Assignment07 - Amazon Fine Food Reviews Analysis_SVM

June 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 [105]: %matplotlib inline
          import warnings
          warnings.filterwarnings("ignore")


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

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

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

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

          from tqdm import tqdm
          import os
```

```
In [106]: # using SQLite Table to read data.
          db_path = '/home/monodeepdas112/Datasets/amazon-fine-food-reviews/database.sqlite'
          # db_path = '/home/monodeepdas112/Datasets/AmazonFineFoodReviews/database.sqlite'
          con = sqlite3.connect(db_path)

          # 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 poi
          # 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 10
        # for tsne assignment you can take 5k data points

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

        # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negat
        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[106]:    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 [107]: display = pd.read_sql_query("""
          SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
          FROM Reviews
          GROUP BY UserId
          HAVING COUNT(*)>1
          """, con)
```

```
In [108]: print(display.shape)
          display.head()
```

```
(80668, 7)
```

```
Out[108]:                UserId   ProductId              ProfileName        Time  Score  \
          0  #oc-R115TNMSPFT9I7  B005ZBZLT4                  Breyton  1331510400      2
          1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
          2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4         Kim Cieszykowski  1348531200      1
          3  #oc-R11O5J5ZVQE25C  B005HG9ESG            Penguin Chick  1346889600      5
          4  #oc-R12KPBODL2B5ZD  B007OSBEV0    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 [109]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[109]:               UserId   ProductId                      ProfileName        Time  \
          80638  AZY10LLTJ71NX  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

                 Score                                               Text  COUNT(*)
          80638      5  I bought this 6 pack because for the price tha...         5
```

```
In [110]: display['COUNT(*)'].sum()
```

```
Out[110]: 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 [111]: display= pd.read_sql_query("""
          SELECT *
          FROM Reviews
          WHERE Score != 3 AND UserId="AR5J8UI46CURR"
          ORDER BY ProductID
          """, con)
          display.head()
```

```
Out[111]:        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 [112]: #Sorting data according to ProductId in ascending order
          sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fa
```

```
In [113]: #Deduplication of entries
          final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep
          final.shape
```

```
Out[113]: (87775, 10)
```

```
In [114]: #Checking to see how much % of data still remains
          (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[114]: 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 [115]: display= pd.read_sql_query("""
          SELECT *
          FROM Reviews
          WHERE Score != 3 AND Id=44737 OR Id=64422
          ORDER BY ProductID
          """, con)

          display.head()
```

```
Out[115]:       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 [116]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [117]: #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[117]: 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 [118]: # 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 afrai
==================================================
```

```
In [119]: # 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 [120]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-al
          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 [121]: # 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 [122]: 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 [123]: #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 [124]: #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 [125]: # 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', 'ourselv
                      "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', 'to
                      's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
                      've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
                      "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                      "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                      'won', "won't", 'wouldn', "wouldn't"])

In [126]: # Combining all the above stundents
          from tqdm import tqdm
          preprocessed_reviews = []
          # tqdm is for printing the status bar
          for sentance in tqdm(final['Text'].values):
              sentance = re.sub(r"http\S+", "", sentance)
              sentance = BeautifulSoup(sentance, 'lxml').get_text()
              sentance = decontracted(sentance)
              sentance = re.sub("\S*\d\S*", "", sentance).strip()
              sentance = re.sub('[^A-Za-z]+', ' ', sentance)
              # https://gist.github.com/sebleier/554280
              sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
              preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87773/87773 [00:33<00:00, 2657.32it/s]
```

```
In [127]: preprocessed_reviews[1500]
```

```
Out[127]: 'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

```
In [128]: ## Similartly you can do preprocessing for review summary also.
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [129]: # #BoW
          # count_vect = CountVectorizer() #in scikit-learn
          # count_vect.fit(preprocessed_reviews)
          # print("some feature names ", count_vect.get_feature_names()[:10])
          # print('='*50)

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

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

```
In [130]: # #bi-gram, tri-gram and n-gram

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

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

5.3 [4.3] TF-IDF

```
In [131]: # tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
          # tf_idf_vect.fit(preprocessed_reviews)
          # print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_n
          # print('='*50)

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

5.4 [4.4] Word2Vec

```
In [132]: # # Train your own Word2Vec model using your own text corpus
          # i=0
          # list_of_sentences=[]
          # for sentence in preprocessed_reviews:
          #     list_of_sentences.append(sentence.split())
```

```
In [133]: # # Using Google News Word2Vectors

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


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

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

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

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

```
#         w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative30
#         print(w2v_model.wv.most_similar('great'))
#         print(w2v_model.wv.most_similar('worst'))
#     else:
#         print("you don't have gogole's word2vec file, keep want_to_train_w2v = Tru
```

```
In [134]: # 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])
```

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

[4.4.1.1] Avg W2v

```
In [135]: # # average Word2Vec
          # # compute average word2vec for each review.
          # sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
          # for sent in tqdm(list_of_sentance): # for each review/sentence
          #     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might nee
          #     cnt_words =0; # num of words with a valid vector in the sentence/review
          #     for word in sent: # for each word in a review/sentence
          #         if word in w2v_words:
          #             vec = w2v_model.wv[word]
          #             sent_vec += vec
          #             cnt_words += 1
          #     if cnt_words != 0:
          #         sent_vec /= cnt_words
          #     sent_vectors.append(sent_vec)
          # print(len(sent_vectors))
          # print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

```
In [136]: # # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
          # model = TfidfVectorizer()
          # tf_idf_matrix = model.fit_transform(preprocessed_reviews)
          # # we are converting a dictionary with word as a key, and the idf as a value
          # dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [137]: # # 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 = tf

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

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 c
<li>When you are working with SGDClassifier with hinge loss and trying to find the AUC
    score, you would have to use <a href='https://scikit-learn.org/stable/modules/generated/skl
<li>Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce
```

the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

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

```
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>When you are working on the linear kernel with BOW or TFIDF please print the top 10 best
```

features for each of the positive and negative classes.

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

Note: Data Leakage

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

7 Applying SVM

```
In [138]: from sklearn.linear_model import SGDClassifier
          from sklearn.svm import SVC
          from sklearn.calibration import CalibratedClassifierCV
          from sklearn.metrics import confusion_matrix
          from sklearn.metrics import roc_curve, auc
          from sklearn.metrics import roc_auc_score
          from sklearn.model_selection import train_test_split
          from sklearn.model_selection import StratifiedKFold
          import pprint
          from sklearn.pipeline import Pipeline
          import os.path
          import pickle
          import math

          import warnings
          warnings.filterwarnings('ignore')
```

7.0.1 [5.0.0] Splitting up the Dataset into D_train and D_test

```
In [139]: num_data_points = 100000

In [140]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(preprocessed_reviews[:num_da

In [141]: prettytable_data = []
```

7.0.2 [5.0.1] Defining some functions to increase code reusability and readability

```
In [142]: '''Creating Custom Vectorizers for TFIDF - W2Vec and Avg - W2Vec'''
          class Tfidf_W2Vec_Vectorizer(object):
              def __init__(self, w2vec_model):
                  if(w2v_model is None):
                      raise Exception('Word 2 Vector model passed to Tfidf_W2Vec Vectorizer is
                  self.tfidf = TfidfVectorizer(max_features=300)
                  self.dictionary = None
                  self.tfidf_feat = None

                  self.word2vec = w2vec_model

              def fit(self, X):
                  '''X : list'''
                  #Initializing the TFIDF Vectorizer
                  self.tfidf.fit_transform(X)
                  # we are converting a dictionary with word as a key, and the idf as a value
                  self.dictionary = dict(zip(self.tfidf.get_feature_names(), list(self.tfidf.i
                  self.tfidf_feat = self.tfidf.get_feature_names()

                  return self
              def transform(self, X):
                  '''X : list'''
                  return np.array([
                          np.mean([self.word2vec[w] * self.dictionary[word]*(X.cout(word)/len(
                                   for w in words if w in self.word2vec and w in self.tfidf_fea
                                  [np.zeros(300)], axis=0)
                          for words in X
                      ])

          class Avg_W2Vec_Vectorizer(object):
              def __init__(self, w2vec_model):
                  if(w2v_model is None):
                      raise Exception('Word 2 Vector model passed to Avg_W2Vec Vectorizer is N
                  self.word2vec = w2vec_model

              def fit(self, X):
                  return self

              def transform(self, X):
                  '''X : list'''
                  return np.array([
                      np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                              or [np.zeros(300)], axis=0)
                      for words in X
                  ])
```

```
In [143]: def get_vectorizer(vectorizer, train, W2V_model=None):
              if(vectorizer=='BOW'):
                  vectorizer = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000
              if(vectorizer=='TFIDF'):
                  vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=5000
              if(vectorizer=='TFIDF-W2Vec'):
                  vectorizer = Tfidf_W2Vec_Vectorizer(W2V_model)
              if(vectorizer=='Avg-W2Vec'):
                  vectorizer = Avg_W2Vec_Vectorizer(W2V_model)

              vectorizer.fit(train)
              return vectorizer
```

```
In [144]: '''Perform Simple Cross Validation'''
          def perform_hyperparameter_tuning(X, Y, vectorizer, penalty, results_path, retrain=Fa
              #If the pandas dataframe with the hyperparameter info exists then return it

              if(retrain==False):
                  # If Cross Validation results exists then return them
                  if(os.path.exists(results_path)):
                      return pd.read_csv(results_path)
```

```
    else:
        # If no data exists but retrain=False then mention accordingly
        print('Retrain is set to be False but no Cross Validation Results DataFra
else:
    # else perform hyperparameter tuning
    print('Performing Hyperparameter Tuning...\n')
    # regularization parameter
    alpha = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 1
    hyperparameters = {
        'svm__penalty' : penalty,
        'svm__alpha' : alpha
    }

    penalties = []
    alpha_values = []

    train_scores = []
    test_scores = []

    train_mean_score = []
    test_mean_score = []

    # Initializing KFold
    skf = StratifiedKFold(n_splits=3)
    X = np.array(X)
    Y = np.array(Y)

    for reg_param in hyperparameters['svm__alpha']:
        for penalty in hyperparameters['svm__penalty']:

            #Performing Cross Validation
            for train_index, test_index in skf.split(X, Y):
                Dx_train, Dx_cv = X[train_index], X[test_index]
                Dy_train, Dy_cv = Y[train_index], Y[test_index]

                #Initializing the Vectorizer
                vectorizer = get_vectorizer(vectorizer, Dx_train.tolist(), W2V_m

                #Transforming the data to features
                x_train = vectorizer.transform(Dx_train.tolist())
                x_cv = vectorizer.transform(Dx_cv.tolist())

                #Initializing the LR model
                svm = SGDClassifier(penalty=penalty,
                                    alpha=reg_param,
                                    max_iter=1000, verbose=0)

                # Fit the model
                svm.fit(x_train, Dy_train)

                # Calibrating the svm model to output probablity class labels
                calib_svm = CalibratedClassifierCV(base_estimator=svm, method="i
                calib_svm.fit(x_train, Dy_train)

                #Prediction
                train_results = calib_svm.predict_proba(x_train)
                cv_results = calib_svm.predict_proba(x_cv)

                try:
                    train_score = roc_auc_score(Dy_train, train_results[:, 1])
                    test_score = roc_auc_score(Dy_cv, cv_results[:, 1])

                    #storing the results to form a dataframe
                    train_scores.append(train_score)
                    test_scores.append(test_score)

                except Exception as e:
                    print('Error Case : ', e)
                    print(('Actual, Predicted'))
                    [print((Dy_cv[i], cv_results[i, 1])) for i in range(len(Dy_c

                print('CV iteration : alpha={0}, penalty={1}, train_score={2}, te
                  .format(reg_param, penalty, train_score, test_score))

            train_mean_score.append(sum(train_scores)/len(train_scores))
            test_mean_score.append(sum(test_scores)/len(test_scores))

            penalties.append(penalty)
            alpha_values.append(reg_param)

            print('C={0}, penalty={1}, train_score={2}, test_score={3}'
                  .format(reg_param, penalty, sum(train_scores)/len(train_scores

            train_scores = []
            test_scores = []

    # Creating a DataFrame from the saved data for visualization
    results_df = pd.DataFrame({'alpha' : alpha_values, 'penalty' : penalties,
                               'train_score' : train_mean_score,
                               'test_score': test_mean_score})

    #writing the results to csv after performing hyperparameter tuning

    results_df.to_csv(results_path)

    return results_df
```

```
In [145]: def analyse_results(df):
              # plotting error curves
              fig = plt.figure(figsize=(15, 5))
              ax = fig.gca()

              mini = df.loc[df['penalty'] == 'l1']
              plt.subplot(1, 2, 1)
              plt.plot([math.log10(i) for i in mini.alpha.tolist()], mini.train_score.tolist()
              plt.plot([math.log10(i) for i in mini.alpha.tolist()], mini.test_score.tolist(),
              plt.grid(True)
              plt.xlabel('log10 of Hyperparameter alpha')
              plt.ylabel('Area Under ROC Curve')
              plt.title('AUC ROC Curve : Penalty = '.format('l1'))
              plt.legend(loc='best')

              mini = df.loc[df['penalty'] == 'l2']
              plt.subplot(1, 2, 2)
              plt.plot([math.log10(i) for i in mini.alpha.tolist()], mini.train_score.tolist()
              plt.plot([math.log10(i) for i in mini.alpha.tolist()], mini.test_score.tolist(),
              plt.grid(True)
              plt.xlabel('log10 of Hyperparameter alpha')
              plt.ylabel('Area Under ROC Curve')
              plt.title('AUC ROC Curve : Penalty = '.format('l2'))
              plt.legend(loc='best')

              plt.show()

              # return the best parameters
              mmax = 0
              ind_max = 0
              for index, row in df.iterrows():
                  if(row['test_score']>mmax):
                      mmax=row['test_score']
                      ind_max = index


              best_params = {
                  'svm__alpha': df.loc[ind_max, 'alpha'],
                  'svm__penalty':df.loc[ind_max, 'penalty']
              }

              return best_params

In [146]: def retrain_with_best_params(data, labels, best_params, vec_name, model_path, word2ve
              if(os.path.exists(model_path)):
                  print('Loading Model....')
                  with open(model_path, 'rb') as input_file:
                      calib_svm = pickle.load(input_file)
```

```
    else:
        svm = SGDClassifier(penalty=best_params['svm__penalty'], alpha=best_params['s

        print('Initializing Vectorizer')
        vectorizer = get_vectorizer(vectorizer=vec_name, train=data, W2V_model=word2v
        print('Training Model....')
        svm.fit(vectorizer.transform(data), np.array(labels))

        calib_svm = CalibratedClassifierCV(base_estimator=svm, method="isotonic", cv=
        calib_svm.fit(vectorizer.transform(data), np.array(labels))

        print('Saving Trained Model....')
        with open(model_path,'wb') as file:
            pickle.dump(calib_svm, file)
    return calib_svm
```

