Assignment09 - Amazon Fine Food Reviews Analysis_RF

June 16, 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 [59]: %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 [60]: # using SQLite Table to read data.
         db_path = '/home/monodeepdas112/Datasets/amazon-fine-food-reviews/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 poin
         # you can change the number to any other number based on your computing power
```

```
        # filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT
        # for tsne assignment you can take 5k data points

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 10

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

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

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

```
Out[60]:    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 [61]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [62]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[62]:                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 [63]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[63]:               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 [64]: display['COUNT(*)'].sum()
```

```
Out[64]: 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 [65]: display= pd.read_sql_query("""
         SELECT *
         FROM Reviews
         WHERE Score != 3 AND UserId="AR5J8UI46CURR"
         ORDER BY ProductID
         """, con)
         display.head()
```

```
Out[65]:        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 [66]: #Sorting data according to ProductId in ascending order
         sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fa

In [67]: #Deduplication of entries
         final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
         final.shape

Out[67]: (87775, 10)

In [68]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

Out[68]: 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 [69]: display= pd.read_sql_query("""
         SELECT *
         FROM Reviews
         WHERE Score != 3 AND Id=44737 OR Id=64422
         ORDER BY ProductID
         """, con)

         display.head()
```

```
Out[69]:       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 [70]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [71]: #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[71]: 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 [72]: # printing some random reviews
         sent_0 = final['Text'].values[0]
         print(sent_0)
         print("="*50)

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

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

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

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

```
In [73]: # 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 [74]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         from bs4 import BeautifulSoup

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

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

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

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

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

```
In [75]: # 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 [76]: 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 [77]: #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 [78]: #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 [79]: # https://gist.github.com/sebleier/554280
         # we are removing the words from the stop words list: 'no', 'nor', 'not'
         # <br /><br /> ==> after the above steps, we are getting "br br"
         # we are including them into stop words list
         # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

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

```
In [80]: # 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 [00:32<00:00, 2708.34it/s]
```

```
In [81]: preprocessed_reviews[1500]
```

```
Out[81]: 'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

```
In [82]: ## Similartly you can do preprocessing for review summary also.
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [83]: # #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 [84]: # #bi-gram, tri-gram and n-gram

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

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

5.3 [4.3] TF-IDF

```
In [85]: # 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_nam
         # print('='*50)

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

5.4 [4.4] Word2Vec

```
In [86]: # # Using Google News Word2Vectors

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


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

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

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

         # elif want_to_use_google_w2v and is_your_ram_gt_16g:
         #     if os.path.isfile('GoogleNews-vectors-negative300.bin'):
         #         w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300
         #         print(w2v_model.wv.most_similar('great'))
         #         print(w2v_model.wv.most_similar('worst'))
         #     else:
         #         print("you don't have gogole's word2vec file, keep want_to_train_w2v = True
In [87]: # 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 [88]: # # average Word2Vec
         # # compute average word2vec for each review.
         # sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         # for sent in tqdm(list_of_sentance): # for each review/sentence
         #     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need
         #     cnt_words =0; # num of words with a valid vector in the sentence/review
         #     for word in sent: # for each word in a review/sentence
         #         if word in w2v_words:
         #             vec = w2v_model.wv[word]
         #             sent_vec += vec
         #             cnt_words += 1
         #     if cnt_words != 0:
         #         sent_vec /= cnt_words
         #     sent_vectors.append(sent_vec)
         # print(len(sent_vectors))
         # print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

```
In [89]: # # 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 [90]: # # TF-IDF weighted Word2Vec
         # tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfi

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

6 [5] Assignment 9: Random Forests

```
<li><strong>Apply Random Forests & GBDT on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vector
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)
    <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>Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.
    </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='3d_plot.JPG' width=500px> with X-axis as <strong>n_estimators</strong>, Y-axis as <s
```

```
        <p style="text-align:center;font-size:30px;color:red;"><strong>(or)</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='heat_map.JPG' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.hea
<li>You choose either of the plotting techniques out of 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and f
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

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

6.1 [5.1] Applying RF

```
In [91]: 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
         from sklearn.ensemble import RandomForestClassifier
         import pprint
         import os.path
         import pickle
         import math

         import xgboost as xgb
         from wordcloud import WordCloud, STOPWORDS
         from PIL import Image
         import urllib
         import requests

