05 Amazon Fine Food Reviews Analysis_Logistic Regression

August 6, 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")
from bs4 import BeautifulSoup


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/
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.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

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
from sklearn.externals import joblib
```

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

In [2]:
```
# 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 point
# 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 5
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100

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

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

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

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

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

                 Summary                                              Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
```

```
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

```
In [3]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [4]: print(display.shape)
        display.head(3)
```

```
(80668, 7)
```

```
Out[4]:               UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1

                                                        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
```

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

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

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

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

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

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

```
In [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
```

```
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head(3)
```

Out[7]:
```
       Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2

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

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  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 ...
```

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

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

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

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

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

Out[9]: (87775, 10)

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

Out[10]: 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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

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

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

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

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

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

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

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

```
(87773, 10)
```

Out[13]:
```
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 [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)
```

```
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 [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)
```

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

In [16]:
```
# 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)
```

```
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 [14]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

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

In [0]:
```
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 oth
==================================================
```

In [0]:
```
#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 [0]:
```
#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 [15]:
```
# 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', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
            '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 [16]: # Combining all the above stundents

         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 stopw
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87773/87773 [00:34<00:00, 2507.97it/s]
```

```
In [17]: preprocessed_reviews[1500]
```

```
Out[17]: 'way hot blood took bite jig lol'
```

5 [4] Splitting the Data

```
In [18]: x = preprocessed_reviews
         y = final["Score"].values
```

Splitting the data as train data, cross validation data and test data

```
In [19]: # splitting the data into 3 parts for furhter process,
         # train data, cross validation data and test data

         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)         # t
         x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size=0.30) # t
```

```
In [20]: # 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))
```

```
43008
18433
26332
```

6 [4] Featurization

6.1 [4.1] BAG OF WORDS

```
In [75]: #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', 'absorbed', 'acai'
==================================================
(43008, 5000) (43008,)
(18433, 5000) (18433,)
(26332, 5000) (26332,)
==================================================
```

```
In [36]: ############## writing bag of words

         joblib.dump(x_train_bow, 'x_tr_bow100k.pkl')
         joblib.dump(x_test_bow , 'x_te_bow100k.pkl')
         joblib.dump(x_cv_bow   , 'x_cv_bow100k.pkl')

         joblib.dump(y_train, 'y_train_bow100k.pkl')
         joblib.dump(y_test , 'y_test_bow100k.pkl')
         joblib.dump(y_cv   , 'y_cv_bow100k.pkl')

Out[36]: ['y_cv_bow100k.pkl']
```

6.2 [4.3] TF-IDF

```
In [76]: # 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', 'absorbed', 'acai
==================================================
After featurization

(43008, 5000) (43008,)
(18433, 5000) (18433,)
(26332, 5000) (26332,)
==================================================
```

```
In [50]: ############## writing tfidf features

         joblib.dump(x_train_tf, 'x_tr_tfidf100k.pkl')
         joblib.dump(x_test_tf , 'x_te_tfidf100k.pkl')
         joblib.dump(x_cv_tf   , 'x_cv_tfidf100k.pkl')

         joblib.dump(y_train, 'y_train_tfidf100k.pkl')
         joblib.dump(y_test , 'y_test_tfidf100k.pkl')
         joblib.dump(y_cv   , 'y_cv_tfidf100k.pkl')
```

```
Out[50]: ['y_cv_tfidf100k.pkl']
```

6.3 [4.4] Word2Vec

```
In [21]: # 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 [24]: # 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 [25]: 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  12472
sample words  ['excellent', 'tea', 'canada', 'could', 'find', 'two', 'makers', 'get', 'maker',
```

6.4 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v converting train data

In [27]:
```
# 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(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%|| 43008/43008 [01:13<00:00, 586.27it/s]


(43008, 200)
[ 6.49759204e-05 -4.01210485e-04 -1.96987505e-04  1.86674397e-04
 -3.93062613e-04 -3.95692781e-04  1.28592659e-04 -5.81469275e-04
  4.70423633e-04  2.20779065e-04  5.01036455e-04 -5.79889586e-06
 -6.81360504e-04  4.06504592e-04  4.39158857e-04 -4.52771211e-04
 -5.18123185e-04 -2.06430349e-04  3.01051062e-05 -6.26732314e-04
 -1.15786870e-04 -1.24883969e-04 -9.64300387e-05 -7.65322069e-05
 -3.96786912e-04  3.01352922e-04 -7.21676791e-05  9.52072904e-04
  1.56492573e-04 -3.05378818e-04 -1.98475498e-04 -4.89442053e-04
  2.84798705e-04  1.18249424e-05  1.09017731e-05 -1.21266353e-04
  7.28418105e-04 -1.91253396e-04 -6.03933902e-04  2.90955751e-04
  1.36362929e-04  2.19257390e-04 -1.94731879e-05 -3.81956561e-04
  4.75944406e-04 -6.16348574e-04  4.22344096e-04 -5.76920092e-04
  9.38924194e-05 -4.10423917e-04 -3.86084836e-04  1.58965829e-05
 -5.95699610e-04 -6.37844918e-04  2.46262401e-04 -2.44120830e-05
 -4.46473554e-04  4.92073369e-04 -2.72515400e-05  6.03172991e-05
  2.50730610e-04 -5.44709353e-04 -5.96124858e-04  3.65358407e-05
  4.39838551e-04  2.33081671e-04 -3.85497251e-04 -2.03161001e-04
  5.20842854e-04  6.23440148e-05  1.80451692e-04  5.92701396e-04
 -5.86813549e-05  1.91423026e-04 -8.23013243e-04 -1.56216588e-04
  1.01790418e-04  1.70859852e-05  4.27651107e-04  6.32332563e-05
  3.45331337e-04 -4.99068909e-05 -4.79262580e-04 -2.71710742e-05
 -4.93192732e-04 -2.26665398e-04  7.39147905e-04 -7.67093902e-04
 -3.63686532e-04 -2.86699055e-04 -2.38230654e-04  2.56952322e-04
```

```
  4.74037767e-04 -1.93245609e-04 -4.58647327e-05 -5.07645326e-04
 -2.81645342e-04 -1.64829760e-04  3.80434117e-04 -4.32361914e-04
 -1.63783560e-04 -2.08929845e-04 -4.40110459e-04  5.16464518e-06
 -1.50112883e-04 -2.15631427e-04 -2.44774946e-04  3.76047367e-04
 -1.29596779e-05  1.04429203e-03 -4.19369234e-05  8.51692114e-05
 -1.57659392e-04  4.71947652e-05 -4.84818069e-04  3.47919394e-04
  4.87278653e-04 -1.19406088e-04 -1.69411334e-04 -6.71505000e-04
  3.93459353e-04 -2.99256446e-04  1.24246934e-04 -1.02825654e-04
 -9.63593158e-04  4.81979884e-04 -2.55358686e-04  2.21120963e-04
 -4.71681782e-04  1.10714206e-04  1.66992197e-05 -1.68547732e-05
 -1.11216779e-03  3.79408049e-04  3.68347192e-04 -7.85754039e-05
  1.78414812e-05 -3.09244814e-04 -1.20281006e-04  5.31448837e-04
 -1.04655137e-04  4.15186560e-04  4.10789243e-04  1.16622274e-04
  4.51028825e-04  3.63063307e-04 -6.65880064e-04  4.33785901e-04
 -2.72301937e-04  3.06393756e-04 -3.62828852e-04 -3.99346100e-04
  3.43876338e-04  2.58125382e-04  8.58824790e-04 -7.05141304e-05
 -2.11588138e-04 -3.38762977e-04 -6.63754285e-05 -8.89204010e-05
 -1.92854404e-04 -6.99194435e-05  2.50763883e-04 -3.00660717e-05
  4.03485260e-04 -1.12916895e-04  1.85531900e-04 -2.63258137e-05
 -2.85088588e-04 -5.12822579e-04 -4.13335508e-04  2.02946790e-04
 -6.27907082e-04 -1.67283369e-04  6.13648988e-05  5.52147776e-04
 -2.67722344e-04 -1.64403850e-04  2.22268113e-04 -2.85741298e-05
 -2.51584106e-04 -2.62495413e-04 -7.14193520e-05  5.01744832e-04
 -7.97117339e-04 -1.63002565e-04  2.16393481e-04  1.76400293e-04
 -8.76360656e-05 -2.95655794e-04 -4.68348361e-05 -3.35334574e-04
 -9.11909135e-05  3.41075151e-04  2.57341519e-04 -4.75211574e-04
  2.78015619e-04 -6.22203942e-04 -1.07132159e-03  3.19772211e-04]
```

In [28]:
```python
type(sent_vectors_train)
```

Out[28]:
```
numpy.ndarray
```

converting cv data

In [29]:
```python
list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [30]:
```python
# 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
```

```python
                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%|| 18433/18433 [00:31<00:00, 577.48it/s]


(18433, 200)
[ 1.16855281e-04 -3.17491519e-04 -1.20220506e-04 -1.44613717e-04
 -3.00735857e-04 -2.26813029e-04 -1.63127571e-04 -1.92485348e-04
 -1.52865692e-04  2.05376198e-04  3.99788965e-04 -3.93996413e-05
 -1.64488527e-04  2.39554695e-04 -2.15464873e-04 -2.93143705e-04
  2.33918112e-04  2.65208670e-05  7.98673136e-06  5.01255722e-05
 -1.38347057e-04  1.61930907e-04 -9.12943366e-05 -6.90486334e-05
 -2.93101682e-04  1.18370465e-04 -1.87933830e-04 -2.12501105e-04
  1.36790226e-04 -2.97779194e-04  3.18305654e-04 -4.27578008e-04
 -1.57667365e-04  4.07461678e-05  8.11535602e-05 -3.34954983e-04
 -1.06396723e-04 -1.11633577e-04  1.34797164e-04  2.50672578e-04
  5.27413341e-05  3.33075107e-04  4.76527938e-04  1.73796322e-04
 -3.15455396e-04  2.27128396e-04 -2.32001786e-04  7.72218942e-04
  6.65636070e-05 -2.64314131e-04 -2.09448180e-04 -1.55663125e-04
 -8.03119160e-05 -3.00346969e-04 -5.76618763e-05 -7.46521552e-06
 -1.63014051e-04  3.33713006e-04  2.45533076e-04  4.86178487e-04
 -2.21784122e-04  2.30954482e-04  2.92317986e-04 -3.17879837e-04
 -1.82280624e-04  1.34920397e-04  4.65926286e-04  4.47713329e-04
 -2.66699079e-04  3.01376215e-04  1.28268994e-04  3.27684475e-04
  4.12212841e-04 -2.01420264e-04  1.27540320e-04  2.95746912e-05
 -1.74014317e-04 -1.51584254e-05 -2.89460410e-05  8.31956131e-05
  2.94056315e-04 -1.78071995e-04 -2.30242140e-04 -3.51065372e-04
 -4.05077673e-04 -3.60692959e-04  5.73650984e-05 -2.05483241e-04
 -2.60306689e-04 -5.75437872e-04 -2.44564539e-04 -9.17338782e-05
 -1.88531971e-04 -1.17027580e-04  4.17310726e-04 -8.44653098e-06
  1.29817186e-04  1.09044024e-04 -4.61346121e-04  5.79029070e-04
 -2.97756321e-05 -2.02839742e-05 -2.73696678e-04 -1.09954133e-04
  7.32544295e-06 -1.69653606e-04 -1.86089656e-05 -6.13508766e-05
 -3.21827493e-04  5.43409527e-04  7.67897939e-04 -5.77435642e-05
 -3.33748260e-04 -2.44599624e-04 -7.63199139e-05 -2.95591318e-04
 -3.38269246e-04  2.35837105e-05 -1.81418251e-05 -6.85736599e-04
  2.82524910e-04  2.73430324e-04 -3.29154650e-05 -1.16294565e-04
 -2.83789730e-04  1.28844578e-04  6.90119586e-04  3.25888230e-04
  4.49833889e-05  3.01234772e-04 -2.13973865e-04  1.50048761e-04
 -3.50690820e-04  2.07118102e-05  2.67075853e-04  2.77949239e-04
  7.48069786e-04  2.63766269e-05  1.71999314e-04 -1.41238918e-04
  5.54458029e-05  1.16587590e-04  6.45711543e-04  2.91366527e-04
```

```
  2.68627309e-04 -4.03567672e-04  2.20319736e-04  4.97608611e-05
 -3.25726945e-04  2.75439078e-05  4.83691089e-04  3.97197683e-04
 -5.13893904e-04 -1.91445200e-05  2.17059488e-04  3.81562655e-05
 -5.69926097e-05  8.22620650e-05 -2.39010675e-04  5.21588617e-04
  1.09451513e-04 -3.06703335e-05  2.30939834e-04  3.01193417e-04
 -4.19023019e-04 -1.60373562e-05  1.10394496e-04 -1.66738490e-05
  1.71218834e-04  3.19005379e-04 -4.93753787e-04 -3.26593563e-04
  8.20631689e-04  2.13464973e-04  1.91991036e-04  2.12138453e-04
 -3.00663762e-04 -6.82194242e-04  4.32936232e-04 -7.13383038e-06
  4.78109275e-04  1.60036091e-04  2.59564179e-04  1.16896527e-04
  2.22516338e-04  1.61123798e-05 -3.41609875e-04 -1.21746125e-05
 -3.70990824e-05  1.58981626e-04  5.04312956e-05  2.39257983e-04
  5.31763164e-05 -9.11195824e-05 -3.36089798e-04 -1.32401313e-05
  2.30864783e-04 -5.51974032e-04  1.76539208e-04 -1.26959005e-04]
```

