07affrsvm

July 14, 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 [104]:
```
%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.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import SGDClassifier
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 prettytable import PrettyTable
```

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 10

# 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 (100000, 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=Fa
```

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

Out[68]:
```
(87775, 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]:
```
87.775
```

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

In [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()
```

```
(87773, 10)
```

```
Out[72]: 1    73592
         0    14181
         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 [73]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
==================================================
```

In [74]:
```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [75]:
```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
```

In [76]:
```
# 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 [77]: sent_1500 = decontracted(sent_1500)
         print(sent_1500)
         print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

```
In [78]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [79]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

```
In [80]: # 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', '
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
                     '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', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                    '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 [81]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews_linear = []
         # 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_linear.append(sentance.strip())
```

```
100%|| 87773/87773 [01:02<00:00, 1394.63it/s]
```

```
In [82]: preprocessed_reviews_linear[1500]
```

```
Out[82]: 'way hot blood took bite jig lol'
```

4.2 [4] Splitting the data

```
In [83]: X = preprocessed_reviews_linear
         Y = final['Score'].values
```

```
In [84]: # Here we are splitting the data(X ,Y) into train and test data
         # X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=F
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30) # this is ra
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [85]: #BoW
         vectorizer = CountVectorizer(min_df = 10, max_features=500)
         vectorizer.fit(X_train) # fit has to happen only on train data
         print(vectorizer.get_feature_names()[:20])# printing some feature names
         print("="*50)

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_bow = vectorizer.transform(X_train)
```

```
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, Y_train.shape)
print(X_test_bow.shape, Y_test.shape)
```

```
['able', 'absolutely', 'actually', 'add', 'added', 'aftertaste', 'ago', 'almonds', 'almost', 'a
==================================================
After vectorizations
(61441, 500) (61441,)
(26332, 500) (26332,)
```

5.2 [4.3] TF-IDF

```
In [86]: tfidf_vect = TfidfVectorizer(min_df=10, max_features=500)
         tfidf_vect.fit(X_train)
         print("some sample features ",tfidf_vect.get_feature_names()[0:10])
         print('='*50)

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_tfidf = tfidf_vect.transform(X_train)
         X_test_tfidf = tfidf_vect.transform(X_test)

         print("After vectorizations")
         print(X_train_tfidf.shape, Y_train.shape)
         print(X_test_tfidf.shape, Y_test.shape)
```

```
some sample features  ['able', 'absolutely', 'actually', 'add', 'added', 'aftertaste', 'ago',
==================================================
After vectorizations
(61441, 500) (61441,)
(26332, 500) (26332,)
```

5.3 [4.4] Word2Vec

```
In [87]: # 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 [88]: # this line of code trains your w2v model on the give list of sentances, fitting the
         w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=-1)
```

```
In [89]: 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  14852
sample words  ['taco', 'bell', 'chipotle', 'sauce', 'texture', 'weak', 'ranch', 'dressing', 'po
```

5.4 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

5.4.1 Converting Train data set

In [90]:
```python
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this lis
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
    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%|| 61441/61441 [03:02<00:00, 337.05it/s]


(61441, 50)
[ 2.63500837e-04 -4.71244666e-05 -2.04688599e-04  3.25983955e-04
  5.97030866e-04 -5.26451226e-04 -7.06264522e-04 -3.84245742e-04
  2.04620731e-03  6.91808340e-04  5.11131255e-04  3.67206511e-04
  8.09698584e-04 -2.01199772e-03 -5.11942920e-04 -4.64604782e-04
 -1.02907764e-03 -6.86631172e-04 -9.57603061e-05  7.20890763e-04
  8.76426566e-04 -4.61501996e-04 -9.18631274e-04 -8.65748106e-04
  1.42489640e-03  6.16011964e-04  8.40154041e-04  2.86267662e-04
 -2.44217904e-04 -1.21405519e-03  1.55787608e-04  3.27642129e-04
  8.99536835e-04 -4.94158634e-04 -9.31732433e-04 -9.09608785e-04
  1.01206961e-03  3.18477795e-04 -1.40491453e-03 -3.54866666e-04
  7.08927958e-04  8.94185796e-04  4.80626939e-04 -1.93990849e-04
 -1.17328240e-04 -6.85300373e-04 -3.85334496e-04  7.27677279e-04
  2.82423397e-05  2.23435819e-04]
```

5.4.2 Converting Test data set

```
In [91]: list_of_sentance_test=[]
         for sentance in X_test:
             list_of_sentance_test.append(sentance.split())
```

```
In [92]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(list_of_sentance_test): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             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%|| 26332/26332 [01:17<00:00, 340.23it/s]


(26332, 50)
[ 1.12836534e-03 -2.42954393e-03 -1.68316450e-03  3.24406996e-04
 -1.19081466e-03 -1.66998638e-03 -1.13681587e-03 -3.43135857e-03
  2.10292448e-03  1.29747941e-03 -2.28080867e-03 -8.38301923e-04
  2.84105598e-03  2.93152874e-03  3.30750103e-03 -1.46111451e-03
  2.08108103e-03  2.12491713e-03  1.61272147e-03  1.43155152e-03
  3.15933547e-05  1.90111310e-03 -1.24895058e-03  5.51244267e-04
 -1.09153419e-03  3.12229006e-03  4.18250845e-04  9.39566418e-04
 -2.01835629e-03 -2.21382575e-03  3.65478190e-04 -7.37338664e-04
 -1.64482396e-03 -9.73850981e-04  4.30961995e-04  1.70413744e-03
 -3.68045460e-03  3.15657977e-04  2.24773609e-04  2.30846297e-03
 -2.86681693e-03 -2.44923227e-04  3.67860761e-04  1.35688275e-03
  7.71064707e-04 -1.57244381e-04  6.75860816e-04  1.70759759e-03
 -2.78552547e-04  1.69529338e-03]
```

[4.4.1.2] TFIDF weighted W2v

5.4.3 Converting Train data

```
In [93]: # 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 = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [94]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

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

```
100%|| 61441/61441 [41:23<00:00, 23.96it/s]
```

5.4.4 Converting Test data

```
In [95]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

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

```
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors_test.append(sent_vec)
        row += 1
```

```
100%|| 26332/26332 [18:22<00:00, 23.89it/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 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.
<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.

7 Applying SVM

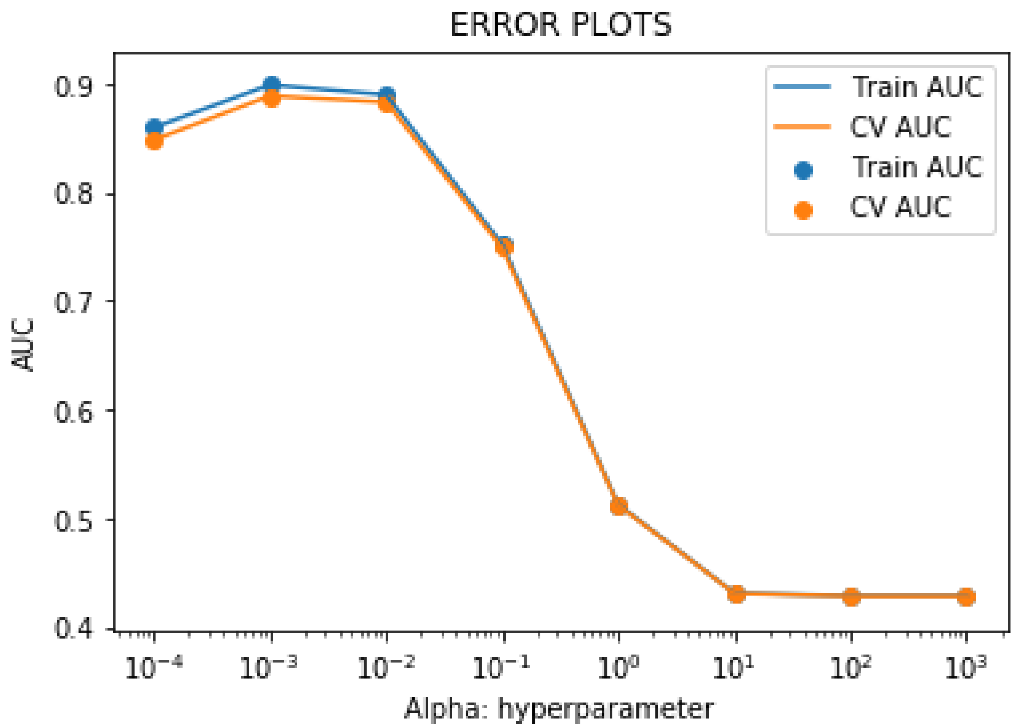
7.1 [5.1] Linear SVM

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

7.1.2 Hyperparameter tuning using GridSearch

In [96]:
```
#clf = SGDClassifier()
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l2'), parameters, cv=3, scorin
grid.fit(X_train_bow, Y_train)

print("best alpha = ", grid.best_params_)

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

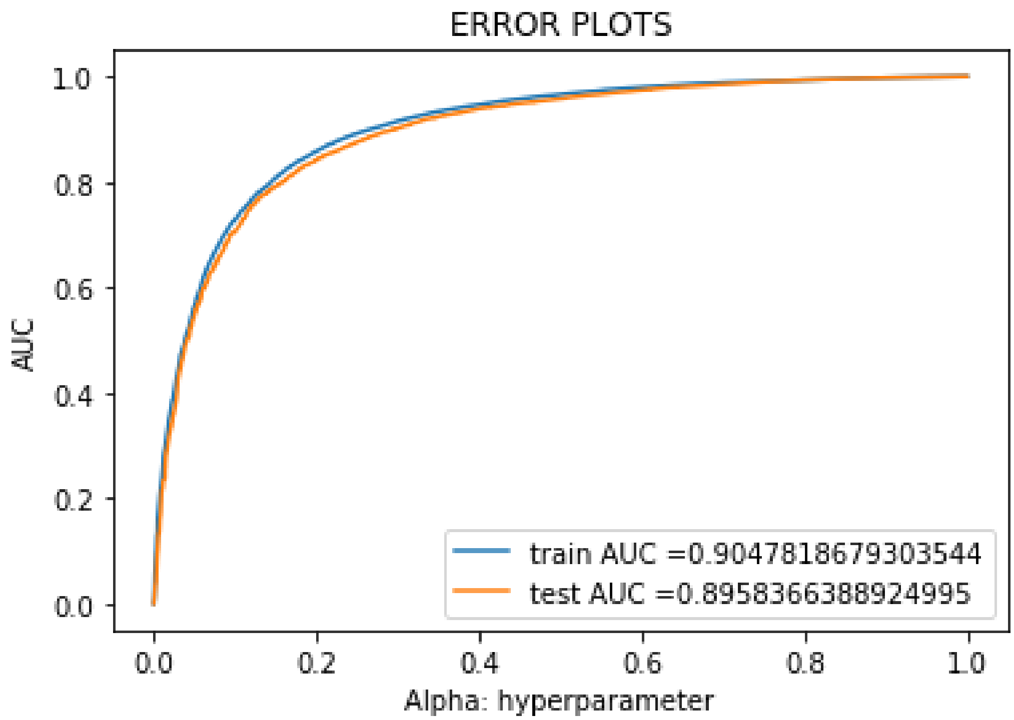
```
best alpha =  {'alpha': 0.001}
```




```
In [97]: a = grid.best_params_
         optimal_a1 = a.get('alpha')
```