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

6.1.1 [5.0.0] Splitting up the Dataset into D_train and D_test

```
In [92]: num_data_points = 50000
```

```
In [93]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(preprocessed_reviews[:num_data
```

```
In [94]: prettytable_data = []
```

6.1.2 [5.0.1] Defining some functions to increase code reusability and readability

```
In [95]: '''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 N
                 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.id
                 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(X
                                  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 No
                 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 [96]: 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 [97]: '''Perform Simple Cross Validation'''
         def perform_hyperparameter_tuning(X, Y, vectorizer, results_path, retrain=False, W2V_
             #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 = {
                     'n_estimators' : [1, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200]
                 }

                 n_estimators = []

                 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 estimators in hyperparameters['n_estimators']:

    #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
        clf = RandomForestClassifier(n_estimators=estimators, class_weight='ba

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

        #Prediction
        train_results = clf.predict_proba(x_train)
        cv_results = clf.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))]

        print('CV iteration : n_estimators={0}, train_score={1}, test_score={2
          .format(estimators, train_score, test_score))

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

```
            n_estimators.append(estimators)

            print('CV : n_estimators={0}, train_score={1}, test_score={2}'
                  .format(estimators, sum(train_scores)/len(train_scores), sum(test_s

            train_scores = []
            test_scores = []

        # Creating a DataFrame from the saved data for visualization
        results_df = pd.DataFrame({
            'n_estimators' : n_estimators,
            '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 [98]: def analyse_results(df):
             # plotting error curves
             fig = plt.figure()
             ax = fig.gca()

             plt.plot([i for i in df.n_estimators.tolist()], df.test_score.tolist(), '-o', c='
             plt.plot([i for i in df.n_estimators.tolist()], df.train_score.tolist(), '-o', c=
             plt.grid(True)
             plt.xlabel('Hyperparameter : n_estimators')
             plt.ylabel('Area Under ROC Curve')
             plt.title('AUC ROC Curve for Random Forest Classifier')
             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 = {
                 'n_estimators':df.loc[ind_max, 'n_estimators']
             }
```

```
    return best_params
```

```
In [99]: def retrain_with_best_params(data, labels, best_params, vec_name, model_path, word2vec
             if(os.path.exists(model_path)):
                 print('Loading Model....')
                 with open(model_path, 'rb') as input_file:
                     clf = pickle.load(input_file)
             else:
                 clf = RandomForestClassifier(n_estimators=best_params['n_estimators'], class_w

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

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

```
In [100]: 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 [101]: 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))
```

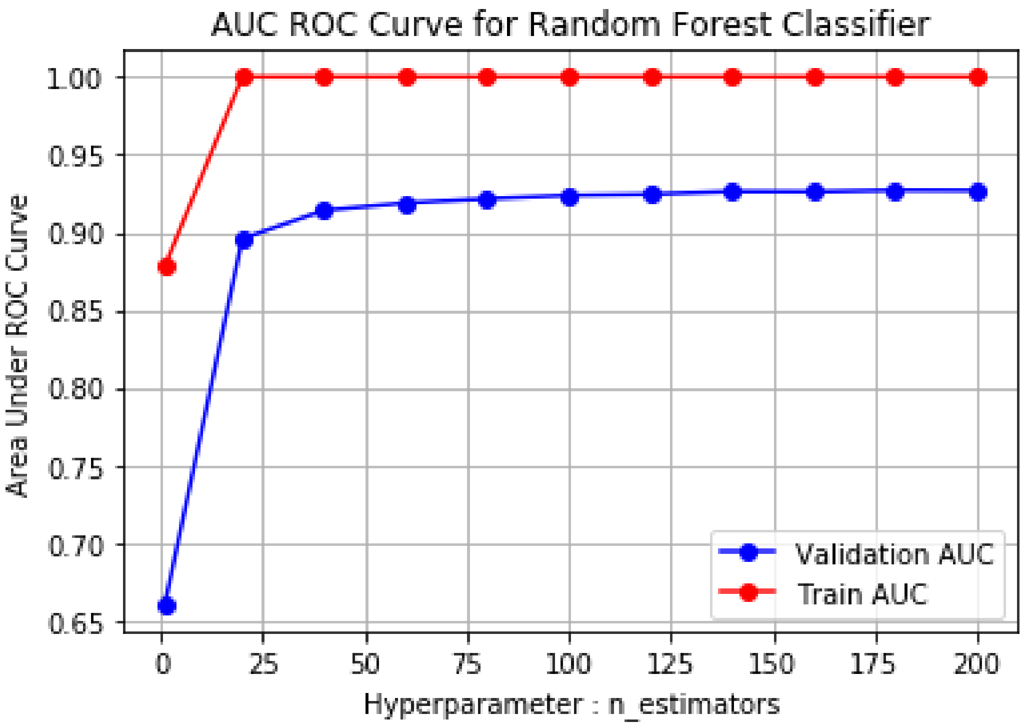
```
    #Finding out the ROC_AUC_SCORE
    train_roc_auc_score = roc_auc_score(np.array(Dy_train), train_score[:, 1])
    print('Area Under the Curve for Train : ', train_roc_auc_score)
    test_roc_auc_score = roc_auc_score(np.array(Dy_test), test_score[:, 1])
    print('Area Under the Curve for Test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