```
In [147]: def plot_confusion_matrix(model, data, labels, dataset_label):
              pred = model.predict(data)
              conf_mat = confusion_matrix(labels, pred)

              strings = strings = np.asarray([['TN = ', 'FP = '],
                                              ['FN = ', 'TP = ']])

              labels = (np.asarray(["{0}{1}".format(string, value)
                                    for string, value in zip(strings.flatten(),
                                                             conf_mat.flatten())])
                       ).reshape(2, 2)

              fig, ax = plt.subplots()
              ax.set(xlabel='Predicted', ylabel='Actual', title='Confusion Matrix : {0}'.forma
              sns.heatmap(conf_mat, annot=labels, fmt="", cmap='YlGnBu', ax=ax)
              ax.set_xlabel('Predicted')
              ax.set_ylabel('Actual')
              ax.set_xticklabels(['False', 'True'])
              ax.set_yticklabels(['False', 'True'])
              plt.show()
```

```
In [148]: def plot_AUC_ROC(model, vectorizer, Dx_train, Dx_test, Dy_train, Dy_test):

              #predicting probability of Dx_test, Dx_train
              test_score = model.predict_proba(vectorizer.transform(Dx_test))
              train_score = model.predict_proba(vectorizer.transform(Dx_train))

              #Finding out the ROC_AUC_SCORE
              train_roc_auc_score = roc_auc_score(np.array(Dy_train), train_score[:, 1])
              print('Area Under the Curve for Train : ', train_roc_auc_score)
              test_roc_auc_score = roc_auc_score(np.array(Dy_test), test_score[:, 1])
              print('Area Under the Curve for Test : ', test_roc_auc_score)
```

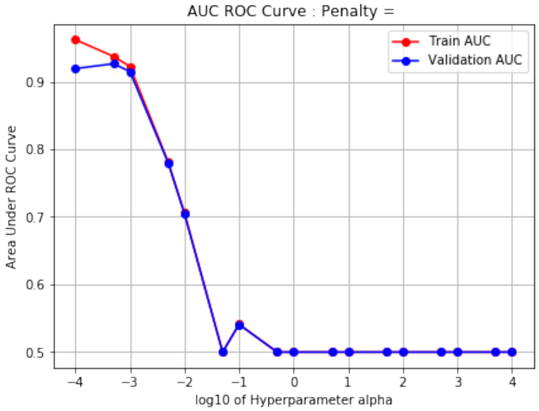
```
    #Plotting with matplotlib.pyplot
    #ROC Curve for D-train
    train_fpr, train_tpr, thresholds = roc_curve(np.array(Dy_train), train_score[:, 
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)

    # #ROC Curve for D-test
    test_fpr, test_tpr, thresholds = roc_curve(np.array(Dy_test), test_score[:, 1])
    plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))

    plt.legend()
    plt.xlabel("FPR : False Positive Ratio")
    plt.ylabel("TPF : True Positive Ratio")
    plt.title("Area Under ROC Curve")
    plt.show()

    plot_confusion_matrix(model, vectorizer.transform(Dx_train), np.array(Dy_train), 
    plot_confusion_matrix(model, vectorizer.transform(Dx_test), np.array(Dy_test), '
    return train_roc_auc_score, test_roc_auc_score
```

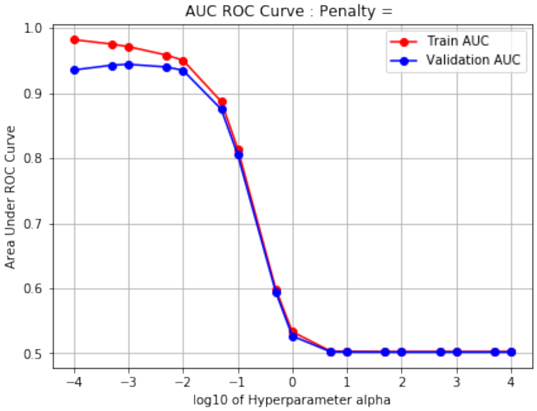
7.1 [5.1] Linear SVM

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

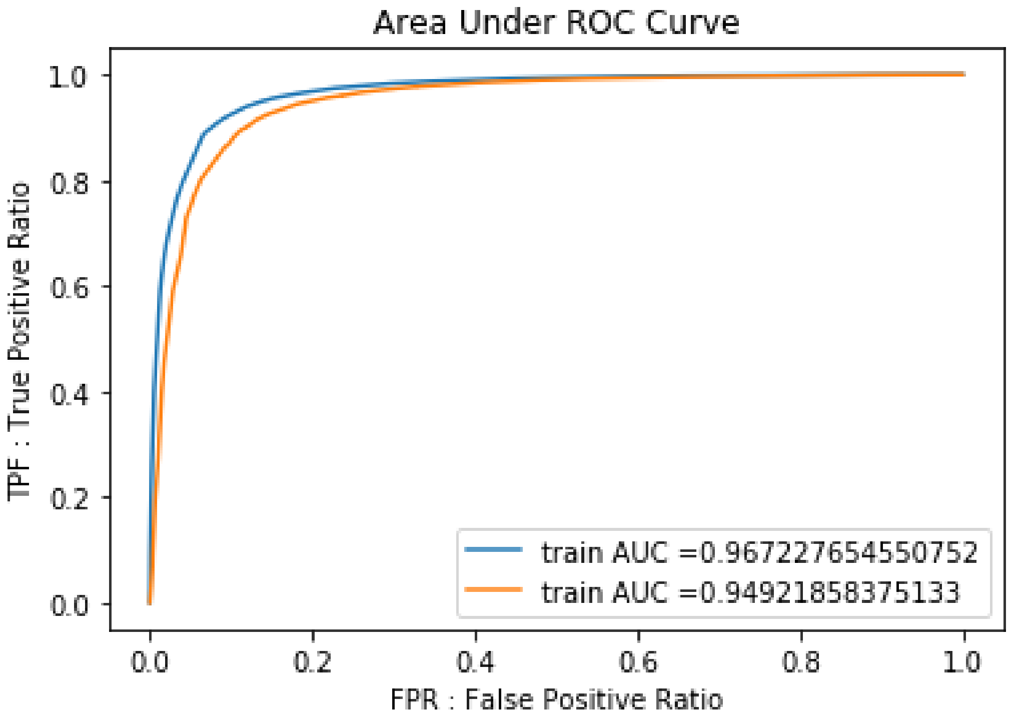
In [45]:

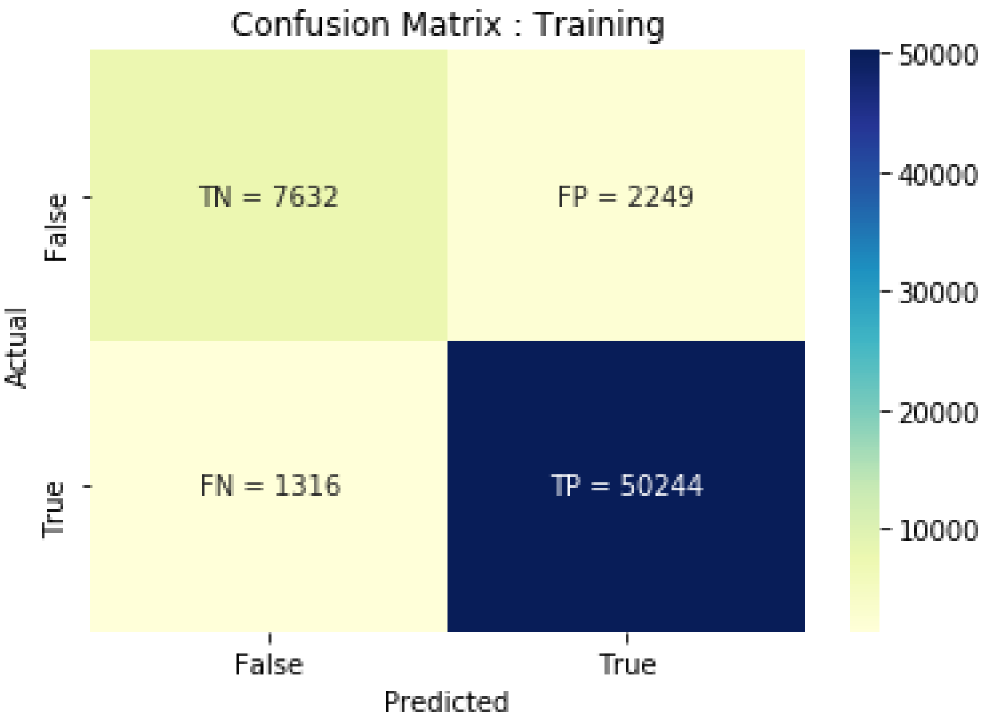
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment7/BOW_svm_results.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='BOW', 
                                           penalty=['l1', 'l2'], results_path=csv_pat
# Analysing best parameters
best_parameters = analyse_results(cv_results)
pprint.pprint(best_parameters)
# retraining the model with best parameters
model_path = 'saved_models/Assignment7/{0}_svm.pkl'.format('BOW')
calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'BOW', 

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='BOW')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_te

# appending the data results
prettytable_data.append(['BOW', 'SVM', best_parameters['svm__penalty'], best_paramete
```



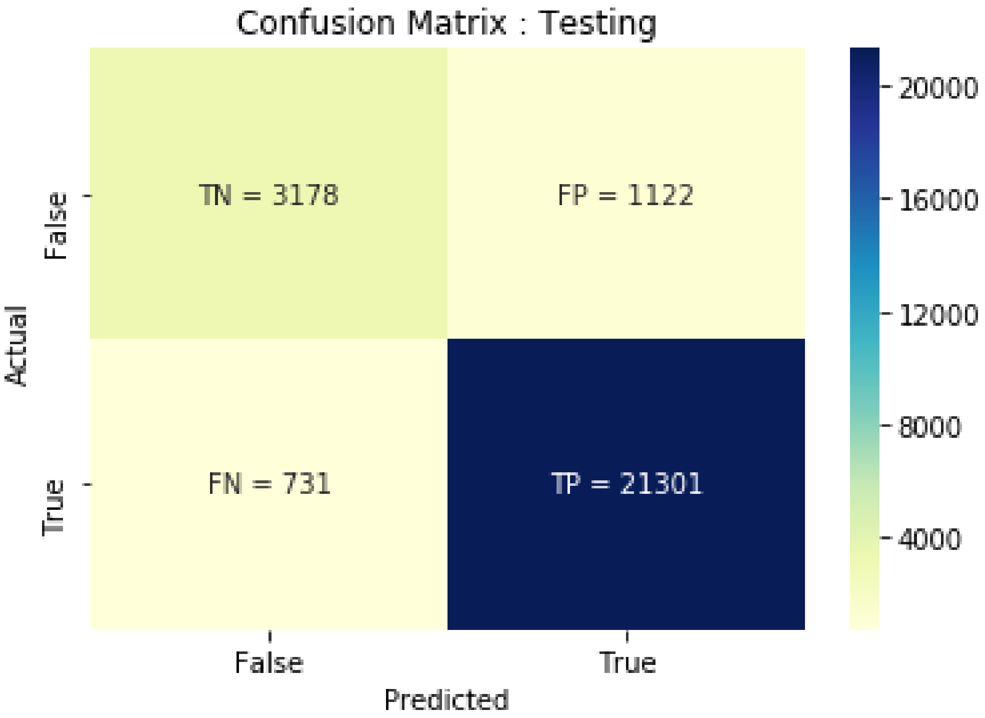


```
{'svm__alpha': 0.001, 'svm__penalty': 'l2'}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.967227654550752
Area Under the Curve for Test :  0.94921858375133
```