7.1.3 Testing with Test data

```
In [98]: 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)

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

         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_prob

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

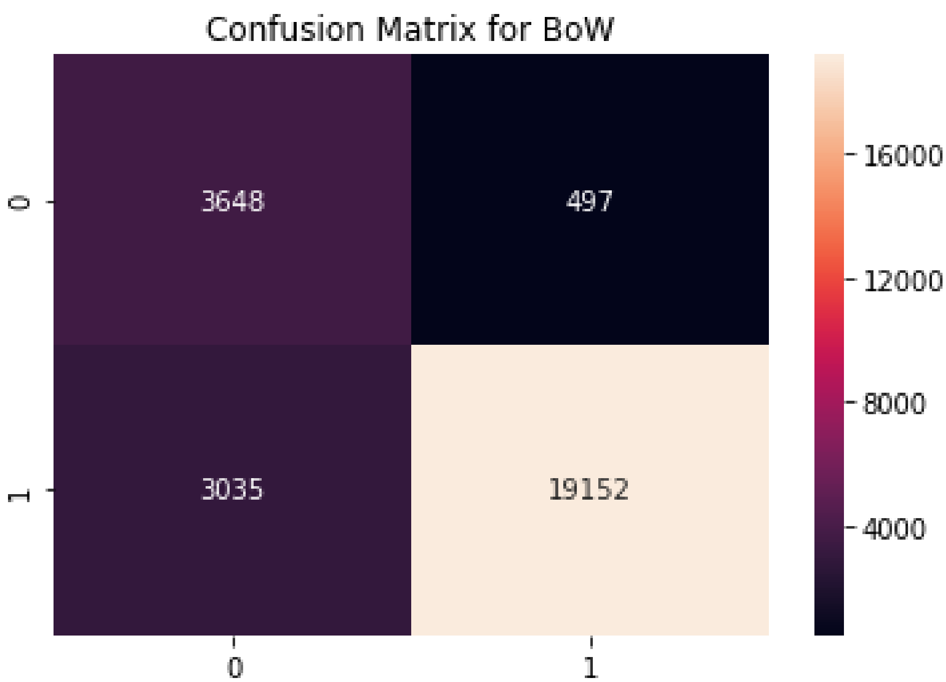


In [99]:
```
clf = SGDClassifier(loss='hinge',penalty='l2',alpha = optimal_a1, class_weight='balanc

clf.fit(X_train_bow,Y_train)

predb = clf.predict(X_test_bow)

accb = accuracy_score(Y_test, predb) * 100
preb = precision_score(Y_test, predb) * 100
recb = recall_score(Y_test, predb) * 100
f1b = f1_score(Y_test, predb) * 100

print('\nAccuracy=%f%%' % (accb))
print('\nprecision=%f%%' % (preb))
print('\nrecall=%f%%' % (recb))
print('\nF1-Score=%f%%' % (f1b))
```

```
Accuracy=84.570105%

precision=95.264400%

recall=85.875962%
```

```
F1-Score=90.326881%
```

```
In [97]: cm = confusion_matrix(Y_test,predb)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix for BoW')
         plt.show()
```



7.1.4 [5.1.2] Feature Importance on BOW

[5.1.2.1] Top 10 important features of positive class

```
In [93]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-g

         def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
             class_labels = classifier.classes_
             feature_names = vectorizer.get_feature_names()
             topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[-n:]


             for coef, feat in reversed(topn_class1):
                 print (class_labels[1], coef, feat)


         most_informative_feature_for_binary_classification(vectorizer, clf)
```

```
1 0.9181492919480334 delicious
1 0.8738416412711003 perfect
1 0.8302002084228136 amazing
1 0.8299716137622165 awesome
1 0.8089792623963026 excellent
1 0.7656861458189953 wonderful
1 0.742627724914713 yummy
1 0.7209145254707702 best
1 0.715741699515429 loves
1 0.6815166231651966 hooked
```

[5.1.2.2] Top 10 important features of negative class

```
In [94]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-g

         def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
             class_labels = classifier.classes_
             feature_names = vectorizer.get_feature_names()
             topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[:n]

             for coef, feat in topn_class2:
                 print( class_labels[0], coef, feat)


         most_informative_feature_for_binary_classification(vectorizer, clf)
```

```
0 -1.0721017904081567 worst
0 -0.970527883938402 disappointing
0 -0.9476806225087374 terrible
0 -0.8482250911352915 rip
0 -0.8397777435290229 disappointed
0 -0.8129548722478339 awful
0 -0.7958488238363879 bland
0 -0.781353210214403 unfortunately
0 -0.7693155703667182 shame
0 -0.7460851222941215 threw
```

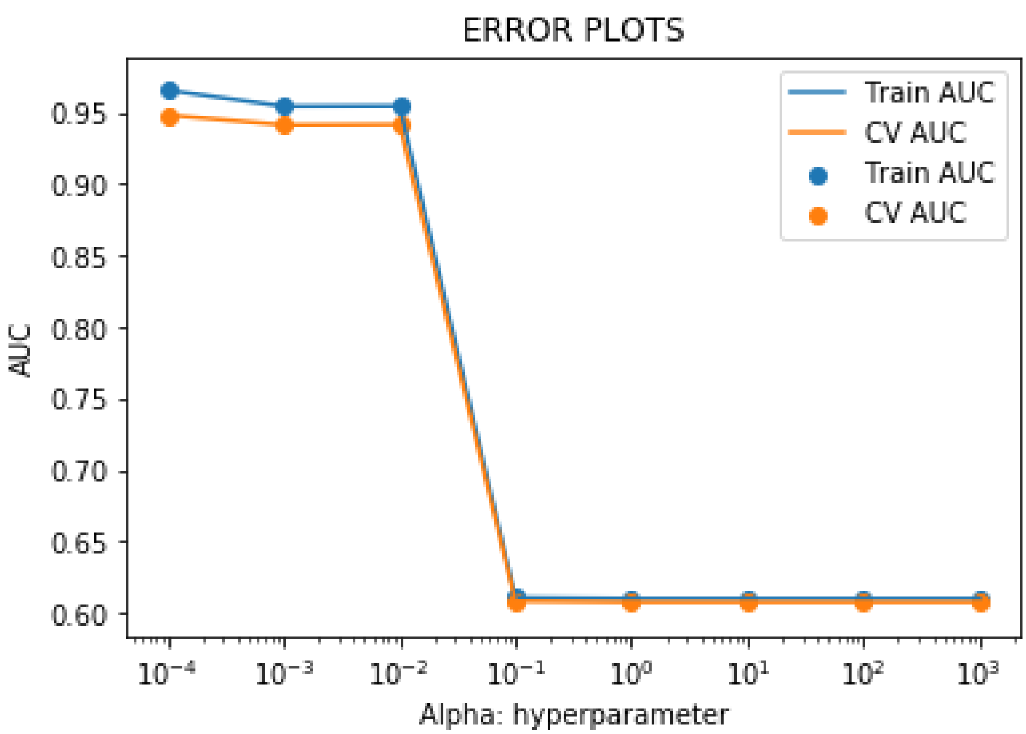
7.1.5 [5.1.3] Applying Linear SVM on TFIDF, SET 2

7.1.6 Hyperparameter tuning using GridSearch

```
In [98]: #clf = SGDClassifier()
         alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
         grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l2'), parameters, cv=3, scorin
         grid.fit(X_train_tfidf, Y_train)
```

```
print("best alpha = ", grid.best_params_)

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

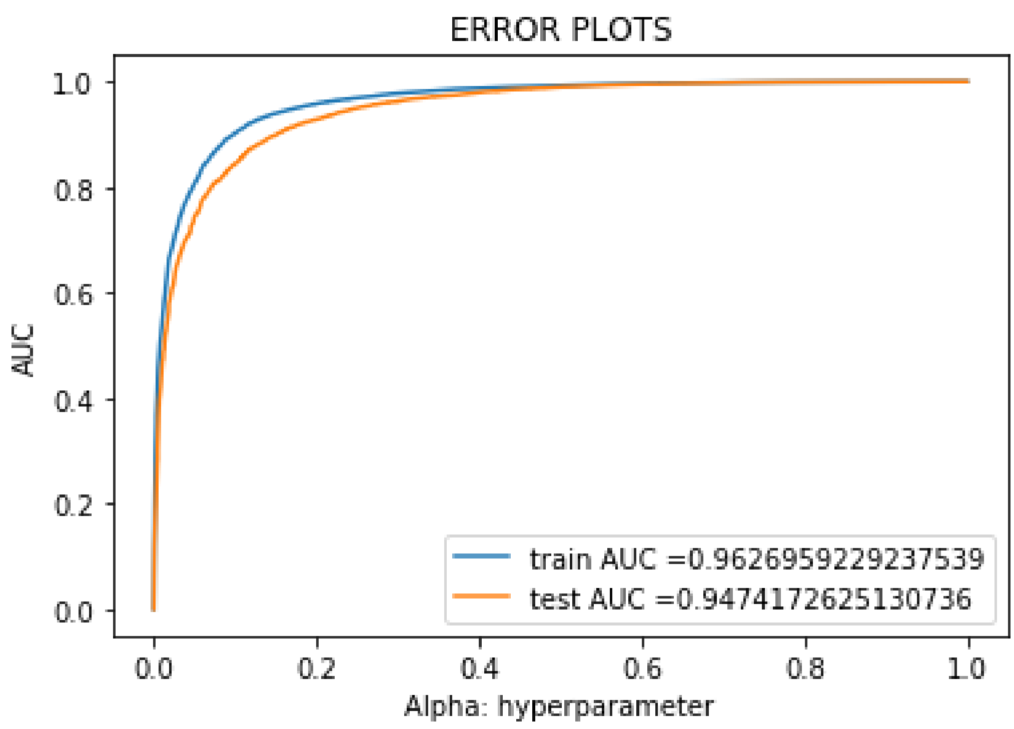
```
best alpha =  {'alpha': 0.0001}
```




```
In [99]: a = grid.best_params_
         optimal_a1 = a.get('alpha')
```