converting test data

In [31]:
```
list_of_sentance_test=[]
for sentance in x_test:
    list_of_sentance_test.append(sentance.split())
```

In [32]:
```
# 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(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%|| 26332/26332 [00:45<00:00, 580.50it/s]


(26332, 200)
[ 2.44657602e-04 -3.82770538e-06 -1.65797182e-04 -2.67196673e-04
 -2.94189998e-04 -1.83520926e-05  3.71838370e-04 -6.27543121e-04
 -3.13828263e-04  8.88660410e-05  4.74972029e-04  3.59352684e-04
 -5.77392770e-04  1.75332721e-05  7.57770055e-04 -4.03996464e-04
 -2.27566947e-04  9.08476440e-05  6.16209931e-05 -1.01442970e-04
```

```
  1.20363843e-04  1.83817013e-04  4.15593002e-05  3.65880583e-04
 -4.37059493e-04 -4.09149505e-04 -1.86081647e-04 -2.20838061e-04
 -1.30986020e-04 -2.30593454e-04 -3.46377352e-04  3.37129751e-04
 -1.34481979e-04 -3.37700057e-04 -7.34359438e-04  2.67073188e-04
 -5.80207992e-05 -6.77376479e-04 -1.98991525e-04  4.16251993e-04
  2.12386361e-05  3.10751511e-04 -5.02228562e-06 -8.27586873e-05
  5.37937663e-04 -3.40164600e-05  6.28020082e-04  7.09034767e-04
  1.39479924e-05 -2.54827616e-04  2.23274684e-04  2.94947641e-04
  2.01781208e-04 -5.52410298e-04 -3.91599290e-04  2.15113375e-04
 -5.10607037e-05  1.70028141e-04  3.81539782e-04 -7.18643207e-05
 -4.07492746e-05 -2.76094272e-04 -3.45940657e-04 -4.15176242e-04
 -2.04200532e-04  2.46420894e-04  2.54971217e-04  2.90581947e-04
 -8.46184150e-06  9.04217328e-04  3.72461346e-05  6.80280825e-04
 -9.49248293e-06 -4.46303687e-04 -2.60605956e-04  2.09473945e-04
 -3.67717922e-04 -1.93539318e-04  3.54892679e-04 -5.13934420e-05
  1.29198031e-04 -4.78597200e-04 -1.48448663e-04  3.49202202e-05
  2.11545251e-04 -5.69994623e-04  1.05719753e-04 -3.67277438e-04
 -3.46832804e-05 -3.87744320e-05  1.15512507e-04  1.34058130e-04
 -1.26421437e-04  6.47118757e-05  3.60218393e-04 -3.57974019e-04
  2.80502381e-04  4.25098313e-04 -5.28584237e-04 -1.68389445e-04
 -1.56414708e-04 -1.25821981e-04 -3.10378266e-04 -4.98106714e-04
  5.09119139e-05  1.25391903e-04 -3.47765965e-04 -4.88682558e-04
 -2.86776462e-04  6.17311537e-04  3.23533846e-04 -2.53852807e-04
 -6.68744390e-05  1.32988724e-04 -3.97592542e-04  6.38576833e-05
  2.85213471e-05 -1.19353820e-04 -2.52574070e-04  2.15630278e-04
  1.52658307e-04  2.37003188e-04  9.95455691e-05 -5.94131340e-04
 -3.51306094e-04  6.42164267e-05  3.79612742e-04  1.56837149e-05
  9.24768980e-05  1.49291122e-04  3.93892253e-05  2.24530714e-04
 -3.06337150e-04 -5.01079782e-05  1.53975183e-04 -2.33681872e-04
  3.98191616e-04  5.33963904e-05 -1.99949089e-05  3.16721285e-04
  2.13203452e-04 -1.83147157e-04  4.20808562e-04  2.63011787e-04
  5.59326849e-05 -1.50269860e-04 -4.58394687e-05 -2.38366545e-04
 -3.21102811e-04 -4.37414560e-05 -4.63514072e-04 -1.52878932e-04
  3.93371165e-05 -1.91150235e-04 -3.09785450e-05  2.20481448e-04
  4.35853871e-04  1.24440086e-04  1.65341065e-04  1.93958413e-04
  2.65359659e-04 -1.32494880e-04 -1.30938928e-05 -3.77085680e-05
  1.46215818e-04  2.04161928e-04 -9.91059767e-05  2.80344438e-04
 -4.50549979e-04  2.29512434e-04  3.36159345e-04 -7.74333848e-04
  6.19113282e-05 -2.36805226e-04  2.89910481e-04 -1.89431828e-04
 -2.28071919e-04 -6.28124876e-04 -1.09216140e-06  3.66677064e-04
  7.00320656e-04  9.59689828e-04 -4.33444239e-04  2.69893637e-04
  3.54069038e-04 -2.41929240e-04 -2.40426738e-04 -6.54208008e-05
 -2.58772029e-04  1.70471071e-04  5.56216792e-05 -1.64616814e-04
 -2.68826317e-04  4.80182444e-04 -9.58171792e-05  6.74981845e-05
 -1.74109869e-04 -2.35014294e-04  1.31011510e-05 -2.38138648e-04]
```

In [33]:
```
############## writing average w2v for 100k data points with 200 features ###########
```

```
joblib.dump(sent_vectors_train, 'sent_vectors_train_100k.pkl')
joblib.dump(sent_vectors_test , 'sent_vectors_test_100k.pkl')
joblib.dump(sent_vectors_cv   , 'sent_vectors_cv_100k.pkl')

joblib.dump(y_train, 'y_train.pkl')
joblib.dump(y_test , 'y_test.pkl')
joblib.dump(y_cv   , 'y_cv.pkl')
```

Out[33]: ['y_cv.pkl']

[4.4.1.2] TFIDF weighted W2v converting train data

In [34]:
```
# 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 [35]:
```
# 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(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%|| 43008/43008 [17:18<00:00, 41.42it/s]
```

converting cv data

```
In [36]: # 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_cv = []; # the tfidf-w2v for each sentence/review is stored in thi
         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%|| 18433/18433 [07:39<00:00, 40.12it/s]
```

converting test data

```
In [37]: # 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(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%|| 26332/26332 [10:45<00:00, 40.77it/s]
```

```
In [38]: ############## writing average w2v with 100k datapoints and 200 features ############

         joblib.dump(tfidf_sent_vectors_train, 'tfidf_sent_vectors_train_100k.pkl')
         joblib.dump(tfidf_sent_vectors_test , 'tfidf_sent_vectors_test_100k.pkl')
         joblib.dump(tfidf_sent_vectors_cv   , 'tfidf_sent_vectors_cv_100k.pkl')

         joblib.dump(y_train, 'y_train.pkl')
         joblib.dump(y_test , 'y_test.pkl')
         joblib.dump(y_cv   , 'y_cv.pkl')
```