```
In [102]: def generate_wordcloud(words, mask):
              word_cloud = WordCloud(width = 512, height = 512, background_color='white', stop
              plt.figure(figsize=(10,8),facecolor = 'white', edgecolor='blue')
              plt.imshow(word_cloud)
              plt.axis('off')
              plt.tight_layout(pad=0)
              plt.show()
```

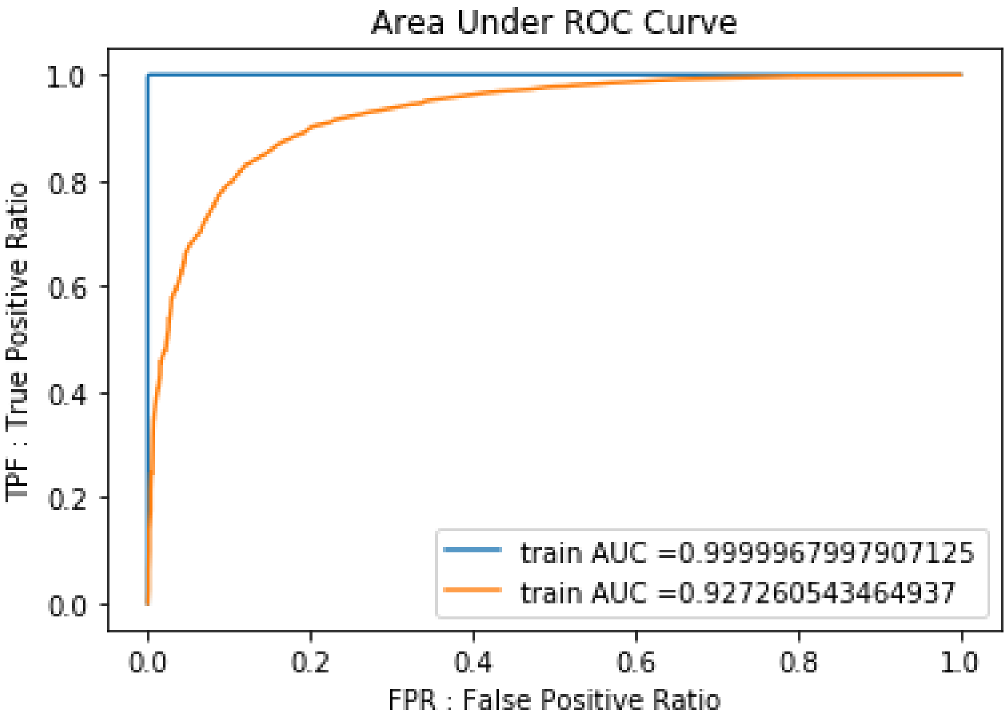
6.1.3 [5.1.1] Applying Random Forests on BOW, SET 1

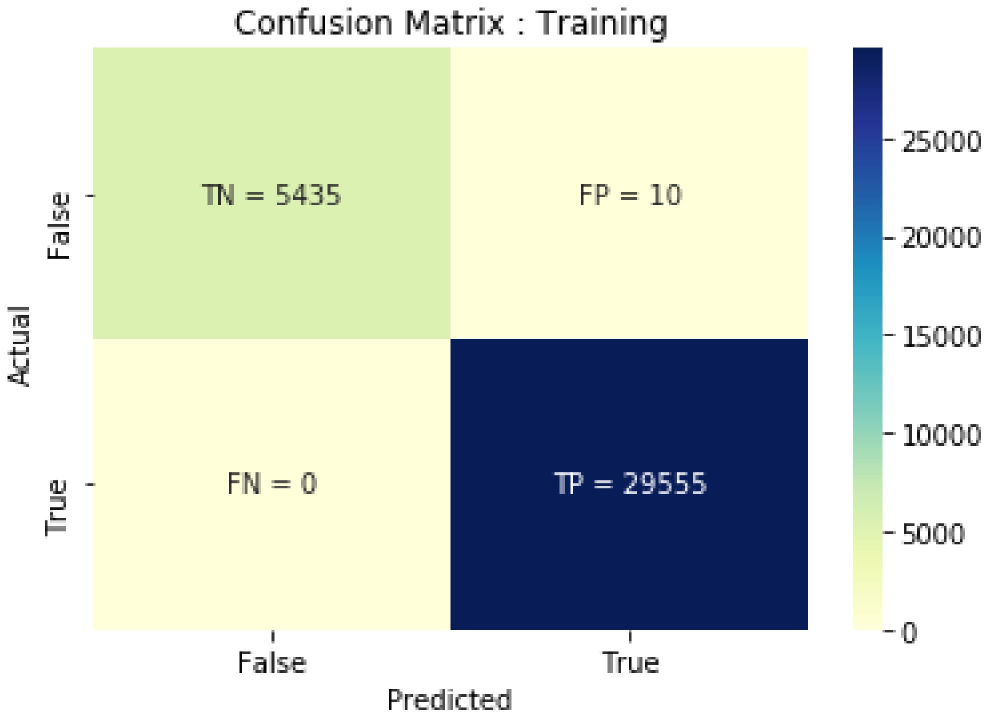
```
In [103]: # Please write all the code with proper documentation
          csv_path = 'saved_models/Assignment9/BOW_rf_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='BOW',
                                                     results_path=csv_path, 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/Assignment9/{0}_rf.pkl'.format('BOW')
          clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'BOW', model_pat