7.1.7 Testing with Test data

In [100]:
```
clf = SGDClassifier(loss='hinge', penalty='l2',alpha = optimal_a1)
calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
calibrator.fit(X_train_tfidf, Y_train)

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

train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, calibrator.p
test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, calibrator.pred

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfi
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf,
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

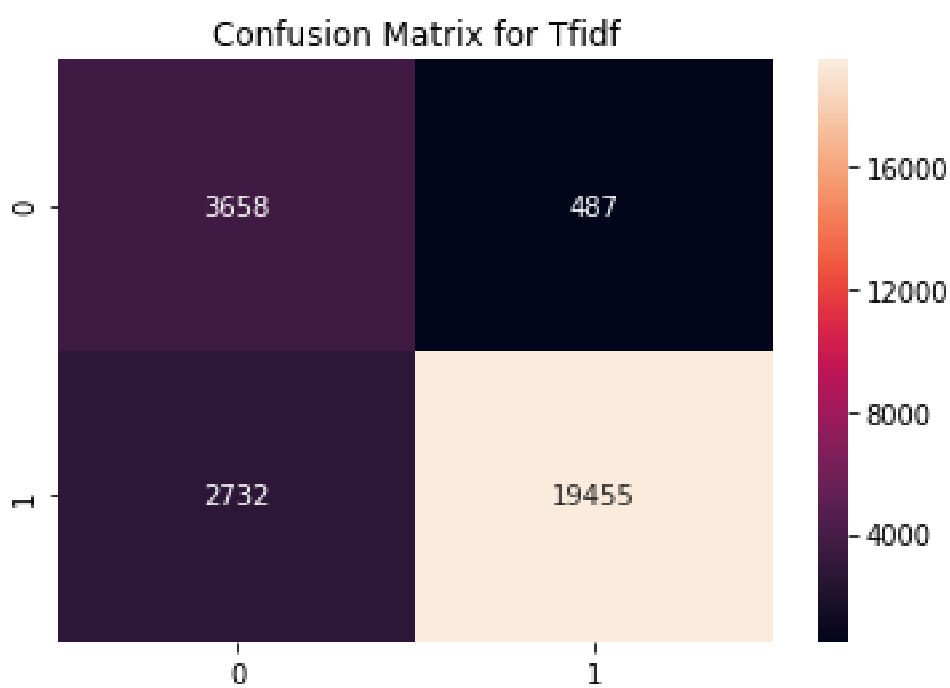


In [100]:
```
clf = SGDClassifier(loss='hinge',penalty='l2',alpha = optimal_a1, class_weight='bala

clf.fit(X_train_tfidf,Y_train)
```

```
predt = clf.predict(X_test_tfidf)

acct = accuracy_score(Y_test, predt) * 100
pret = precision_score(Y_test, predt) * 100
rect = recall_score(Y_test, predt) * 100
f1t = f1_score(Y_test, predt) * 100

print('\nAccuracy=%f%%' % (acct))
print('\nprecision=%f%%' % (pret))
print('\nrecall=%f%%' % (rect))
print('\nF1-Score=%f%%' % (f1t))
```

```
Accuracy=80.810421%

precision=96.243418%

recall=80.258035%

F1-Score=87.526845%
```

In [102]:
```
cm = confusion_matrix(Y_test,predt)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix for Tfidf')
plt.show()
```



7.1.8 [5.1.4] Feature Importance on BOW

[5.1.4.1] Top 10 important features of positive class

```
In [103]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-

          def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10)
              class_labels = classifier.classes_
              feature_names = vectorizer.get_feature_names()
              topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[-n:]


              for coef, feat in reversed(topn_class1):
                  print (class_labels[1], coef, feat)


          most_informative_feature_for_binary_classification(tfidf_vect, clf)
```

```
1 6.120183034431552 great
1 4.749801188355211 delicious
1 4.576974033742144 best
1 4.218113589408829 perfect
1 3.8489291276012545 good
1 3.7069623531971945 love
1 3.627367155275993 loves
1 3.625505366449252 excellent
1 3.505946485724742 nice
1 3.406310876328791 wonderful
```

[5.1.4.2] Top 10 important features of negative class

```
In [104]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-

          def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10)
              class_labels = classifier.classes_
              feature_names = vectorizer.get_feature_names()
              topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[:n]

              for coef, feat in topn_class2:
                  print( class_labels[0], coef, feat)


          most_informative_feature_for_binary_classification(tfidf_vect, clf)
```

```
0 -4.782108292102296 not
0 -4.061609691533384 disappointed
0 -3.437486669741958 worst
0 -3.354589762891281 disappointing
```

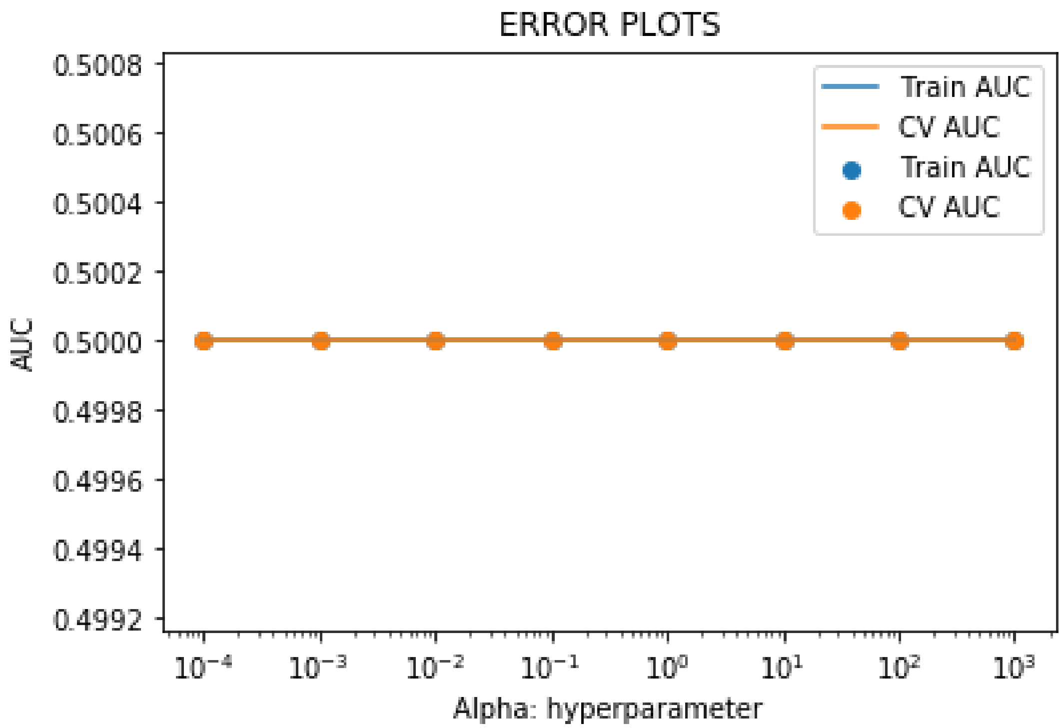
```
0 -3.3083156709202144 terrible
0 -3.161698987004 unfortunately
0 -3.0672651874420467 bland
0 -2.7484599935622622 awful
0 -2.7045733238033116 disappointment
0 -2.6779006901026072 thought
```

7.1.9 [5.1.5] Applying Linear SVM on AVG W2V, SET 3

7.1.10 Hyperparameter tuning using GridSearch

```
In [109]: #clf = SGDClassifier()
          alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
          parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
          grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l1'), parameters, cv=3, scor
          grid.fit(sent_vectors_train, Y_train)

          print("best alpha = ", grid.best_params_)

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

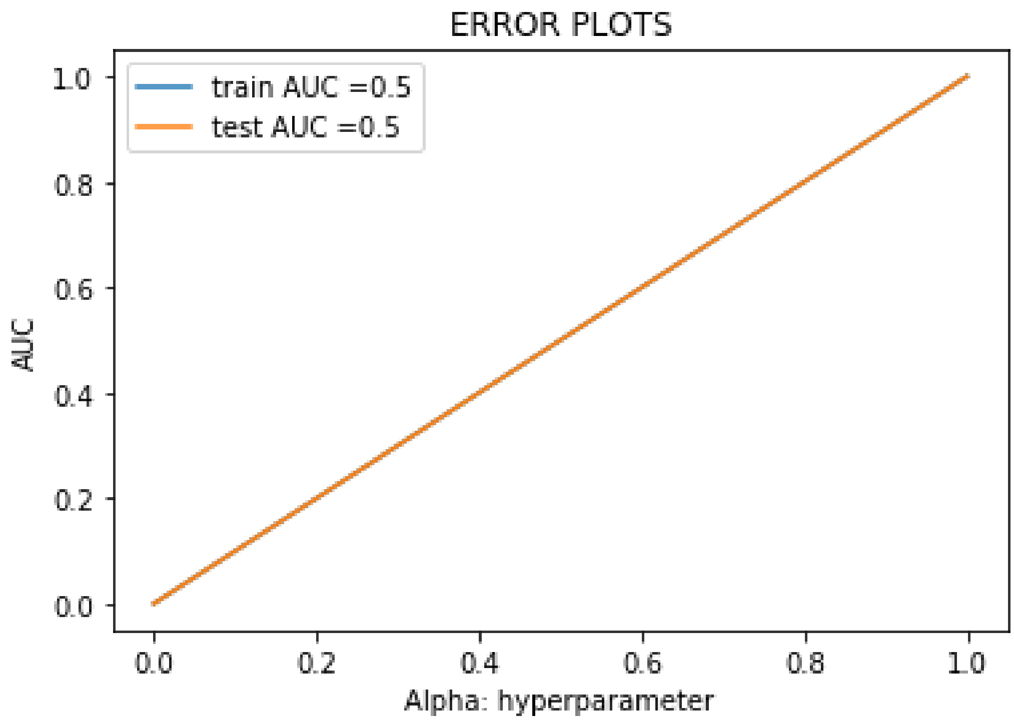
```
best alpha =  {'alpha': 0.0001}
```




```
In [110]: a = grid.best_params_
          optimal_a1 = a.get('alpha')
```