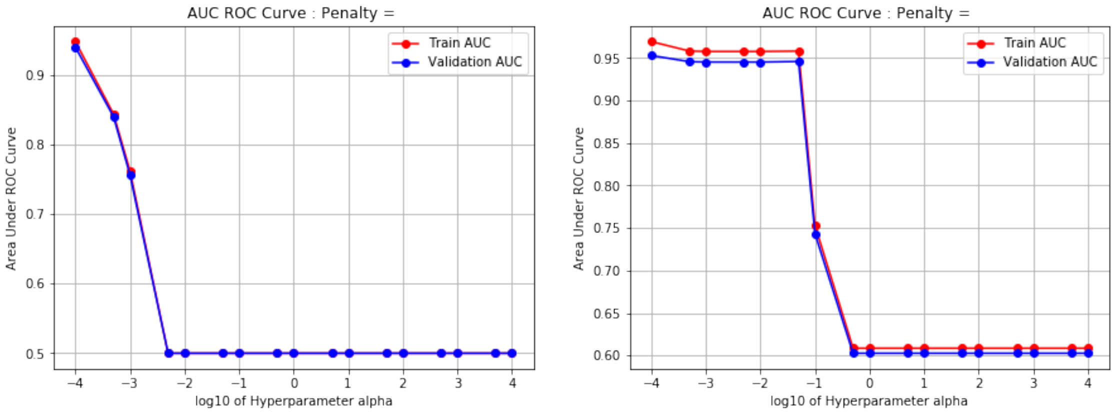
Confusion Matrix : Training



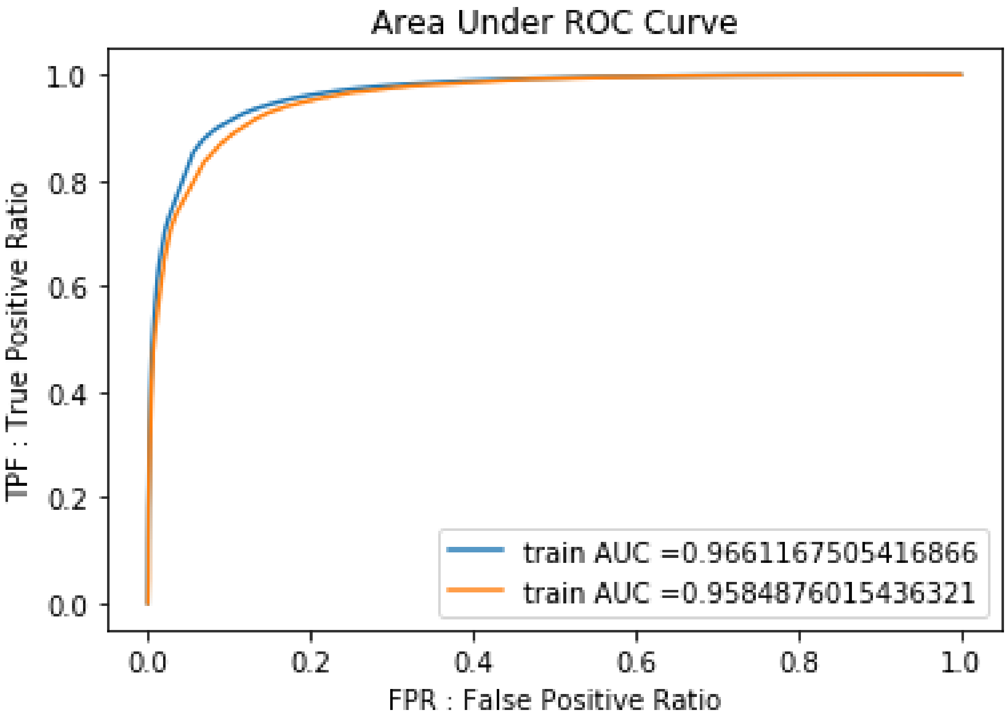
Confusion Matrix : Testing

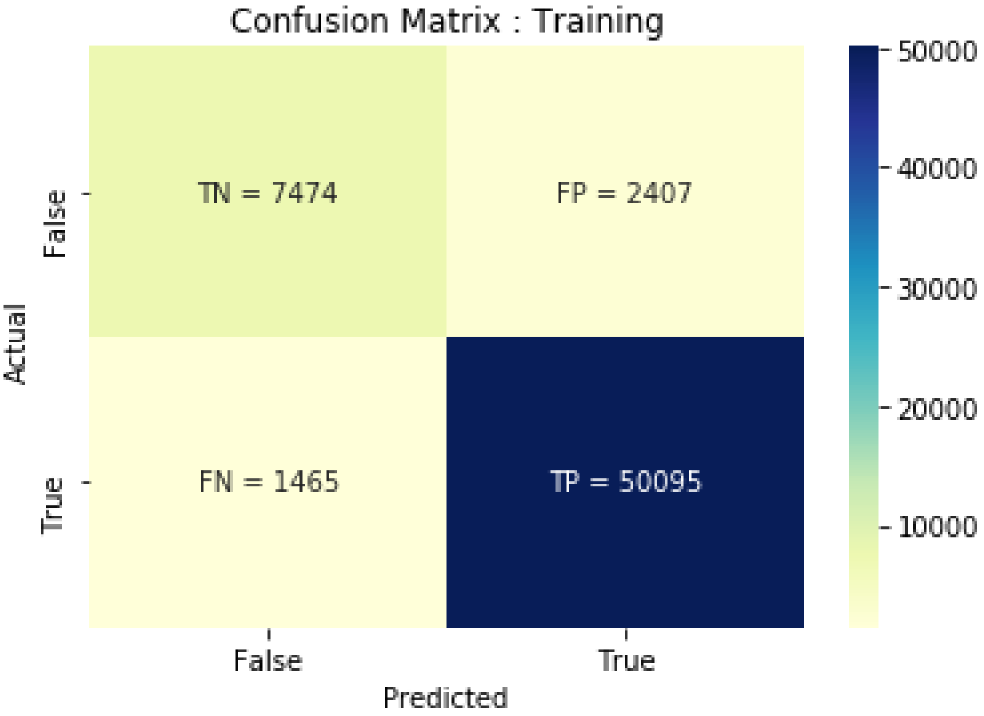
7.1.2 [5.1.2] Applying Linear SVM on TFIDF, SET 2

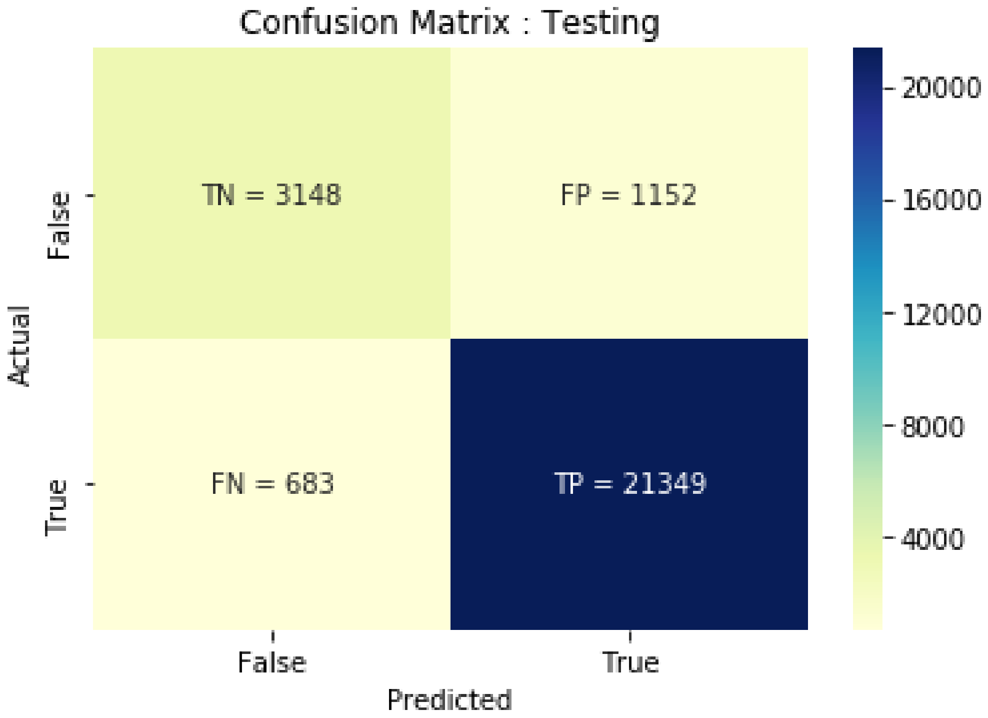
```
In [46]: # Please write all the code with proper documentation
         csv_path = 'saved_models/Assignment7/TFIDF_svm_results.csv'
         cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF'
                                                    penalty=['l1', 'l2'], results_path=csv_pat
         # Analysing best parameters
         best_parameters = analyse_results(cv_results)
         pprint.pprint(best_parameters)
         # retraining the model with best parameters
         model_path = 'saved_models/Assignment7/{0}_svm.pkl'.format('TFIDF')
         calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF

         print('Retraining Vectorizer with Dx_train')
         vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='TFIDF')

         # plotting AUC ROC
         train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_te

         # appending the data results
         prettytable_data.append(['TFIDF', 'SVM', best_parameters['svm__penalty'], best_parame
```



```
{'svm__alpha': 0.0001, 'svm__penalty': 'l2'}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9661167505416866
Area Under the Curve for Test :  0.9584876015436321
```


Area Under ROC Curve
TPF : True Positive Ratio
FPR : False Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
train AUC =0.9661167505416866
train AUC =0.9584876015436321


Confusion Matrix : Training
TN = 7474
FP = 2407
FN = 1465
TP = 50095
Actual
False
True
Predicted
False
True
50000
40000
30000
20000
10000



7.2 Preparing/Training Google Word2Vec

```
In [149]: is_your_ram_gt_16g=True
          want_to_use_google_w2v = True
          want_to_train_w2v = False

          path_to_word2vec = '/home/monodeepdas112/Datasets/GoogleNews-vectors-negative300.bin

          if want_to_train_w2v:

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

              # min_count = 5 considers only words that occured atleast 5 times
              w2v_model=Word2Vec(list_of_sentences,min_count=5,size=300, workers=4)
              print(w2v_model.wv.most_similar('great'))
              print('='*50)
              print(w2v_model.wv.most_similar('worst'))

          elif want_to_use_google_w2v and is_your_ram_gt_16g:
```

```
    if os.path.isfile(path_to_word2vec):
        print('Preparing to load pre-trained Word2Vec model !')
        w2v_model=KeyedVectors.load_word2vec_format(path_to_word2vec, binary=True)
        print('Successfully loaded model into memory !!')
        print('Words similar to "similar" : ', w2v_model.wv.most_similar('great'))
        print('Words similar to "worst" : ',w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have google's word2vec file, keep want_to_train_w2v = True,
```

```
Preparing to load pre-trained Word2Vec model !
Successfully loaded model into memory !!
Words similar to "similar" :  [('terrific', 0.798933207988739), ('fantastic', 0.79352122545242
Words similar to "worst" :  [('Worst', 0.6146091222763062), ('weakest', 0.6143776774406433), (
```

7.2.1 [5.1.3] Applying Linear SVM on AVG W2V, SET 3

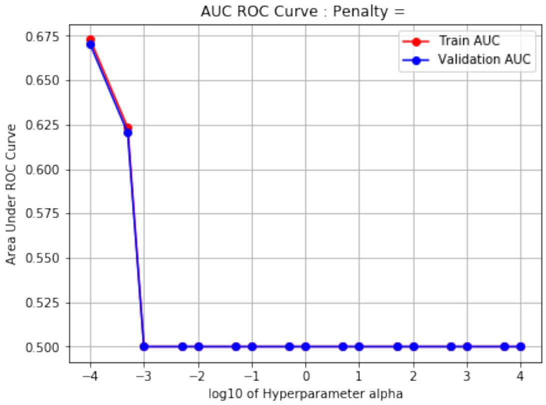
```
In [151]: # Please write all the code with proper documentation
          csv_path = 'saved-models/Assignment7/Avg-W2Vec_svm_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='Avg-W
                                                     penalty=['l1', 'l2'], results_path=csv_pa
                                                     retrain=True, W2V_model=w2v_model)
          # Analysing best parameters
          best_parameters = analyse_results(cv_results)
          pprint.pprint(best_parameters)
          # retraining the model with best parameters
          model_path = 'saved-models/Assignment7/{0}_svm.pkl'.format('Avg-W2Vec')
          calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-

          print('Retraining Vectorizer with Dx_train')
          vectorizer_obj = get_vectorizer(W2V_model = w2v_model, train=Dx_train, vectorizer='A

          # plotting AUC ROC
          train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_

          # appending the data results
          prettytable_data.append(['Avg-W2Vec', 'SVM', best_parameters['svm__penalty'], best_pa
```

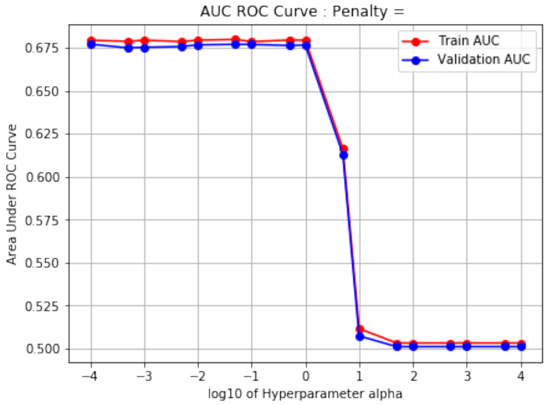
```
Performing Hyperparameter Tuning...

CV iteration : alpha=0.0001, penalty=l1, train_score=0.6703388284636733, test_score=0.67055206
CV iteration : alpha=0.0001, penalty=l1, train_score=0.6744887575909243, test_score=0.66904747
CV iteration : alpha=0.0001, penalty=l1, train_score=0.6739659263124029, test_score=0.67098238
C=0.0001, penalty=l1, train_score=0.6729311707890003, test_score=0.6701939741168363
CV iteration : alpha=0.0001, penalty=l2, train_score=0.6792123546646881, test_score=0.68076453
CV iteration : alpha=0.0001, penalty=l2, train_score=0.6794995618465143, test_score=0.67575352
CV iteration : alpha=0.0001, penalty=l2, train_score=0.6793057074173449, test_score=0.67464754
C=0.0001, penalty=l2, train_score=0.6793392079761825, test_score=0.6770552011126162
```

```
CV iteration : alpha=0.0005, penalty=l1, train_score=0.6119264138170805, test_score=0.620767772
CV iteration : alpha=0.0005, penalty=l1, train_score=0.6250971783219387, test_score=0.615530346
CV iteration : alpha=0.0005, penalty=l1, train_score=0.6329104139263981, test_score=0.624757962
C=0.0005, penalty=l1, train_score=0.623311335355139, test_score=0.6203520270975692
CV iteration : alpha=0.0005, penalty=l2, train_score=0.6771563486547317, test_score=0.674740149
CV iteration : alpha=0.0005, penalty=l2, train_score=0.6807477766784049, test_score=0.676215757
CV iteration : alpha=0.0005, penalty=l2, train_score=0.6781584267051505, test_score=0.673869788
C=0.0005, penalty=l2, train_score=0.6786875173460958, test_score=0.6749418983491936
CV iteration : alpha=0.001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l1, train_score=0.5, test_score=0.5
C=0.001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l2, train_score=0.6786576717718611, test_score=0.6770984190
CV iteration : alpha=0.001, penalty=l2, train_score=0.6803193796155272, test_score=0.6760576690
CV iteration : alpha=0.001, penalty=l2, train_score=0.6791883534486836, test_score=0.6725684547
C=0.001, penalty=l2, train_score=0.6793884682786907, test_score=0.6752415142773499
CV iteration : alpha=0.005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l1, train_score=0.5, test_score=0.5
C=0.005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l2, train_score=0.6778929387043878, test_score=0.6765621645
CV iteration : alpha=0.005, penalty=l2, train_score=0.6778761089853999, test_score=0.6749764077
CV iteration : alpha=0.005, penalty=l2, train_score=0.6799144660882157, test_score=0.6755156484
C=0.005, penalty=l2, train_score=0.6785611712593345, test_score=0.6756847402625574
CV iteration : alpha=0.01, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l1, train_score=0.5, test_score=0.5
C=0.01, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l2, train_score=0.6798839467981954, test_score=0.67951406806
CV iteration : alpha=0.01, penalty=l2, train_score=0.6793136244788004, test_score=0.67644740491
CV iteration : alpha=0.01, penalty=l2, train_score=0.6789878464290215, test_score=0.67402287096
C=0.01, penalty=l2, train_score=0.6793951392353392, test_score=0.6766614479813766
CV iteration : alpha=0.05, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l1, train_score=0.5, test_score=0.5
C=0.05, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l2, train_score=0.6792524977734355, test_score=0.67746756816
CV iteration : alpha=0.05, penalty=l2, train_score=0.6797386670812211, test_score=0.67658316643
CV iteration : alpha=0.05, penalty=l2, train_score=0.6804177737695805, test_score=0.67685235678
C=0.05, penalty=l2, train_score=0.6798029795414123, test_score=0.6769676971310686
CV iteration : alpha=0.1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l1, train_score=0.5, test_score=0.5
C=0.1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l2, train_score=0.6777599594118677, test_score=0.679050269175
CV iteration : alpha=0.1, penalty=l2, train_score=0.6783404356543575, test_score=0.675418462910
CV iteration : alpha=0.1, penalty=l2, train_score=0.6794960446870558, test_score=0.676268499960
C=0.1, penalty=l2, train_score=0.678532146584427, test_score=0.6769124106822711
```