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

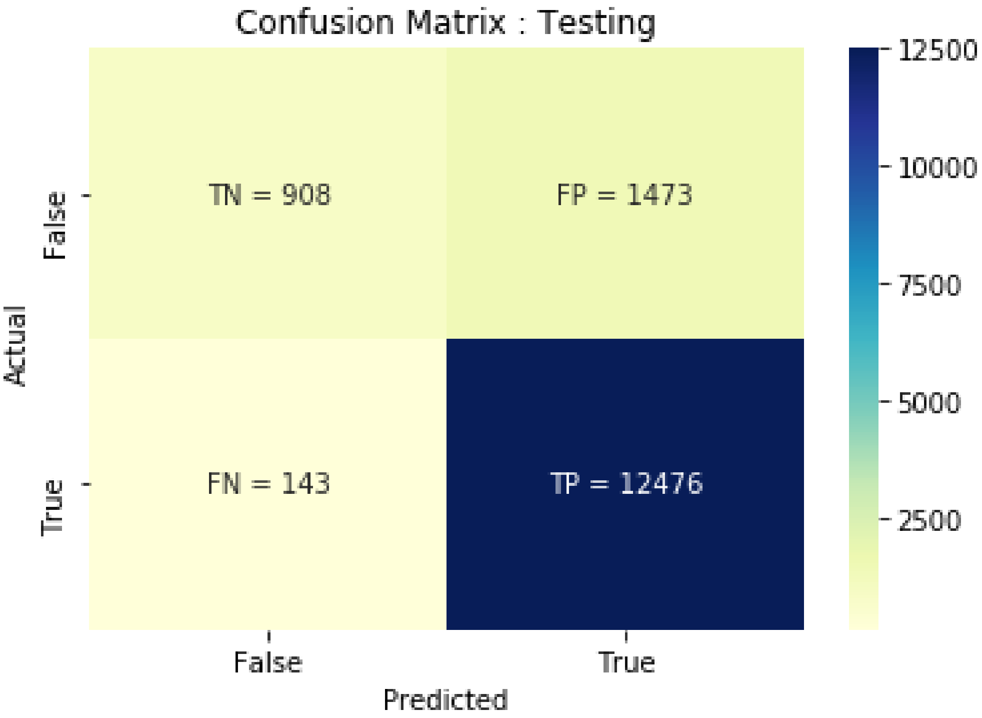
```
vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='BOW')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

# appending the data results
prettytable_data.append(['BOW', 'Random Forests', best_parameters['n_estimators'], t
```



```
{'n_estimators': 180}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9999967997907125
Area Under the Curve for Test :  0.927260543464937
```


