07 Amazon Fine Food Reviews Analysis_Support Vector Machines

August 7, 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
from sklearn.externals import joblib
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from prettytable import PrettyTable
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from bs4 import BeautifulSoup
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

```
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\externals\joblib\__init__.py:15: DeprecationWa
  warnings.warn(msg, category=DeprecationWarning)
```

```
In [61]: # using SQLite Table to read data.
         con = sqlite3.connect('database.sqlite')

         # filtering only positive and negative reviews i.e.
         # not taking into consideration those reviews with Score=3
         # SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data poin
         # 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
         # for tsne assignment you can take 5k data points

         filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 300

         # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negati
         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 (30000, 10)
```

```
Out[61]:    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 [62]: display = pd.read_sql_query("""
         SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
         FROM Reviews
         GROUP BY UserId
         HAVING COUNT(*)>1
         """, con)
```

```
In [63]: print(display.shape)
         display.head()
```

```
(80668, 7)
```

```
Out[63]:                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 [64]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[64]:               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 [65]: display['COUNT(*)'].sum()
```

```
Out[65]: 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 [66]: display= pd.read_sql_query("""
         SELECT *
         FROM Reviews
         WHERE Score != 3 AND UserId="AR5J8UI46CURR"
         ORDER BY ProductID
         """, con)
         display.head()
```

```
Out[66]:        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 [67]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fal
```

In [68]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[68]:
```
(28072, 10)
```

In [69]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[69]:
```
93.57333333333332
```

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 [70]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[70]:
```
      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 [71]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [72]:
```
#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()
```

```
(28072, 10)
```

```
Out[72]: 1    23606
         0     4466
         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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.c
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
```

```
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

```
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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

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

        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)
```

```
In [73]: # 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)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
```

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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

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

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wa
```

In [74]:
```
# 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', 'througl
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'c
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'ar
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'r
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

```
In [75]: # 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 stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 28072/28072 [00:11<00:00, 2417.97it/s]
```

```
In [76]: preprocessed_reviews[1500]
```

```
Out[76]: 'favorite stevia product subscribe save queried customer service nunaturals gmo use ye
```

5 [4] Featurization

Before we featurize the data, we need to split it

```
In [95]: len(preprocessed_reviews)
```

```
Out[95]: 28072
```

```
In [96]: x = preprocessed_reviews
         y = final["Score"].values
```

```
In [97]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)         # th
         x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size=0.30) # th
```

```
In [98]: # number of rows in earch data set, train, cross validation and test data respectivel
         print(len(x_train))
         print(len(x_cv))
         print(len(x_test))
```

```
13755
5895
8422
```

5.1 [4.1] BAG OF WORDS

In [99]:
```
#BoW
count_vect = CountVectorizer(max_features=5000) #in scikit-learn
count_vect.fit(x_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

x_train_bow = count_vect.transform(x_train)
x_test_bow  = count_vect.transform(x_test)
x_cv_bow    = count_vect.transform(x_cv)

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("="*50)
```

```
some feature names  ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'acai', 'accept',
==================================================
(13755, 5000) (13755,)
(5895, 5000) (5895,)
(8422, 5000) (8422,)
==================================================
```

5.2 [4.2] Bi-Grams and n-Grams.

In [ ]:
```
#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/mod

# 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_
```

In [45]:
```
count_vect = CountVectorizer(max_features=5000)
```

5.3 [4.3] TF-IDF

In [44]:
```
tf_vect = TfidfVectorizer(max_features=5000)
```

In [100]:
```
# TFIDF using scikit-learn

tf_idf = TfidfVectorizer(max_features=5000) #arguments: ngram_range=(1,2), min_df=10
```

```
tf_idf.fit(x_train)

print("some sample features",tf_idf.get_feature_names()[0:10])
print('='*50)

# we use fit() method to learn the vocabulary from x_train
# and now transform text data to vectors using transform() method

x_train_tf = tf_idf.transform(x_train)
x_cv_tf    = tf_idf.transform(x_cv)
x_test_tf  = tf_idf.transform(x_test)

print("After featurization\n")

print(x_train_tf.shape, y_train.shape)
print(x_cv_tf.shape, y_cv.shape)
print(x_test_tf.shape, y_test.shape)
print("="*50)
```

```
some sample features ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'acai', 'accept',
==================================================
After featurization

(13755, 5000) (13755,)
(5895, 5000) (5895,)
(8422, 5000) (8422,)
==================================================
```

5.4 [4.4] Word2Vec

In [101]:
```
# Train your own Word2Vec model using your own text corpus

list_of_sentance_train =[]

for sentance in x_train:
    list_of_sentance_train.append(sentance.split())
```

In [103]:
```
# this line of code trains your w2v model on the give list of sentances
w2v_model = Word2Vec(list_of_sentance_train,min_count=5,size=200, workers=-1)
```

In [104]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  7020
sample words  ['oriental', 'flavor', 'ramen', 'noodles', 'taste', 'okay', 'almost', 'always',
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [105]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this li
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(200) # 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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|| 13755/13755 [00:17<00:00, 782.56it/s]


(13755, 200)
[-9.00761910e-05 -2.92332302e-04 -3.64117966e-04 -2.21249823e-04
 -3.71155061e-04 -1.75637893e-05  3.74915594e-04  3.40630784e-04
  1.79394720e-04 -7.08610593e-05  2.18776651e-04  1.18206092e-05
  6.06097452e-05  3.55778427e-05  2.11918737e-04  2.02481608e-04
 -1.42583727e-04  2.57509147e-04 -1.99871613e-04 -1.28527605e-05
  1.88563937e-04 -3.81251904e-04 -2.37057650e-04  1.10871460e-04
 -3.76407627e-04 -1.07960612e-04  2.16393899e-04  5.57827552e-04
  2.50106664e-04  1.89427344e-04  1.47648877e-04  2.79277855e-04
 -7.60078361e-05 -4.75778237e-04 -8.82904378e-05 -1.70498198e-04
  2.82945688e-04 -5.64672944e-05 -1.39797379e-04  1.83657961e-06
 -2.29309206e-05  3.51572300e-05 -1.36398293e-04  4.54556446e-05
 -2.91588842e-04  4.01331570e-05  3.50647675e-04  2.02204152e-04
 -2.21906808e-04  2.94453237e-06 -4.83398943e-04 -3.99281641e-04
  3.67727747e-06  4.40284634e-04 -4.29759525e-04 -3.82446066e-05
  2.88011379e-04 -2.67093085e-04 -3.26985834e-05  3.10476817e-04
 -2.92499441e-04  4.95380538e-05  5.06681562e-05 -2.16222923e-04
  3.45034219e-04  2.11628618e-04  2.10695574e-04  2.13573692e-04
  2.80809663e-05 -4.36915464e-05  1.39832561e-04 -1.09295273e-04
  2.12021941e-04 -2.03261401e-04 -4.59095908e-04  6.34224246e-05
  4.72327613e-04  1.76885718e-04 -4.56655062e-04 -1.01981331e-05
 -1.51001594e-04  1.35909630e-04 -2.50922638e-04  1.47574581e-04
 -3.06184602e-04 -1.96545006e-04 -3.29716747e-04  5.66938949e-05
  4.27652618e-04 -5.63027880e-04 -8.95648731e-05 -1.29185595e-04
```

```
  2.46287061e-04  4.56684536e-04 -2.95451245e-04  2.17056177e-04
  1.12428838e-04 -9.04549854e-05  5.47614408e-04  2.86027608e-04
  3.27186514e-04 -4.66501289e-04  1.74786506e-04 -3.03260317e-04
 -1.91170134e-04 -3.06914967e-04 -6.03536429e-05  3.07451918e-04
  1.93557736e-04 -4.72528204e-04 -3.60527486e-05  4.37651635e-04
 -2.37077756e-04 -2.93246776e-04  5.99201103e-05 -2.88428598e-04
 -3.24007303e-04  1.83874482e-04 -4.04810137e-05 -2.96387112e-04
 -2.51502093e-05 -7.23965500e-04  6.14376545e-04  9.97742125e-05
  3.83083386e-04  5.15072679e-04 -2.63763988e-04  5.29316661e-05
  1.32605134e-04 -4.29529324e-05 -9.32268296e-06  5.09923423e-04
  1.25755454e-04  2.26705369e-05 -4.98352954e-04 -2.02538787e-04
 -1.23197707e-04 -3.81566835e-04 -2.18472391e-04 -2.30007438e-06
 -2.62730026e-04  3.53896810e-04  2.81311102e-05  3.31527512e-04
  2.28086394e-04  2.37815436e-04  2.54147045e-04  2.29290535e-04
  2.57593418e-05  1.82792842e-04  2.88998562e-04  1.42212000e-04
  3.90222030e-05 -5.48024896e-04  2.11987575e-04 -3.24594408e-06
  7.45662284e-04  1.88994023e-04  2.68666733e-04  2.74086177e-04
 -1.40717736e-04  2.19775286e-04 -1.64815809e-04 -2.66940710e-04
  9.35676858e-05  1.09570369e-05 -5.36828148e-05 -3.98899911e-05
  1.18698443e-04 -2.41640580e-04  1.73009116e-04  1.82288873e-04
  7.26355190e-04  1.91111287e-04  2.91356047e-04 -1.87505445e-04
  3.53083647e-04 -2.40950910e-04 -3.92580552e-04 -4.54655442e-04
 -4.47533511e-04  5.25066622e-04  2.59312577e-04 -7.56136886e-05
 -5.06017065e-05 -5.82820562e-04 -1.69085545e-04 -4.85218770e-04
 -6.27274249e-04 -4.99657058e-05 -2.54047528e-04  1.54455677e-05
 -1.24258371e-04  1.21595671e-04  1.19815171e-05  2.08852349e-04
  8.58886192e-04 -3.10660645e-05 -5.50047359e-04  2.96539471e-04]
```

Cross Validation Data

In [106]:
```
list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [107]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(200) # 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_cv.append(sent_vec)
        sent_vectors_cv = np.array(sent_vectors_cv)
        print(sent_vectors_cv.shape)
        print(sent_vectors_cv[0])
```

```
100%|| 5895/5895 [00:07<00:00, 782.79it/s]


(5895, 200)
[-1.72943286e-04  2.33065958e-04 -3.93048228e-04  1.86227492e-04
 -9.50025594e-04 -1.71142671e-04 -8.30980941e-04  2.14801029e-04
  7.30458321e-04  3.10006510e-04  2.84597380e-04  1.03421665e-04
  9.75337358e-05  1.85543123e-04  1.70488908e-05 -5.48282054e-04
  3.77961987e-05  4.88662155e-04  2.91657374e-05  1.58417170e-04
  6.34737546e-04 -1.52792807e-04  1.62977207e-04 -1.43782366e-04
 -8.26233738e-04  3.87488437e-04  1.11715058e-04 -6.42638875e-04
  1.43383946e-04 -4.48101401e-04  6.86916398e-04  2.67014286e-04
  1.16662906e-04 -7.70075902e-04 -8.36112782e-05  1.03518574e-04
 -3.43826188e-05 -3.89402308e-05  2.80242038e-04 -4.46012944e-04
 -6.11948446e-05 -5.15450232e-04 -1.02576013e-04 -3.32321057e-04
 -3.72662220e-04 -1.93467172e-05 -6.71248545e-04  1.68696781e-04
  3.22605868e-04 -2.82323610e-04 -5.49271894e-04 -1.60492876e-04
 -5.99000010e-04 -1.68614572e-04 -4.89805014e-05  3.49630367e-04
 -9.35323759e-05 -2.76686686e-05  1.48162955e-04  2.40052032e-04
 -3.45234293e-05 -2.36818921e-04  2.43983259e-04  1.69180946e-05
  6.28722722e-04  3.08273853e-04  3.40273512e-04  7.08346145e-05
  3.06462578e-04  5.80866510e-04 -9.33761735e-06  9.28084966e-05
  5.84752614e-04  1.79039760e-04 -4.93926004e-04  1.68528032e-04
 -1.12016614e-04 -5.43359629e-04  1.15839143e-04 -3.30551123e-04
 -6.08178228e-04  8.66315821e-04  6.18003974e-04  5.30455730e-04
 -9.46968365e-05  4.15397315e-04  5.32931806e-04 -1.27469210e-04
  6.70380765e-05 -1.22030692e-03  7.40408224e-05 -3.33231371e-04
 -4.64589406e-05 -3.47960909e-04 -3.83714958e-04  7.28903594e-05
  1.92680941e-04 -6.32308141e-04  6.24818253e-04 -2.16935732e-05
  3.25588239e-04 -4.47343636e-04 -8.59106732e-04  4.87037563e-04
  2.40234382e-04 -2.35072057e-04  8.69794038e-04 -5.11593169e-04
 -4.19912756e-04  1.95884445e-04 -1.80977563e-04 -4.35951075e-04
 -8.64349428e-04  4.82899252e-05  5.69020381e-05 -2.65693907e-04
  1.81314569e-04 -2.04493871e-04 -1.50290856e-04 -4.92498868e-04
 -1.05740858e-04  1.61876752e-06  1.42303583e-04  6.02806849e-04
 -2.78250276e-04  5.51008963e-04 -7.73968272e-04 -6.77751808e-04
  2.25229513e-04 -2.60933712e-04  5.76257057e-04 -3.22460309e-04
 -3.11828530e-04  3.66188589e-04  2.51494128e-04 -4.13332341e-04
  4.23794762e-05 -1.97293454e-04  2.59830094e-04 -4.10831056e-04
 -4.05304576e-04  6.36559420e-04 -1.67391476e-04 -3.61874134e-04
 -2.67149276e-04 -2.66723002e-04 -1.25551941e-04 -3.10606954e-04
 -7.76051831e-05  3.64927086e-04 -6.30279054e-04 -4.82852305e-04
 -2.21936495e-04  2.34555128e-04 -1.36127715e-04 -4.22813055e-04
```