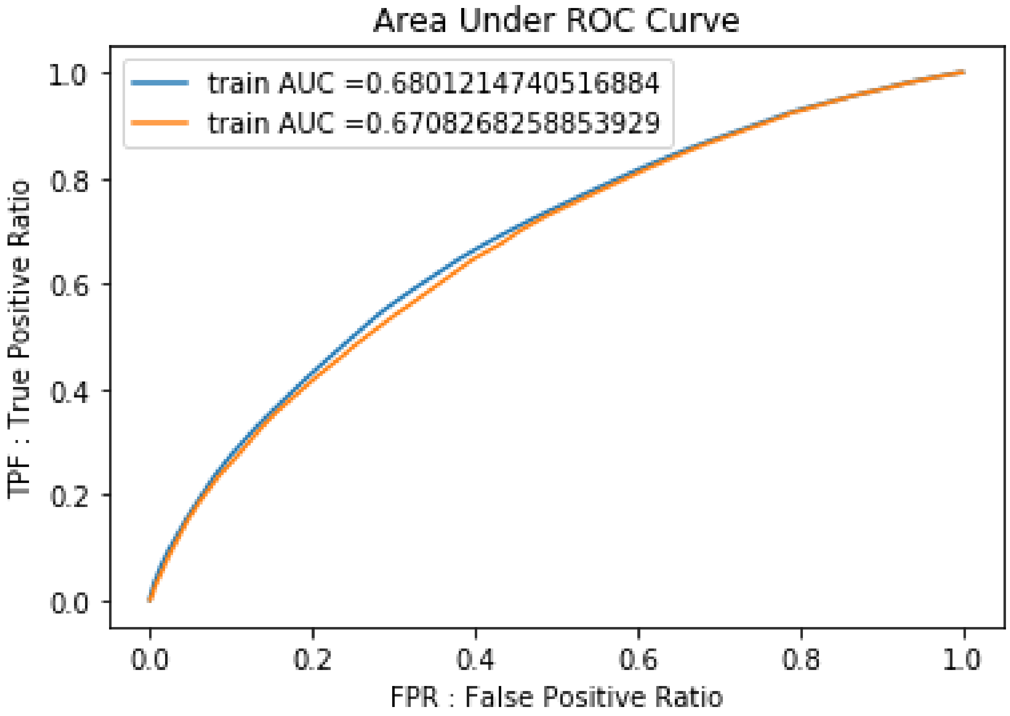
```
CV iteration : alpha=0.5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l1, train_score=0.5, test_score=0.5
C=0.5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l2, train_score=0.6790064164093121, test_score=0.678159172987
CV iteration : alpha=0.5, penalty=l2, train_score=0.6797806961078291, test_score=0.675912837285
CV iteration : alpha=0.5, penalty=l2, train_score=0.6795691692959644, test_score=0.674808641444
C=0.5, penalty=l2, train_score=0.6794520939377019, test_score=0.6762935505725977
CV iteration : alpha=1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l1, train_score=0.5, test_score=0.5
C=1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l2, train_score=0.6785175573498236, test_score=0.67864957307893
CV iteration : alpha=1, penalty=l2, train_score=0.6793057738488302, test_score=0.67588843518468
CV iteration : alpha=1, penalty=l2, train_score=0.6802346917752361, test_score=0.67522384815301
C=1, penalty=l2, train_score=0.67935267432463, test_score=0.6765872854722095
CV iteration : alpha=5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l1, train_score=0.5, test_score=0.5
C=5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l2, train_score=0.616577146886382, test_score=0.612881813392445
CV iteration : alpha=5, penalty=l2, train_score=0.6164770209896607, test_score=0.61037642505884
CV iteration : alpha=5, penalty=l2, train_score=0.61611767945365, test_score=0.615816873521407
C=5, penalty=l2, train_score=0.6163906157765643, test_score=0.6130250373242317
CV iteration : alpha=10, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l1, train_score=0.5, test_score=0.5
C=10, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l2, train_score=0.523433558502062, test_score=0.51552179392870
CV iteration : alpha=10, penalty=l2, train_score=0.507762881789551, test_score=0.50580053745737
CV iteration : alpha=10, penalty=l2, train_score=0.50368048682253, test_score=0.500850698431823
C=10, penalty=l2, train_score=0.511625642371381, test_score=0.5073910099392996
CV iteration : alpha=50, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l1, train_score=0.5, test_score=0.5
C=50, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l2, train_score=0.502824484413134, test_score=0.50201074017444
CV iteration : alpha=50, penalty=l2, train_score=0.5033239523126721, test_score=0.5006029164740
CV iteration : alpha=50, penalty=l2, train_score=0.50368048682253, test_score=0.500850698431823
C=50, penalty=l2, train_score=0.5032763078494454, test_score=0.5011547850267878
CV iteration : alpha=100, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l1, train_score=0.5, test_score=0.5
C=100, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l2, train_score=0.502824484413134, test_score=0.5020107401744
CV iteration : alpha=100, penalty=l2, train_score=0.5033239523126721, test_score=0.500602916474
CV iteration : alpha=100, penalty=l2, train_score=0.50368048682253, test_score=0.50085069843182
C=100, penalty=l2, train_score=0.5032763078494454, test_score=0.5011547850267878
```

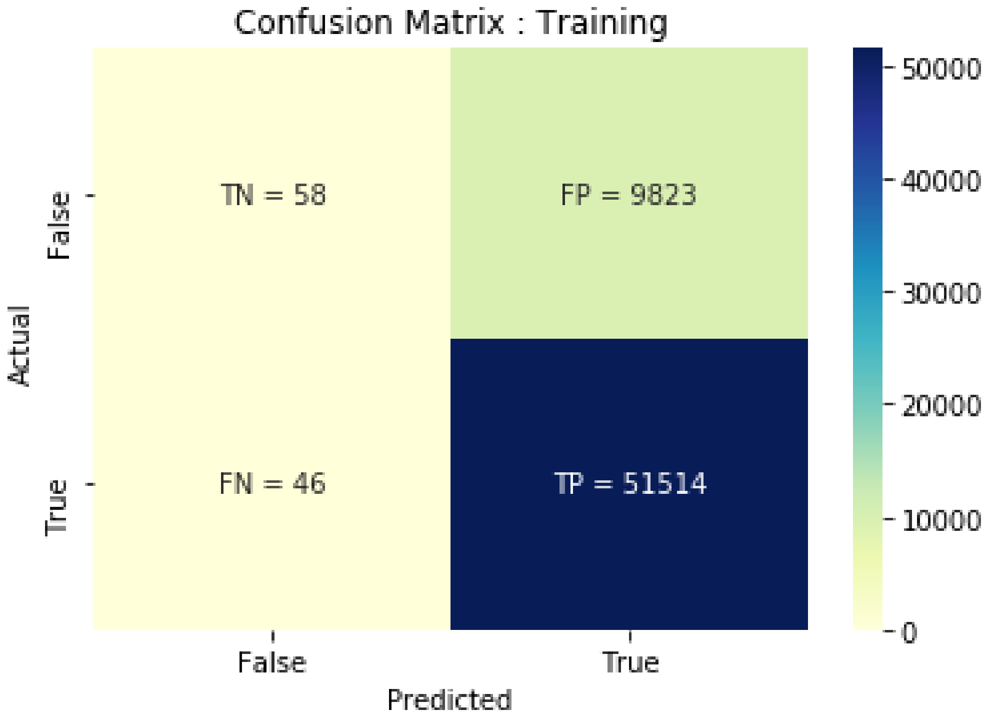
```
CV iteration : alpha=500, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l1, train_score=0.5, test_score=0.5
C=500, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l2, train_score=0.502824484413134, test_score=0.5020107401744
CV iteration : alpha=500, penalty=l2, train_score=0.5033239523126721, test_score=0.500602916474
CV iteration : alpha=500, penalty=l2, train_score=0.50368048682253, test_score=0.50085069843182
C=500, penalty=l2, train_score=0.5032763078494454, test_score=0.5011547850267878
CV iteration : alpha=1000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l1, train_score=0.5, test_score=0.5
C=1000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l2, train_score=0.502824484413134, test_score=0.502010740174
CV iteration : alpha=1000, penalty=l2, train_score=0.5033239523126721, test_score=0.50060291647
CV iteration : alpha=1000, penalty=l2, train_score=0.50368048682253, test_score=0.5008506984318
C=1000, penalty=l2, train_score=0.5032763078494454, test_score=0.5011547850267878
CV iteration : alpha=5000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l1, train_score=0.5, test_score=0.5
C=5000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l2, train_score=0.502824484413134, test_score=0.502010740174
CV iteration : alpha=5000, penalty=l2, train_score=0.5033239523126721, test_score=0.50060291647
CV iteration : alpha=5000, penalty=l2, train_score=0.50368048682253, test_score=0.5008506984318
C=5000, penalty=l2, train_score=0.5032763078494454, test_score=0.5011547850267878
CV iteration : alpha=10000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l1, train_score=0.5, test_score=0.5
C=10000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l2, train_score=0.502824484413134, test_score=0.50201074017
CV iteration : alpha=10000, penalty=l2, train_score=0.5033239523126721, test_score=0.5006029164
CV iteration : alpha=10000, penalty=l2, train_score=0.50368048682253, test_score=0.500850698431
C=10000, penalty=l2, train_score=0.5032763078494454, test_score=0.5011547850267878
```

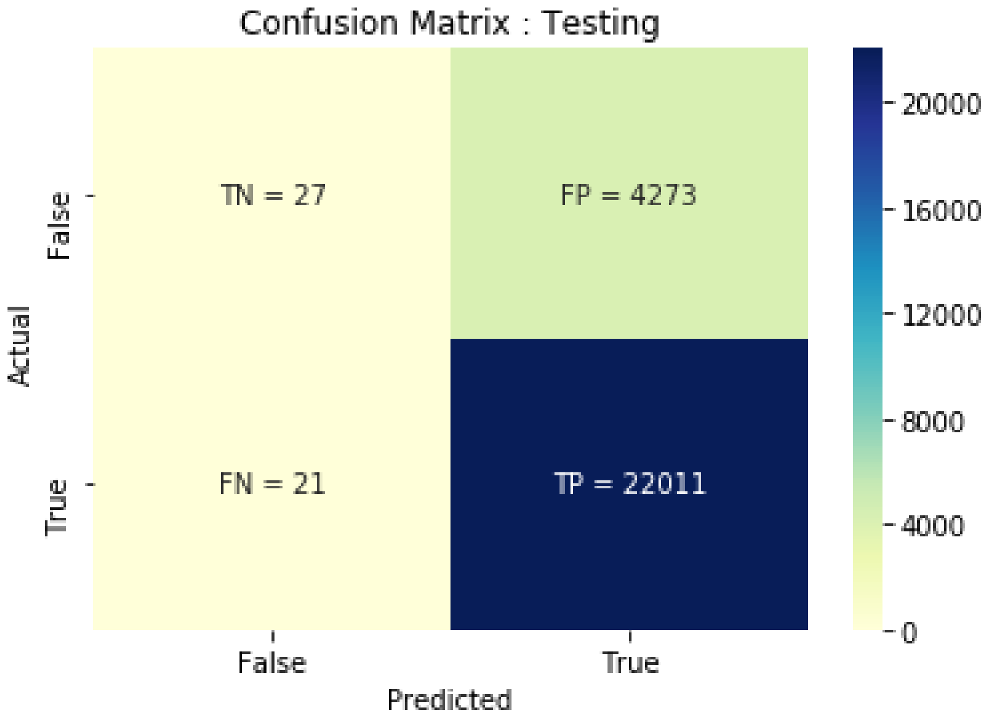




```
{'svm__alpha': 0.0001, 'svm__penalty': 'l2'}
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.6801214740516884
Area Under the Curve for Test :  0.6708268258853929
```




Confusion Matrix : Training
Actual
False
True
TN = 58
FP = 9823
FN = 46
TP = 51514
False
True
Predicted
50000
40000
30000
20000
10000
0


Confusion Matrix : Testing
Actual
False
True
TN = 27
FP = 4273
FN = 21
TP = 22011
False
True
Predicted
20000
16000
12000
8000
4000
0

7.2.2 [5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

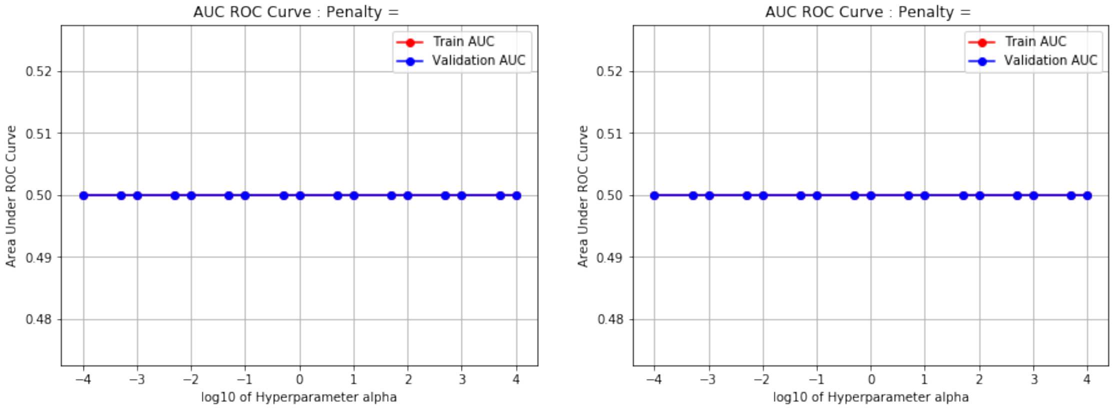
```
In [152]: # Please write all the code with proper documentation
          csv_path = 'saved-models/Assignment7/TFIDF-W2Vec_svm_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF-
                                                     penalty=['l1', 'l2'], results_path=csv_pa
                                                     retrain=True, W2V_model=w2v_model)
          # Analysing best parameters
          best_parameters = analyse_results(cv_results)
          pprint.pprint(best_parameters)
          # retraining the model with best parameters
          model_path = 'saved-models/Assignment7/{0}_svm.pkl'.format('TFIDF-W2Vec')
          calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFID

          print('Retraining Vectorizer with Dx_train')
          vectorizer_obj = get_vectorizer(W2V_model = w2v_model, train=Dx_train, vectorizer='T

          # plotting AUC ROC
          train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_

          # appending the data results
          prettytable_data.append(['TFIDF-W2Vec', 'SVM', best_parameters['svm__penalty'], best_
```

```
Performing Hyperparameter Tuning...

CV iteration : alpha=0.0001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0001, penalty=l1, train_score=0.5, test_score=0.5
C=0.0001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0001, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0001, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0001, penalty=l2, train_score=0.5, test_score=0.5
C=0.0001, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0005, penalty=l1, train_score=0.5, test_score=0.5
C=0.0005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0005, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0005, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.0005, penalty=l2, train_score=0.5, test_score=0.5
C=0.0005, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l1, train_score=0.5, test_score=0.5
C=0.001, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.001, penalty=l2, train_score=0.5, test_score=0.5
C=0.001, penalty=l2, train_score=0.5, test_score=0.5
```