7.1.11 Testing with Test data

```
In [111]: clf = SGDClassifier(loss='hinge', penalty='l1',alpha = optimal_a1)
          calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
          calibrator.fit(sent_vectors_train, Y_train)

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

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

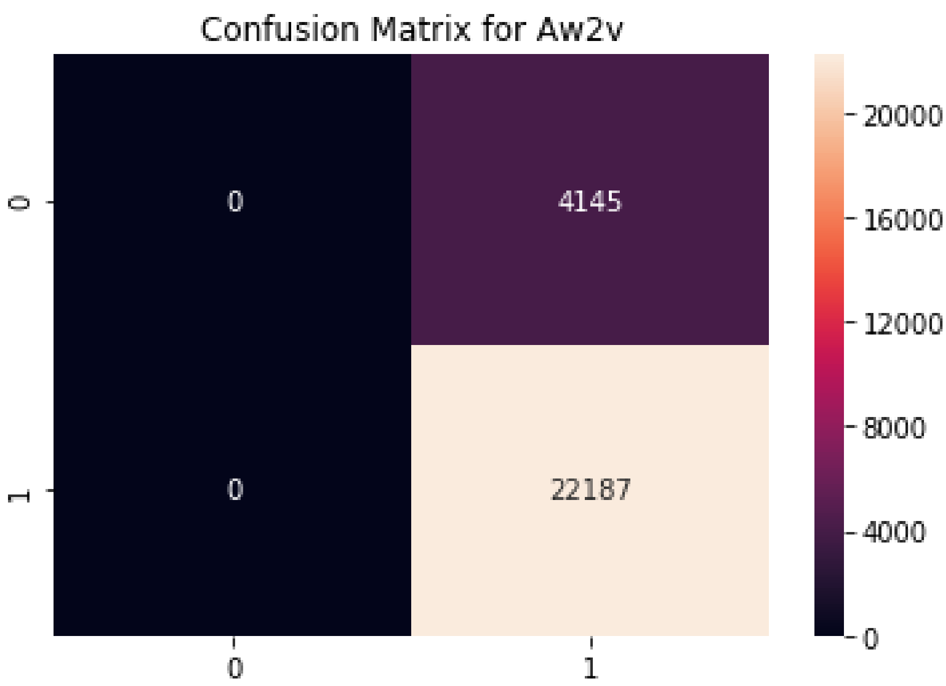



```
In [101]: clf = SGDClassifier(loss='hinge',penalty='l1',alpha = optimal_a1, class_weight='balan

          clf.fit(sent_vectors_train,Y_train)

          preda = clf.predict(sent_vectors_test)

          acca = accuracy_score(Y_test, preda) * 100
          prea = precision_score(Y_test, preda) * 100
          reca = recall_score(Y_test, preda) * 100
          f1a = f1_score(Y_test, preda) * 100

          print('\nAccuracy=%f%%' % (acca))
          print('\nprecision=%f%%' % (prea))
          print('\nrecall=%f%%' % (reca))
          print('\nF1-Score=%f%%' % (f1a))
```

```
Accuracy=83.890324%

precision=83.890324%

recall=100.000000%
```

F1-Score=91.239519%

```
In [113]: cm = confusion_matrix(Y_test,preda)
          sns.heatmap(cm, annot=True,fmt='d')
          plt.title('Confusion Matrix for Aw2v')
          plt.show()
```



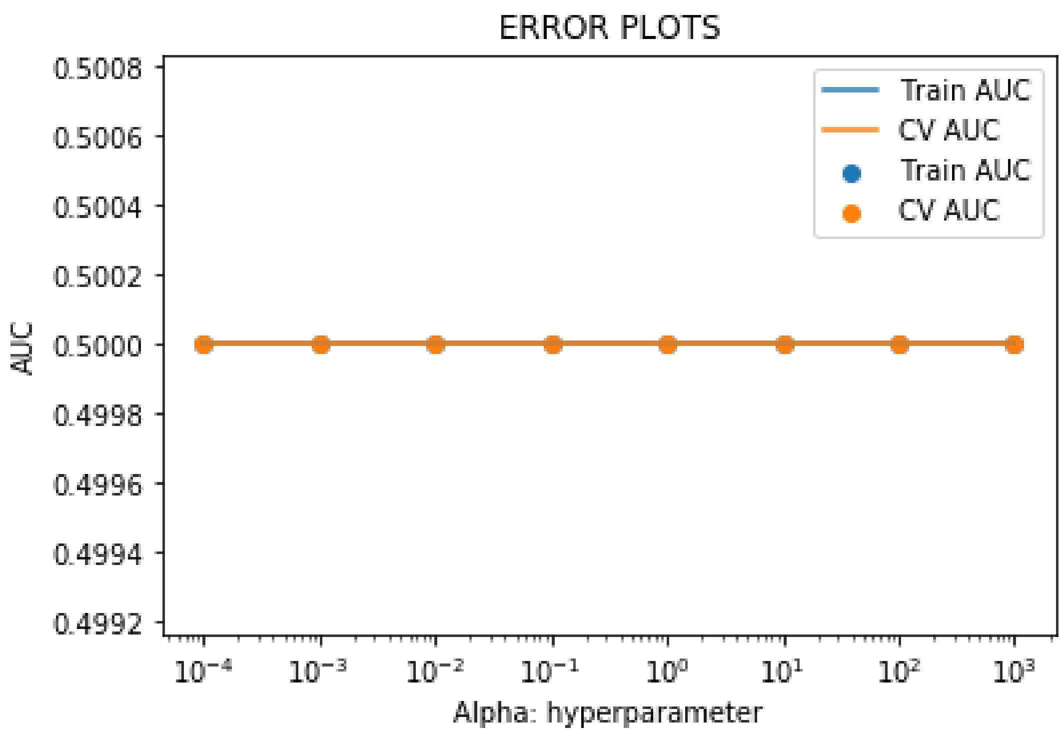
7.1.12 [5.1.6] Applying Linear SVM on TFIDF W2V, SET 4

7.1.13 Hyperparameter tuning using GridSearch

```
In [114]: #clf = SGDClassifier()
          alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
          parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
          grid = GridSearchCV(SGDClassifier(loss='hinge',penalty='l1'), parameters, cv=3, scor
          grid.fit(tfidf_sent_vectors_train, Y_train)

          print("best alpha = ", grid.best_params_)

          train_auc_tfw2v = grid.cv_results_['mean_train_score']
          cv_auc_tfw2v = grid.cv_results_['mean_test_score']

          plt.plot(alpha, train_auc_tfw2v, label='Train AUC')
          plt.scatter(alpha, train_auc_tfw2v, label='Train AUC')
```

```
plt.plot(alpha, cv_auc_tfw2v, label='CV AUC')
plt.scatter(alpha, cv_auc_tfw2v, label='CV AUC')

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

```
best alpha =  {'alpha': 0.0001}
```

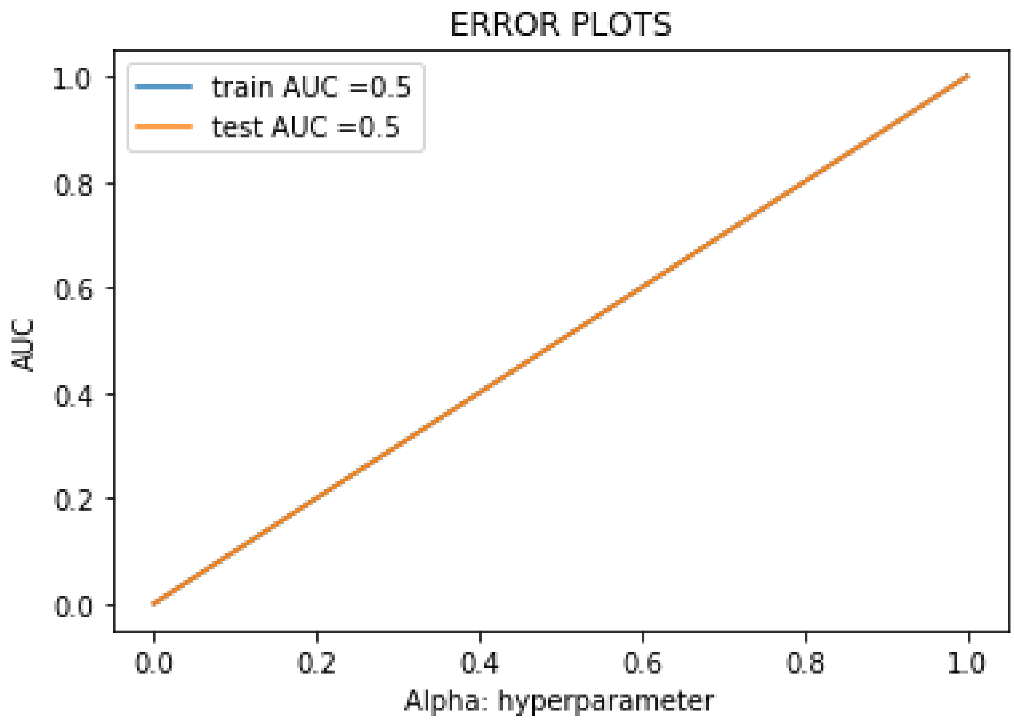



```
In [115]: a = grid.best_params_
          optimal_a1 = a.get('alpha')
```