```
  3.19052713e-04 -1.22711205e-03  3.30606024e-04 -2.69116443e-04
  2.20288617e-04  5.45829339e-04 -2.65943961e-04 -5.30649962e-04
  3.80745774e-04 -5.48207553e-04 -1.11275595e-03 -3.73404400e-04
  1.26329257e-04 -8.44232231e-05  1.25580559e-03  1.29049635e-04
  5.50798704e-04  1.63944062e-04 -7.45891074e-05  5.26805755e-04
  6.78192360e-04 -4.44844528e-04  2.42533582e-04 -2.63343316e-04
 -3.57838713e-04  4.99101692e-04  7.97595180e-04  4.83030329e-05
  7.13418929e-04  4.83542705e-05 -3.26886070e-04 -5.50761074e-04
 -8.41393919e-04 -5.67116590e-04 -4.50204845e-04 -5.87314459e-04
  4.10897809e-04  1.80218086e-04 -7.12570649e-04  7.51321768e-04
 -7.19675645e-04 -2.40071196e-04  7.89994684e-04 -6.37209351e-04]
```

Test data

In [108]:
```python
list_of_sentance_test=[]
for sentance in x_test:
    list_of_sentance_test.append(sentance.split())
```

In [109]:
```python
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this lis
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(200) # 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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|| 8422/8422 [00:10<00:00, 793.88it/s]


(8422, 200)
[ 7.56628712e-05  2.33125975e-04 -4.28982770e-05  3.80961916e-04
  2.74211369e-04 -2.89915214e-04 -1.72577530e-04  7.57987176e-05
 -9.88171052e-04 -5.95456885e-04 -3.36130733e-04  2.05238510e-04
 -7.34663772e-05 -8.65988760e-05  1.16635535e-04 -3.90440241e-04
  6.61382495e-05 -2.51071069e-04 -2.78100119e-04  1.36762184e-04
  8.07753466e-04 -1.37217285e-07 -6.59875634e-04 -1.74387750e-04
  1.55726051e-04  1.42601141e-04 -2.30125423e-04  1.34983433e-04
  1.66459499e-04 -4.15164860e-04  2.05689871e-04  7.82860911e-04
```

```
 -5.36991723e-04 -6.70718518e-04 -7.43675870e-06 -7.68515364e-04
 -5.68124136e-04  3.74013524e-04  3.06913661e-04  4.34778029e-04
  7.09164381e-04  7.04325141e-04  5.26852487e-04 -8.78738271e-05
  1.35036256e-04 -1.68654950e-04  3.09632410e-04 -3.50150847e-04
  4.35908344e-04  8.10596801e-04  3.83364742e-04  7.65667173e-05
 -3.45624091e-05  3.38748063e-04 -3.66037237e-04  1.14811770e-04
 -8.34770569e-05 -1.06361749e-04  9.61345310e-05  2.93489639e-04
 -6.62605700e-04 -1.23908918e-04 -4.10609900e-04 -4.08236043e-05
  3.95849590e-05 -4.70118214e-04  3.46633891e-04  3.35426492e-04
 -9.77863658e-04 -1.93781684e-04  6.13494449e-04 -2.86982341e-04
 -3.01091236e-04 -2.95207613e-04  3.32411478e-04 -6.79631973e-05
  4.85263721e-04  3.46821544e-05  9.09904490e-05 -6.33175509e-04
  1.58315260e-04  6.34350387e-05 -6.89556170e-05  4.91456405e-04
 -1.12035043e-04 -4.88330519e-04  1.06785029e-04 -9.29378953e-04
 -2.93696406e-04 -5.78907675e-05  1.25791553e-03  5.50506926e-05
  4.47381990e-04 -1.02492663e-04 -4.03328859e-04 -4.93854683e-04
  3.12384468e-04  1.91102269e-04 -5.40427747e-04  1.60525672e-04
  1.02180906e-03 -4.49642263e-04 -6.61402932e-04 -6.24833772e-04
  7.92020905e-04 -1.74878101e-04 -5.64168475e-04 -1.48075742e-04
 -2.79788342e-04  1.82239567e-04  3.23911076e-04  5.65255492e-04
 -4.78489892e-05 -8.36661196e-04 -1.75742316e-05 -4.59181919e-04
 -9.45212620e-05  5.36737956e-04  4.34077641e-05  2.09270487e-04
  1.05327604e-04 -4.81552796e-04 -1.35854287e-04  3.50996774e-04
  3.49270544e-04  6.14974657e-05  8.87185197e-04  4.02329094e-04
  9.77524299e-05 -7.20421246e-04 -3.38108985e-05 -3.27162676e-05
 -4.05832279e-04 -2.33026840e-04 -1.83487835e-04 -2.75077896e-04
 -1.07796363e-04 -6.55724526e-04  3.50419628e-04 -3.14911951e-04
 -5.17354019e-04  1.01577706e-04 -5.06661231e-04 -6.66278744e-05
  9.77219597e-04  5.46369755e-04 -2.91577437e-04 -6.11993591e-04
 -5.66581492e-04  2.59935038e-04  5.58595259e-04  5.12019251e-05
  1.86463933e-04 -4.49512413e-04  7.24785784e-05 -6.72036659e-05
  2.28061874e-04  7.61935164e-05 -1.86970729e-04  7.62670255e-04
 -5.40844789e-04 -4.27220359e-04  7.85065019e-04  4.12776405e-04
  1.99165127e-04 -2.47585834e-04 -1.82281564e-04 -3.95555738e-04
  4.83856517e-04 -4.81060768e-04 -1.91160531e-04  2.03892826e-04
 -2.95112946e-04  1.56244332e-04 -3.62552050e-04 -3.40054112e-04
  7.75642387e-04 -2.66032769e-04  8.49692794e-04 -4.89494663e-04
 -2.15434053e-04 -2.09663562e-04 -4.19324776e-04 -5.17966028e-04
  5.62673564e-05  1.24812829e-05  1.84923083e-05  8.47501057e-05
 -5.62601927e-05 -3.54604895e-04  4.66069088e-04  4.33029173e-04
 -7.46839411e-04  1.66305171e-04  1.29593666e-04  3.18689665e-04
  5.75635197e-04  3.03099430e-04 -8.14777704e-05  3.87012430e-04]
```

[4.4.1.2] TFIDF weighted W2v Train data

In [110]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
```

```
tf_idf_matrix_train = model.fit_transform(x_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_train = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [111]:
```
# 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 = tfid

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(200) # 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_train[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_train.append(sent_vec)
    row += 1
```

```
100%|| 13755/13755 [02:23<00:00, 95.77it/s]
```

Cross Validation data

In [112]:
```
# 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 = tfid

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in th
row=0;
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(200) # 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_train[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_cv.append(sent_vec)
            row += 1
```

```
100%|| 5895/5895 [01:00<00:00, 101.32it/s]
```

Test data

```
In [113]: # 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 = tfid

          tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in
          row=0;
          for sent in tqdm(list_of_sentance_test): # for each review/sentence
              sent_vec = np.zeros(200) # 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_train[word]*(sent.count(word)/len(sent))
                      sent_vec += (vec * tf_idf)
                      weight_sum += tf_idf
              if weight_sum != 0:
                  sent_vec /= weight_sum
              tfidf_sent_vectors_test.append(sent_vec)
              row += 1
```

```
100%|| 8422/8422 [01:26<00:00, 97.68it/s]
```

6 [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 vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vector
```

```
    </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.

6.0.1 Loading various datasets using joblib

BoW

```
In [3]: x_train_bow = joblib.load('x_tr_bow100k.pkl')
        x_test_bow  = joblib.load('x_te_bow100k.pkl')
        x_cv_bow    = joblib.load('x_cv_bow100k.pkl')

        y_train = joblib.load('y_train.pkl')
        y_test  = joblib.load('y_test.pkl')
        y_cv    = joblib.load('y_cv.pkl')
```

TF-IDF

```
In [26]: x_train_tf = joblib.load('x_tr_tfidf100k.pkl')
         x_test_tf  = joblib.load('x_te_tfidf100k.pkl')
         x_cv_tf    = joblib.load('x_cv_tfidf100k.pkl')

         y_train = joblib.load('y_train.pkl')
         y_test  = joblib.load('y_test.pkl')
         y_cv    = joblib.load('y_cv.pkl')
```

Average W2V

```
In [37]: sent_vectors_train = joblib.load('sent_vectors_train_100k.pkl')
         sent_vectors_test  = joblib.load('sent_vectors_test_100k.pkl')
         sent_vectors_cv    = joblib.load('sent_vectors_cv_100k.pkl')

         y_train = joblib.load('y_train.pkl')
         y_test  = joblib.load('y_test.pkl')
         y_cv    = joblib.load('y_cv.pkl')
```

TF-IDF W2V

```
In [38]: tfidf_sent_vectors_train = joblib.load('tfidf_sent_vectors_train_100k.pkl')
         tfidf_sent_vectors_test  = joblib.load('tfidf_sent_vectors_test_100k.pkl')
         tfidf_sent_vectors_cv    = joblib.load('tfidf_sent_vectors_cv_100k.pkl')

         y_train = joblib.load('y_train.pkl')
         y_test  = joblib.load('y_test.pkl')
         y_cv    = joblib.load('y_cv.pkl')
```

6.0.2 Feature Importance

7 Applying SVM

7.1 [5.1] Linear SVM

7.1.1 [5.1.1] Applying Linear SVM on BOW, SET 1

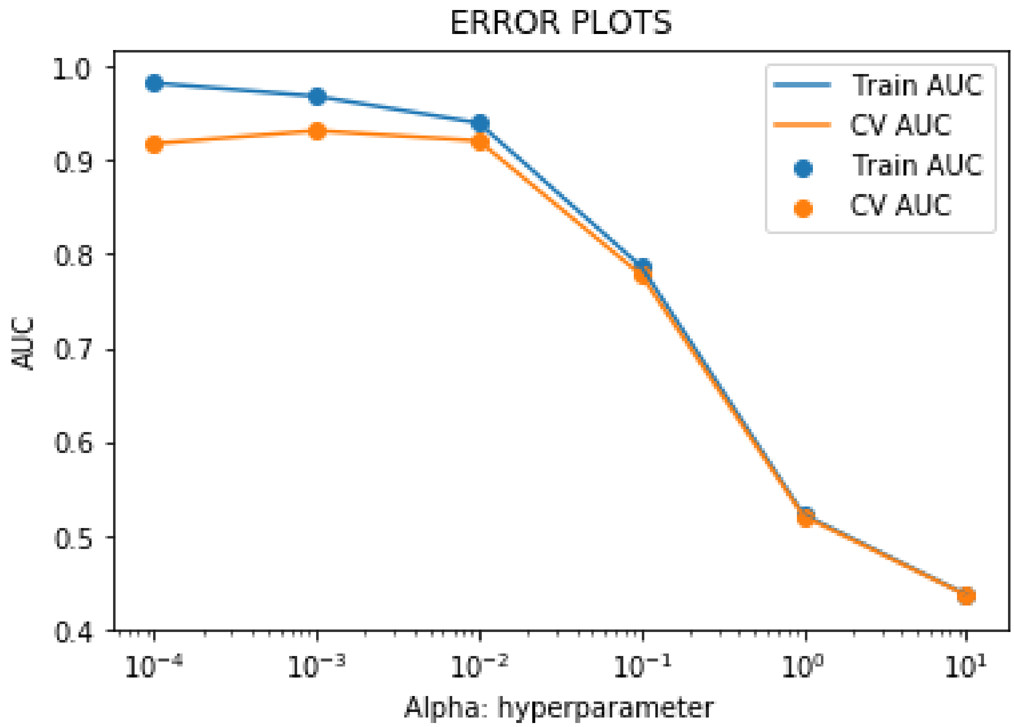
7.1.2 Hyperparameter tunin using GridSearchCV

Here we are using SGDClassfier with hinge loss which almost works like SVM