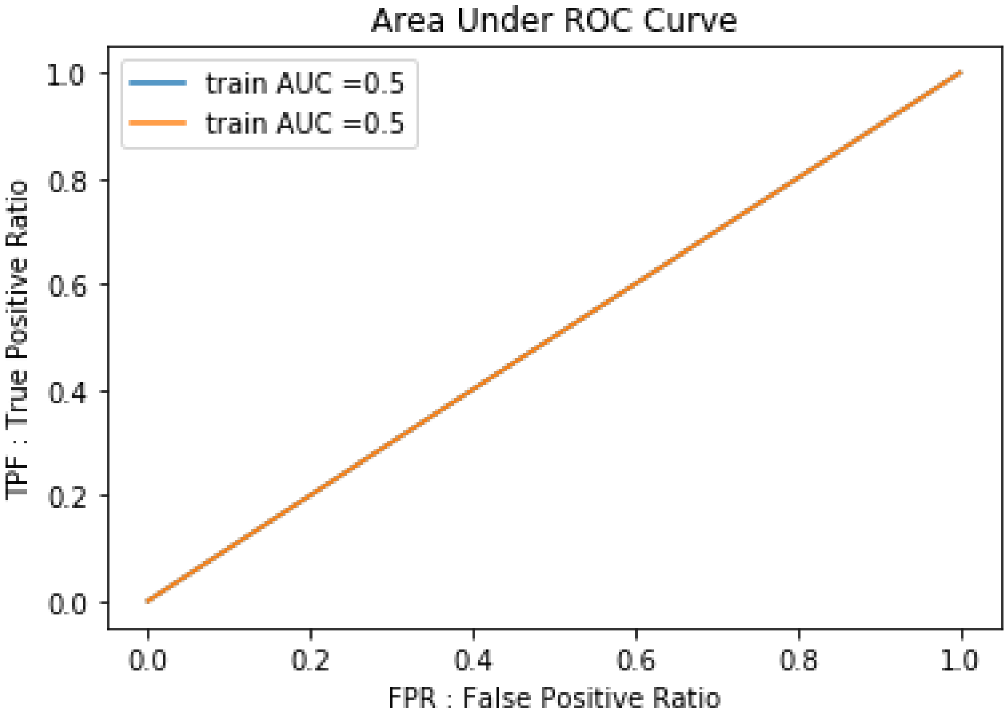
```
CV iteration : alpha=0.005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l1, train_score=0.5, test_score=0.5
C=0.005, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.005, penalty=l2, train_score=0.5, test_score=0.5
C=0.005, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l1, train_score=0.5, test_score=0.5
C=0.01, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.01, penalty=l2, train_score=0.5, test_score=0.5
C=0.01, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l1, train_score=0.5, test_score=0.5
C=0.05, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.05, penalty=l2, train_score=0.5, test_score=0.5
C=0.05, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l1, train_score=0.5, test_score=0.5
C=0.1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.1, penalty=l2, train_score=0.5, test_score=0.5
C=0.1, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l1, train_score=0.5, test_score=0.5
C=0.5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=0.5, penalty=l2, train_score=0.5, test_score=0.5
C=0.5, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l1, train_score=0.5, test_score=0.5
C=1, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=1, penalty=l2, train_score=0.5, test_score=0.5
C=1, penalty=l2, train_score=0.5, test_score=0.5
```

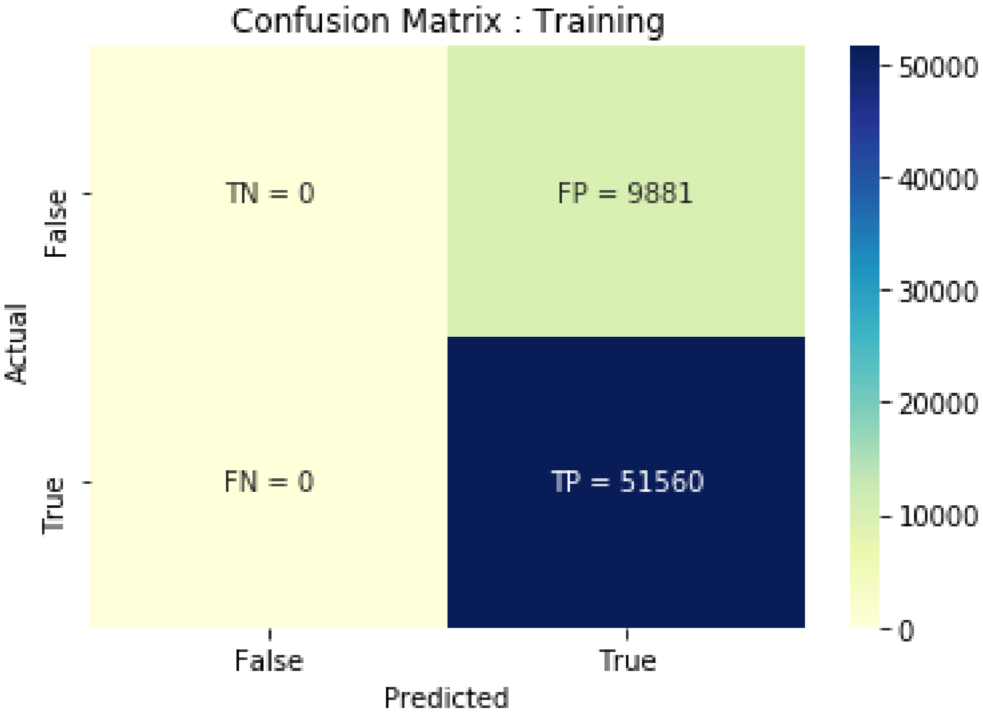
```
CV iteration : alpha=5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l1, train_score=0.5, test_score=0.5
C=5, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=5, penalty=l2, train_score=0.5, test_score=0.5
C=5, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l1, train_score=0.5, test_score=0.5
C=10, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=10, penalty=l2, train_score=0.5, test_score=0.5
C=10, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l1, train_score=0.5, test_score=0.5
C=50, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=50, penalty=l2, train_score=0.5, test_score=0.5
C=50, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l1, train_score=0.5, test_score=0.5
C=100, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=100, penalty=l2, train_score=0.5, test_score=0.5
C=100, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l1, train_score=0.5, test_score=0.5
C=500, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=500, penalty=l2, train_score=0.5, test_score=0.5
C=500, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l1, train_score=0.5, test_score=0.5
C=1000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=1000, penalty=l2, train_score=0.5, test_score=0.5
C=1000, penalty=l2, train_score=0.5, test_score=0.5
```

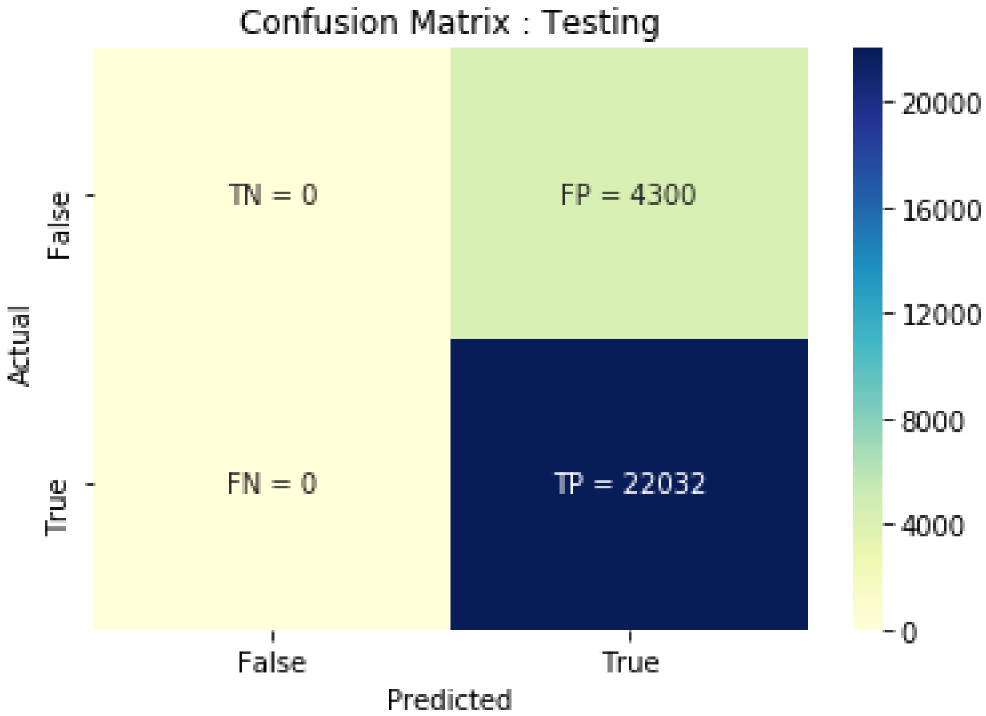
```
CV iteration : alpha=5000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l1, train_score=0.5, test_score=0.5
C=5000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=5000, penalty=l2, train_score=0.5, test_score=0.5
C=5000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l1, train_score=0.5, test_score=0.5
C=10000, penalty=l1, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l2, train_score=0.5, test_score=0.5
CV iteration : alpha=10000, penalty=l2, train_score=0.5, test_score=0.5
C=10000, penalty=l2, train_score=0.5, test_score=0.5
```



```
{'svm__alpha': 0.0001, 'svm__penalty': 'l1'}
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.5
Area Under the Curve for Test :  0.5
```


Area Under ROC Curve
train AUC =0.5
train AUC =0.5
TPF : True Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
FPR : False Positive Ratio


Confusion Matrix : Training
TN = 0
FP = 9881
FN = 0
TP = 51560
Actual
False
True
False
True
Predicted
50000
40000
30000
20000
10000
0



7.3 [5.2] RBF SVM

```
In [50]: num_data_points = 40000
```

```
In [51]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(preprocessed_reviews[:num_data
```

7.4 A great article that helped me a lot in understanding the parameters of SVC in depth

https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769

```
In [52]: ## '''Perform Simple Cross Validation'''
         def perform_hyperparameter_tuning_rbf(X, Y, vectorizer, results_path, retrain=False, 
             #If the pandas dataframe with the hyperparameter info exists then return it

             if(retrain==False):
                 # If Cross Validation results exists then return them
                 if(os.path.exists(results_path)):
                     return pd.read_csv(results_path)
                 else:
                     # If no data exists but retrain=False then mention accordingly
                     print('Retrain is set to be False but no Cross Validation Results DataFra
             else:
                 # else perform hyperparameter tuning
```

```
print('Performing Hyperparameter Tuning...\n')
# regularization parameter
hyperparameters = {
    'svm__C' : [0.0001, 0.01, 1, 100, 10000],
}

C_values = []

train_scores = []
test_scores = []

train_mean_score = []
test_mean_score = []

# Initializing KFold
skf = StratifiedKFold(n_splits=3)
X = np.array(X)
Y = np.array(Y)

for C in hyperparameters['svm__C']:

    #Performing Cross Validation
    for train_index, test_index in skf.split(X, Y):
        Dx_train, Dx_cv = X[train_index], X[test_index]
        Dy_train, Dy_cv = Y[train_index], Y[test_index]

        #Initializing the Vectorizer
        vectorizer = get_vectorizer(vectorizer, Dx_train.tolist(), W2V_model)

        #Transforming the data to features
        x_train = vectorizer.transform(Dx_train.tolist())
        x_cv = vectorizer.transform(Dx_cv.tolist())

        #Initializing the LR model
        calib_svm = SVC(kernel='rbf', C=C, max_iter=1000, verbose=False, proba

        # Fit the model
        calib_svm.fit(x_train, Dy_train)

        #Prediction
        train_results = calib_svm.predict_proba(x_train)
        cv_results = calib_svm.predict_proba(x_cv)

        try:
            train_score = roc_auc_score(Dy_train, train_results[:, 1])
            test_score = roc_auc_score(Dy_cv, cv_results[:, 1])

            #storing the results to form a dataframe
```

```
                             train_scores.append(train_score)
                             test_scores.append(test_score)

                         except Exception as e:
                             print('Error Case : ', e)
                             print(('Actual, Predicted'))
                             [print((Dy_cv[i], cv_results[i, 1])) for i in range(len(Dy_cv))]


                     train_mean_score.append(sum(train_scores)/len(train_scores))
                     test_mean_score.append(sum(test_scores)/len(test_scores))

                     C_values.append(C)

                     print('C={0}, train_score={1}, test_score={2}'
                           .format(C, sum(train_scores)/len(train_scores), sum(test_scores)/len

                     train_scores = []
                     test_scores = []

                 # Creating a DataFrame from the saved data for visualization
                 results_df = pd.DataFrame({'C' : C_values, 'train_score' : train_mean_score,
                                            'test_score': test_mean_score})

                 #writing the results to csv after performing hyperparameter tuning
                 try:
                     results_df.to_csv(results_path)
                 except Exception as ex:
                     print(str(ex), "\nError occured while converting DataFrame to CSV after c
                 return results_df

In [53]: def analyse_results(df):
             # plotting error curves
             fig = plt.figure()
             ax = fig.gca()

             plt.plot([math.log10(i) for i in df.C.tolist()], df.test_score.tolist(), '-o', c=
             plt.plot([math.log10(i) for i in df.C.tolist()], df.train_score.tolist(), '-o', c
             plt.grid(True)
             plt.xlabel('log10 of "C"')
             plt.ylabel('Area Under ROC Curve')
             plt.title('AUC ROC Curve for Logistic Regression')
             plt.legend(loc='best')
             plt.show()

             # return the best parameters
             mmax = 0
             ind_max = 0
             for index, row in df.iterrows():
                 if(row['test_score']>mmax):
                     mmax=row['test_score']
                     ind_max = index


             best_params = {
                 'svm__C': df.loc[ind_max, 'C']
             }

             return best_params

In [54]: def retrain_with_best_params(data, labels, best_params, vec_name, model_path, word2ve
             if(os.path.exists(model_path)):
                 print('Loading Model....')
                 with open(model_path, 'rb') as input_file:
                     calib_svm = pickle.load(input_file)
             else:
                 calib_svm = SVC(kernel='rbf', C=best_params['svm__C'], max_iter=1000, verbose=

                 print('Initializing Vectorizer')
                 vectorizer = get_vectorizer(vectorizer=vec_name, train=data, W2V_model=word2ve
                 print('Training Model....')
                 calib_svm.fit(vectorizer.transform(data), np.array(labels))

                 print('Saving Trained Model....')
                 with open(model_path,'wb') as file:
                     pickle.dump(calib_svm, file)
             return calib_svm

In [55]: def plot_confusion_matrix(model, data, labels, dataset_label):
             pred = model.predict(data)
             conf_mat = confusion_matrix(labels, pred)

             strings = strings = np.asarray([['TN = ', 'FP = '],
                                             ['FN = ', 'TP = ']])

             labels = (np.asarray(["{0}{1}".format(string, value)
                                   for string, value in zip(strings.flatten(),
                                                            conf_mat.flatten())])
                      ).reshape(2, 2)

             fig, ax = plt.subplots()
             ax.set(xlabel='Predicted', ylabel='Actual', title='Confusion Matrix : {0}'.format
             sns.heatmap(conf_mat, annot=labels, fmt="", cmap='YlGnBu', ax=ax)
             ax.set_xlabel('Predicted')
             ax.set_ylabel('Actual')
             ax.set_xticklabels(['False', 'True'])
```