```
Out[38]: ['y_cv.pkl']
```

7 [5] Assignment 5: Apply Logistic Regression

```
<li><strong>Apply Logistic Regression 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>Hyper paramter tuning (find best hyper parameters corresponding the algorithm that
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaico
<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>Pertubation Test</strong>
    <ul>
<li>Get the weights W after fit your model with the data X i.e Train data.</li>
<li>Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse
```

matrix, X.data+=e)

```
<li>Fit the model again on data X' and get the weights W'</li>
<li>Add a small eps value(to eliminate the divisible by zero error) to W and W i.e
```

W=W+10^-6 and W′ = W′+10^-6

```
<li>Now find the % change between W and W' (| (W-W') / (W) |)*100)</li>
<li>Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in t
<li> Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is suc
        <li> Print the feature names whose % change is more than a threshold x(in our example i
    </ul>
</li>
<br>
<li><strong>Sparsity</strong>
    <ul>
<li>Calculate sparsity on weight vector obtained after using L1 regularization</li>
    </ul>
</li>
<br><font color='red'>NOTE: Do sparsity and multicollinearity for any one of the vectorizers. B
<br>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Get top 10 important features for both positive and negative classes separately.</li>
    </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 for
    <img src='summary.JPG' width=400px>
```


Note: Data Leakage

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

8 Applying Logistic Regression

8.1 [5.1] Logistic Regression on BOW, SET 1

8.1.1 [5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

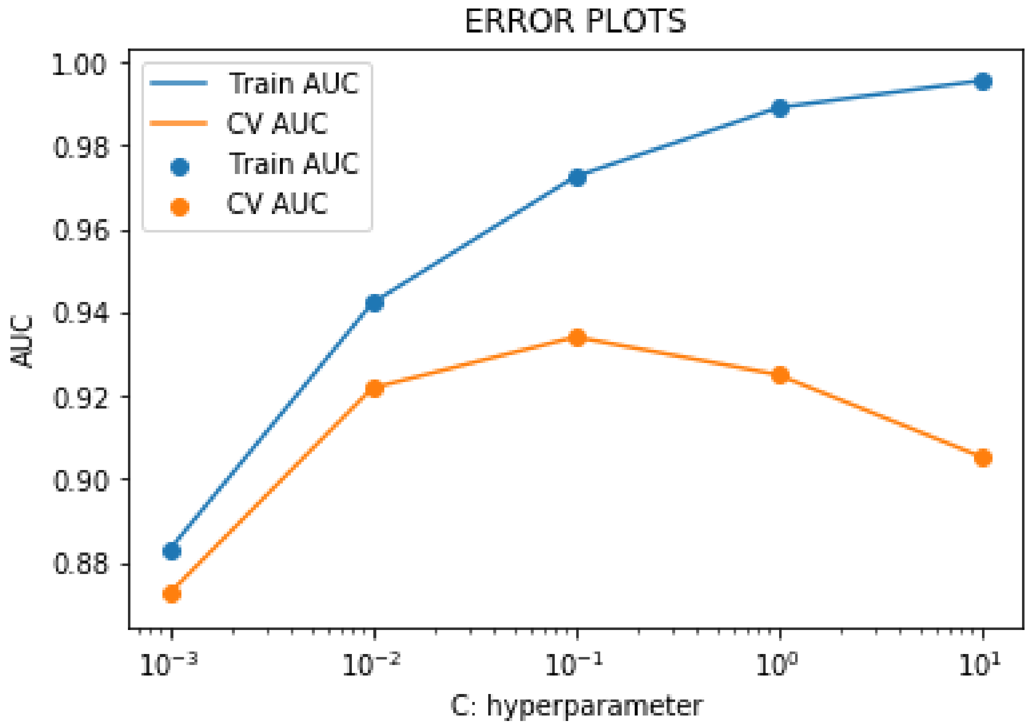
8.1.2 Hyperparameter tuning using GridSearchCV

In [77]:
```
lam = [0.001,0.01,0.1,1,10]
clf = LogisticRegression()
param_grid = {'C':lam}

grid = GridSearchCV(estimator = clf,param_grid=param_grid ,cv = 3,n_jobs = -1, scoring
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_a1 = a.get('C')
```

```
best C =  {'C': 0.1}
Accuracy on train data =  93.40263192425276
```

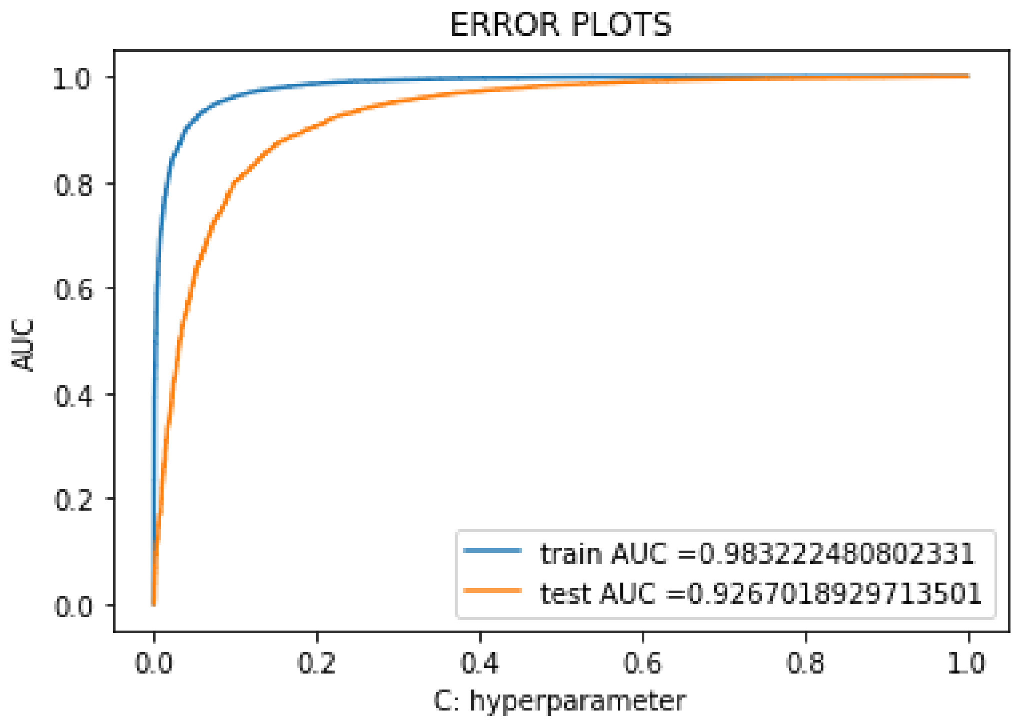
In [78]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc

bow_auc_train      = grid.cv_results_['mean_train_score']
bow_auc_cv         = grid.cv_results_['mean_test_score']

plt.plot(lam, bow_auc_train, label='Train AUC')
plt.scatter(lam, bow_auc_train, label='Train AUC')

plt.plot(lam, bow_auc_cv, label='CV AUC')
plt.scatter(lam, bow_auc_cv, label='CV AUC')

plt.legend()
plt.xlabel("C: hyperparameter")
```

```
plt.ylabel("AUC")
plt.xscale('log')
plt.title("ERROR PLOTS")
plt.show()
```



"Here we can observe that for our hyperparameter C best value is 0.1"
Testing with test data

In [79]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk

clf = LogisticRegression()
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_tes

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



[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

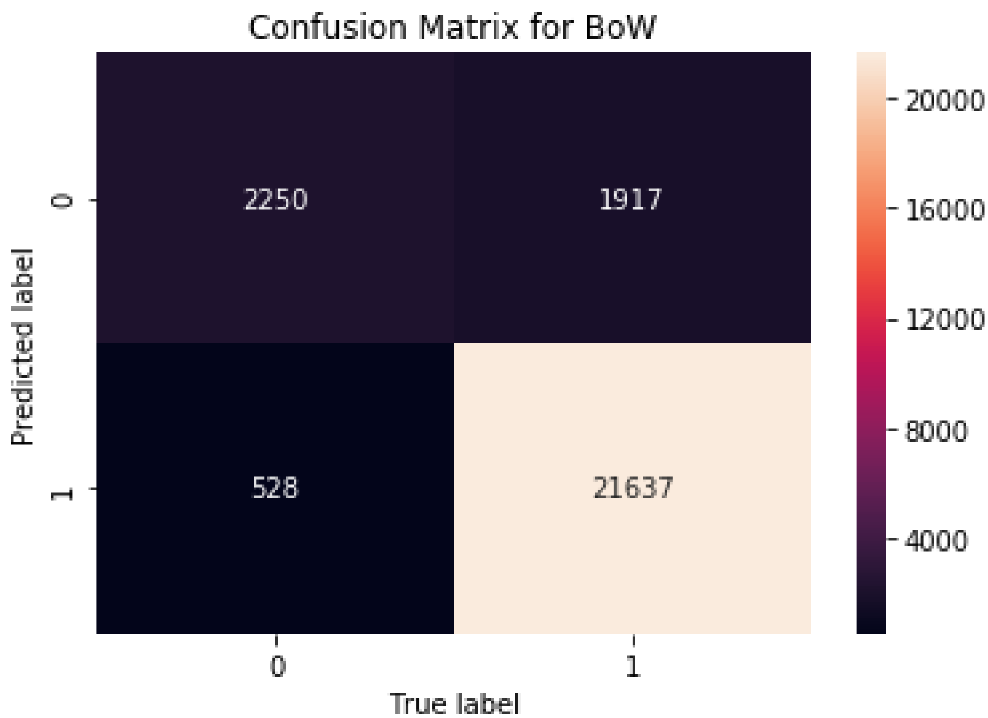
Calculating Confusion Matrix

In [80]:
```
clf = LogisticRegression(C = optimal_a1 , penalty='l1')
clf.fit(x_train_bow, y_train)

pred = clf.predict(x_test_bow)

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

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

```
Accuracy  = 90.684338%

precision = 91.714419%
```

```
recall    = 97.733852%

F1-Score  = 94.628506%
```

```
In [67]: cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.ylabel('Predicted label')
         plt.xlabel('True label')
         plt.title('Confusion Matrix for BoW')
         plt.show()
```