```
In [25]: alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10]
         parameters = {'alpha': alpha}
         grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l2'), parameters, cv=3, scorin
         grid.fit(x_train_bow, y_train)

         print("best alpha = ", grid.best_params_)
         print("Accuracy on train data = ", grid.best_score_*100)
         a = grid.best_params_
         optimal_a1 = a.get('alpha')

best alpha =  {'alpha': 0.001}
Accuracy on train data =  93.12955584948898
```

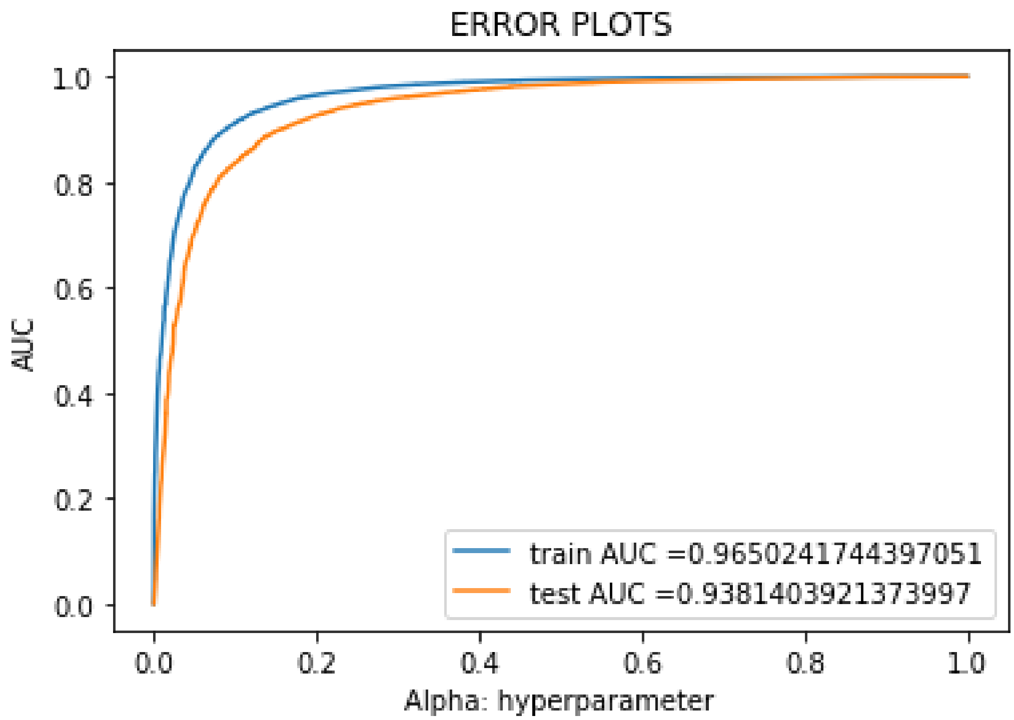
```
In [26]: train_auc_bow = grid.cv_results_['mean_train_score']
         cv_auc_bow = grid.cv_results_['mean_test_score']

         plt.plot(alpha, train_auc_bow, label='Train AUC')
         plt.scatter(alpha, train_auc_bow, label='Train AUC')
         plt.plot(alpha, cv_auc_bow, label='CV AUC')
```

```
plt.scatter(alpha, cv_auc_bow, label='CV AUC')

plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.xscale('log')
plt.show()
```



Testing with test data

```
In [27]: clf = SGDClassifier(loss='hinge',alpha = optimal_a1, penalty='l2')
         calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
         calibrator.fit(x_train_bow, y_train)

         train_fpr_bow, train_tpr_bow, thresholds_bow = roc_curve(y_train, calibrator.predict_p
         test_fpr_bow, test_tpr_bow, thresholds_bow = roc_curve(y_test, calibrator.predict_prol

         plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, trai
         plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC ="+str(auc(test_fpr_bow, test_tp
         plt.legend()
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



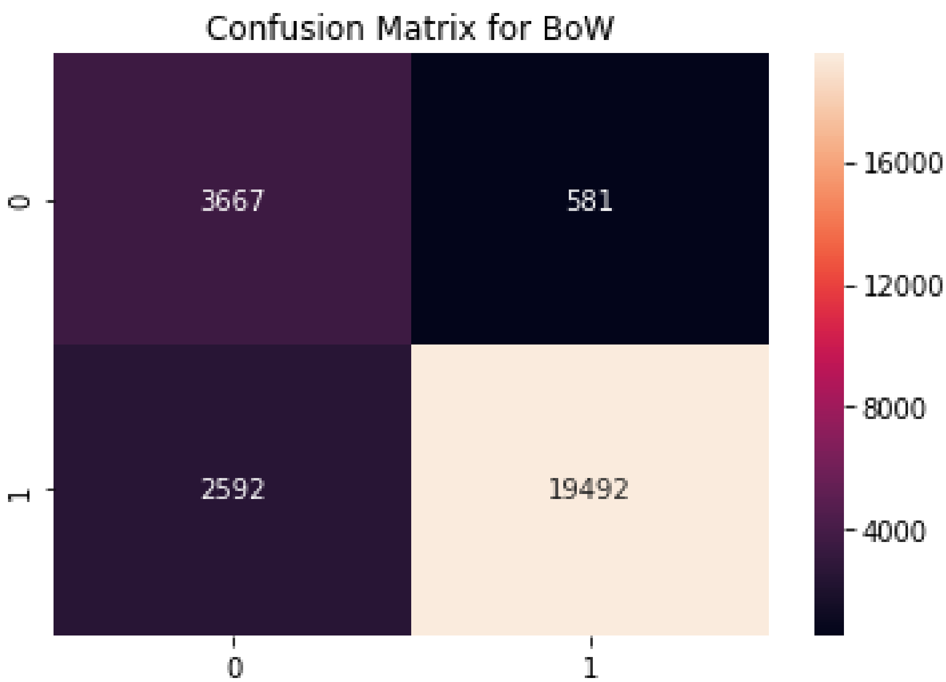
7.1.3 Calculating Confusion Matrix

```
In [28]: clf = SGDClassifier(loss='hinge',penalty='l2',alpha = optimal_a1, class_weight='balanc
         clf.fit(x_train_bow,y_train)
         predb = clf.predict(x_test_bow)

         acc_b = accuracy_score(y_test, predb) * 100
         pre_b = precision_score(y_test, predb) * 100
         rec_b = recall_score(y_test, predb) * 100
         f1_b  = f1_score(y_test, predb) * 100

         print('\nAccuracy = %f%%' % (acc_b))
         print('\nprecision= %f%%' % (pre_b))
         print('\nrecall    = %f%%' % (rec_b))
         print('\nF1-Score = %f%%' % (f1_b))
```

```
Accuracy = 87.950023%

precision= 97.105565%

recall    = 88.262996%
```

F1-Score = 92.473373%

```
In [29]: cm = confusion_matrix(y_test,predb)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix for BoW')
         plt.show()
```



7.1.4 Feature Importance

```
In [30]: def show_most_informative_features(vectorizer, clf, n=10):
             feature_names = vectorizer.get_feature_names()
             coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
             top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
             print("\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))

         show_most_informative_features(count_vect,clf)
         #Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informat
```

```
                Negative                                        Positive
---------------------------------------------------------------------------------------------
        -1.1031         disappointing                                   0.9288          delicious
```

```
-0.9491        worst                                0.8818        perfect
-0.8483        disappointed                         0.8611        amazing
-0.8334        disappointment                       0.8473        excellent
-0.7542        bland                                0.7450        loves
-0.7510        awful                                0.7419        great
-0.7420        terrible                             0.7314        best
-0.6804        threw                                0.6756        awesome
-0.6730        unfortunately                        0.6744        hooked
-0.6574        sadly                                0.6678        pleased
```

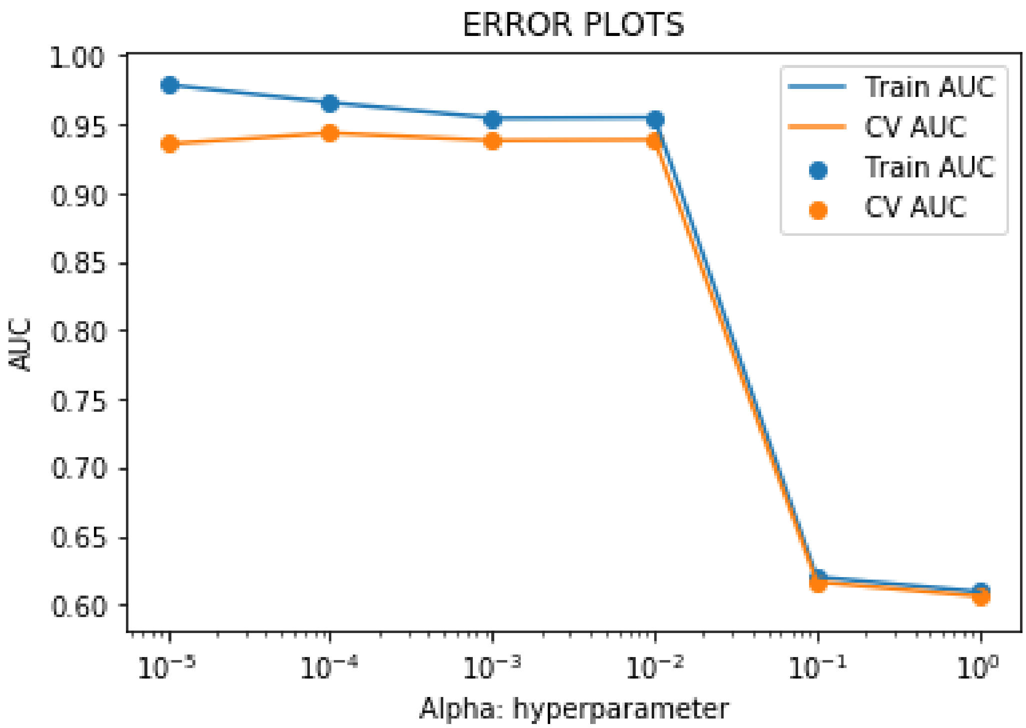
7.1.5 [5.1.2] Applying Linear SVM on TFIDF, SET 2

7.1.6 Hyperparameter tuning using GridSearchCV

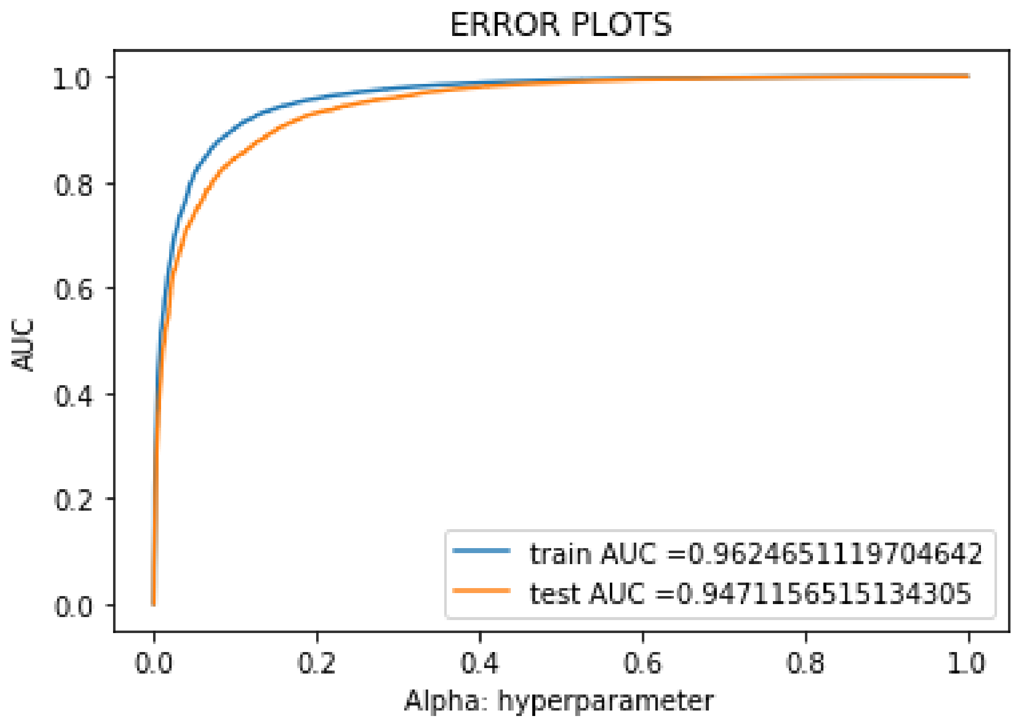
```
In [31]: alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]
         parameters = {'alpha': alpha}
         grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l2'), parameters, cv=3, scori
         grid.fit(x_train_tf, y_train)

         print("best alpha = ", grid.best_params_)
         print("Accuracy on train data = ", grid.best_score_*100)
         a = grid.best_params_
         optimal_a2 = a.get('alpha')
```

```
best alpha =  {'alpha': 0.0001}
Accuracy on train data =  94.3725832910788
```

```
In [32]: train_auc_tf = grid.cv_results_['mean_train_score']
         cv_auc_tf = grid.cv_results_['mean_test_score']

         plt.plot(alpha, train_auc_tf, label='Train AUC')
         plt.scatter(alpha, train_auc_tf, label='Train AUC')
         plt.plot(alpha, cv_auc_tf, label='CV AUC')
         plt.scatter(alpha, cv_auc_tf, label='CV AUC')

         plt.legend()
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.xscale('log')
         plt.show()
```