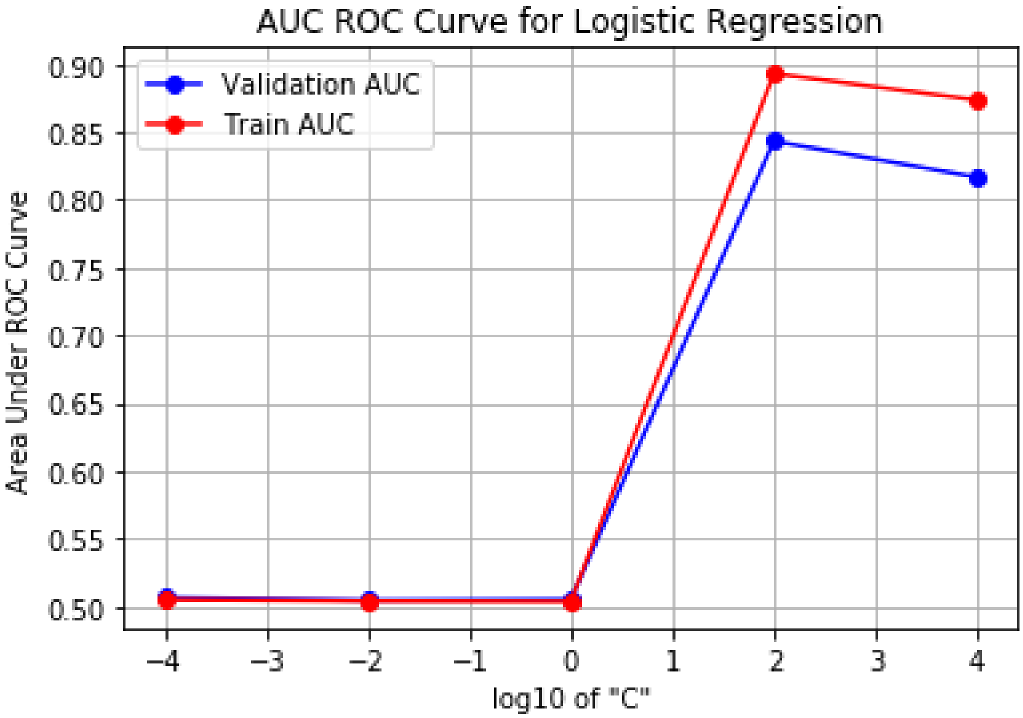
```
    ax.set_yticklabels(['False', 'True'])
    plt.show()
```

In [56]:
```
def plot_AUC_ROC(model, vectorizer, Dx_train, Dx_test, Dy_train, Dy_test):

    #predicting probability of Dx_test, Dx_train
    test_score = model.predict_proba(vectorizer.transform(Dx_test))
    train_score = model.predict_proba(vectorizer.transform(Dx_train))

    #Finding out the ROC_AUC_SCORE
    train_roc_auc_score = roc_auc_score(np.array(Dy_train), train_score[:, 1])
    print('Area Under the Curve for Train : ', train_roc_auc_score)
    test_roc_auc_score = roc_auc_score(np.array(Dy_test), test_score[:, 1])
    print('Area Under the Curve for Test : ', test_roc_auc_score)

    #Plotting with matplotlib.pyplot
    #ROC Curve for D-train
    train_fpr, train_tpr, thresholds = roc_curve(np.array(Dy_train), train_score[:, 1]
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))

    # #ROC Curve for D-test
    test_fpr, test_tpr, thresholds = roc_curve(np.array(Dy_test), test_score[:, 1])
    plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))

    plt.legend()
    plt.xlabel("FPR : False Positive Ratio")
    plt.ylabel("TPF : True Positive Ratio")
    plt.title("Area Under ROC Curve")
    plt.show()

    plot_confusion_matrix(model, vectorizer.transform(Dx_train), np.array(Dy_train),
    plot_confusion_matrix(model, vectorizer.transform(Dx_test), np.array(Dy_test), 'Te
    return train_roc_auc_score, test_roc_auc_score
```

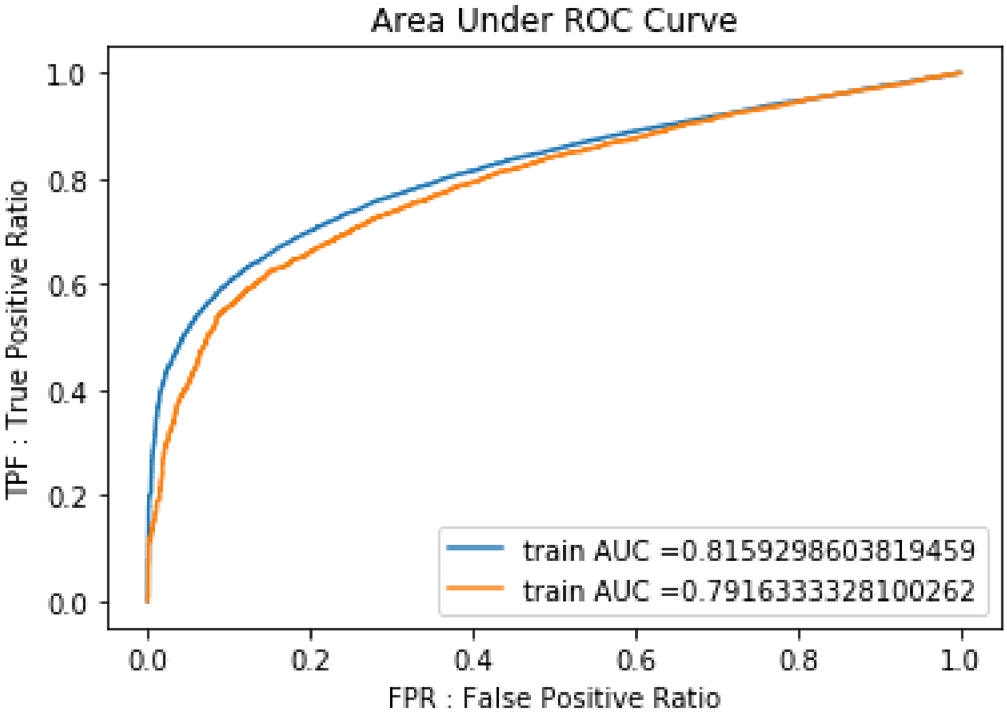
7.4.1 [5.2.1] Applying RBF SVM on BOW, SET 1

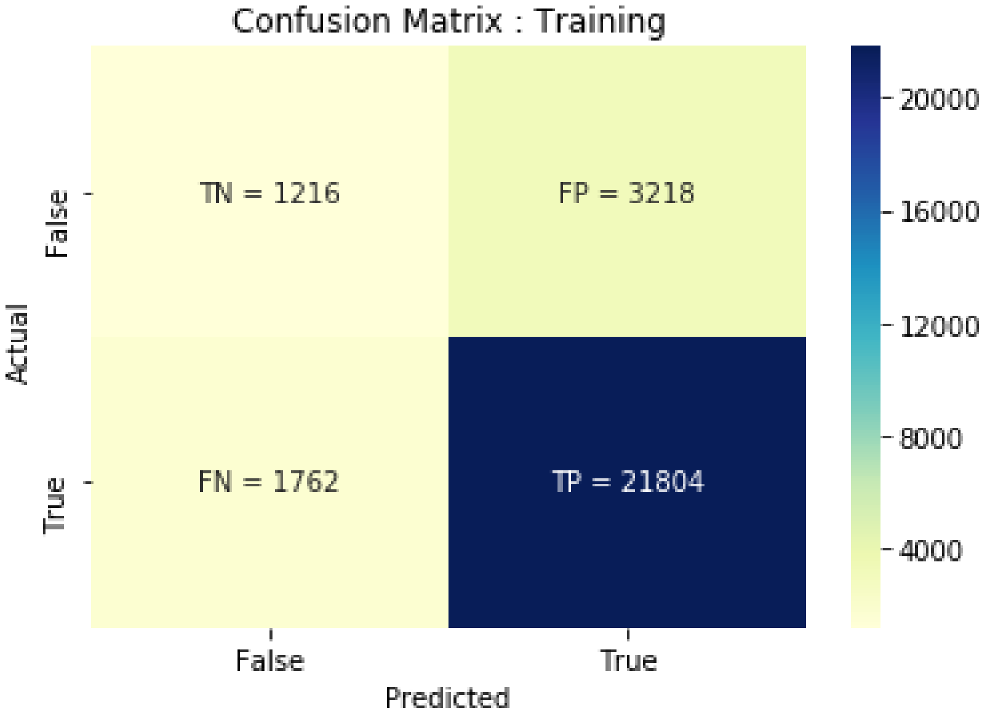
In [57]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment7/BOW_svm_rbf_results.csv'
cv_results = perform_hyperparameter_tuning_rbf(X=Dx_train, Y=Dy_train, vectorizer='BOW
                                           results_path=csv_path, retrain=False, W2V_
# Analysing best parameters
best_parameters = analyse_results(cv_results)
pprint.pprint(best_parameters)
# retraining the model with best parameters
model_path = 'saved_models/Assignment7/{0}_svm_rbf.pkl'.format('BOW')
calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'BOW',

print('Retraining Vectorizer with Dx_train')
```

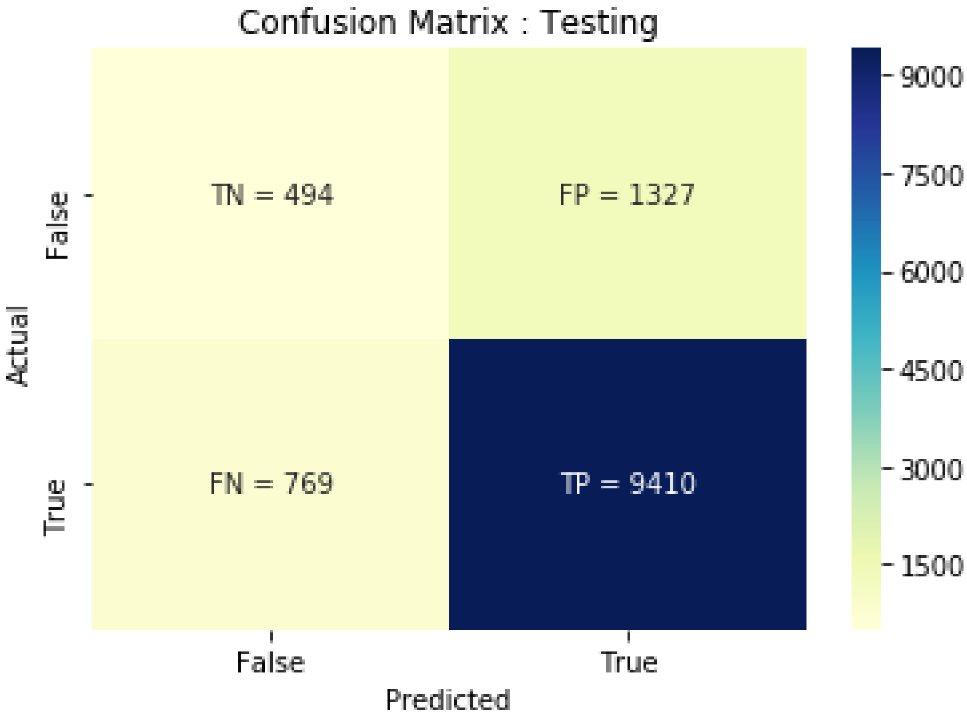
```
vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='BOW')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_te