```
In [68]: clf = LogisticRegression(C=100, penalty='l1')
         clf.fit(x_train_bow, y_train)

         pred = clf.predict(x_test_bow)
         ac1 = accuracy_score(y_test, pred) * 100
         er1 = np.around(100 - ac1, decimals = 2)

         w = clf.coef_
         s1 = np.count_nonzero(w)

         #=====================================================================
```

```
clf = LogisticRegression(C=10, penalty='l1')
clf.fit(x_train_bow, y_train)

pred = clf.predict(x_test_bow)
ac2 = accuracy_score(y_test, pred) * 100
er2 = np.around(100 - ac2, decimals = 2)

w = clf.coef_
s2 = np.count_nonzero(w)

#=======================================================================

clf = LogisticRegression(C=1, penalty='l1')
clf.fit(x_train_bow, y_train)

pred = clf.predict(x_test_bow)
ac3 = accuracy_score(y_test, pred) * 100
er3 = np.around(100 - ac3, decimals = 2)

w = clf.coef_
s3 = np.count_nonzero(w)

#=======================================================================

clf = LogisticRegression(C=0.1, penalty='l1')
clf.fit(x_train_bow, y_train)

pred = clf.predict(x_test_bow)
ac4 = accuracy_score(y_test, pred) * 100
er4 = np.around(100 - ac4, decimals = 2)

w = clf.coef_
s4 = np.count_nonzero(w)

#=======================================================================

clf = LogisticRegression(C=0.01, penalty='l1')
clf.fit(x_train_bow, y_train)

pred = clf.predict(x_test_bow)
ac5 = accuracy_score(y_test, pred) * 100
er5 = np.around(100 - ac5, decimals = 2)

w = clf.coef_
s5 = np.count_nonzero(w)

#=======================================================================
```

```
In [69]: x = PrettyTable()

         c = [100,10,1,0.1,0.01]

         x.field_names = ['C','Train_Error(%)','Sparsity']

         x.add_row([c[0],er1,s1])
         x.add_row([c[1],er2,s2])
         x.add_row([c[2],er3,s3])
         x.add_row([c[3],er4,s4])
         x.add_row([c[4],er5,s5])

         print(x)
```

```
+------+----------------+----------+
|  C   | Train_Error(%) | Sparsity |
+------+----------------+----------+
| 100  |     11.07      |   4966   |
|  10  |     10.27      |   4702   |
|  1   |      8.69      |   3107   |
| 0.1  |      9.29      |   700    |
| 0.01 |     13.71      |    93    |
+------+----------------+----------+
```

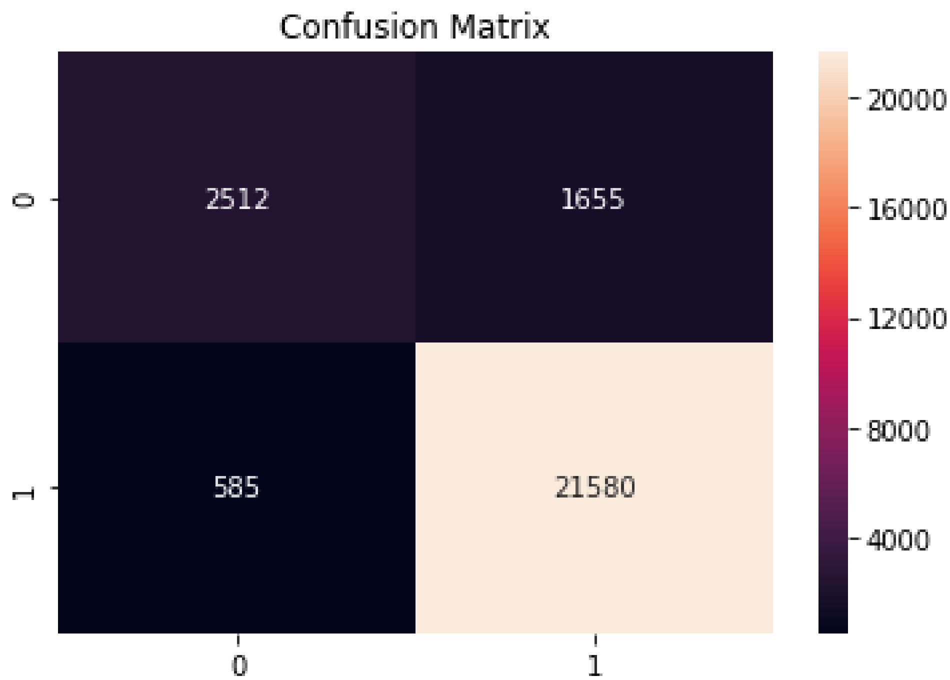
8.1.3 [5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

```
In [81]: clf = LogisticRegression(C = optimal_a1)
         clf.fit(x_train_bow,y_train)

         pred = clf.predict(x_test_bow)

         acc_b2 = accuracy_score(y_test, pred) * 100
         pre_b2 = precision_score(y_test, pred) * 100
         rec_b2 = recall_score(y_test, pred) * 100
         f1_b2  = f1_score(y_test, pred) * 100

         print('\nAccuracy = %f%%' % (acc_b2))
         print('\nprecision= %f%%' % (pre_b2))
         print('\nrecall    = %f%%' % (rec_b2))
         print('\nF1-Score = %f%%' % (f1_b2))
```

```
Accuracy = 91.432478%

precision= 92.652114%

recall    = 97.530306%
```

```
F1-Score = 95.028647%
```

```
In [72]: import seaborn as sns
         cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix')
         plt.show()
```



[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

```
In [73]: # from sklearn.linear_model import LogisticRegression

         clf = LogisticRegression(C= optimal_a1, penalty= 'l2')
         clf.fit(x_train_bow,y_train)
         y_pred = clf.predict(x_test_bow)
         print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
         print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 91.493%
Non Zero weights: 5000
```

```
In [74]: from scipy.sparse import find
         #Weights before adding random noise
```

```
weights1 = find(clf.coef_[0])[2]
print(weights1[:50])
```

```
[-0.11828416  0.14784608  0.07844733  0.2899714  -0.12574554 -0.06580674
  0.08502243 -0.1820233  -0.02990117 -0.01156399  0.08286648  0.29266685
 -0.03601395  0.00134779 -0.00861052  0.30527248  0.12111194  0.05425803
 -0.06000727 -0.13465612 -0.19783663  0.09692817 -0.05466848 -0.01151854
  0.1452461   0.15460764 -0.04499032 -0.0272331  -0.23572383  0.14167968
 -0.08776724  0.14026028 -0.12931243  0.26623561  0.09463407  0.13276691
  0.62350468  0.40163391  0.110437    0.61814222 -0.1640716  -0.04199796
  0.11461488  0.07611953  0.03151373 -0.31722006  0.2159016  -0.07740436
  0.41345863 -0.05410631]
```

In [75]:
```
x_train_bow_new = x_train_bow
#Random noise
epsilon = np.random.uniform(low=-0.0001, high=0.0001, size=(find(x_train_bow_new)[0].s

#Getting the postions(row and column) and value of non-zero datapoints
a,b,c = find(x_train_bow_new)

#Introducing random noise to non-zero datapoints
x_train_bow_new[a,b] = epsilon + x_train_bow_new[a,b]
```

In [76]:
```
#Training on train data having random noise
# from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= optimal_a1, penalty= 'l2')
clf.fit(x_train_bow_new, y_train)
y_pred = clf.predict(x_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Accuracy on test set: 91.091%
Non Zero weights: 5000
```

In [77]:
```
from scipy.sparse import find
#Weights after adding random noise
weights2 = find(clf.coef_[0])[2]
print(weights2[:50])
```

```
[ 0.08638771  0.19871468  0.08928028  0.34156782 -0.10146011 -0.02845346
 -0.0086347  -0.19486528  0.09572047 -0.07636405  0.05525308  0.21327691
  0.04630791 -0.00578565 -0.13085844  0.29748709  0.0240218   0.00803406
  0.00335476 -0.15299587 -0.12499382  0.14523545  0.0106188  -0.19629824
  0.13226826  0.06888225  0.06502459 -0.12984293 -0.1148442   0.09788302
 -0.40338227  0.06874388 -0.08917456  0.37464932  0.08582865  0.09176393
  0.5867344   0.40284308  0.07626824  0.56084165 -0.04817533  0.02209232
```

```
 0.16642448  0.03925806  0.06757161 -0.16457494  0.24596967 -0.23649219
 0.28331048 -0.07432764]
```

```
In [78]: print(weights2.size)
```

```
5000
```

```
In [79]: weights_diff = (abs(weights1 - weights2)/weights1) * 100

In [80]: print(weights_diff[np.where(weights_diff > 30)].size)
```

```
1922
```

8.1.4 [5.1.3] Feature Importance on BOW, SET 1

Printing Most informative features.