7.1.14 Testing with Test data

```
In [116]: clf = SGDClassifier(loss='hinge', penalty='l1',alpha = optimal_a1)
          calibrator = CalibratedClassifierCV(base_estimator=clf, cv=3, method='isotonic')
          calibrator.fit(tfidf_sent_vectors_train, Y_train)

          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
          # not the predicted outputs
```

```
train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, calibrator.pr
test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, calibrator.predi

plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw2
plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, t
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

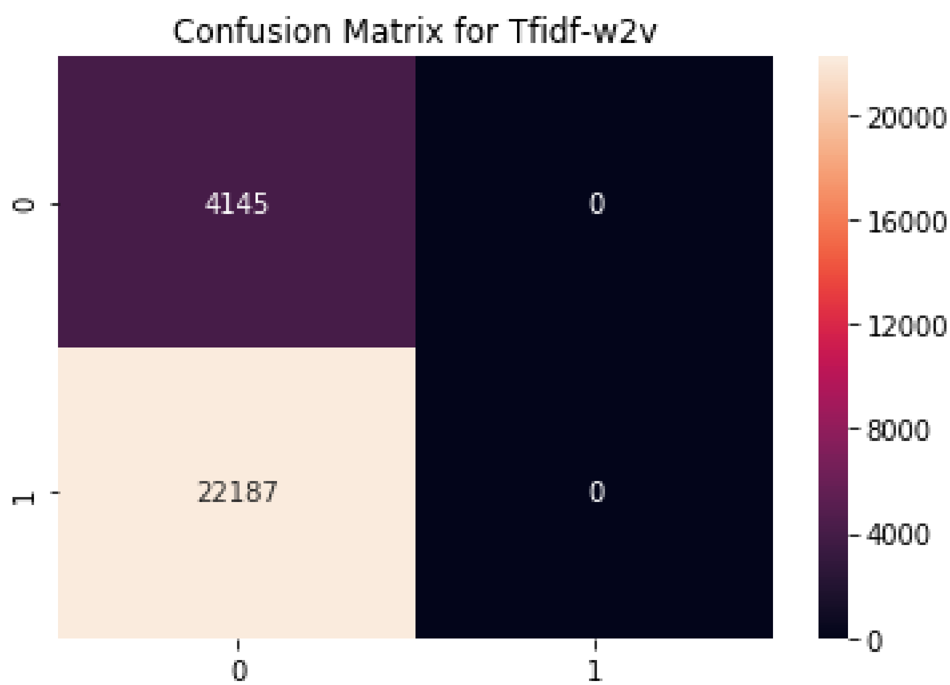


In [102]:
```
clf = SGDClassifier(loss='hinge',penalty='l1',alpha = optimal_a1, class_weight='balan

clf.fit(tfidf_sent_vectors_train,Y_train)

predw = clf.predict(tfidf_sent_vectors_test)

accw = accuracy_score(Y_test, predw) * 100
prew = precision_score(Y_test, predw) * 100
recw = recall_score(Y_test, predw) * 100
f1w = f1_score(Y_test, predw) * 100

print('\nAccuracy=%f%%' % (accw))
```

```
print('\nprecision=%f%%' % (prew))
print('\nrecall=%f%%' % (recw))
print('\nF1-Score=%f%%' % (f1w))
```

```
Accuracy=83.890324%

precision=83.890324%

recall=100.000000%

F1-Score=91.239519%
```

In [119]:
```
cm = confusion_matrix(Y_test,predw)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix for Tfidf-w2v')
plt.show()
```



7.2 [5.2] RBF SVM

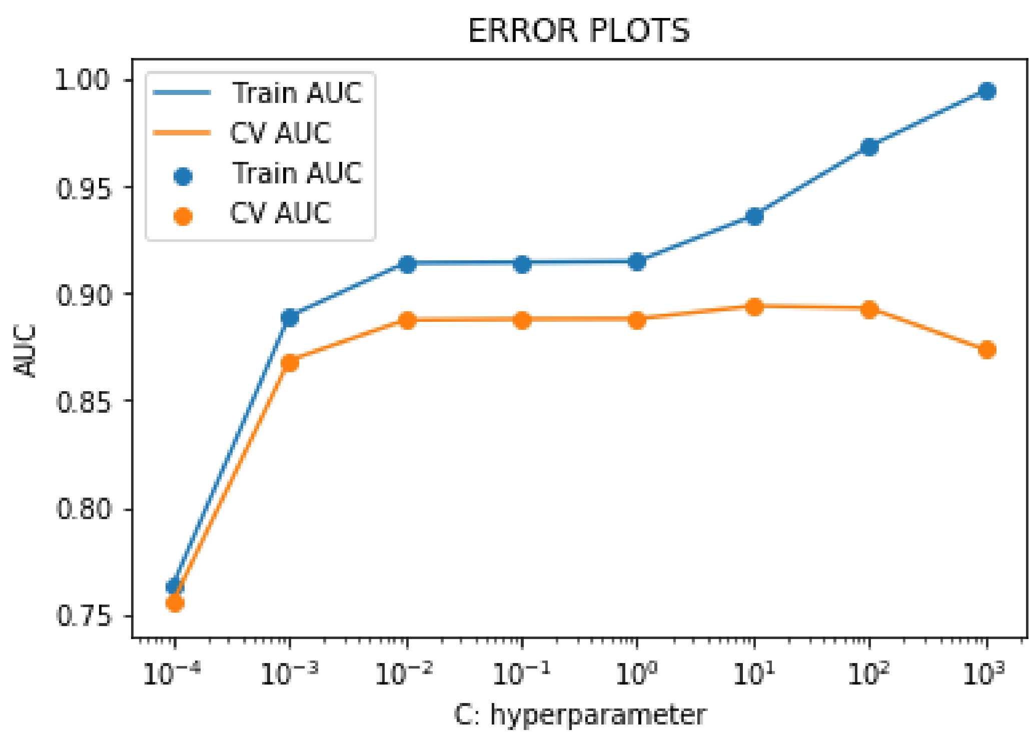
7.2.1 [5.2.1] Applying RBF SVM on BOW, SET 1

7.2.2 Hyperparameter tuning using GridSearch

In [37]:
```
#clf = SVC()
C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
```

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

print("best C = ", grid.best_params_)

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

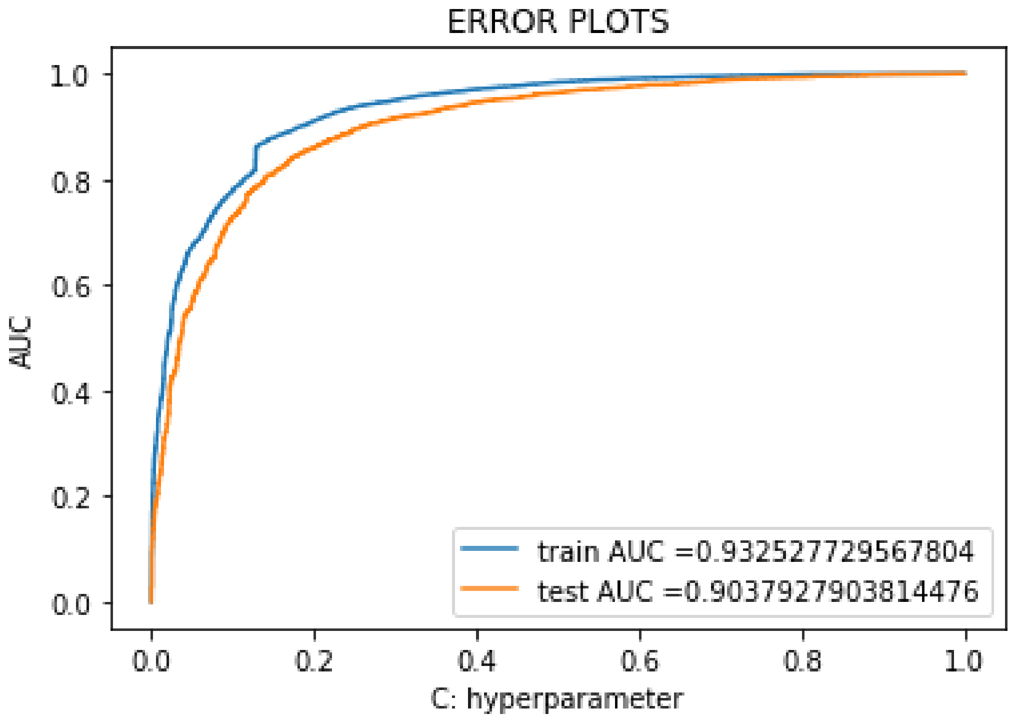
```
best C =  {'C': 10}
```



```
In [38]: a = grid.best_params_
         optimal_a2 = a.get('C')
```

7.2.3 Testing with Test data

```
In [179]: clf = SVC(kernel='rbf', C = optimal_a2, probability=True)
          clf.fit(X_train_bow, Y_train)

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

          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()
```

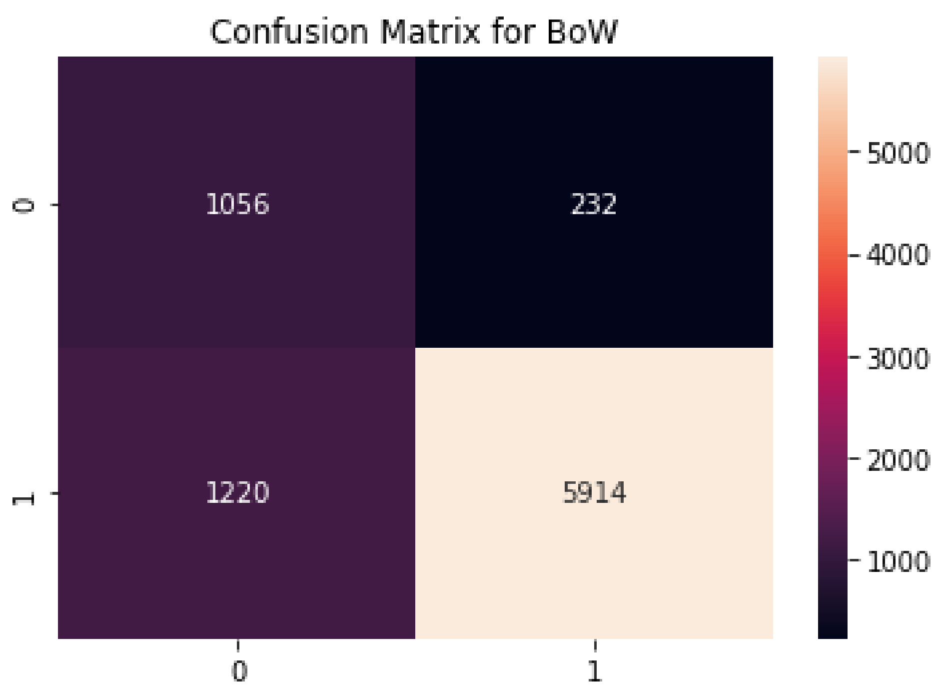