# appending the data results
prettytable_data.append(['BOW', 'SVM-rbf', best_parameters['svm__C'], None, train_sco
```



```
{'svm__C': 100.0}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.8159298603819459
Area Under the Curve for Test :  0.7916333328100262
```

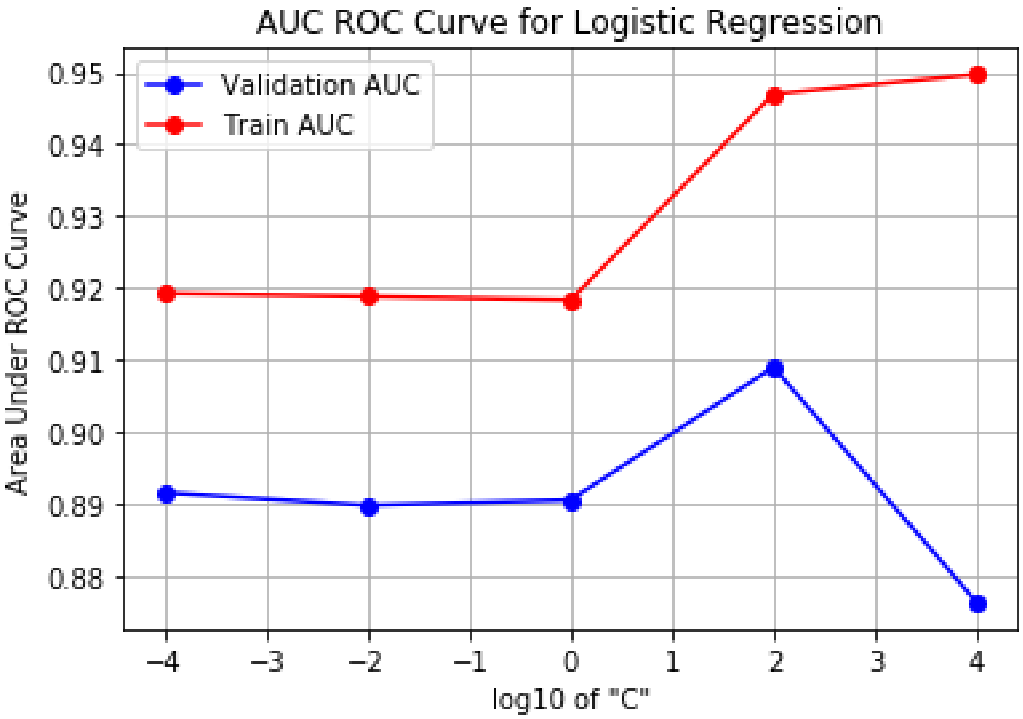

Area Under ROC Curve
TPF : True Positive Ratio
FPR : False Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
train AUC =0.8159298603819459
train AUC =0.7916333328100262


Confusion Matrix : Training
TN = 1216
FP = 3218
FN = 1762
TP = 21804
Actual
False
True
Predicted
False
True
20000
16000
12000
8000
4000

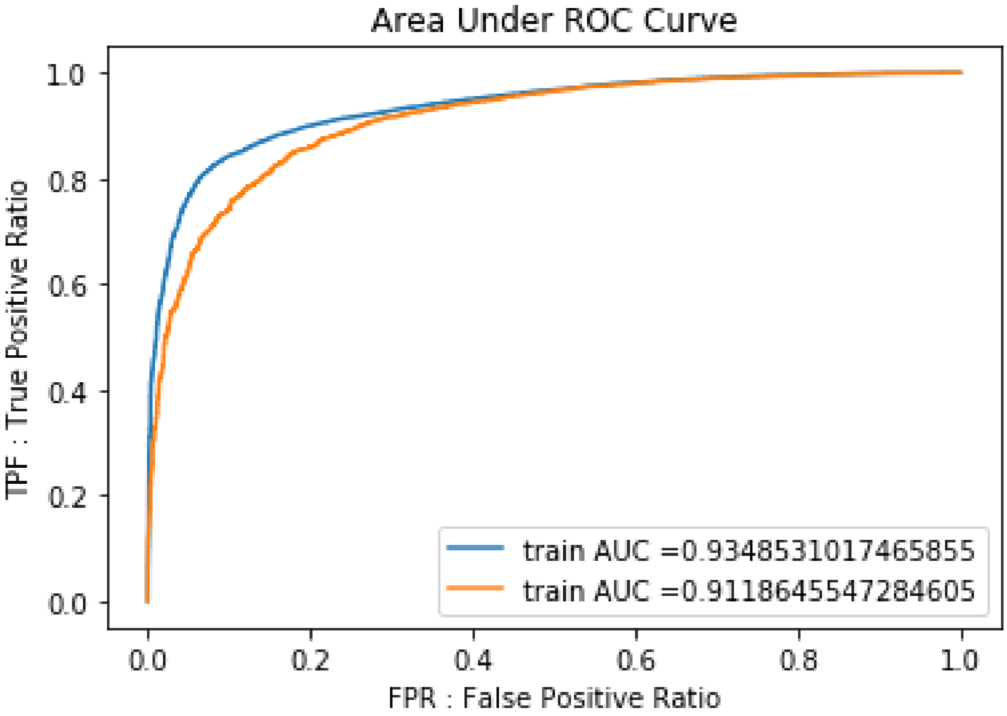


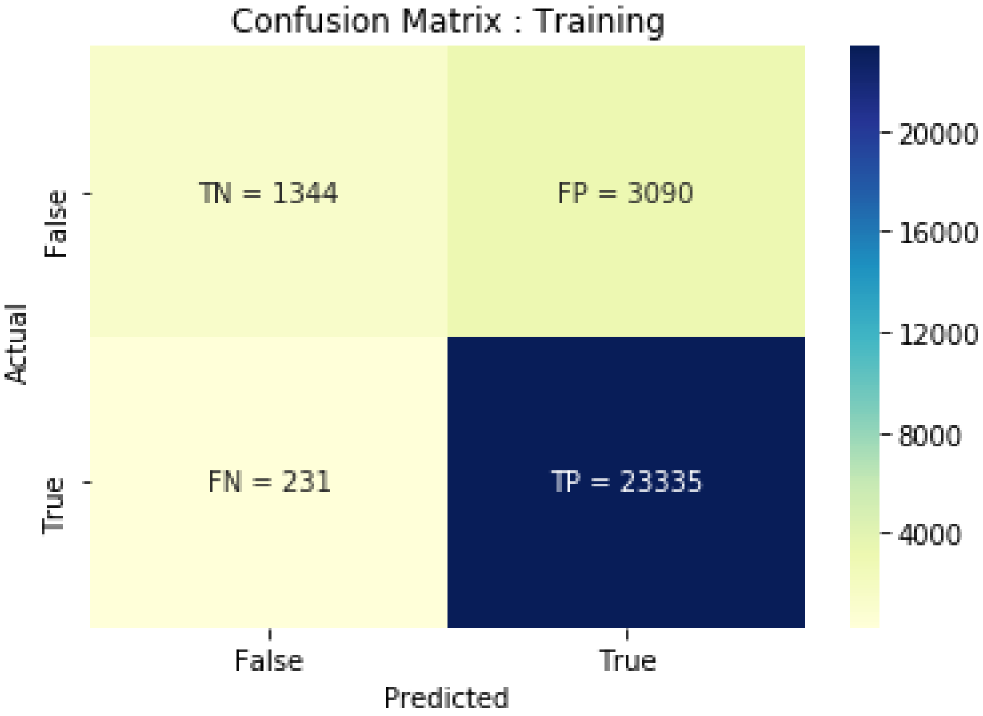
7.4.2 [5.2.2] Applying RBF SVM on TFIDF, SET 2

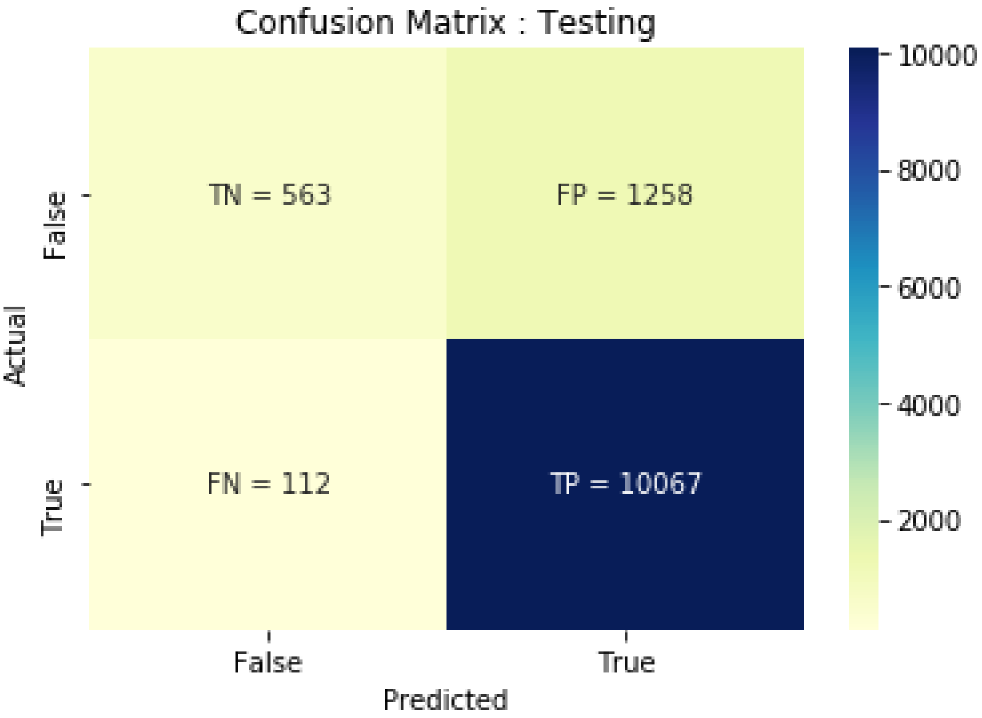
```
In [58]: # Please write all the code with proper documentation
         csv_path = 'saved_models/Assignment7/TFIDF_svm_rbf_results.csv'
         cv_results = perform_hyperparameter_tuning_rbf(X=Dx_train, Y=Dy_train, vectorizer='TF
                                                    results_path=csv_path, retrain=False, W2V_
         # Analysing best parameters
         best_parameters = analyse_results(cv_results)
         pprint.pprint(best_parameters)
         # retraining the model with best parameters
         model_path = 'saved_models/Assignment7/{0}_svm_rbf.pkl'.format('TFIDF')
         calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF

         print('Retraining Vectorizer with Dx_train')
         vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='TFIDF')

         # plotting AUC ROC
         train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_te

         # appending the data results
         prettytable_data.append(['TFIDF', 'SVM-rbf', best_parameters['svm__C'], None, train_sc
```



```
{'svm__C': 100.0}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9348531017465855
Area Under the Curve for Test :  0.9118645547284605
```


Area Under ROC Curve
TPF : True Positive Ratio
FPR : False Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
train AUC =0.9348531017465855
train AUC =0.9118645547284605


Confusion Matrix : Training
TN = 1344
FP = 3090
FN = 231
TP = 23335
Actual
False
True
Predicted
False
True
20000
16000
12000
8000
4000

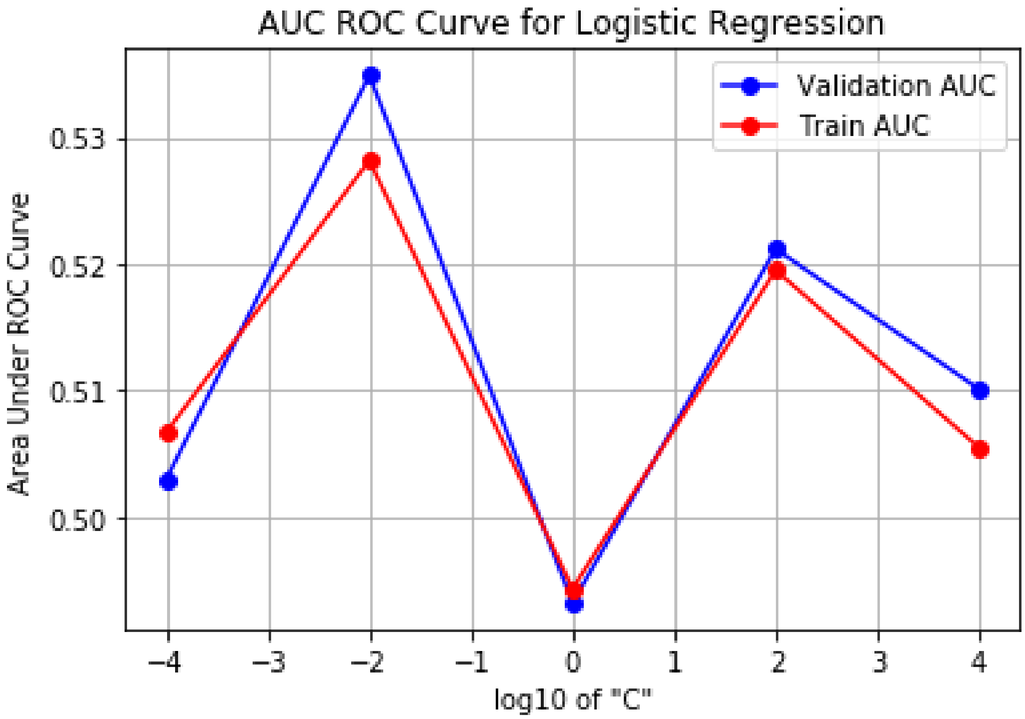


7.4.3 [5.2.3] Applying RBF SVM on AVG W2V, SET 3

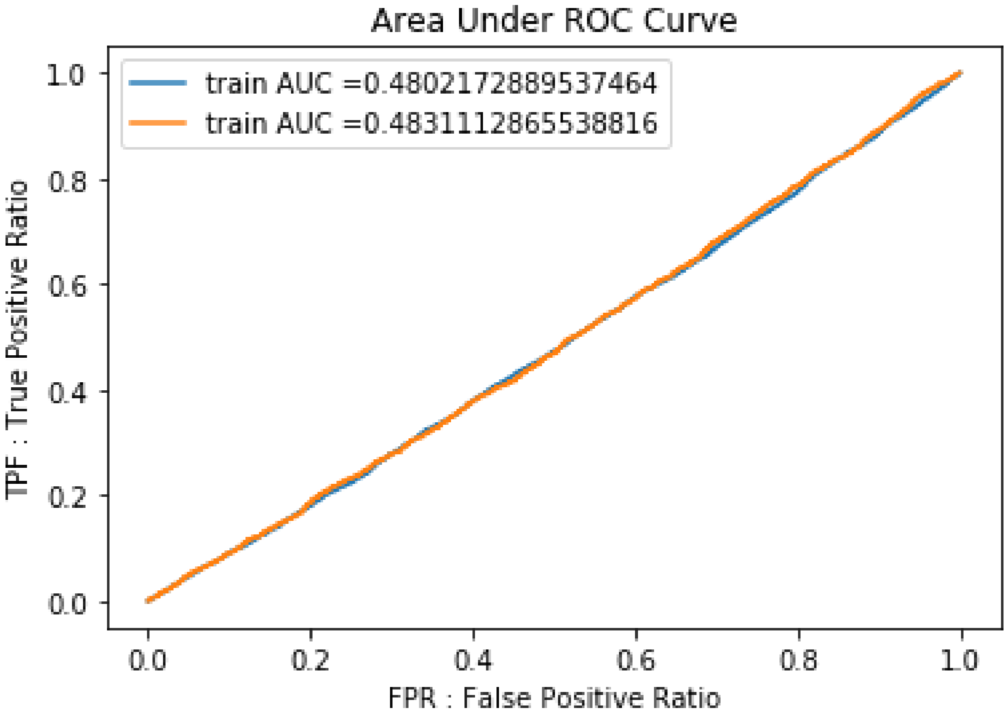
```
In [59]: # Please write all the code with proper documentation
         csv_path = 'saved_models/Assignment7/Avg-W2Vec_svm_rbf_results.csv'
         cv_results = perform_hyperparameter_tuning_rbf(X=Dx_train, Y=Dy_train, vectorizer='Av
                                                    results_path=csv_path, retrain=True, W2V_mo
         # Analysing best parameters
         best_parameters = analyse_results(cv_results)
         pprint.pprint(best_parameters)
         # retraining the model with best parameters
         model_path = 'saved_models/Assignment7/{0}_svm_rbf.pkl'.format('Avg-W2Vec')
         calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-W2

         print('Retraining Vectorizer with Dx_train')
         vectorizer_obj = get_vectorizer(W2V_model = w2v_model, train=Dx_train, vectorizer='Av

         # plotting AUC ROC
         train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_te

         # appending the data results
         prettytable_data.append(['Avg-W2Vec', 'SVM-rbf', best_parameters['svm__C'], None, trai
```

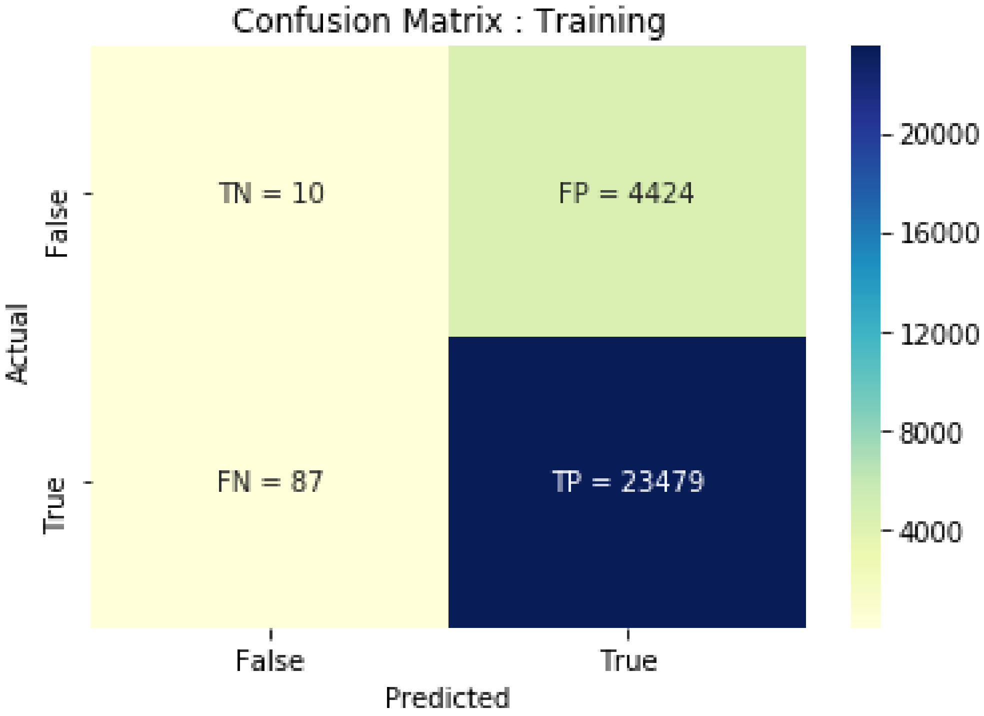
```
Performing Hyperparameter Tuning...
```

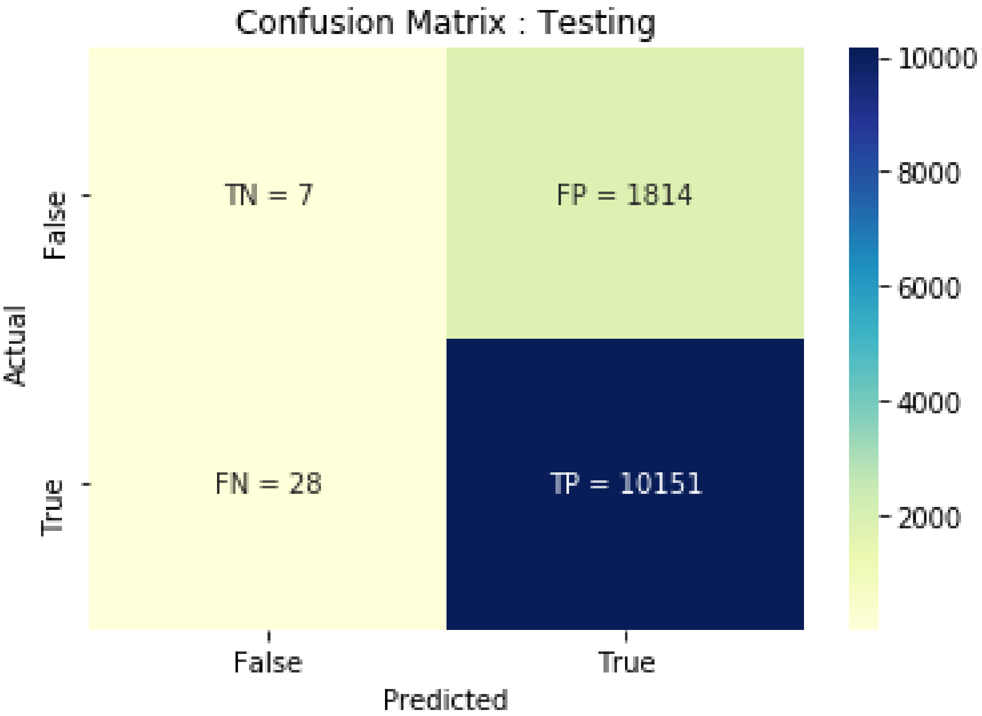
```
C=0.0001, train_score=0.5066795583911804, test_score=0.5029801141328885
C=0.01, train_score=0.528162579761382, test_score=0.5349074065240971
C=1, train_score=0.49431304584432945, test_score=0.49320339706620997
C=100, train_score=0.5194949951403993, test_score=0.5211663363414948
C=10000, train_score=0.5054711810816822, test_score=0.5100912423750392
```