```
In [83]: 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
-------------------------------------------------------------------------------------------
        -1.5615         worst                                   1.2520          delicious
        -1.5389         disappointing                           1.2110          perfect
        -1.5187         terrible                                1.1229          great
        -1.3937         awful                                   1.1224          excellent
        -1.3256         disappointed                            1.0423          amazing
        -1.3158         disappointment                          1.0195          nice
        -1.2438         horrible                                0.9989          loves
        -1.2159         threw                                   0.9902          best
        -1.1506         waste                                   0.9741          wonderful
        -1.1315         rip                                     0.9671          highly
```

8.2 [5.2] Logistic Regression on TFIDF, SET 2

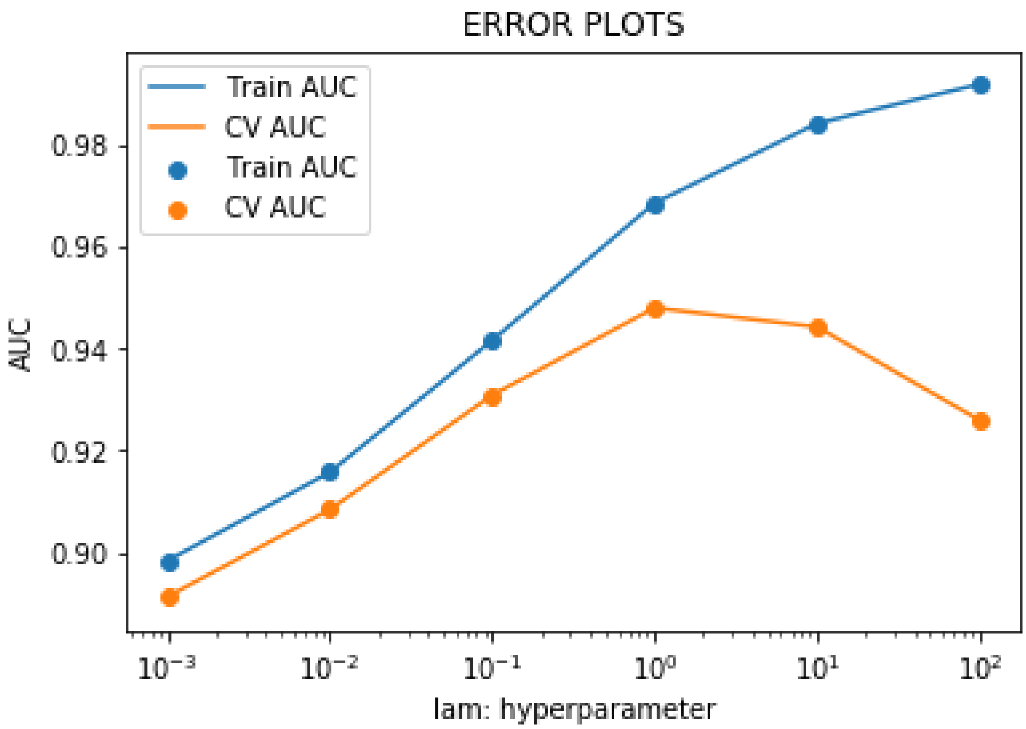
8.2.1 [5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

Hyperparameter tuning using GridSearchCV

In [82]:
```
lam = [0.001,0.01,0.1,1,10,100]
clf = LogisticRegression()
param_grid = {'C':lam}

grid = GridSearchCV(estimator = clf,param_grid=param_grid ,cv = 5,n_jobs = -1, scoring
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_a1 = a.get('C')
```

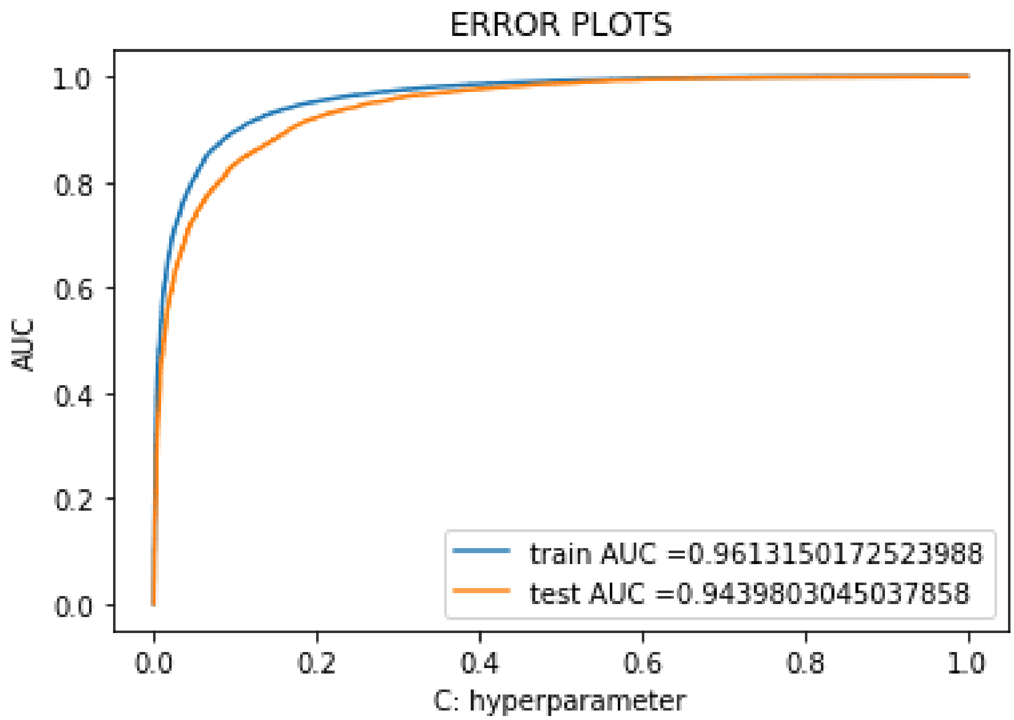
```
best C =  {'C': 1}
Accuracy on train data =  94.79363113403443
```

In [83]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc


tf_auc_train = grid.cv_results_['mean_train_score']
tf_auc_cv    = grid.cv_results_['mean_test_score']


plt.plot(lam, tf_auc_train, label='Train AUC')
plt.scatter(lam, tf_auc_train, label='Train AUC')

plt.plot(lam, tf_auc_cv, label='CV AUC')
plt.scatter(lam, tf_auc_cv, label='CV AUC')

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



Testing with test data

```
In [84]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk

         clf = LogisticRegression(penalty='l1')
         clf.fit(x_train_tf, y_train)

         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_proba

         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()
```



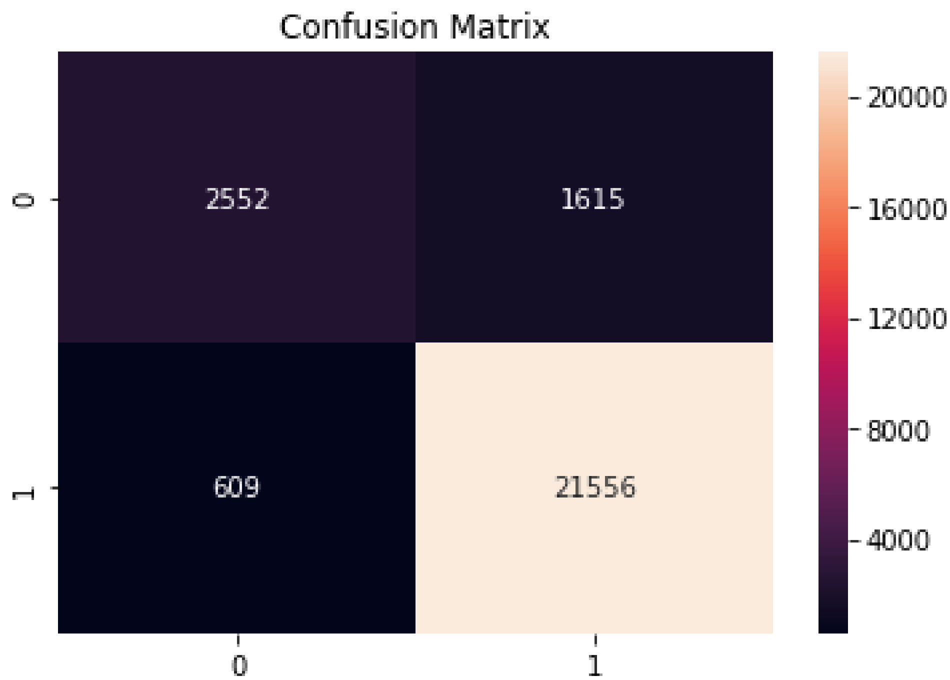
Calculating Confusin Matrix

In [85]:
```
clf = LogisticRegression(C = optimal_a1 , penalty='l1')

clf.fit(x_train_tf,y_train)

pred = clf.predict(x_test_tf)

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

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

```
Accuracy = 91.652742%

precision= 92.905784%

recall    = 97.503166%
```

```
F1-Score = 95.148974%
```

```
In [93]: cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix')
         plt.show()
```

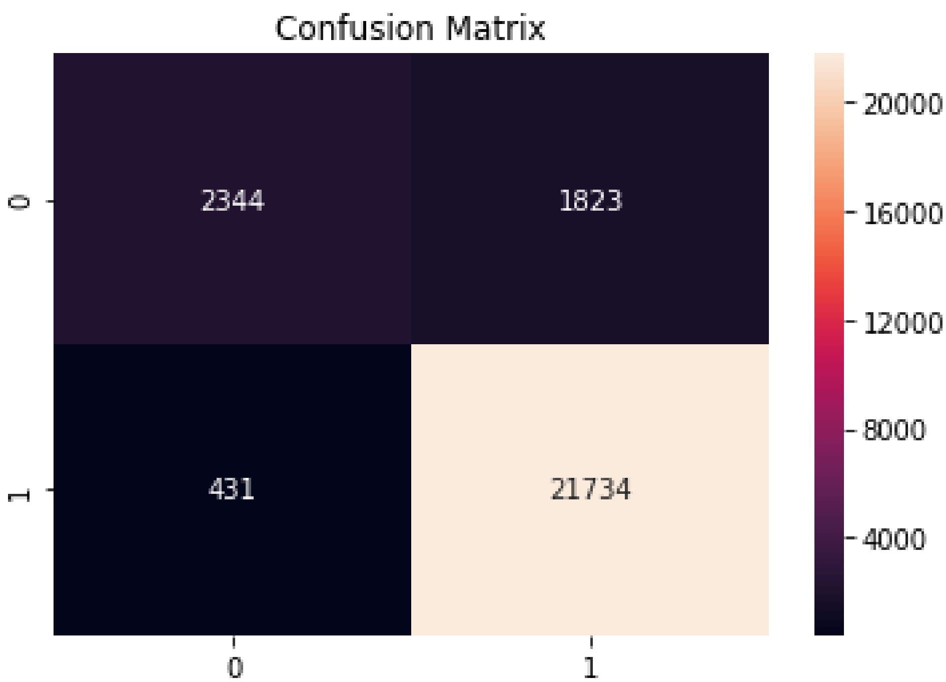


8.2.2 [5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

```
In [86]: clf = LogisticRegression(C = optimal_a1)
         clf.fit(x_train_tf,y_train)
         pred = clf.predict(x_test_tf)

         acc_tf2 = accuracy_score(y_test, pred) * 100
         pre_tf2 = precision_score(y_test, pred) * 100
         rec_tf2 = recall_score(y_test, pred) * 100
         f1_tf2  = f1_score(y_test, pred) * 100

         print('\nAccuracy = %f%%' % (acc_tf2))
         print('\nprecision= %f%%' % (pre_tf2))
         print('\nrecall    = %f%%' % (rec_tf2))
         print('\nF1-Score = %f%%' % (f1_tf2))
```

```
Accuracy = 91.493240%

precision= 92.086087%

recall    = 98.317351%

F1-Score = 95.099755%
```