Area Under ROC Curve
TPF : True Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
train AUC =0.9999967997907125
train AUC =0.927260543464937
0.0
0.2
0.4
0.6
0.8
1.0
FPR : False Positive Ratio


Confusion Matrix : Training
TN = 5435
FP = 10
FN = 0
TP = 29555
Actual
False
True
False
True
Predicted
25000
20000
15000
10000
5000
0

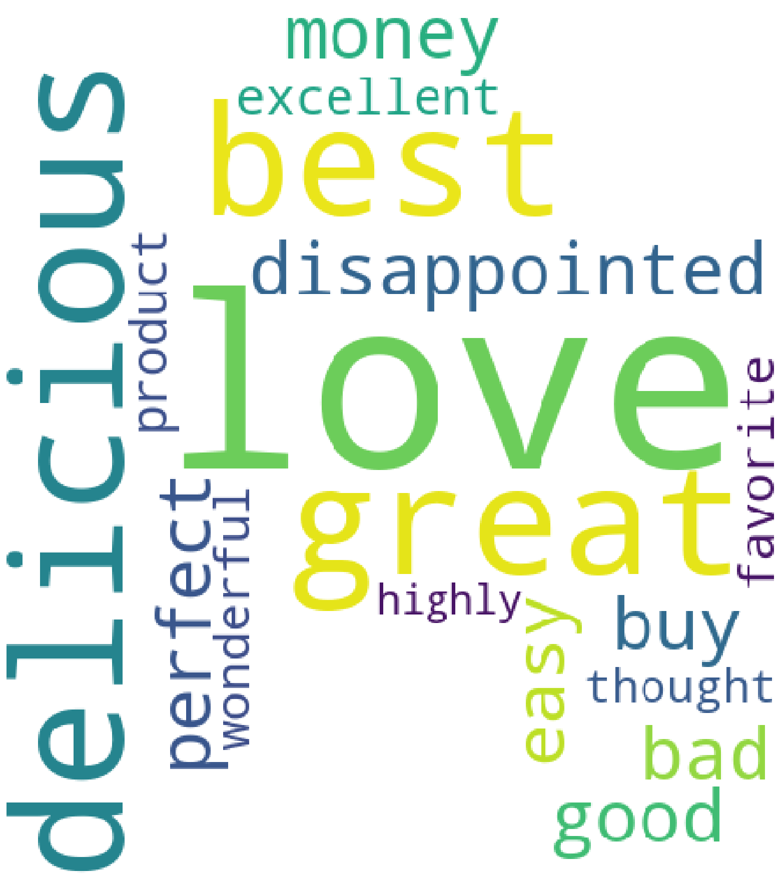


6.1.4 [5.1.2] Wordcloud of top 20 important features from SET 1

6.1.5 Reference article

https://blog.goodaudience.com/how-to-generate-a-word-cloud-of-any-shape-in-python-7bce27a55f6e