```
{'svm__C': 0.01}
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.4802172889537464
Area Under the Curve for Test :  0.4831112865538816
```


Area Under ROC Curve
train AUC =0.4802172889537464
train AUC =0.4831112865538816
TPF : True Positive Ratio
FPR : False Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix : Training
TN = 10
FP = 4424
FN = 87
TP = 23479
Actual
False
True
Predicted
False
True
20000
16000
12000
8000
4000

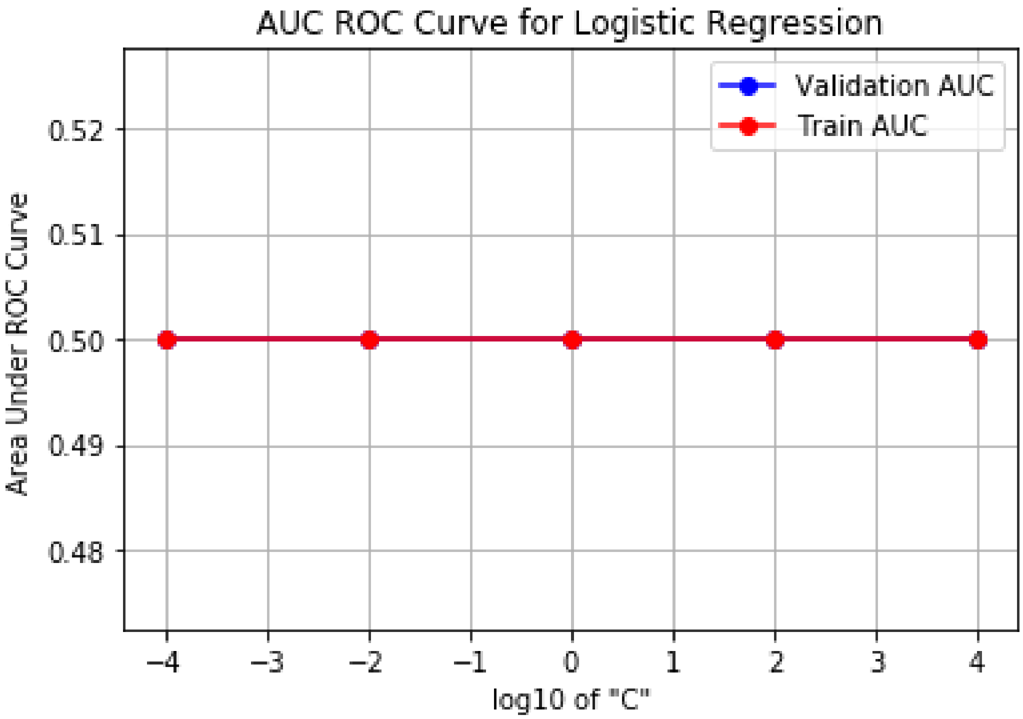


7.4.4 [5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

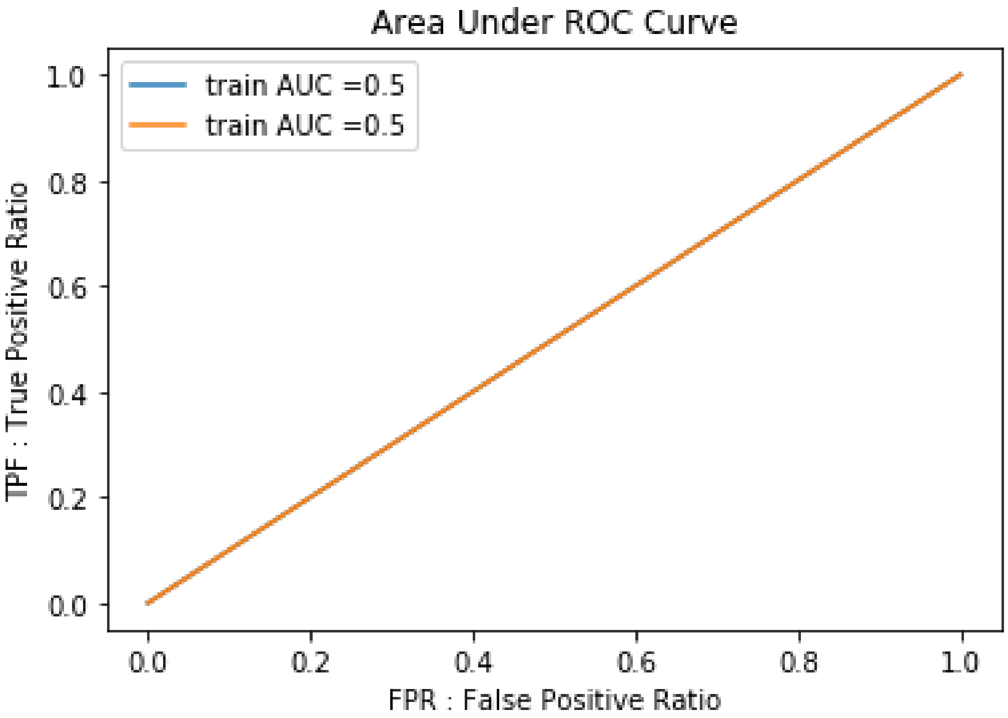
```
In [60]: # Please write all the code with proper documentation
         csv_path = 'saved_models/Assignment7/TFIDF-W2Vec_svm_rbf_results.csv'
         cv_results = perform_hyperparameter_tuning_rbf(X=Dx_train, Y=Dy_train, vectorizer='TF
                                                    results_path=csv_path, retrain=True, W2V_mo
         # Analysing best parameters
         best_parameters = analyse_results(cv_results)
         pprint.pprint(best_parameters)
         # retraining the model with best parameters
         model_path = 'saved_models/Assignment7/{0}_svm_rbf.pkl'.format('TFIDF-W2Vec')
         calibrated_svm = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF-

         print('Retraining Vectorizer with Dx_train')
         vectorizer_obj = get_vectorizer(W2V_model = w2v_model, train=Dx_train, vectorizer='TF

         # plotting AUC ROC
         train_score, test_score = plot_AUC_ROC(calibrated_svm, vectorizer_obj, Dx_train, Dx_te

         # appending the data results
         prettytable_data.append(['TFIDF-W2Vec', 'SVM-rbf', best_parameters['svm__C'], None, t
```

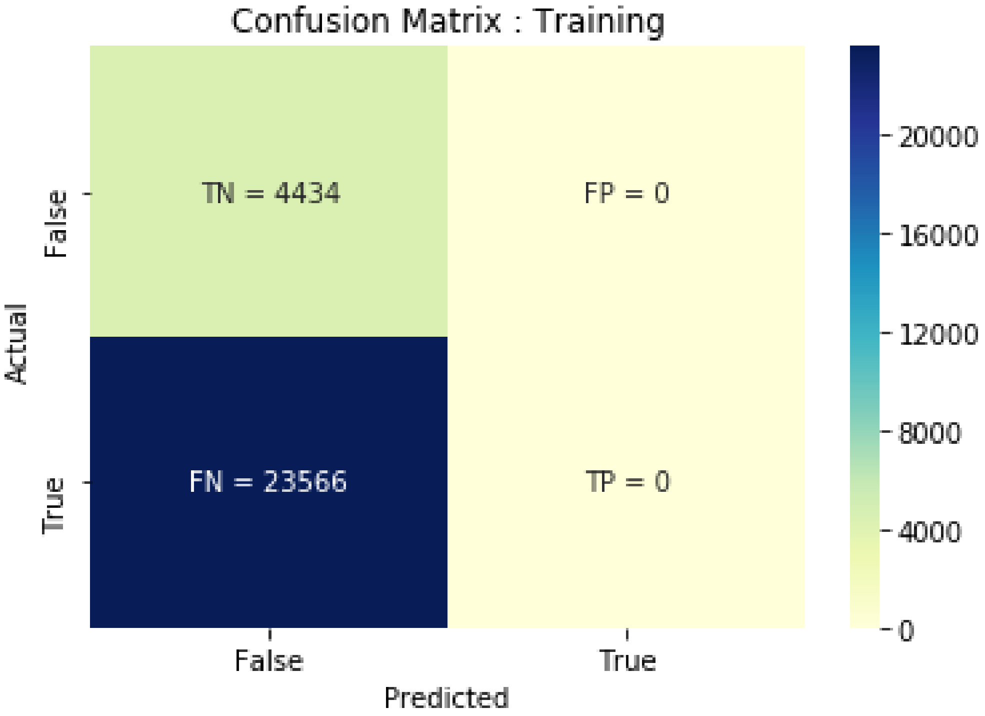
```
Performing Hyperparameter Tuning...
```

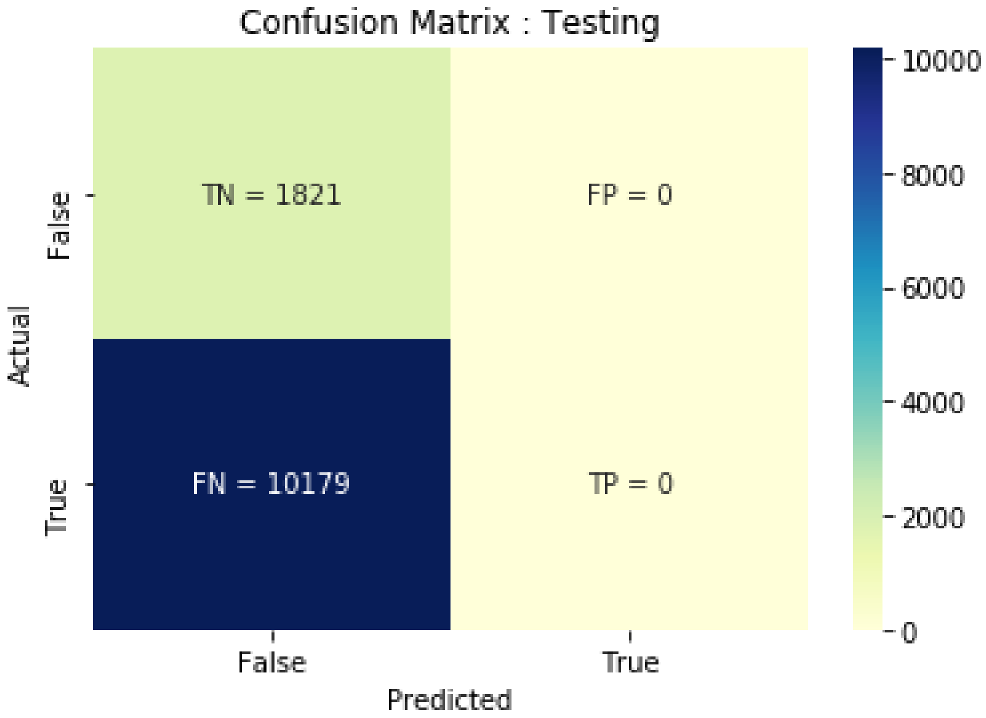
```
C=0.0001, train_score=0.5, test_score=0.5
C=0.01, train_score=0.5, test_score=0.5
C=1, train_score=0.5, test_score=0.5
C=100, train_score=0.5, test_score=0.5
C=10000, train_score=0.5, test_score=0.5
```



```
{'svm__C': 0.0001}
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.5
Area Under the Curve for Test :  0.5
```


Area Under ROC Curve
train AUC =0.5
train AUC =0.5
TPF : True Positive Ratio
FPR : False Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix : Training
TN = 4434
FP = 0
FN = 23566
TP = 0
Actual
False
True
Predicted
False
True
20000
16000
12000
8000
4000
0



8 [6] Conclusions

```
In [61]: from prettytable import PrettyTable
```

```
In [62]: # Please compare all your models using Prettytable library
         x = PrettyTable()

         x.field_names = ["Vectorizer", "Model", "Penalty", "Hyper parameter: 1/C", "Train AUC"
         [x.add_row(i) for i in prettytable_data]
         print(x)
```

```
+-------------+---------+---------+----------------------+--------------------+---------------
|  Vectorizer |  Model  | Penalty | Hyper parameter: 1/C |     Train AUC      |       Test AUC
+-------------+---------+---------+----------------------+--------------------+---------------
|     BOW     |   SVM   |    l2   |        0.001         | 0.967227654550752  |  0.94921858375
|    TFIDF    |   SVM   |    l2   |        0.0001        | 0.9661167505416866 | 0.958487601543
|  Avg-W2Vec  |   SVM   |    l2   |        0.0001        | 0.6553656030031227 | 0.649246101864
| TFIDF-W2Vec |   SVM   |    l1   |        0.0001        |        0.5         |        0.5
|     BOW     | SVM-rbf |  100.0  |         None         | 0.8159298603819459 | 0.791633332810
|    TFIDF    | SVM-rbf |  100.0  |         None         | 0.9348531017465855 | 0.911864554728
|  Avg-W2Vec  | SVM-rbf |   0.01  |         None         | 0.4802172889537464 | 0.483111286553
| TFIDF-W2Vec | SVM-rbf |  0.0001 |         None         |        0.5         |        0.5
+-------------+---------+---------+----------------------+--------------------+---------------
```