In [98]:
```
cm = confusion_matrix(y_test,pred)
sns.heatmap(cm, annot=True,fmt='d')
plt.title('Confusion Matrix')
plt.show()
```



8.2.3 [5.2.3] Feature Importance on TFIDF, SET 2

[5.2.3.1] Top 10 important features of positive class from SET 2

In [100]:
```
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-informa
```

```
                Negative                                                        Positive
----------------------------------------------------------------------------------------------
        -6.8340         not                                                     9.8811          great
        -5.9786         worst                                                   7.4512          best
        -5.9097         disappointed                                            6.8551          delicious
        -5.5184         disappointing                                           6.0728          perfect
        -5.2156         terrible                                                5.4930          good
        -5.1953         awful                                                   5.4258          excellent
        -4.9687         disappointment                                          5.4215          wonderful
        -4.9091         horrible                                                5.3577          love
        -4.4228         unfortunately                                           5.2965          loves
        -4.1887         threw                                                   5.1406          nice
```

8.3 [5.3] Logistic Regression on AVG W2V, SET 3

8.3.1 [5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

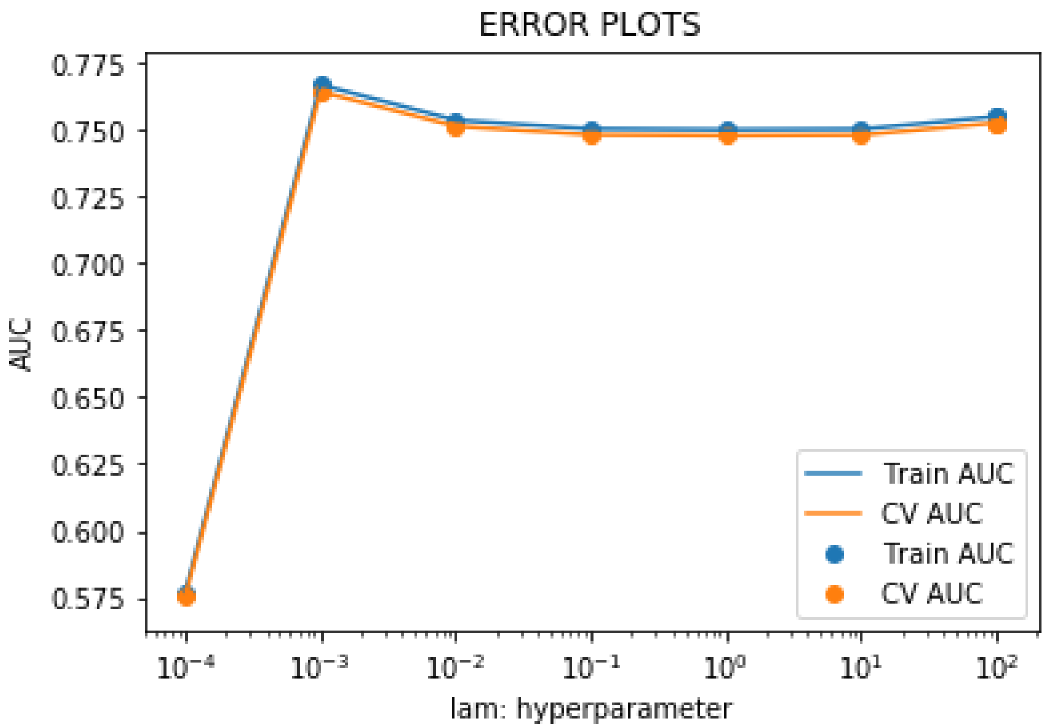
Hyperparameter tuning using GridSearchCV

In [69]:
```
lam = [0.0001, 0.001,0.01,0.1,1,10,100,100]
clf = LogisticRegression()
param_grid = {'C':lam}

grid = GridSearchCV(estimator = clf,param_grid=param_grid ,cv = 5,n_jobs = -1, scoring
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_a1 = a.get('C')
```

```
best C =  {'C': 0.001}
Accuracy on train data =  76.39517856084187
```

In [70]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc

aw2v_auc_train = grid.cv_results_['mean_train_score']
aw2v_auc_cv    = grid.cv_results_['mean_test_score']


plt.plot(lam, aw2v_auc_train, label='Train AUC')
```

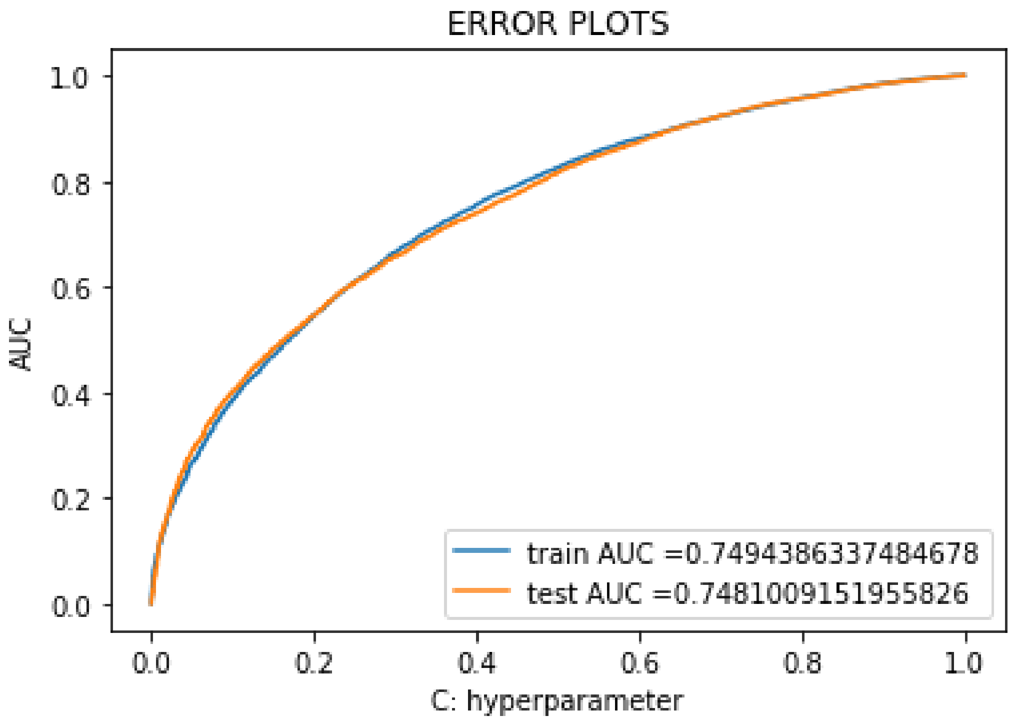
```
plt.scatter(lam, aw2v_auc_train, label='Train AUC')

plt.plot(lam, aw2v_auc_cv, label='CV AUC')
plt.scatter(lam, aw2v_auc_cv, label='CV AUC')

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



Testing with test data

```
In [71]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk

         clf = LogisticRegression()
         clf.fit(sent_vectors_train, y_train)

         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()
```



L1 regularisation

Note: Whenever I add "class_weight = 'balanced'" in logistic regression accuracy, precision, recall and f1 score drops drastically. But if I remove this then each metric scores improve alot.

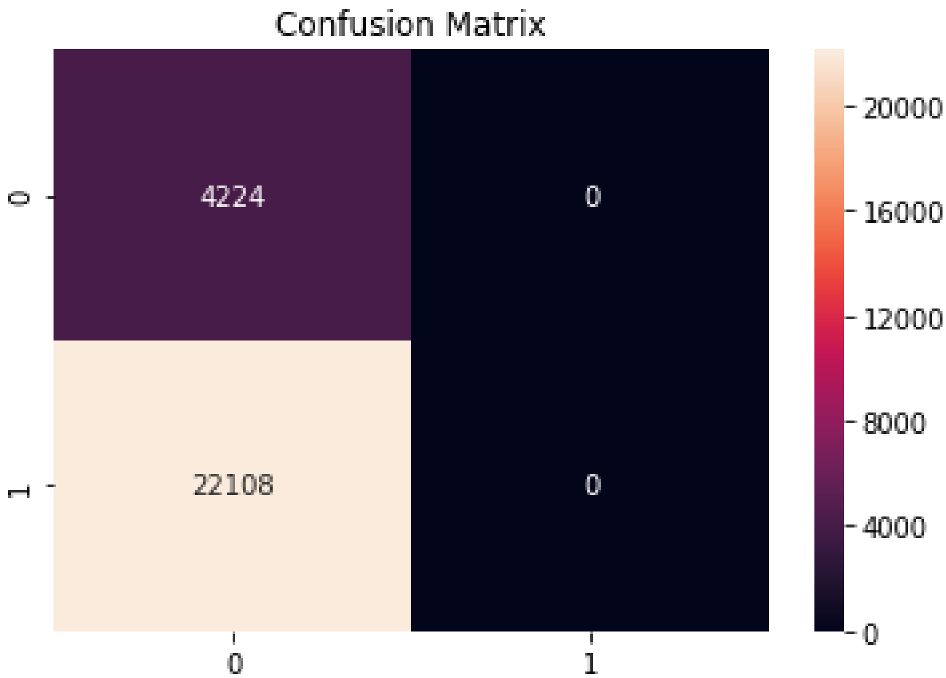
```
In [73]: ################ without class_weight='balanced' ##########

         clf = LogisticRegression(C = optimal_a1,penalty='l1')
         clf.fit(sent_vectors_train,y_train)
         pred = clf.predict(sent_vectors_test)

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

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

```
Accuracy = 83.958681%

precision= 83.958681%

recall    = 100.000000%

F1-Score = 91.279934%
```

```
In [72]: ################## with class_weight='balanced' ###############

         clf = LogisticRegression(C = optimal_a1,penalty='l1', class_weight='balanced')
         clf.fit(sent_vectors_train,y_train)
         pred = clf.predict(sent_vectors_test)