In [33]:
```
clf = SGDClassifier(loss='hinge',alpha = optimal_a2, penalty='l2')
calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
calibrator.fit(x_train_tf, y_train)

train_fpr_tf, train_tpr_tf, thresholds_tf = roc_curve(y_train, calibrator.predict_prol
test_fpr_tf, test_tpr_tf, thresholds_tf = roc_curve(y_test, calibrator.predict_proba(:

plt.plot(train_fpr_tf, train_tpr_tf, label="train AUC ="+str(auc(train_fpr_tf, train_
plt.plot(test_fpr_tf, test_tpr_tf, label="test AUC ="+str(auc(test_fpr_tf, test_tpr_t
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



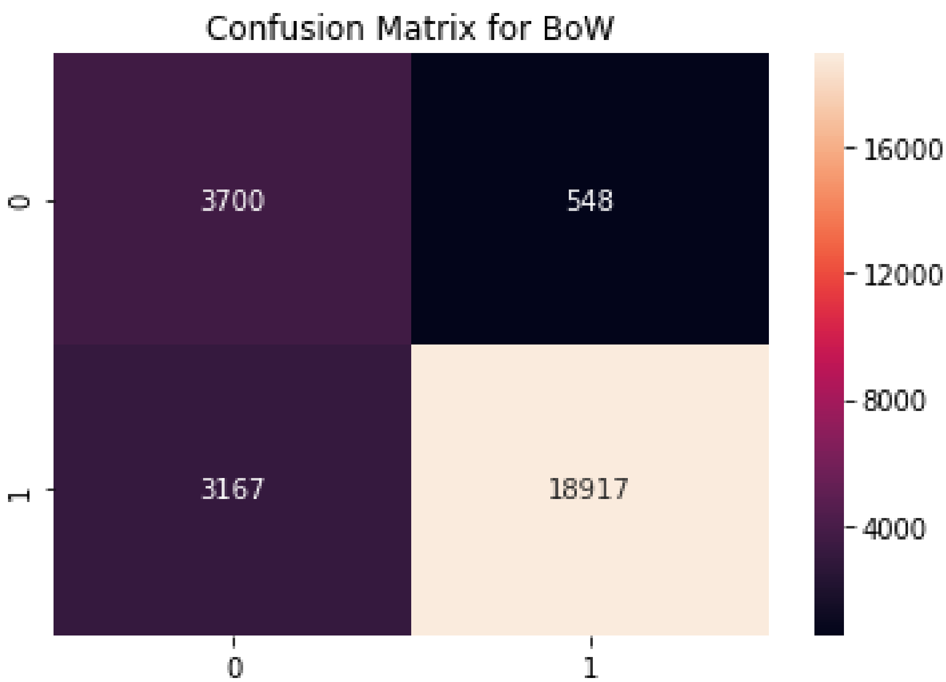
7.1.7 Calculating Confusion Matrix

```
In [34]: clf = SGDClassifier(loss='hinge',penalty='l1',alpha = optimal_a2, class_weight='balanc
         clf.fit(x_train_tf, y_train)
         predb = clf.predict(x_test_tf)

         acc_tf = accuracy_score(y_test, predb) * 100
         pre_tf = precision_score(y_test, predb) * 100
         rec_tf = recall_score(y_test, predb) * 100
         f1_tf  = f1_score(y_test, predb) * 100

         print('\nAccuracy = %f%%' % (acc_tf))
         print('\nprecision= %f%%' % (pre_tf))
         print('\nrecall    = %f%%' % (rec_tf))
         print('\nF1-Score = %f%%' % (f1_tf))
```

```
Accuracy = 85.891691%

precision= 97.184690%

recall    = 85.659301%
```

```
F1-Score = 91.058750%
```

```
In [35]: cm = confusion_matrix(y_test,predb)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix for BoW')
         plt.show()
```



7.1.8 Feature Importance

```
In [36]: def show_most_informative_features(vectorizer, clf, n=10):
             feature_names = vectorizer.get_feature_names()
             coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
             top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
             print("\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))

         show_most_informative_features(tf_idf,clf)
         #Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informat
```

```
                Negative                                                Positive
-----------------------------------------------------------------------------------------------
        -8.5233         disappointing                                   8.6825          great
```

```
-7.9091        cancelled                                       8.4323        delicious
-7.4764        worst                                           8.0965        perfect
-6.2319        terrible                                        7.4714        best
-6.0844        not                                             7.1161        amazing
-5.9477        disappointment                                  5.9856        loves
-5.7856        disappointed                                    5.8308        excellent
-5.5398        whatsoever                                      5.6244        wonderful
-5.3891        awful                                           5.6224        highly
-5.2084        horrible                                        5.2478        good
```

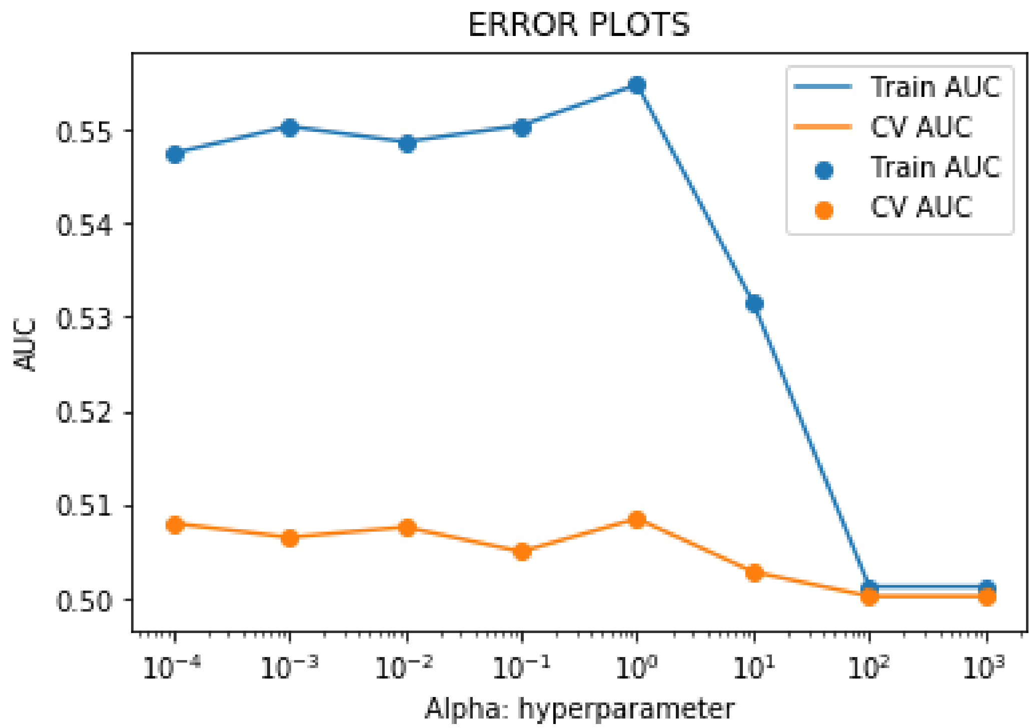
7.1.9 [5.1.3] Applying Linear SVM on AVG W2V, SET 3

7.1.10 Hyperparameter tunin using GridSearchCV

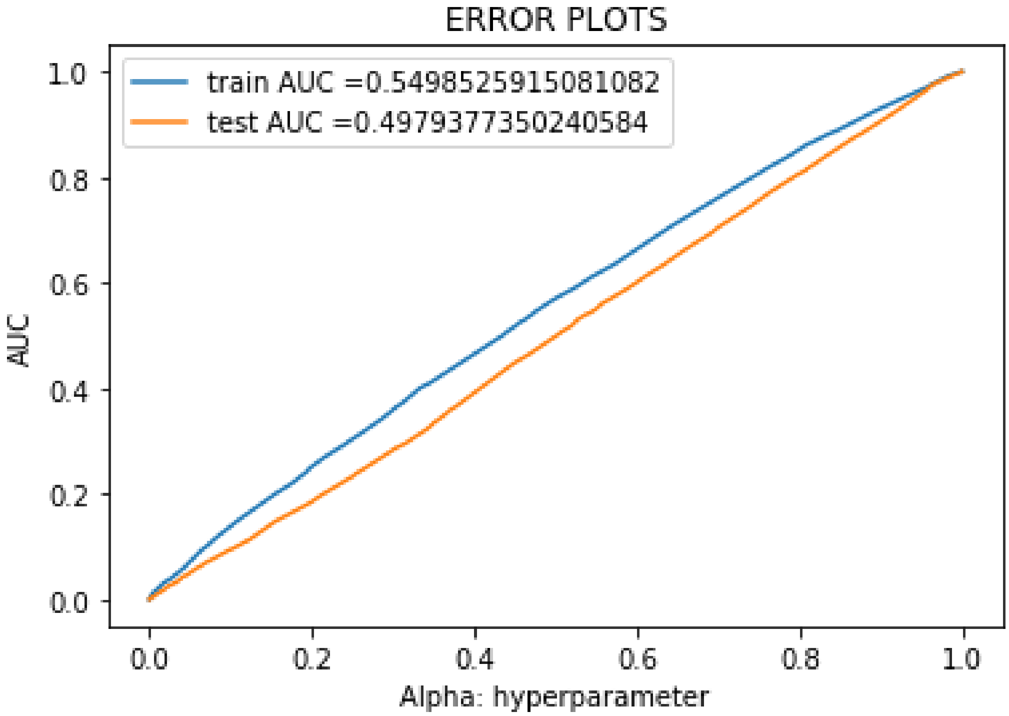
```
In [48]: alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'alpha': alpha}
         grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l2'), parameters, cv=3, scorin
         grid.fit(sent_vectors_train, y_train)

         print("best alpha = ", grid.best_params_)
         print("Accuracy on train data = ", grid.best_score_*100)
         a = grid.best_params_
         optimal_a3 = a.get('alpha')
```

```
best alpha =  {'alpha': 1}
Accuracy on train data =  50.85062079929457
```

```
In [49]: train_auc_aw2v = grid.cv_results_['mean_train_score']
         cv_auc_aw2v = grid.cv_results_['mean_test_score']

         plt.plot(alpha, train_auc_aw2v, label='Train AUC')
         plt.scatter(alpha, train_auc_aw2v, label='Train AUC')
         plt.plot(alpha, cv_auc_aw2v, label='CV AUC')
         plt.scatter(alpha, cv_auc_aw2v, label='CV AUC')

         plt.legend()
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.xscale('log')
         plt.show()
```

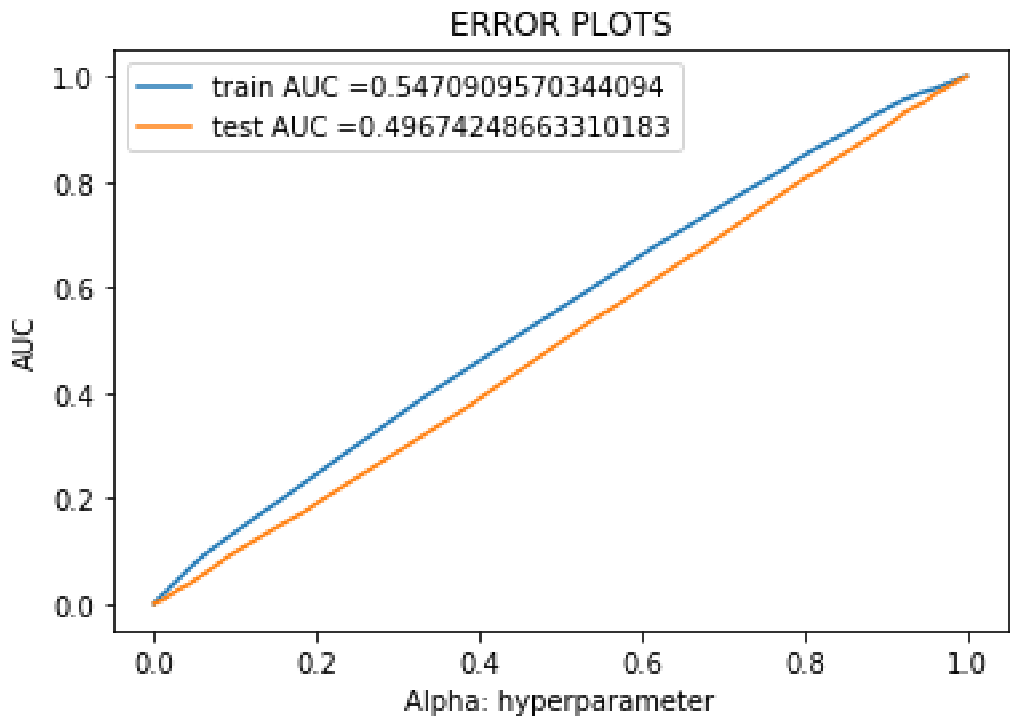


In [50]:
```
clf = SGDClassifier(loss='hinge',alpha = optimal_a3, penalty='l2')
calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
calibrator.fit(sent_vectors_train, y_train)

train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(y_train, calibrator.predic
test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(y_test, calibrator.predict_

plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v, 
plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test_
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Testing with test data

In [52]:
```
clf = SGDClassifier(loss='hinge', penalty='l2',alpha = optimal_a3)
calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
calibrator.fit(sent_vectors_train, y_train)

train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(y_train, calibrator.predi
test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(y_test, calibrator.predict_

plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v, t
plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test_
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



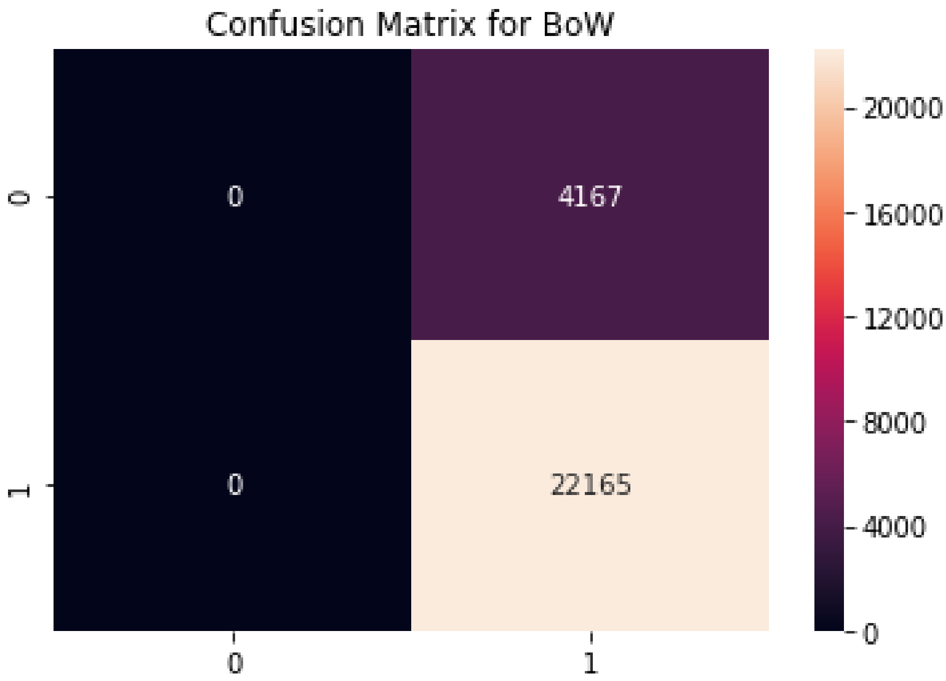
7.1.11 Calculating Confusion Matrix