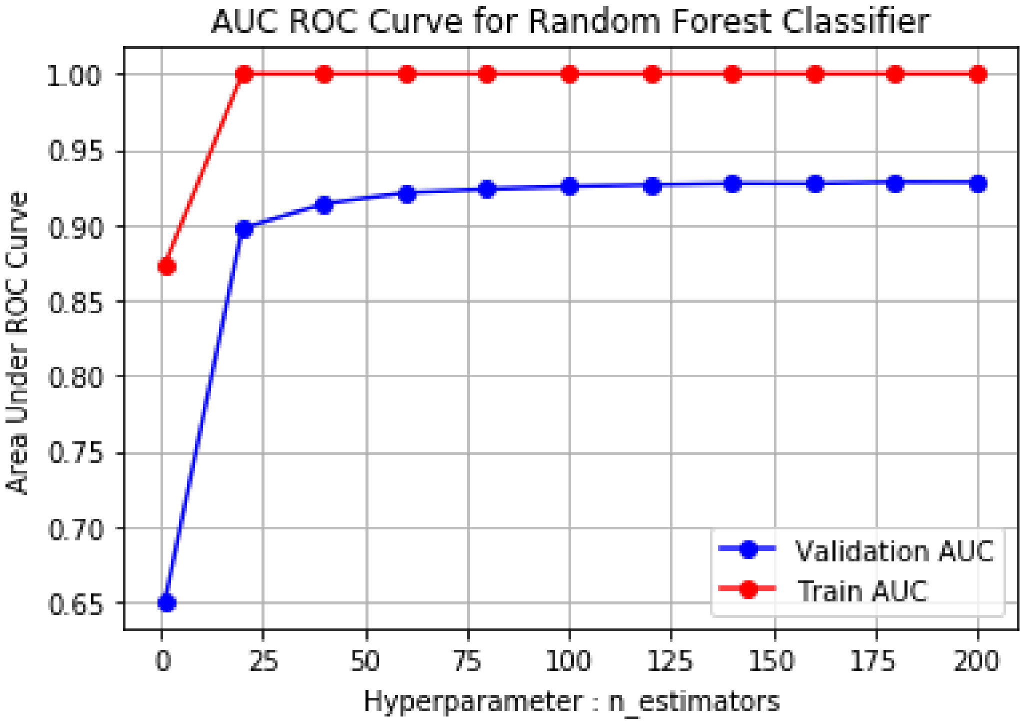
```
In [104]: # Please write all the code with proper documentation
          feature_names = vectorizer_obj.get_feature_names()
          feature_importances = clf.feature_importances_
          features_with_importances = [(feature_names[i], feature_importances[i]) for i in rang

          # sorting the features with importances
          features_with_importances.sort(key = lambda x: abs(x[1]), reverse=True)
          important_features = features_with_importances[:20]

          mask = np.array(Image.open(requests.get('http://www.clker.com/cliparts/O/i/x/Y/q/P/ye

          words = ' '.join([important_features[i][0] for i in range(20)])
          generate_wordcloud(words, mask)
```



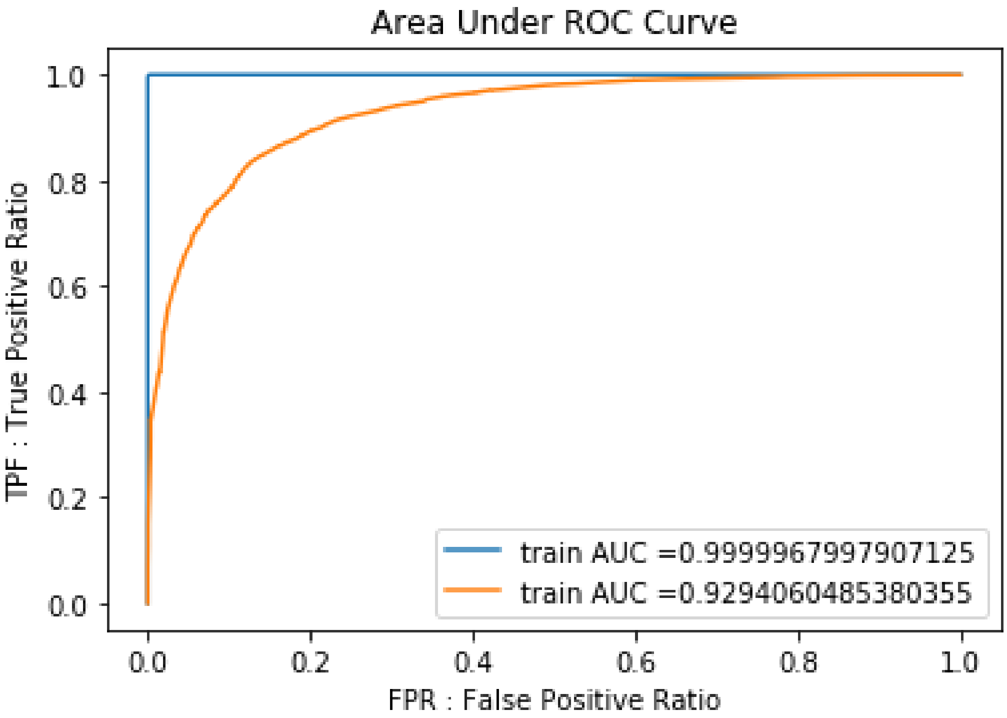
6.1.6 [5.1.3] Applying Random Forests on TFIDF, SET 2

