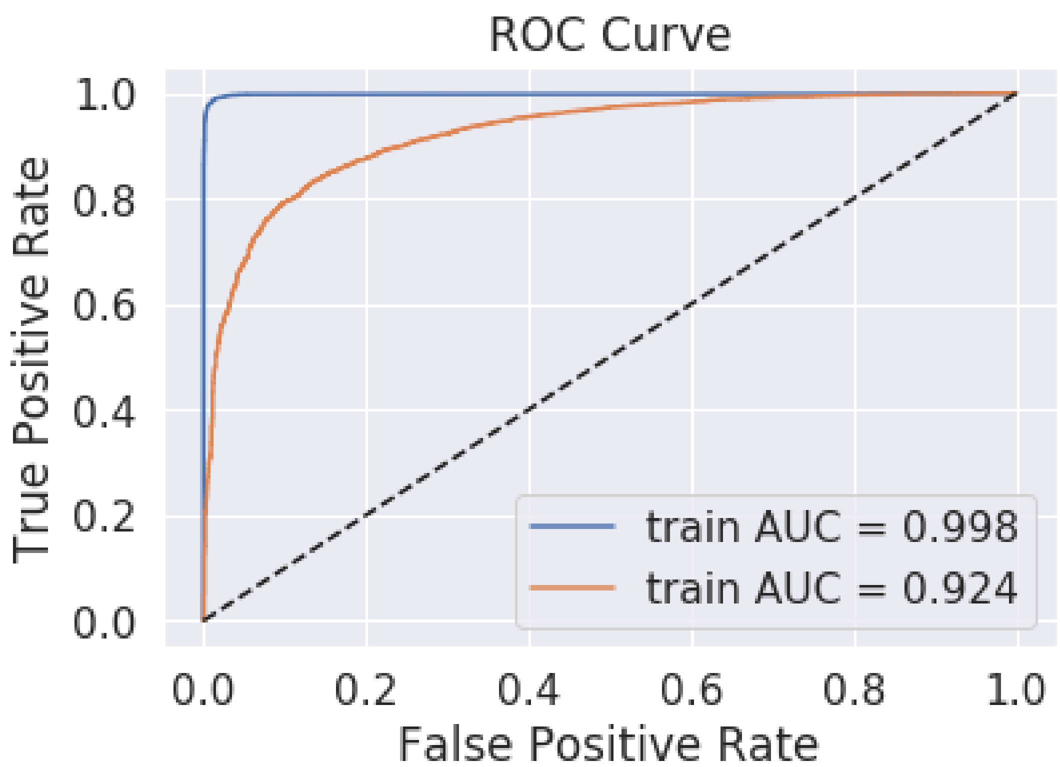



```
In [123]: optimal_svm = svm_optimal('RBFKernel', optimal_hyperparameter, 'l2', train_vect, y_tr

In [124]: # retrain_svm(optimal_svm, train_vect, y_train, test_vect, y_test)

In [125]: cm_fig(optimal_svm, y_test, test_vect)
```

```
[[1331  523]
 [ 841 9305]]
```



In [126]:
```
tfidfw2v_auc2 = error_plot(optimal_svm, train_vect, y_train, test_vect, y_test)
tfidfw2v_auc2
```




```
Out[126]: 0.9235230627445552
```

9 [6] Conclusions

```
In [127]: # Please compare all your models using Prettytable library
```

```
In [128]: from prettytable import PrettyTable

          model_metric = PrettyTable()

          model_metric = PrettyTable(["Model Name", "SVM Type", 'Hyperparameter', 'AUC'])

          model_metric.add_row(["Bag of Words","Linear kernel", optimal_hyperparameter_bow1, bo
          model_metric.add_row(["TF-IDF","Linear kernel", optimal_hyperparameter_tfidf1, tfidf_
          model_metric.add_row(["Avg W2V","Linear kernel", optimal_hyperparameter_avgw2v1, avgw
          model_metric.add_row(["TF-IDF W2V","Linear kernel", optimal_hyperparameter_tfidfw2v1
          model_metric.add_row(["Bag of Words","RBF kernel", optimal_hyperparameter_bow2, bow_a
          model_metric.add_row(["TF-IDF","RBF kernel", optimal_hyperparameter_tfidf2, tfidf_auc
          model_metric.add_row(["Avg W2V","RBF kernel", optimal_hyperparameter_avgw2v2, avgw2v_
          model_metric.add_row(["TF-IDF W2V","RBF kernel", optimal_hyperparameter_tfidfw2v2, ti
```

```python
print(model_metric.get_string(start=0, end=8))
```

```
+--------------+---------------+----------------+--------------------+
|  Model Name  |    SVM Type   | Hyperparameter |        AUC         |
+--------------+---------------+----------------+--------------------+
| Bag of Words | Linear kernel |       10       | 0.9386344698096035 |
|    TF-IDF    | Linear kernel |      100       | 0.9448584206008033 |
|   Avg W2V    | Linear kernel |      100       | 0.9450313781406859 |
|  TF-IDF W2V  | Linear kernel |      100       | 0.9450462375470654 |
| Bag of Words |   RBF kernel  |      0.1       | 0.9131805892863865 |
|    TF-IDF    |   RBF kernel  |       1        | 0.9235218931964408 |
|   Avg W2V    |   RBF kernel  |       1        | 0.9450313781406859 |
|  TF-IDF W2V  |   RBF kernel  |       1        | 0.9235230627445552 |
+--------------+---------------+----------------+--------------------+
```

9.1 [6.1] Observations:

1) Train time: As mentioned in the instructions RBF Kernel has taken significantly higher time and resources to train than the Linear Kernel.

2) For Important features for BOW and TF-IDF using Linear kernel is very accurate in terms of the particular words used in the positive and negative review.

3) Test AUC scores are very simialr to the train AUC scores for the data which indicates the model is doing a better job on the unseen data