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

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

```
Accuracy = 16.041319%

precision= 0.000000%

recall    = 0.000000%

F1-Score = 0.000000%
```

```
In [52]: cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix')
         plt.show()
```



Testing with test data

8.3.2 [5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

```
In [48]: clf = LogisticRegression(C = optimal_a1, class_weight = 'balanced')
         clf.fit(sent_vectors_train,y_train)
         pred = clf.predict(sent_vectors_test)

         acc_aw2v2 = accuracy_score(y_test, pred) * 100
         pre_aw2v2 = precision_score(y_test, pred) * 100
         rec_aw2v2 = recall_score(y_test, pred) * 100
         f1_aw2v2  = f1_score(y_test, pred) * 100

         print('\nAccuracy = %f%%' % (acc_aw2v2))
         print('\nprecision= %f%%' % (pre_aw2v2))
         print('\nrecall    = %f%%' % (rec_aw2v2))
         print('\nF1-Score = %f%%' % (f1_aw2v2))
```

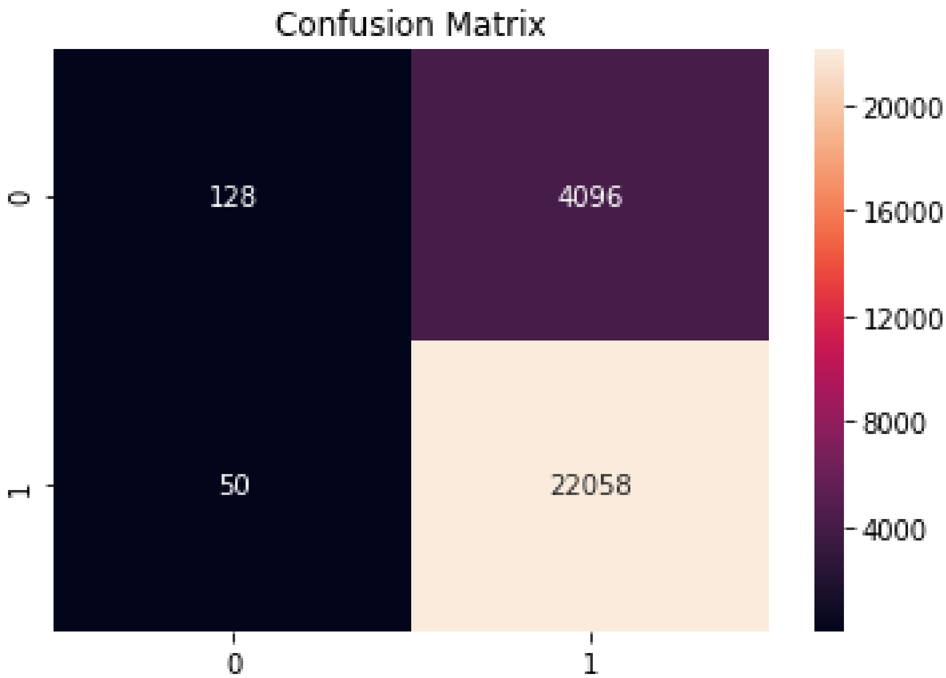
```
Accuracy = 69.178186%

precision= 91.337745%

recall    = 69.920391%
```

```
F1-Score = 79.206805%
```

```
In [66]: clf = LogisticRegression(C = optimal_a1)
         clf.fit(sent_vectors_train,y_train)
         pred = clf.predict(sent_vectors_test)

         acc_aw2v2 = accuracy_score(y_test, pred) * 100
         pre_aw2v2 = precision_score(y_test, pred) * 100
         rec_aw2v2 = recall_score(y_test, pred) * 100
         f1_aw2v2  = f1_score(y_test, pred) * 100

         print('\nAccuracy = %f%%' % (acc_aw2v2))
         print('\nprecision= %f%%' % (pre_aw2v2))
         print('\nrecall    = %f%%' % (rec_aw2v2))
         print('\nF1-Score = %f%%' % (f1_aw2v2))
```

```
Accuracy = 84.254899%

precision= 84.338916%

recall    = 99.773838%

F1-Score = 91.409390%
```

```
In [67]: cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix')
         plt.show()
```



8.4 [5.4] Logistic Regression on TFIDF W2V, SET 4

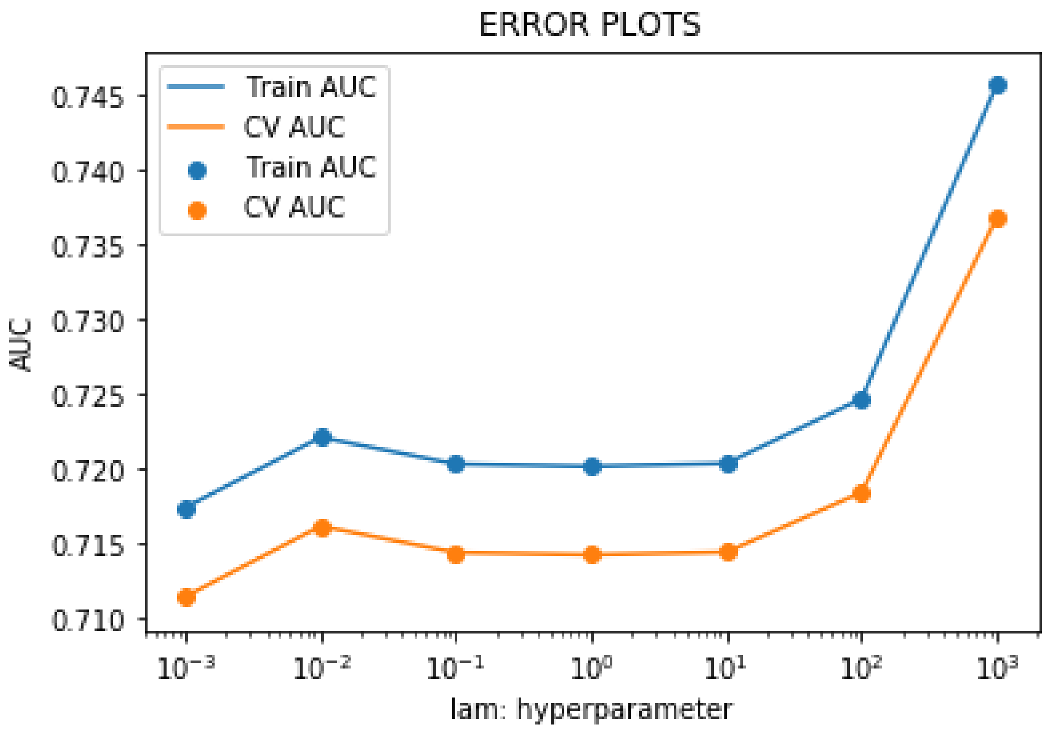
8.4.1 [5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

```
In [55]: lam = [0.001,0.01,0.1,1,10,100, 1000]
         clf = LogisticRegression()
         param_grid = {'C':lam}

         grid = GridSearchCV(estimator = clf,param_grid=param_grid ,cv = 5,n_jobs = -1, scorin
         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_a1 = a.get('C')
```

```
best C =  {'C': 1000}
Accuracy on train data =  73.68707660880133
```

```
In [56]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc

         tfw2v_auc_train = grid.cv_results_['mean_train_score']
         tfw2v_auc_cv    = grid.cv_results_['mean_test_score']
```

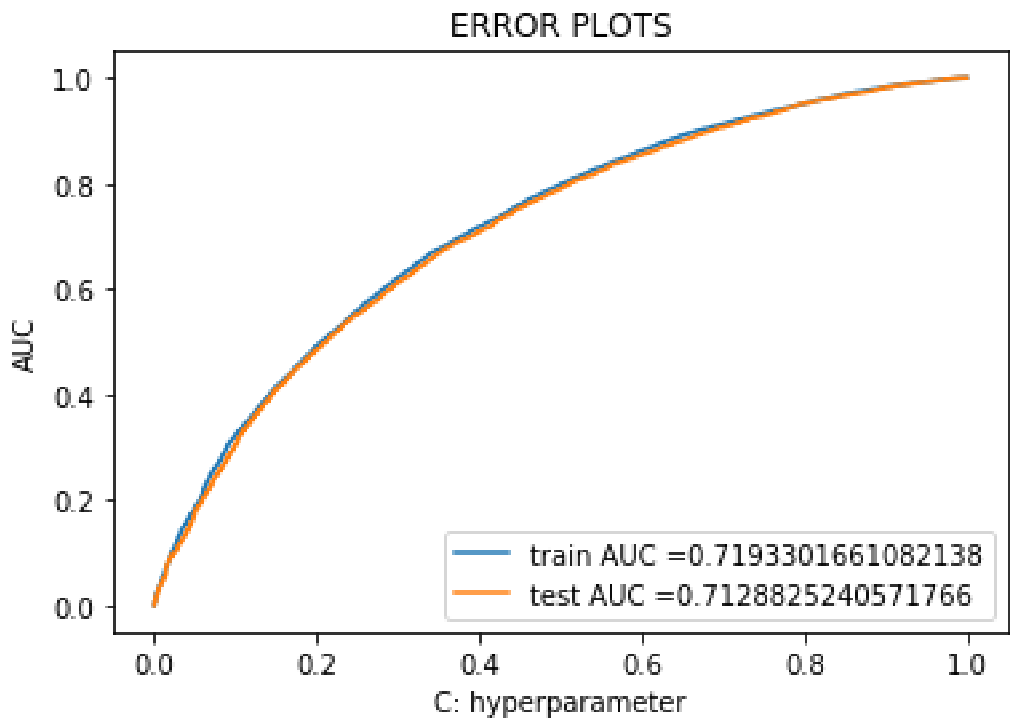
```
plt.plot(lam, tfw2v_auc_train, label='Train AUC')
plt.scatter(lam, tfw2v_auc_train, label='Train AUC')

plt.plot(lam, tfw2v_auc_cv, label='CV AUC')
plt.scatter(lam, tfw2v_auc_cv, label='CV AUC')
plt.legend()
plt.xlabel("lam: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.xscale('log')
plt.show()
```