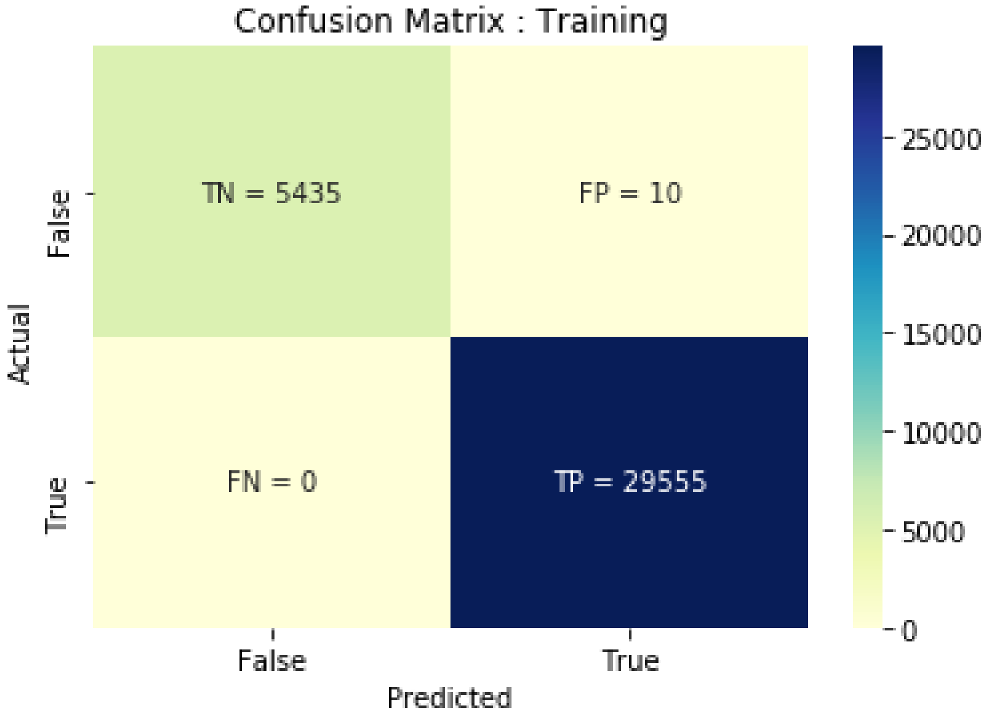
```
In [105]: # Please write all the code with proper documentation
          csv_path = 'saved_models/Assignment9/TFIDF_rf_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF
                                                     results_path=csv_path, 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/Assignment9/{0}_rf.pkl'.format('TFIDF')
          clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF', model_pa

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

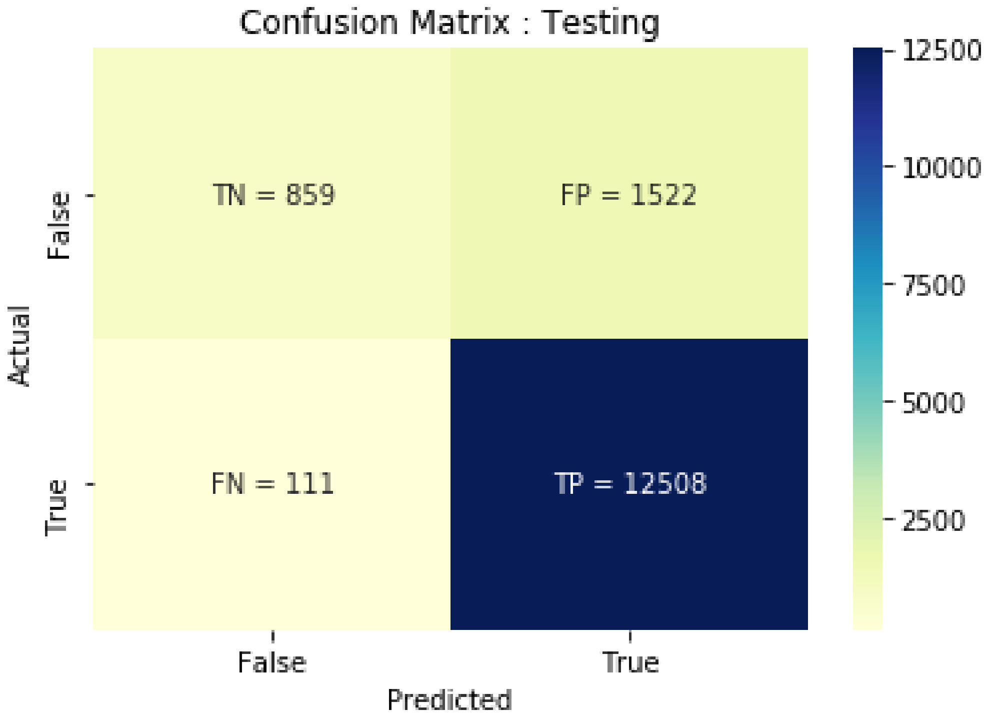
```
vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='TFIDF')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