```
In [39]: clf = SVC(kernel='rbf', C = optimal_a2, class_weight='balanced')
```

```
clf.fit(X_train_bow,Y_train)

predb1 = clf.predict(X_test_bow)

accb1 = accuracy_score(Y_test, predb1) * 100
preb1 = precision_score(Y_test, predb1) * 100
recb1 = recall_score(Y_test, predb1) * 100
f1b1 = f1_score(Y_test, predb1) * 100

print('\nAccuracy=%f%%' % (accb1))
print('\nprecision=%f%%' % (preb1))
print('\nrecall=%f%%' % (recb1))
print('\nF1-Score=%f%%' % (f1b1))
```

```
Accuracy=82.759440%

precision=96.225187%

recall=82.898795%

F1-Score=89.066265%
```

In [40]:
```
cm = confusion_matrix(Y_test,predb1)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix for BoW')
plt.show()
```

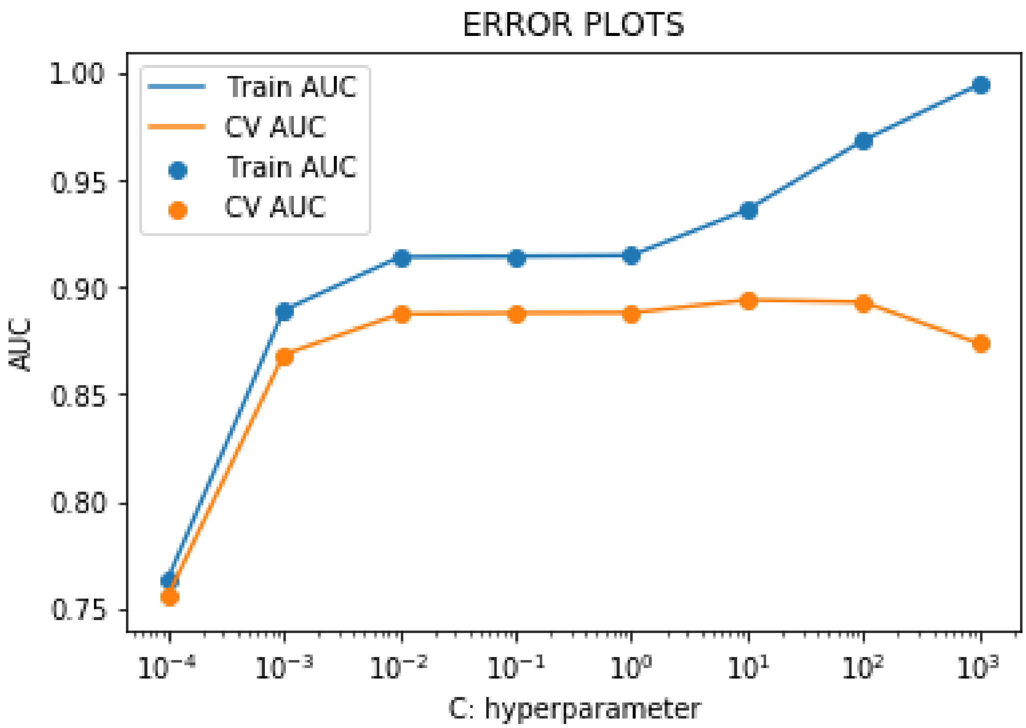


7.2.4 [5.2.2] Applying RBF SVM on TFIDF, SET 2

7.2.5 Hyperparameter tuning using GridSearch

```
In [42]: #clf = SVC()
         C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
         grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=1)
         grid.fit(X_train_tfidf, Y_train)

         print("best C = ", grid.best_params_)

         train_auc_tfidf = grid.cv_results_['mean_train_score']
         cv_auc_tfidf = 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.xscale('log')
         plt.xlabel("C: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```

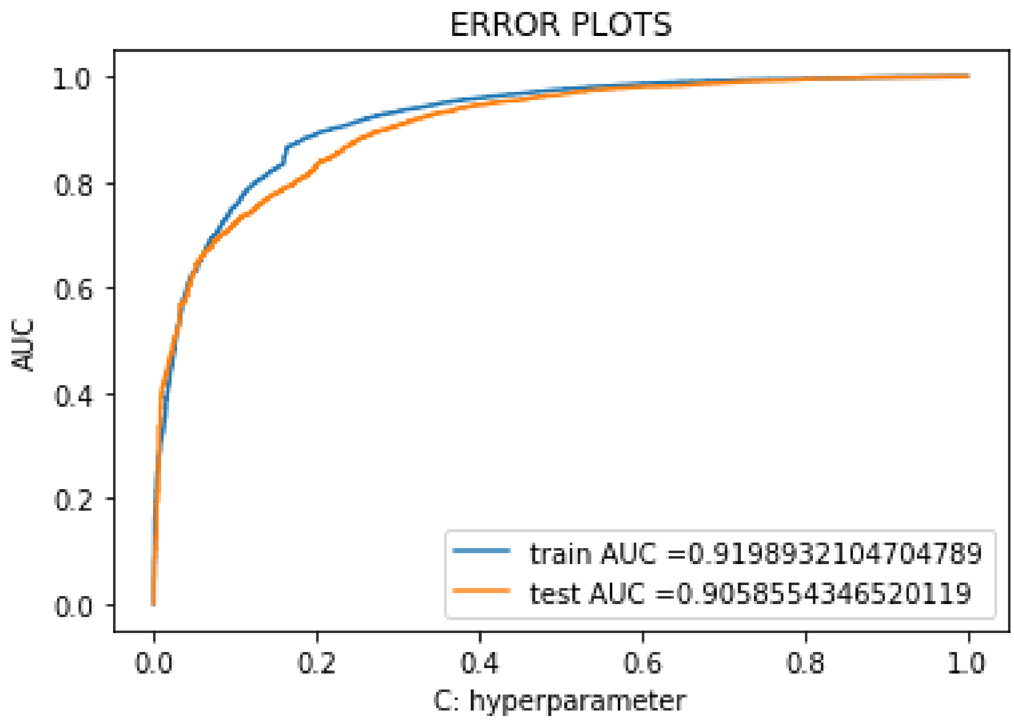
```
best C =  {'C': 1000}
```




```
In [43]: a = grid.best_params_
         optimal_a2 = a.get('C')
```

7.2.6 Testing with Test Data

```
In [44]: clf = SVC(kernel='rbf', C = optimal_a2, probability=True)
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_p
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_prob

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, t
         plt.legend()
         plt.xlabel("C: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```

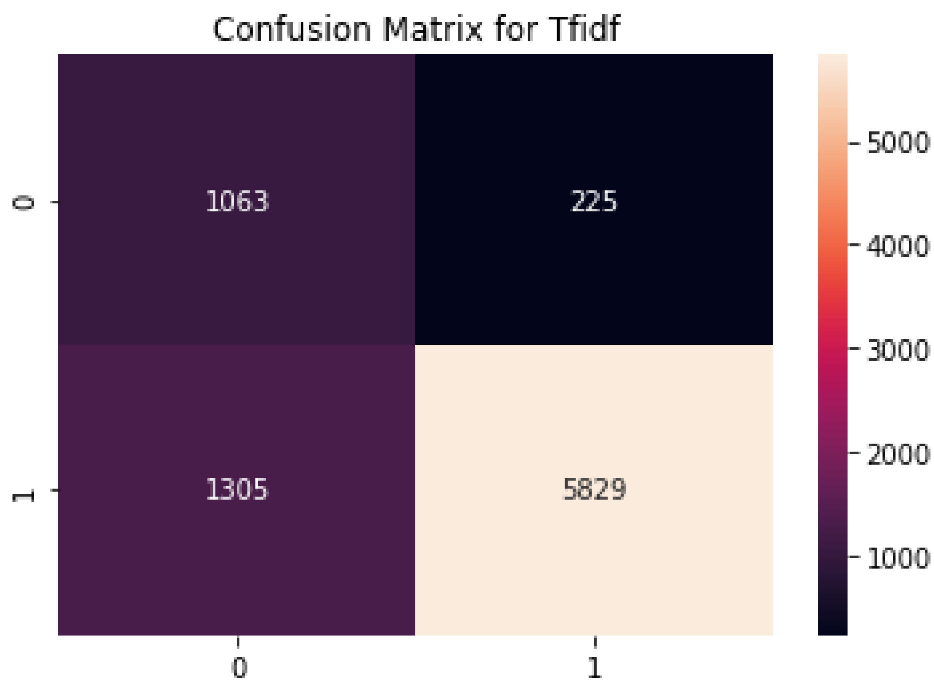


In [45]:
```
clf = SVC(kernel='rbf', C = optimal_a2, class_weight='balanced')

clf.fit(X_train_tfidf,Y_train)

predt1 = clf.predict(X_test_tfidf)

acct1 = accuracy_score(Y_test, predt1) * 100
pret1 = precision_score(Y_test, predt1) * 100
rect1 = recall_score(Y_test, predt1) * 100
f1t1 = f1_score(Y_test, predt1) * 100

print('\nAccuracy=%f%%' % (acct1))
print('\nprecision=%f%%' % (pret1))
print('\nrecall=%f%%' % (rect1))
print('\nF1-Score=%f%%' % (f1t1))
```