```
In [53]: clf = SGDClassifier(loss='hinge',penalty='l2',alpha = optimal_a3, class_weight="balanc
         clf.fit(sent_vectors_train, y_train)
         predb = clf.predict(sent_vectors_test)

         acc_aw2v = accuracy_score(y_test, predb) * 100
         pre_aw2v = precision_score(y_test, predb) * 100
         rec_aw2v = recall_score(y_test, predb) * 100
         f1_aw2v  = f1_score(y_test, predb) * 100

         print('\nAccuracy = %f%%' % (acc_aw2v))
         print('\nprecision= %f%%' % (pre_aw2v))
         print('\nrecall    = %f%%' % (rec_aw2v))
         print('\nF1-Score = %f%%' % (f1_aw2v))
```

```
Accuracy = 84.175148%

precision= 84.175148%

recall    = 100.000000%
```

```
F1-Score = 91.407716%
```

```
In [79]: cm = confusion_matrix(y_test,predb)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix for BoW')
         plt.show()
```



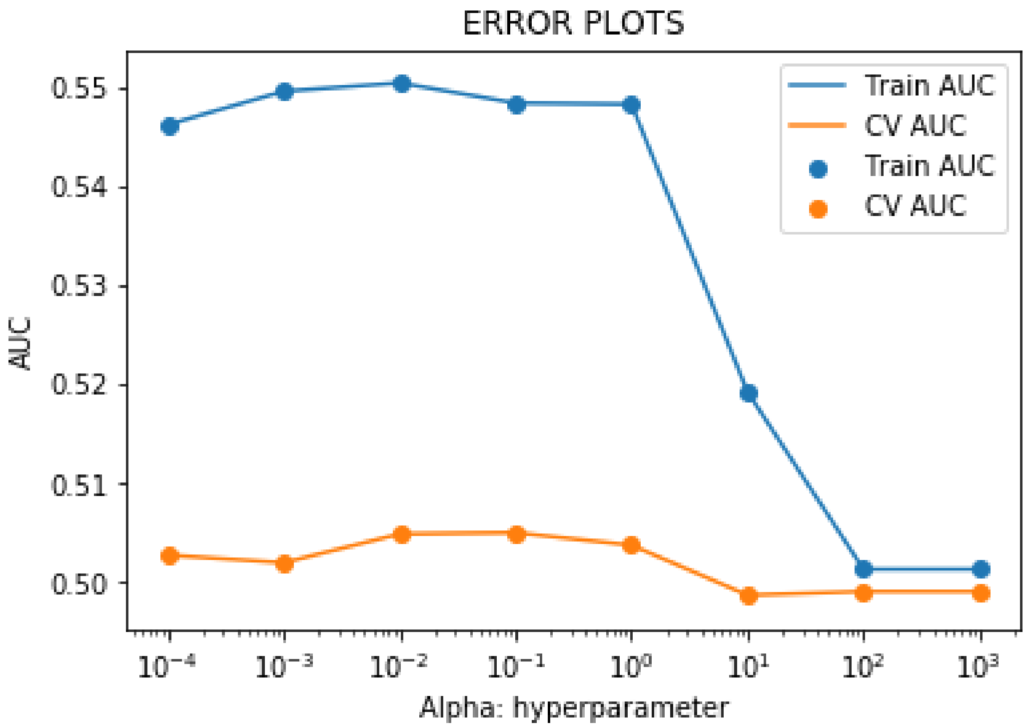
7.1.12 [5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

7.1.13 Hyperparameter tuning using GridSearchCV

```
In [56]: alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'alpha': alpha}
         grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l2'), parameters, cv=3, scorin
         grid.fit(tfidf_sent_vectors_train, y_train)

         print("best alpha = ", grid.best_params_)
         print("Accuracy on train data = ", grid.best_score_*100)
         a = grid.best_params_
         optimal_a4 = a.get('alpha')
```

```
best alpha =  {'alpha': 0.1}
Accuracy on train data =  50.493469155031335
```

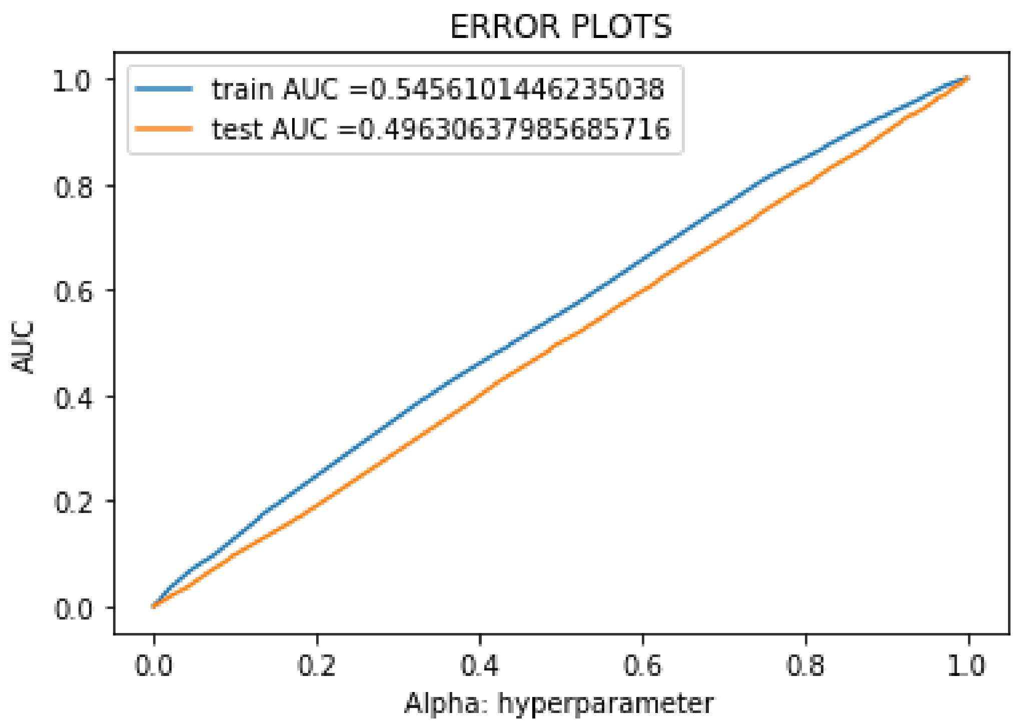
In [57]:
```
train_auc_tw2v = grid.cv_results_['mean_train_score']
cv_auc_tw2v = grid.cv_results_['mean_test_score']

plt.plot(alpha, train_auc_tw2v, label='Train AUC')
plt.scatter(alpha, train_auc_tw2v, label='Train AUC')
plt.plot(alpha, cv_auc_tw2v, label='CV AUC')
plt.scatter(alpha, cv_auc_tw2v, label='CV AUC')

plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.xscale('log')
plt.show()
```



Testing with test data

In [58]:
```
clf = SGDClassifier(loss='hinge',alpha = optimal_a4, penalty='l2')
calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
calibrator.fit(tfidf_sent_vectors_train, y_train)

train_fpr_tw2v, train_tpr_tw2v, thresholds_tw2v = roc_curve(y_train, calibrator.predic
test_fpr_tw2v, test_tpr_tw2v, thresholds_tw2v = roc_curve(y_test, calibrator.predict_
```

```
plt.plot(train_fpr_tw2v, train_tpr_tw2v, label="train AUC ="+str(auc(train_fpr_tw2v, t
plt.plot(test_fpr_tw2v, test_tpr_tw2v, label="test AUC ="+str(auc(test_fpr_tw2v, test_
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



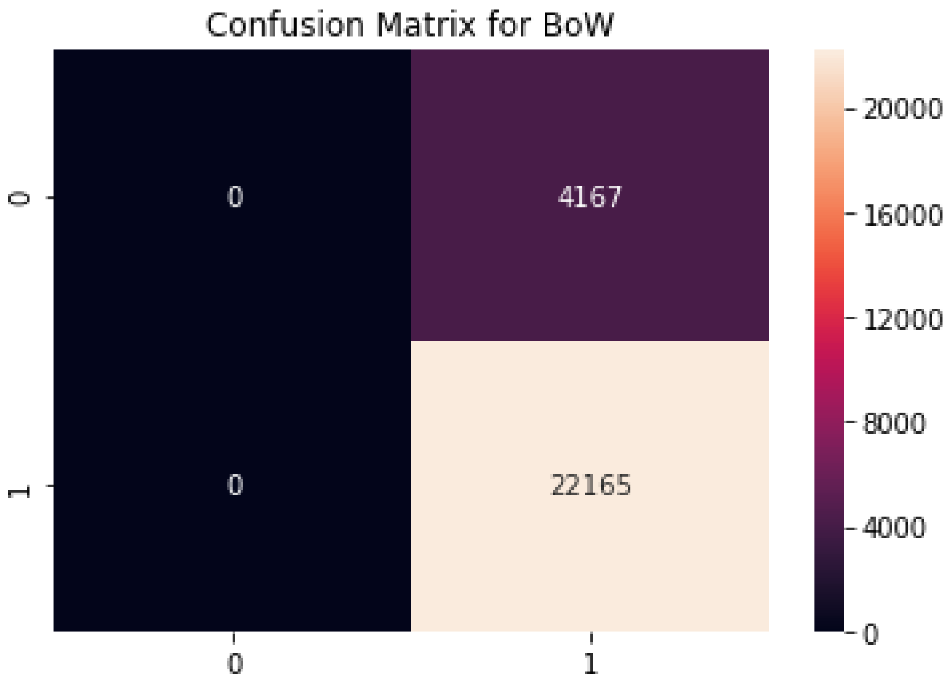
Calculating Confusion Matrix

```
In [59]: clf = SGDClassifier(loss='hinge',penalty='l2',alpha = optimal_a4, class_weight='balanc
         clf.fit(tfidf_sent_vectors_train, y_train)
         predb = clf.predict(tfidf_sent_vectors_test)

         acc_tw2v = accuracy_score(y_test, predb) * 100
         pre_tw2v = precision_score(y_test, predb) * 100
         rec_tw2v = recall_score(y_test, predb) * 100
         f1_tw2v  = f1_score(y_test, predb) * 100

         print('\nAccuracy = %f%%' % (acc_tw2v))
         print('\nprecision= %f%%' % (pre_tw2v))
         print('\nrecall    = %f%%' % (rec_tw2v))
         print('\nF1-Score = %f%%' % (f1_tw2v))
```

```
Accuracy = 84.175148%
```

```
precision= 84.175148%

recall    = 100.000000%

F1-Score = 91.407716%
```

```
In [60]: cm = confusion_matrix(y_test,predb)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix for BoW')
         plt.show()
```



7.2 [5.2] RBF SVM

7.2.1 [5.2.1] Applying RBF SVM on BOW, SET 1

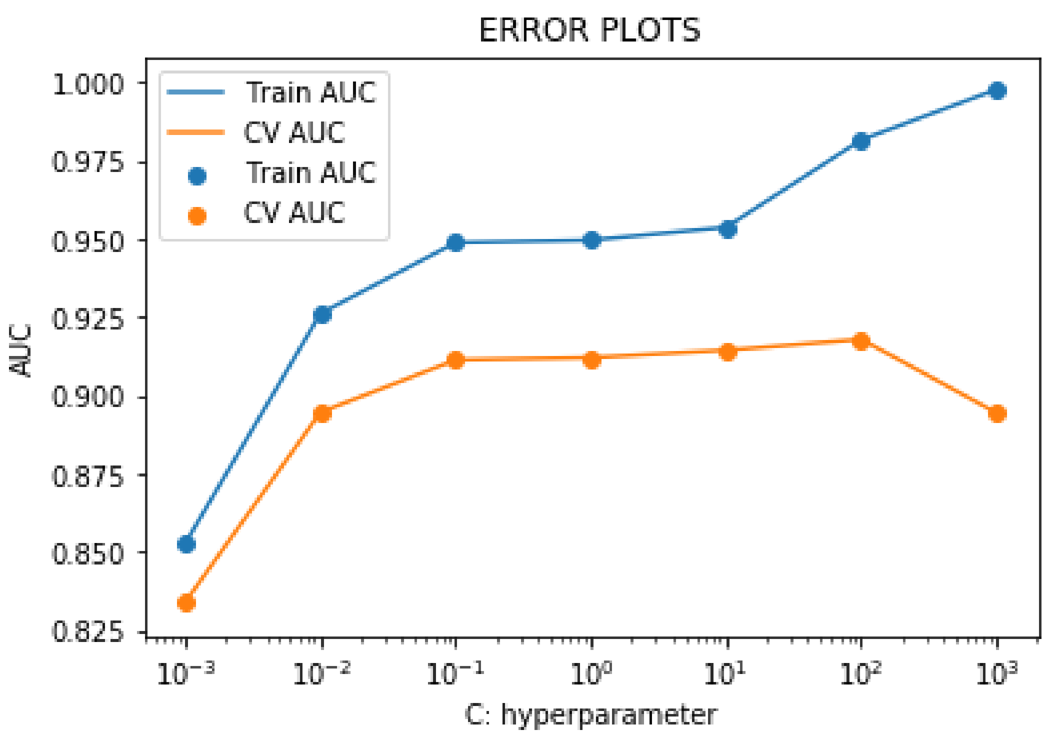
7.2.2 Hyperparameter tuning using GridSearchCV

```
In [115]: C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
          parameters = {'C': C}
          grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=-
          grid.fit(x_train_bow, y_train)

          print("best C = ", grid.best_params_)
          print("Accuracy on train data = ", grid.best_score_*100)
```

```
a = grid.best_params_
optimal_a5 = a.get('C')
```