Testing with test data

```
In [57]: clf = LogisticRegression()
         clf.fit(tfidf_sent_vectors_train, y_train)

         train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(y_train, clf.predict_p
         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 [60]: clf = LogisticRegression(C = optimal_a1,penalty = 'l1') #,class_weight = 'balanced'
         clf.fit(tfidf_sent_vectors_train,y_train)
         pred = clf.predict(tfidf_sent_vectors_test)

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

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

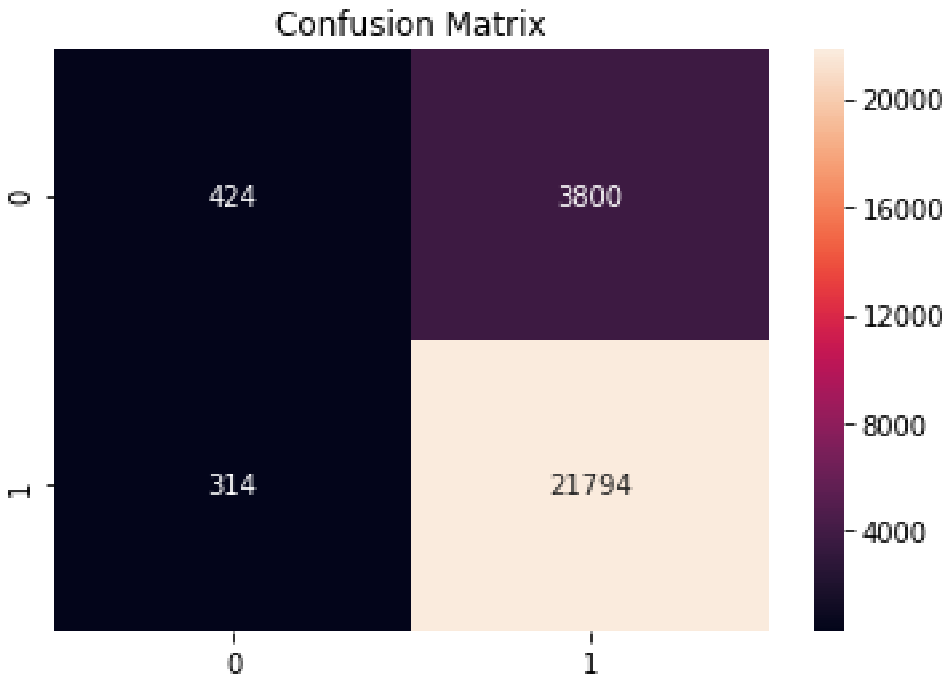
```
Accuracy = 84.376424%

precision= 85.152770%

recall    = 98.579700%
```

```
F1-Score = 91.375624%
```

```
In [59]: clf = LogisticRegression(C = optimal_a1,penalty = 'l1',class_weight = 'balanced') #,c
         clf.fit(tfidf_sent_vectors_train,y_train)
         pred = clf.predict(tfidf_sent_vectors_test)

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

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

```
Accuracy = 68.650311%

precision= 91.698272%

recall    = 68.898136%

F1-Score = 78.679718%
```

```
In [61]: cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix')
         plt.show()
```



8.4.2 [5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [64]:
```
################ without class_weight
################ gives better values for all the metrics

clf = LogisticRegression(C = optimal_a1) #,class_weight = 'balanced'
clf.fit(tfidf_sent_vectors_train,y_train)
pred = clf.predict(tfidf_sent_vectors_test)

acc_tw2v2 = accuracy_score(y_test, pred) * 100
pre_tw2v2 = precision_score(y_test, pred) * 100
rec_tw2v2 = recall_score(y_test, pred) * 100
f1_tw2v2  = f1_score(y_test, pred) * 100

print('\nAccuracy = %f%%' % (acc_tw2v2))
print('\nprecision= %f%%' % (pre_tw2v2))
print('\nrecall    = %f%%' % (rec_tw2v2))
print('\nF1-Score = %f%%' % (f1_tw2v2))
```

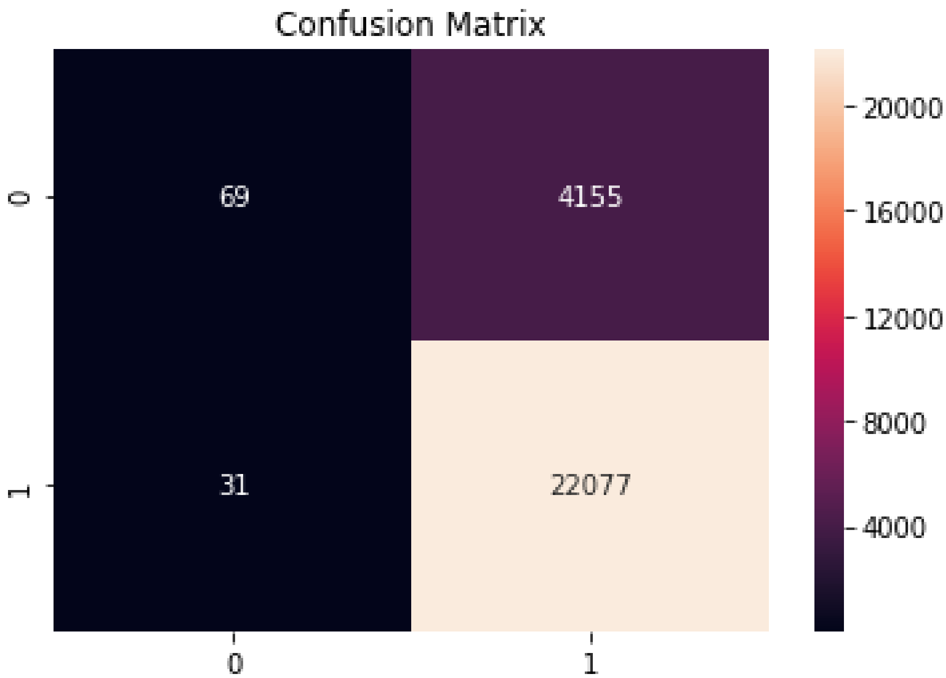
```
Accuracy = 84.102993%

precision= 84.160567%

recall    = 99.859779%
```

```
F1-Score = 91.340505%
```

```
In [63]: ############## including class_weight parameter decreases the performance of model
         ############## as values of performance metrices decreases

         clf = LogisticRegression(C = optimal_a1,class_weight = 'balanced')
         clf.fit(tfidf_sent_vectors_train,y_train)
         pred = clf.predict(tfidf_sent_vectors_test)

         acc_tw2v2 = accuracy_score(y_test, pred) * 100
         pre_tw2v2 = precision_score(y_test, pred) * 100
         rec_tw2v2 = recall_score(y_test, pred) * 100
         f1_tw2v2  = f1_score(y_test, pred) * 100

         print('\nAccuracy = %f%%' % (acc_tw2v2))
         print('\nprecision= %f%%' % (pre_tw2v2))
         print('\nrecall    = %f%%' % (rec_tw2v2))
         print('\nF1-Score = %f%%' % (f1_tw2v2))
```

```
Accuracy = 68.893362%

precision= 91.575551%

recall    = 69.327845%

F1-Score = 78.913631%
```

```
In [65]: cm = confusion_matrix(y_test,pred)
         sns.heatmap(cm, annot=True,fmt='d')
         plt.title('Confusion Matrix')
         plt.show()
```



9 [6] Conclusions

- Considered 100k datapoints for running this model.
- One thing I noticed is that BoW and Tf_idf have better accuracies than average w2v and tf-idf w2v.
- Done Perturbation test and found out that features are collinear.
- Both BoW and TF-IDF models gave best accuracies compared to AVG W2V and TF-IDF W2V
- After taking more features for w2v from 50 to 200, model's performance increases a bit, if we remove parameter(class_weight) then model seems to do better as model accuracy increases where as if we include this parameter, model's accuracy decreases drastically for average w2v.

In [87]:
```
# 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"]
reg= ["L1","L2", "L1", "L2", "L1", "L2", "L1", "L2"]
acc= [acc_b,acc_b2,acc_tf1,acc_tf2,acc_aw2v,acc_aw2v2,acc_tw2v,acc_tw2v2]
pre= [pre_b,pre_b2,pre_tf1,pre_tf2,pre_aw2v,pre_aw2v2,pre_tw2v,pre_tw2v2]
rec= [rec_b,rec_b2,rec_tf1,rec_tf2,rec_aw2v,rec_aw2v2,rec_tw2v,rec_tw2v2]
f1=  [f1_b,f1_b2,f1_tf1,f1_tf2,f1_aw2v,f1_aw2v2,f1_tw2v,f1_tw2v2]


#Initialize Prettytable
ptable = PrettyTable()
```

```
ptable.add_column("Index", number)
ptable.add_column("Model", name)
ptable.add_column("Regularizer", reg)
ptable.add_column("Accuracy%", acc)
ptable.add_column("Precision%", pre)
ptable.add_column("Recall%", rec)
ptable.add_column("F1%", f1)

print(ptable)
```

```
+-------+-----------+-------------+-------------------+-------------------+-------------------
| Index |   Model   | Regularizer |     Accuracy%     |     Precision%    |      Recall%
+-------+-----------+-------------+-------------------+-------------------+-------------------
|   1   |    Bow    |      L1     | 90.68433844751634 | 91.71441911795917 | 97.73385199927628
|   2   |    Bow    |      L2     | 91.43247759380222 | 92.65211412856652 | 97.53030577166638
|   3   |   Tfidf   |      L1     | 91.65274191098284 |  92.9057839841393 |  97.5031662746517
|   4   |   Tfidf   |      L2     | 91.49324016405895 |  92.0860871038807 | 98.31735118509137
|   5   |  Avg W2v  |      L1     | 83.95868145222542 | 83.95868145222542 |       100.0
|   6   |  Avg W2v  |      L2     | 84.25489898222695 | 84.33891565343733 | 99.77383752487788
|   7   | Tfidf W2v |      L1     | 84.37642412274039 | 85.15277018051106 | 98.57969965623303
|   8   | Tfidf W2v |      L2     | 84.10299255658515 | 84.16056724611161 | 99.85977926542428
+-------+-----------+-------------+-------------------+-------------------+-------------------
```