```
Accuracy=81.833294%

precision=96.283449%

recall=81.707317%
```

F1-Score=88.398544%

```
In [53]: cm = confusion_matrix(Y_test,predt1)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix for Tfidf')
         plt.show()
```



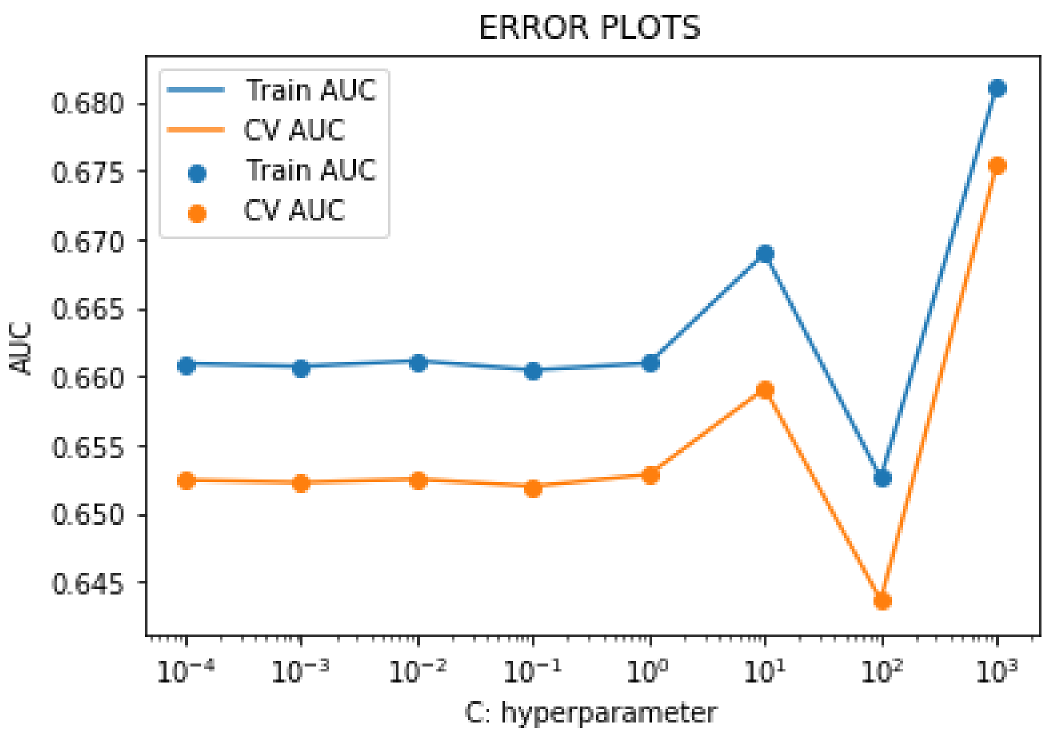
7.2.7 [5.2.3] Applying RBF SVM on AVG W2V, SET 3

7.2.8 Hyperparameter tuning using GridSearch

```
In [47]: #clf = SVC()
         C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
         grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=1)
         grid.fit(sent_vectors_train, Y_train)

         print("best C = ", grid.best_params_)

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

```
best C =  {'C': 1000}
```

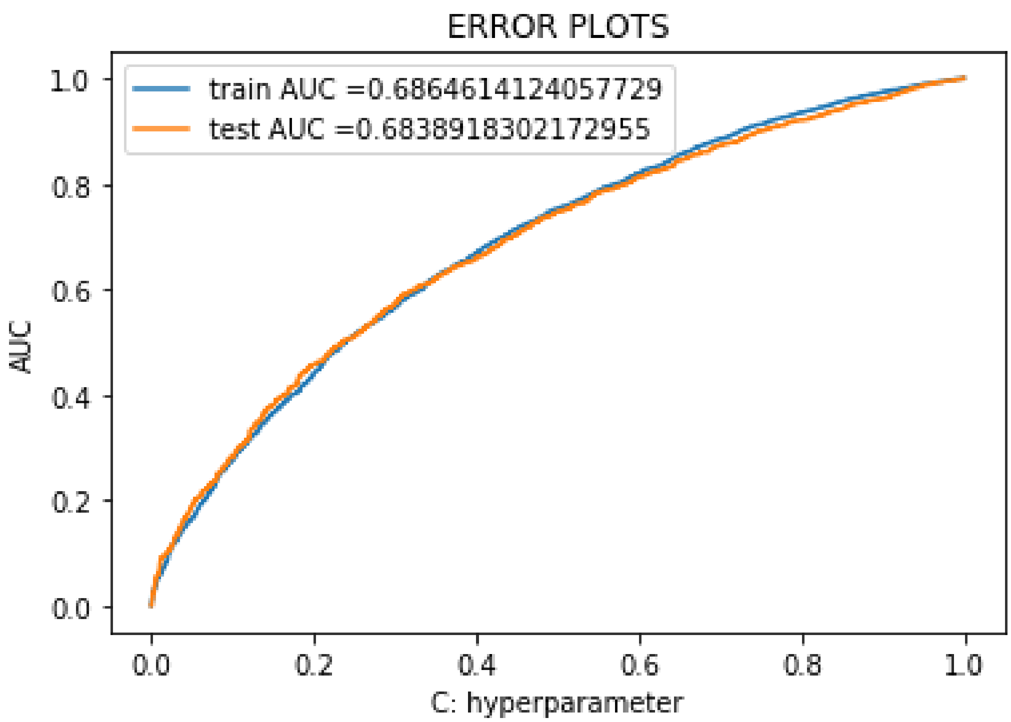



```
In [48]: a = grid.best_params_
         optimal_a2 = a.get('C')
```

7.2.9 Testing with Test Data

```
In [49]: clf = SVC(kernel='rbf', C = optimal_a2, probability=True)
         clf.fit(sent_vectors_train, Y_train)

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

```
train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_proba
test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(se

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

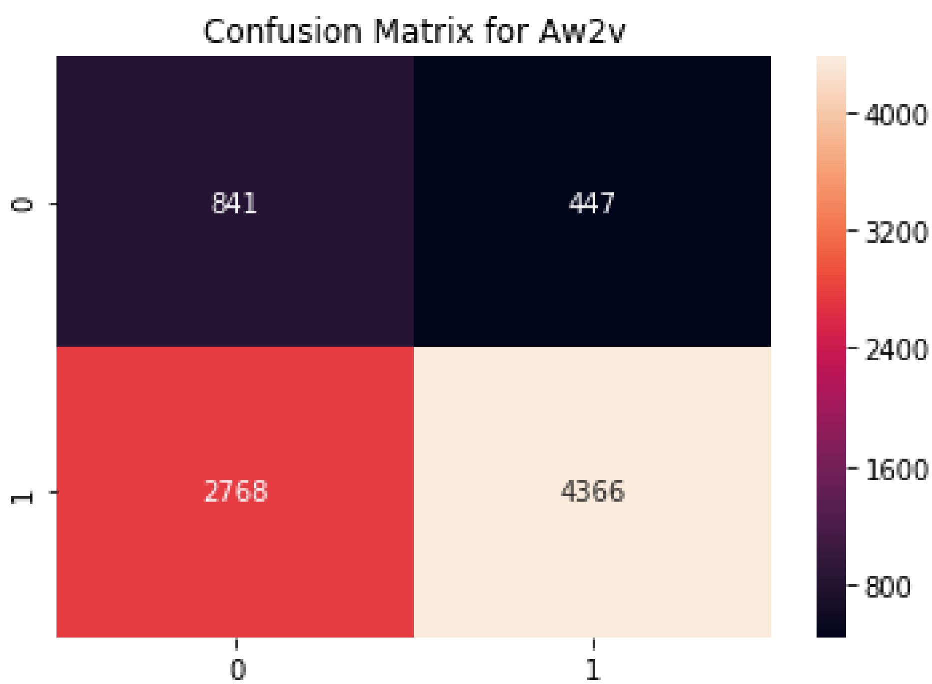



```
In [50]: clf = SVC(kernel='rbf', C = optimal_a2, class_weight='balanced')

         clf.fit(sent_vectors_train,Y_train)

         preda1 = clf.predict(sent_vectors_test)

         acca1 = accuracy_score(Y_test, preda1) * 100
         prea1 = precision_score(Y_test, preda1) * 100
         reca1 = recall_score(Y_test, preda1) * 100
         f1a1 = f1_score(Y_test, preda1) * 100

         print('\nAccuracy=%f%%' % (acca1))
         print('\nprecision=%f%%' % (prea1))
```

```
print('\nrecall=%f%%' % (reca1))
print('\nF1-Score=%f%%' % (f1a1))
```

```
Accuracy=61.826170%

precision=90.712653%

recall=61.199888%

F1-Score=73.089479%
```

In [54]:
```
cm = confusion_matrix(Y_test,preda1)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix for Aw2v')
plt.show()
```



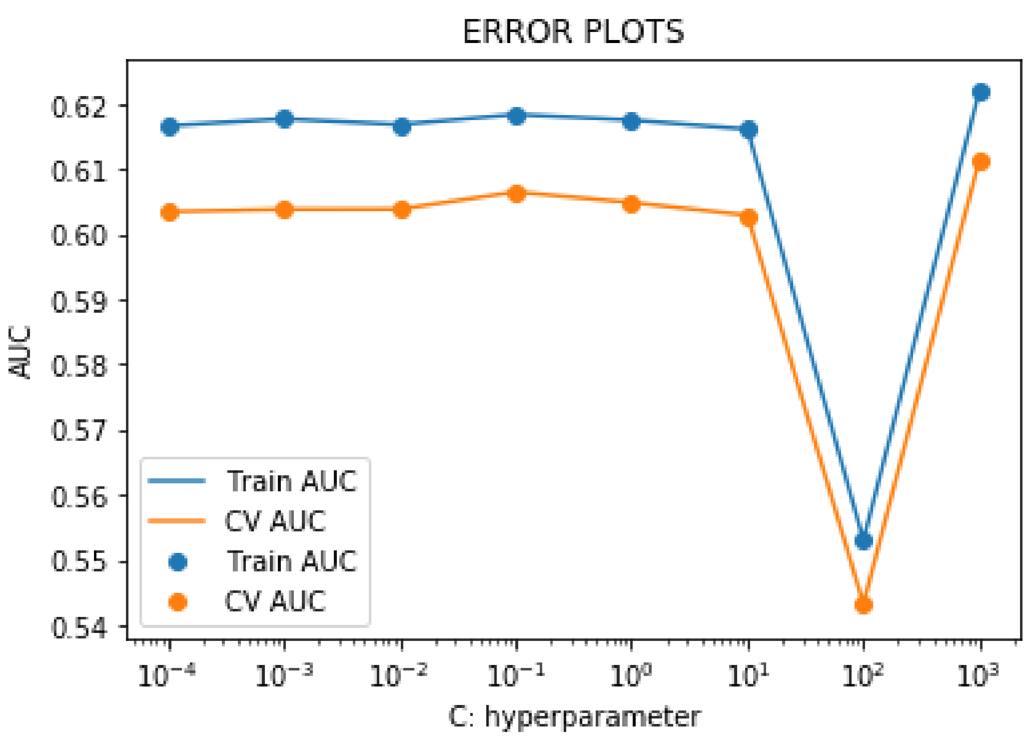
7.2.10 [5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