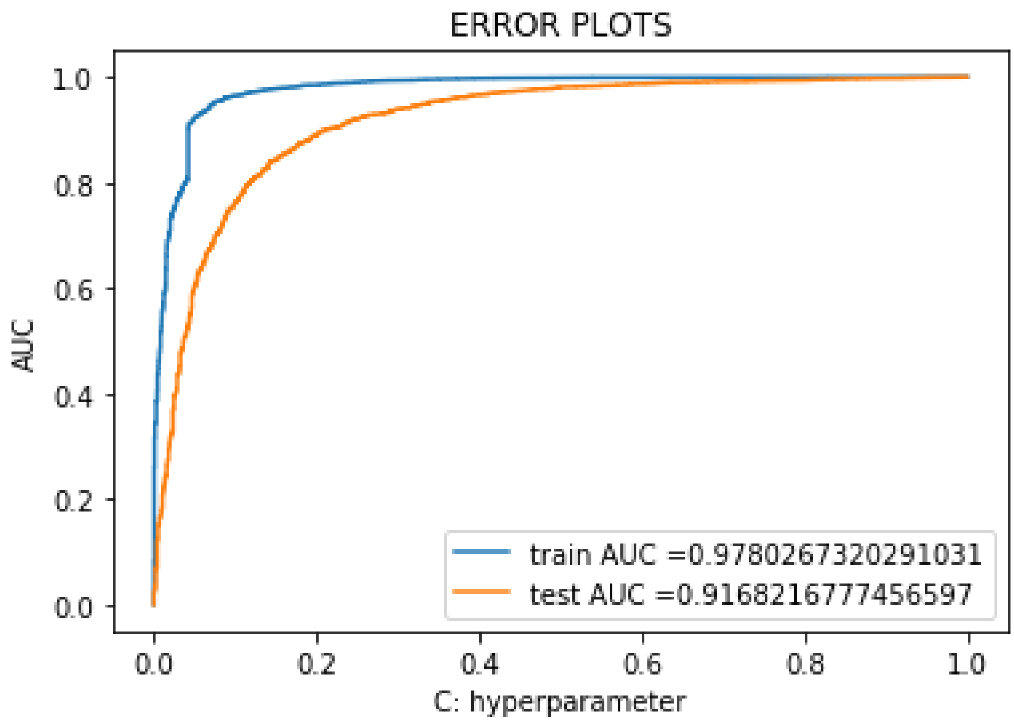
```
best C =  {'C': 100}
Accuracy on train data =  91.79191109269469
```

In [116]:
```
train_auc_bow = grid.cv_results_['mean_train_score']
cv_auc_bow    = grid.cv_results_['mean_test_score']

plt.plot(C, train_auc_bow, label='Train AUC')
plt.scatter(C, train_auc_bow, label='Train AUC')

plt.plot(C, cv_auc_bow, label='CV AUC')
plt.scatter(C, cv_auc_bow, label='CV AUC')

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



Testing

In [117]:
```
clf = SVC(kernel='rbf', C = optimal_a5, probability=True)
clf.fit(x_train_bow, y_train)

train_fpr_bow, train_tpr_bow, thresholds_bow = roc_curve(y_train, clf.predict_proba(x
test_fpr_bow, test_tpr_bow, thresholds_bow = roc_curve(y_test, clf.predict_proba(x_te

plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, tra
plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC ="+str(auc(test_fpr_bow, test_tp
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

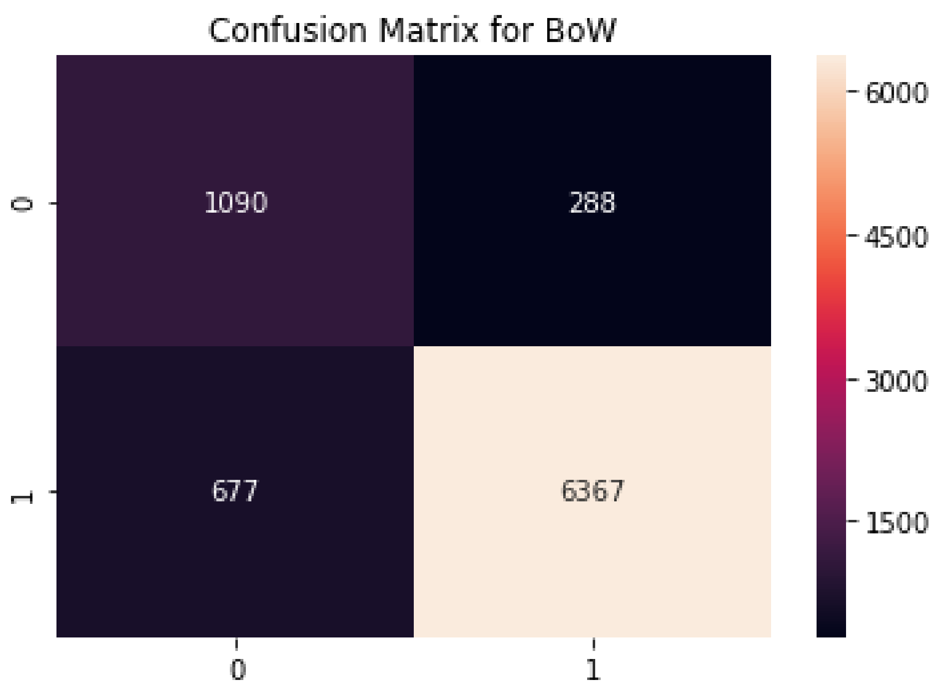


In [118]:
```
clf = SVC(kernel='rbf', C = optimal_a5, class_weight='balanced')
clf.fit(x_train_bow,y_train)

predb1 = clf.predict(x_test_bow)

acc_b1 = accuracy_score(y_test, predb1) * 100
pre_b1 = precision_score(y_test, predb1) * 100
rec_b1 = recall_score(y_test, predb1) * 100
f1_b1 = f1_score(y_test, predb1) * 100
```

```
print('\nAccuracy = %f%%' % (acc_b1))
print('\nprecision= %f%%' % (pre_b1))
print('\nrecall    = %f%%' % (rec_b1))
print('\nF1-Score = %f%%' % (f1_b1))
```

```
Accuracy = 88.411304%

precision= 95.561513%

recall    = 90.442628%

F1-Score = 92.931634%
```

In [168]:
```
cm = confusion_matrix(y_test,predb1)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix for BoW')
plt.show()
```



7.2.3 [5.2.2] Applying RBF SVM on TFIDF, SET 2

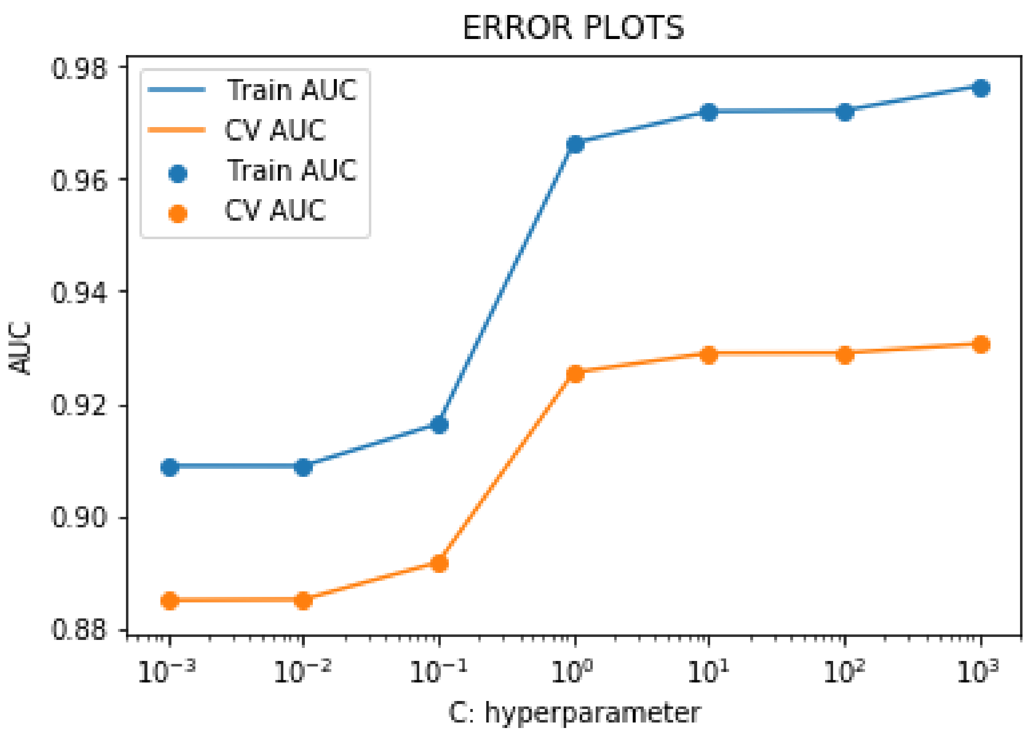
7.2.4 Hyperparameter tuning using GridSearchCV

In [119]:
```
C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
parameters = {'C': C}
```

```
grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=-
grid.fit(x_train_tf, y_train)

print("best C = ", grid.best_params_)
print("Accuracy on train data = ", grid.best_score_*100)
a = grid.best_params_
optimal_a6 = a.get('C')
```

```
best C =  {'C': 1000}
Accuracy on train data =  93.06359300499562
```

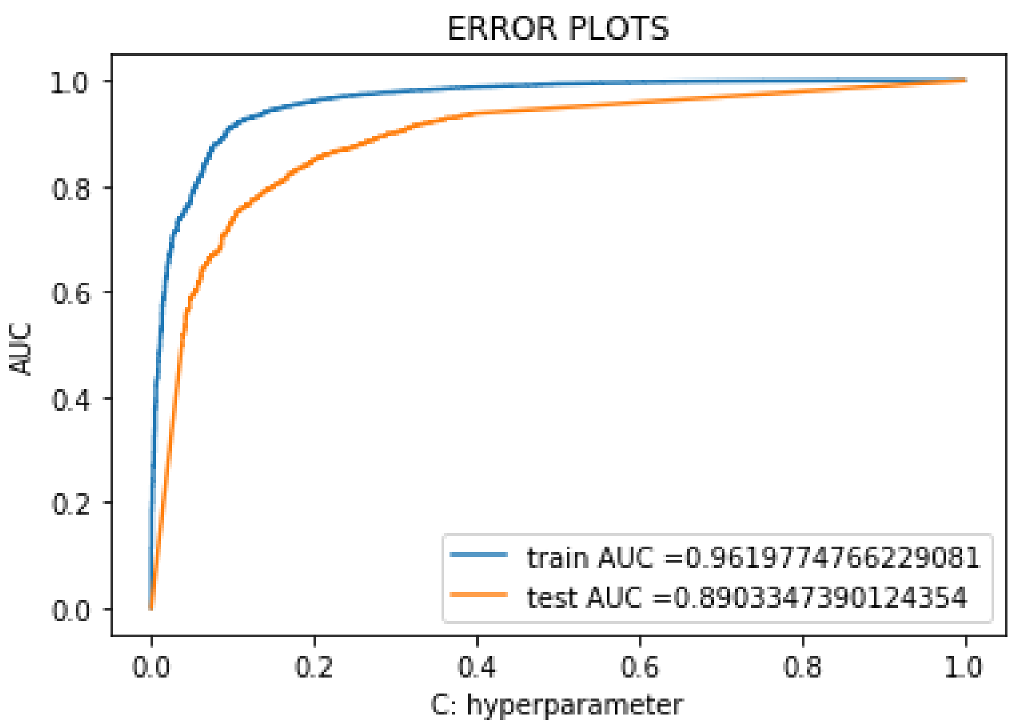
In [120]:
```
train_auc_tf = grid.cv_results_['mean_train_score']
cv_auc_tf     = grid.cv_results_['mean_test_score']

plt.plot(C, train_auc_tf, label='Train AUC')
plt.scatter(C, train_auc_tf, label='Train AUC')

plt.plot(C, cv_auc_tf, label='CV AUC')
plt.scatter(C, cv_auc_tf, label='CV AUC')

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



Testing

```
In [121]: optimal_a6 = 1
```

```
In [122]: clf = SVC(kernel='rbf', C = optimal_a6, probability=True)
          clf.fit(x_train_tf, y_train)

          train_fpr_tf, train_tpr_tf, thresholds_tf = roc_curve(y_train, clf.predict_proba(x_t
          test_fpr_tf, test_tpr_tf, thresholds_tf = roc_curve(y_test, clf.predict_proba(x_test_

          plt.plot(train_fpr_tf, train_tpr_tf, label="train AUC ="+str(auc(train_fpr_tf, train_
          plt.plot(test_fpr_tf, test_tpr_tf, label="test AUC ="+str(auc(test_fpr_tf, test_tpr_t
          plt.legend()
          plt.xlabel("C: hyperparameter")
          plt.ylabel("AUC")
          plt.title("ERROR PLOTS")
          plt.show()
```



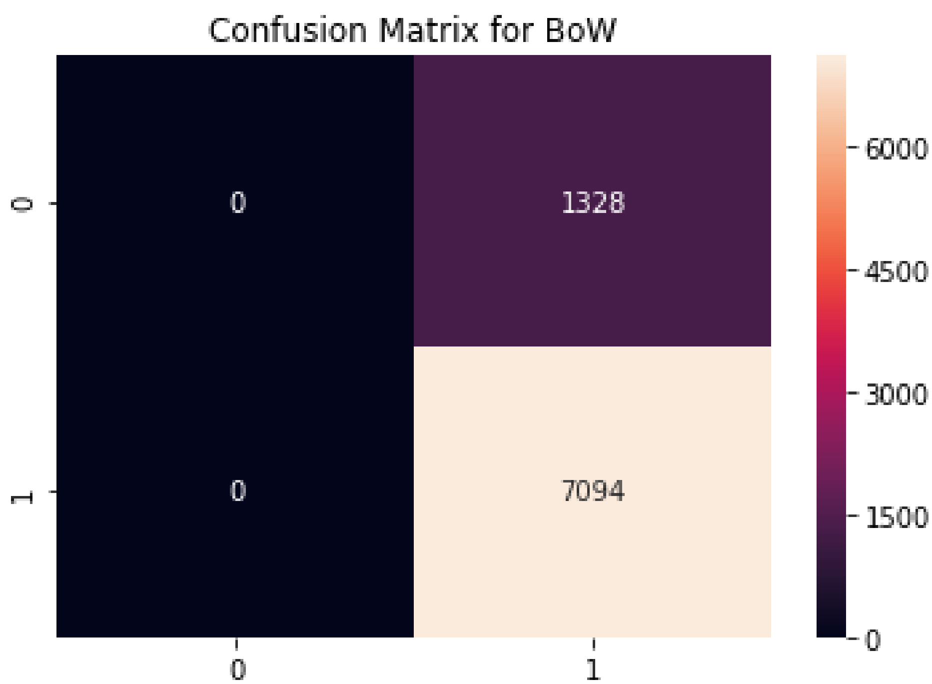
Calculating Confusion Matrix

```
In [123]: clf = SVC(kernel='rbf', C = optimal_a6, class_weight='balanced')
          clf.fit(x_train_tf, y_train)
```

```
predb1 = clf.predict(x_test_tf)

acc_tf1 = accuracy_score(y_test, predb1) * 100
pre_tf1 = precision_score(y_test, predb1) * 100
rec_tf1 = recall_score(y_test, predb1) * 100
f1_tf1 = f1_score(y_test, predb1) * 100

print('\nAccuracy = %f%%' % (acc_tf1))
print('\nprecision= %f%%' % (pre_tf1))
print('\nrecall    = %f%%' % (rec_tf1))
print('\nF1-Score = %f%%' % (f1_tf1))
```

```
Accuracy = 84.231774%

precision= 84.231774%

recall    = 100.000000%

F1-Score = 91.441093%
```

```
In [124]: cm = confusion_matrix(y_test,predb1)
          sns.heatmap(cm, annot=True,fmt='d')
          plt.title('Confusion Matrix for BoW')
          plt.show()
```



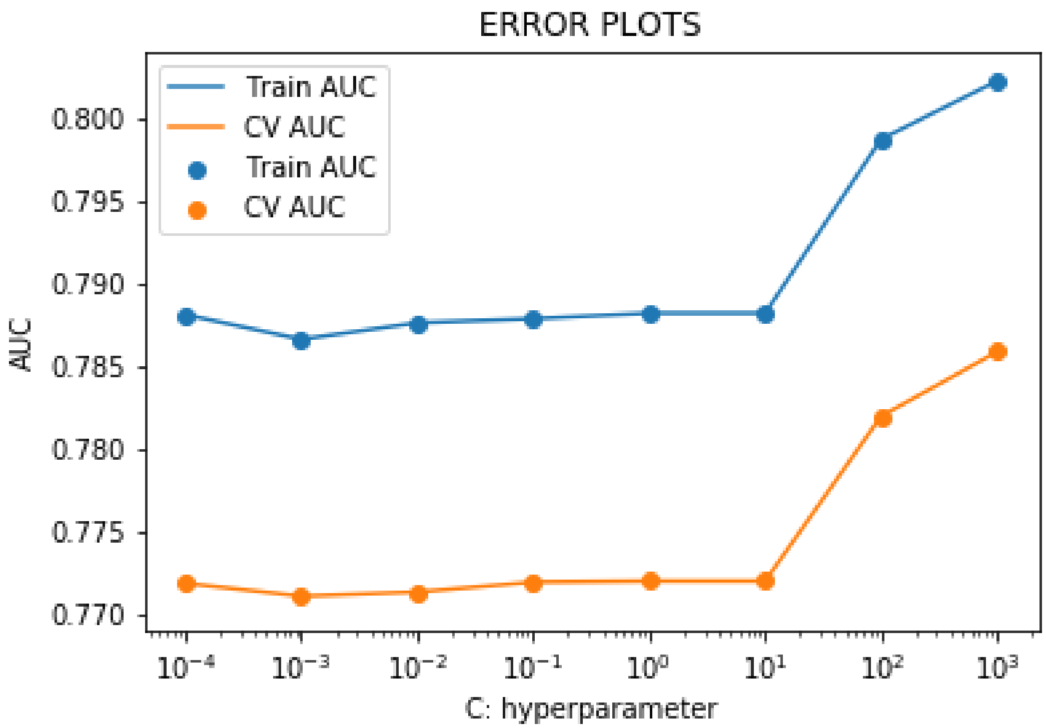
7.2.5 [5.2.3] Applying RBF SVM on AVG W2V, SET 3

7.2.6 Hyperparameter tuning using GridSearchCV

```
In [130]: C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
          parameters = {'C': C}
          grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=-
          grid.fit(sent_vectors_train, y_train)

          print("best C = ", grid.best_params_)
          print("Accuracy on train data = ", grid.best_score_*100)
          a = grid.best_params_
          optimal_a7 = a.get('C')
```

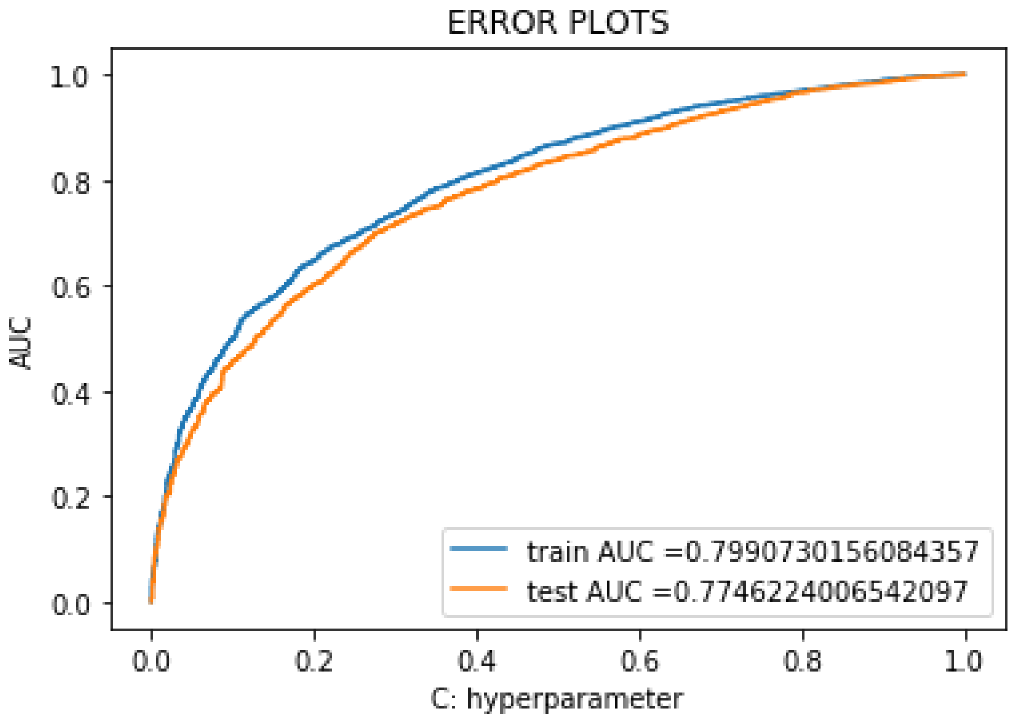
```
best C =  {'C': 1000}
Accuracy on train data =  78.58427699843556
```

```
In [131]: train_auc_aw2v = grid.cv_results_['mean_train_score']
          cv_auc_aw2v    = grid.cv_results_['mean_test_score']

          plt.plot(C, train_auc_aw2v, label='Train AUC')
          plt.scatter(C, train_auc_aw2v, label='Train AUC')

          plt.plot(C, cv_auc_aw2v, label='CV AUC')
          plt.scatter(C, cv_auc_aw2v, label='CV AUC')

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



Testing

In [132]:
```
clf = SVC(kernel='rbf', C = optimal_a7, probability=True)
clf.fit(sent_vectors_train, y_train)

train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(y_train, clf.predict_prob
test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(y_test, clf.predict_proba(s

plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v,
plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Calculating Confusion Matrix

```
In [134]: clf = SVC(kernel='rbf', C = optimal_a7, class_weight='balanced')
          clf.fit(sent_vectors_train,y_train)

          predb1 = clf.predict(sent_vectors_test)

          acc_aw2v1 = accuracy_score(y_test, predb1) * 100
          pre_aw2v1 = precision_score(y_test, predb1) * 100
          rec_aw2v1 = recall_score(y_test, predb1) * 100
          f1_aw2v1 = f1_score(y_test, predb1) * 100

          print('\nAccuracy = %f%%' % (acc_aw2v1))
          print('\nprecision= %f%%' % (pre_aw2v1))
          print('\nrecall    = %f%%' % (rec_aw2v1))
          print('\nF1-Score = %f%%' % (f1_aw2v1))
```

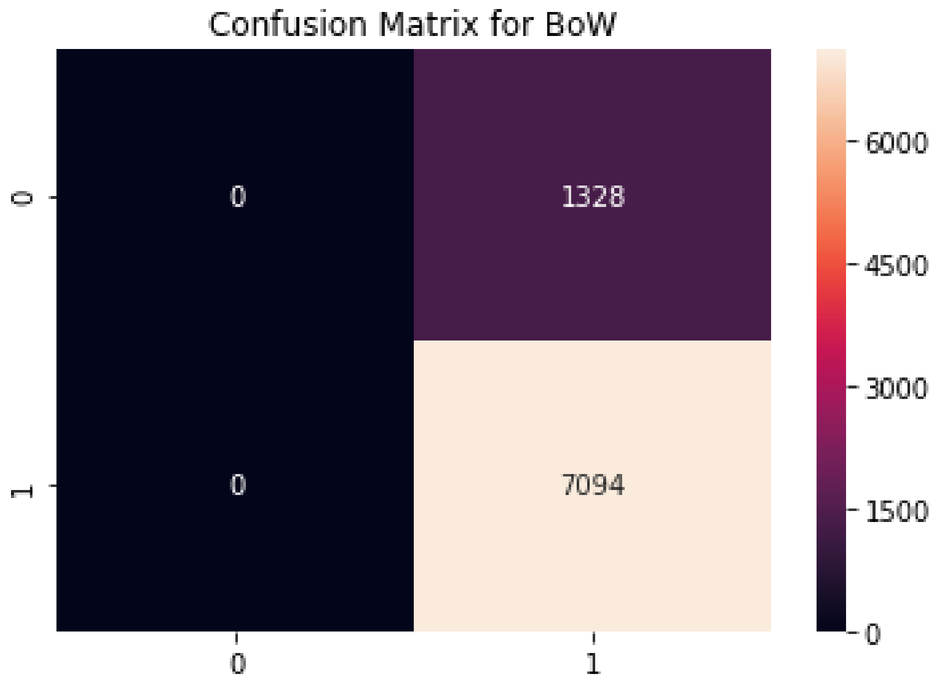
```
Accuracy = 15.768226%

precision= 0.000000%

recall    = 0.000000%
```

```
F1-Score = 0.000000%
```

```
In [135]: clf = SVC(kernel='rbf', C = optimal_a7)
          clf.fit(sent_vectors_train,y_train)

          predb1 = clf.predict(sent_vectors_test)

          acc_aw2v1 = accuracy_score(y_test, predb1) * 100
          pre_aw2v1 = precision_score(y_test, predb1) * 100
          rec_aw2v1 = recall_score(y_test, predb1) * 100
          f1_aw2v1 = f1_score(y_test, predb1) * 100

          print('\nAccuracy = %f%%' % (acc_aw2v1))
          print('\nprecision= %f%%' % (pre_aw2v1))
          print('\nrecall   = %f%%' % (rec_aw2v1))
          print('\nF1-Score = %f%%' % (f1_aw2v1))
```