7.2.11 Hyperparameter tuning with GridSearch

In [52]:
```
#clf = SVC()
C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
parameters = {'C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
```

```
grid = GridSearchCV(SVC(kernel='rbf'), parameters, cv=3, scoring='roc_auc', n_jobs=1)
grid.fit(tfidf_sent_vectors_train, Y_train)

print("best C = ", grid.best_params_)

train_auc_tfw2v = grid.cv_results_['mean_train_score']
cv_auc_tfw2v = grid.cv_results_['mean_test_score']

plt.plot(C, train_auc_tfw2v, label='Train AUC')
plt.scatter(C, train_auc_tfw2v, label='Train AUC')
plt.plot(C, cv_auc_tfw2v, label='CV AUC')
plt.scatter(C, cv_auc_tfw2v, label='CV AUC')

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

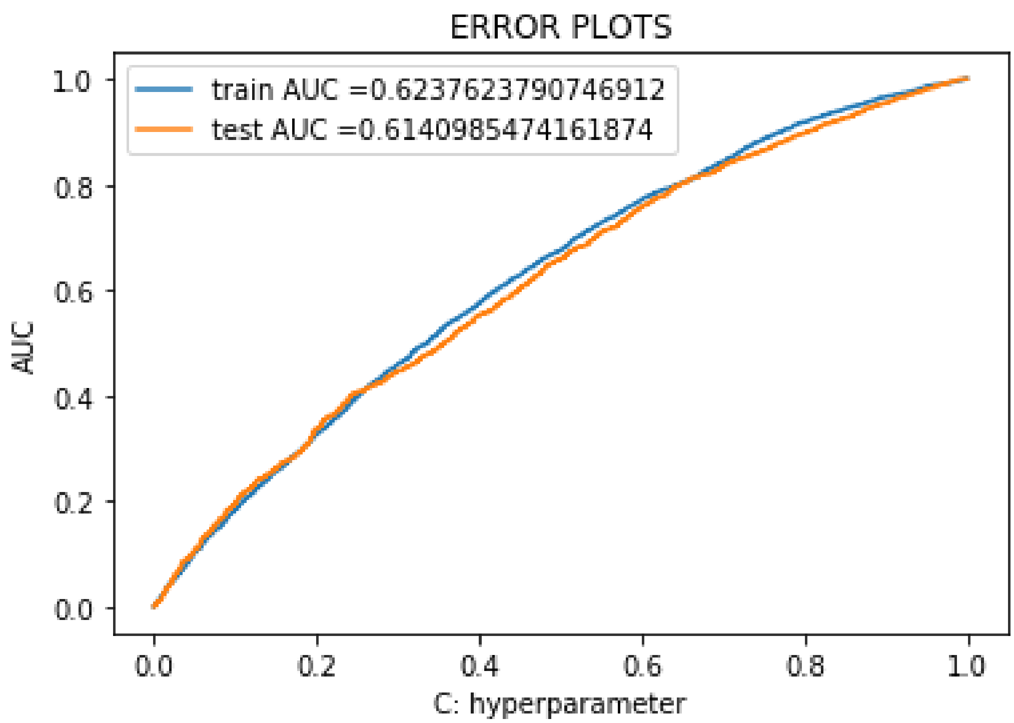
```
best C =  {'C': 1000}
```




```
In [55]: a = grid.best_params_
         optimal_a2 = a.get('C')
```

7.2.12 Testing with Test Data

In [56]:
```
clf = SVC(kernel='rbf', C = optimal_a2, probability=True)
clf.fit(tfidf_sent_vectors_train, Y_train)

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

train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_pr
test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_proba

plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw2v
plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, te
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

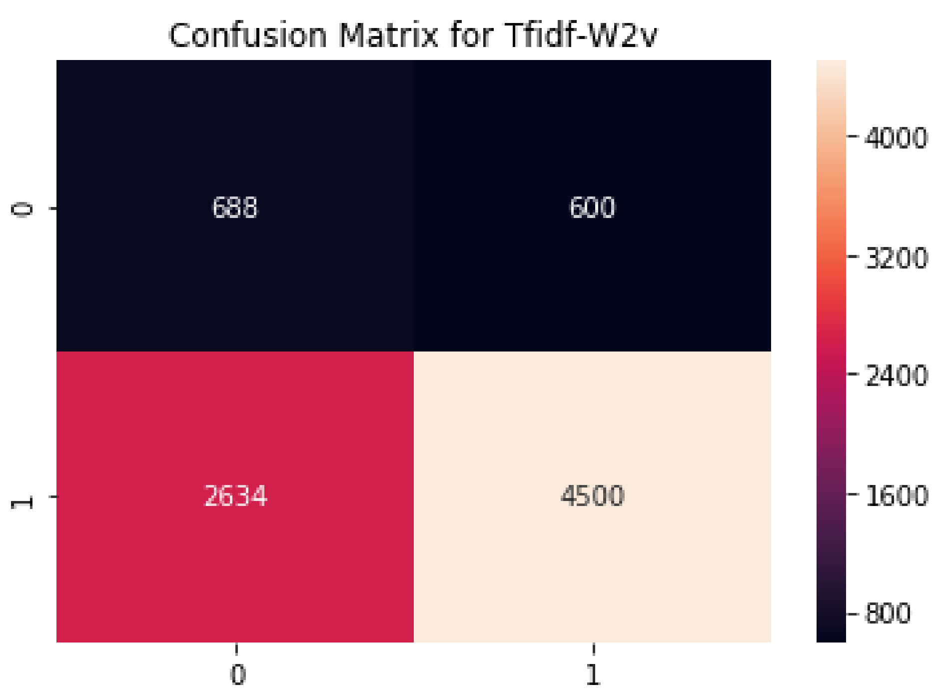


In [57]:
```
clf = SVC(kernel='rbf', C = optimal_a2, class_weight='balanced')

clf.fit(tfidf_sent_vectors_train,Y_train)

predw1 = clf.predict(tfidf_sent_vectors_test)
```

```
accw1 = accuracy_score(Y_test, predw1) * 100
prew1 = precision_score(Y_test, predw1) * 100
recw1 = recall_score(Y_test, predw1) * 100
f1w1 = f1_score(Y_test, predw1) * 100

print('\nAccuracy=%f%%' % (accw1))
print('\nprecision=%f%%' % (prew1))
print('\nrecall=%f%%' % (recw1))
print('\nF1-Score=%f%%' % (f1w1))
```

```
Accuracy=61.600570%

precision=88.235294%

recall=63.078217%

F1-Score=73.565473%
```

In [58]:
```
cm = confusion_matrix(Y_test,predw1)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix for Tfidf-W2v')
plt.show()
```



8 [6] Conclusions

In [106]:
```python
# 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"]
acc= [accb,accb1,acct,acct1,acca,acca1,accw,accw1]
pre= [preb,preb1,pret,pret1,prea,prea1,prew,prew1]
rec= [recb,recb1,rect,rect1,reca,reca1,recw,recw1]
f1= [f1b,f1b1,f1t,f1t1,f1a,f1a1,f1w,f1w1]


#Initialize Prettytable
ptable = PrettyTable()
ptable.add_column("Index", number)
ptable.add_column("Model", name)
ptable.add_column("SVM", svm)
ptable.add_column("Accuracy%", acc)
ptable.add_column("Precision%", pre)
ptable.add_column("Recall%", rec)
ptable.add_column("F1%", f1)

print(ptable)
```

```
+-------+-----------+--------+--------------------+-------------------+--------------------+---
| Index |   Model   |  SVM   |     Accuracy%      |     Precision%    |      Recall%       |
+-------+-----------+--------+--------------------+-------------------+--------------------+---
|   1   |    Bow    | Linear |  84.5701048154337  | 95.26440014061166 | 85.87596197374377  |  90
|   2   |    Bow    |  RBF   | 82.75943956304916  |  96.2251871135698 | 82.89879450518643  | 89
|   3   |   Tfidf   | Linear | 80.81042078079903  | 96.24341783833668 | 80.25803531009507  | 87
|   4   |   Tfidf   |  RBF   | 81.83329375445263  | 96.28344895936571 | 81.70731707317073  | 88
|   5   |  Avg W2v  | Linear | 83.89032356068662  | 83.89032356068662 |       100.0        | 91
|   6   |  Avg W2v  |  RBF   | 61.826169555924956 | 90.71265323083317 | 61.19988786094758  | 73
|   7   | Tfidf W2v | Linear | 83.89032356068662  | 83.89032356068662 |       100.0        | 91
|   8   | Tfidf W2v |  RBF   | 61.60056993588221  | 88.23529411764706 | 63.07821698906644  | 73
+-------+-----------+--------+--------------------+-------------------+--------------------+---
```

1. Here we have used 100k data points for Linear SVM and 30k datapoints for RBF SVM.
2. Wh have used SGDClassifier with Hinge loss for Linear SVM and SVC for RBF SVM.
3. For Linear SVM(BOW) optimal alpha is 0.001 and for rest(TFIDF, AW2V, TFIDF-W2V) is 0.0001
4. For RBF SVM(BOW) optimal C is 10 and for rest optimal C is 1000
5. Linear SVM gives better accuracy value than RBF SVM.
6. Also BOW is better than other models