```
Accuracy = 84.231774%

precision= 84.231774%

recall   = 100.000000%

F1-Score = 91.441093%
```

```
In [136]: cm = confusion_matrix(y_test,predb1)
          sns.heatmap(cm, annot=True,fmt='d')
          plt.title('Confusion Matrix for BoW')
          plt.show()
```



7.2.7 [5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

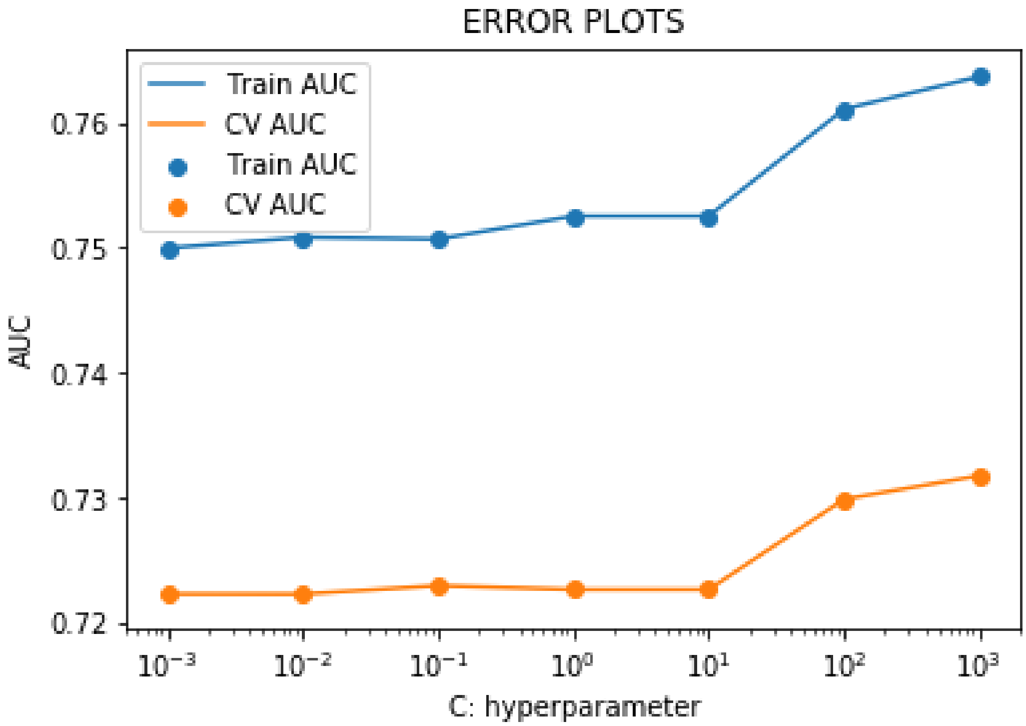
7.2.8 Hyperparameter tuning using GridSearchCV

```
In [137]: C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
          parameters = {'C': C}
          grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=-
          grid.fit(tfidf_sent_vectors_train, y_train)

          print("best C = ", grid.best_params_)
          print("Accuracy on train data = ", grid.best_score_*100)
          a = grid.best_params_
          optimal_a8 = a.get('C')
```

```
best C =  {'C': 1000}
Accuracy on train data =  73.17460039388827
```

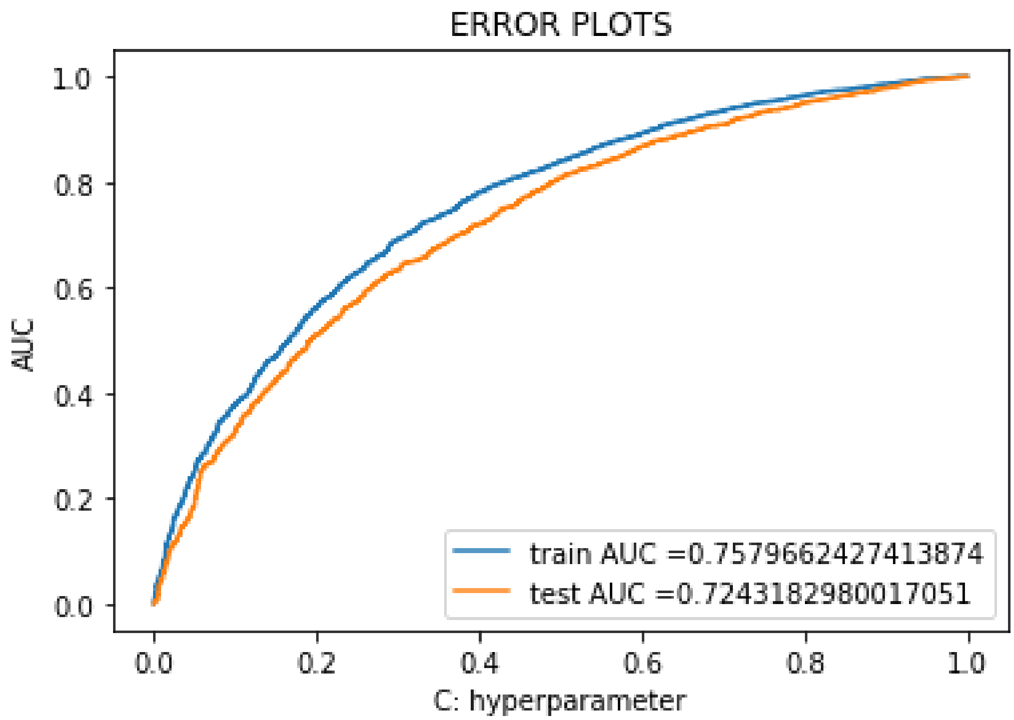
```
In [138]: train_auc_tw2v = grid.cv_results_['mean_train_score']
          cv_auc_tw2v    = grid.cv_results_['mean_test_score']

          plt.plot(C, train_auc_tw2v, label='Train AUC')
          plt.scatter(C, train_auc_tw2v, label='Train AUC')

          plt.plot(C, cv_auc_tw2v, label='CV AUC')
```

```
plt.scatter(C, cv_auc_tw2v, label='CV AUC')

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



Testing

In [139]:
```
clf = SVC(kernel='rbf', C = optimal_a8, probability=True)
clf.fit(tfidf_sent_vectors_train, y_train)

train_fpr_tw2v, train_tpr_tw2v, thresholds_tw2v = roc_curve(y_train, clf.predict_prob
test_fpr_tw2v, test_tpr_tw2v, thresholds_tw2v = roc_curve(y_test, clf.predict_proba(

plt.plot(train_fpr_tw2v, train_tpr_tw2v, label="train AUC ="+str(auc(train_fpr_tw2v,
plt.plot(test_fpr_tw2v, test_tpr_tw2v, label="test AUC ="+str(auc(test_fpr_tw2v, tes
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Calculating Confusion Matrix

```
In [192]: clf = SVC(kernel='rbf', C = optimal_a8, class_weight='balanced')
          clf.fit(tfidf_sent_vectors_train,y_train)

          predb1 = clf.predict(sent_vectors_test)

          acc_tw2v1 = accuracy_score(y_test, predb1) * 100
          pre_tw2v1 = precision_score(y_test, predb1) * 100
          rec_tw2v1 = recall_score(y_test, predb1) * 100
          f1_tw2v1 = f1_score(y_test, predb1) * 100

          print('\nAccuracy = %f%%' % (acc_tw2v1))
          print('\nprecision= %f%%' % (pre_tw2v1))
          print('\nrecall    = %f%%' % (rec_tw2v1))
          print('\nF1-Score = %f%%' % (f1_tw2v1))
```

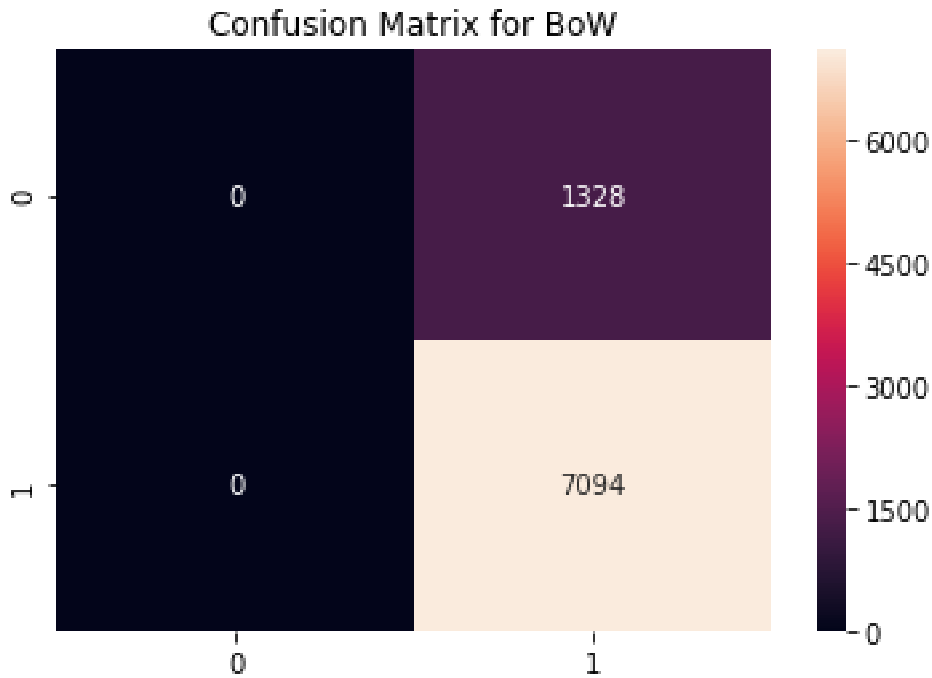
```
Accuracy = 59.926383%

precision= 89.854443%

recall    = 58.716638%
```

```
F1-Score = 71.022581%
```

```
In [140]: clf = SVC(kernel='rbf', C = optimal_a8)
          clf.fit(tfidf_sent_vectors_train,y_train)

          predb1 = clf.predict(sent_vectors_test)

          acc_tw2v1 = accuracy_score(y_test, predb1) * 100
          pre_tw2v1 = precision_score(y_test, predb1) * 100
          rec_tw2v1 = recall_score(y_test, predb1) * 100
          f1_tw2v1 = f1_score(y_test, predb1) * 100

          print('\nAccuracy = %f%%' % (acc_tw2v1))
          print('\nprecision= %f%%' % (pre_tw2v1))
          print('\nrecall    = %f%%' % (rec_tw2v1))
          print('\nF1-Score = %f%%' % (f1_tw2v1))
```

```
Accuracy = 84.231774%

precision= 84.231774%

recall    = 100.000000%

F1-Score = 91.441093%
```

```
In [141]: cm = confusion_matrix(y_test,predb1)
          sns.heatmap(cm, annot=True,fmt='d')
          plt.title('Confusion Matrix for BoW')
          plt.show()
```



8 [6] Conclusions

1. For Liner SVM I have used 100k points and for RBF kernel I have used 30k data points.
2. In terms of accuracy measures BoW is our best model.

In [142]:
```
# Please compare all your models using Prettytable library
number= [1,2,3,4,5,6,7,8]
name= ["Bow", "Bow", "Tfidf", "Tfidf", "Avg W2v", "Avg W2v", "Tfidf W2v", "Tfidf W2v'
svm= ["Linear", "RBF", "Linear", "RBF", "Linear", "RBF", "Linear", "RBF"]
optimal= [optimal_a1, optimal_a2, optimal_a4, optimal_a3, optimal_a5, optimal_a6, opt
acc= [acc_b,acc_b1,acc_tf,acc_tf1,acc_aw2v,acc_aw2v1,acc_tw2v,acc_tw2v1]
pre= [pre_b,pre_b1,pre_tf,pre_tf1,pre_aw2v,pre_aw2v1,pre_tw2v,pre_tw2v1]
rec= [rec_b,rec_b1,rec_tf,rec_tf1,rec_aw2v,rec_aw2v1,rec_tw2v,rec_tw2v1]
f1= [f1_b,f1_b1,f1_tf,f1_tf1,f1_aw2v,f1_aw2v1,f1_tf,f1_tw2v1]


#Initialize Prettytable
pt = PrettyTable()
pt.add_column("Index", number)
pt.add_column("Model", name)
pt.add_column("Optimal", optimal)
pt.add_column("SVM", svm)
pt.add_column("Accuracy%", acc)
pt.add_column("Precision%", pre)
```

```python
pt.add_column("Recall%", rec)
pt.add_column("F1%", f1)

print(pt)# Please compare all your models using Prettytable library
```

Index	Model	Optimal	SVM	Accuracy%	Precision%	Recall%
1	Bow	0.001	Linear	87.95002278596384	97.10556468888556	88.2629958340
2	Bow	0.0001	RBF	88.41130372833057	95.56151325588323	90.4426275725
3	Tfidf	0.1	Linear	85.8916907185174	97.1846904700745	85.6593008512
4	Tfidf	1	RBF	84.23177392543339	84.23177392543339	100.0
5	Avg W2v	100	Linear	84.175148108765	84.175148108765	100.0
6	Avg W2v	1	RBF	84.23177392543339	84.23177392543339	100.0
7	Tfidf W2v	1000	Linear	84.175148108765	84.175148108765	100.0
8	Tfidf W2v	1000	RBF	84.23177392543339	84.23177392543339	100.0