# appending the data results
prettytable_data.append(['TFIDF', 'Random Forests', best_parameters['n_estimators'],
```



```
{'n_estimators': 180}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9999967997907125
Area Under the Curve for Test :  0.9294060485380355
```

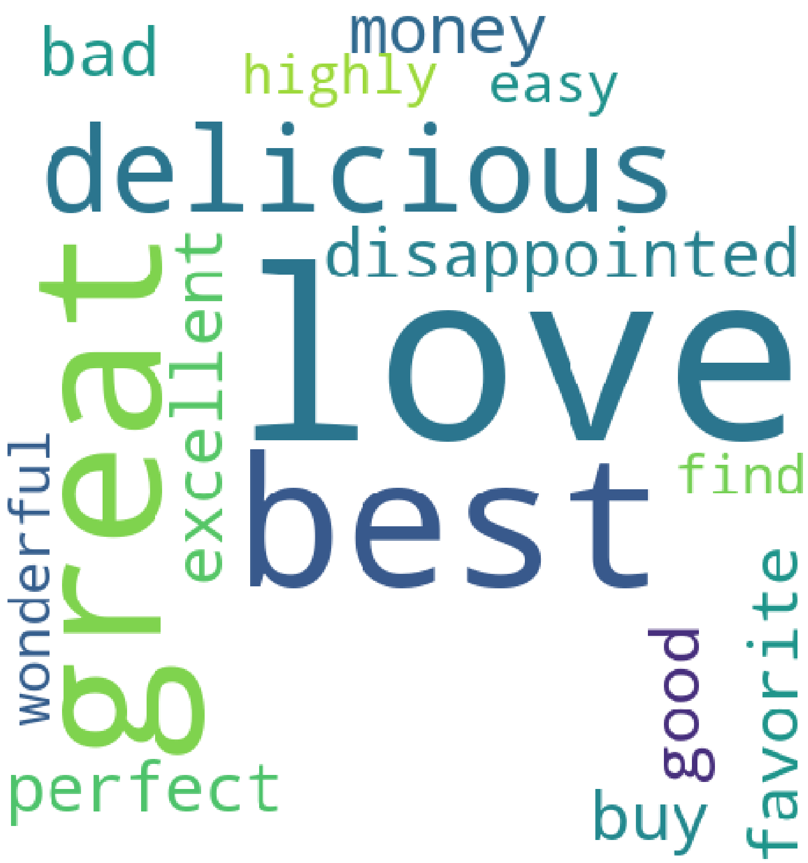

Area Under ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
TPF : True Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0
FPR : False Positive Ratio
train AUC =0.9999967997907125
train AUC =0.9294060485380355


Confusion Matrix : Training
TN = 5435
FP = 10
FN = 0
TP = 29555
False
True
Actual
False
True
Predicted
25000
20000
15000
10000
5000
0



6.1.7 [5.1.4] Wordcloud of top 20 important features from SET 2

```
In [106]: # Please write all the code with proper documentation
          feature_names = vectorizer_obj.get_feature_names()
          feature_importances = clf.feature_importances_
          features_with_importances = [(feature_names[i], feature_importances[i]) for i in rang

          # sorting the features with importances
          features_with_importances.sort(key = lambda x: abs(x[1]), reverse=True)
          important_features = features_with_importances[:20]

          mask = np.array(Image.open(requests.get('http://www.clker.com/cliparts/O/i/x/Y/q/P/ye

          words = ' '.join([important_features[i][0] for i in range(20)])
          generate_wordcloud(words, mask)
```



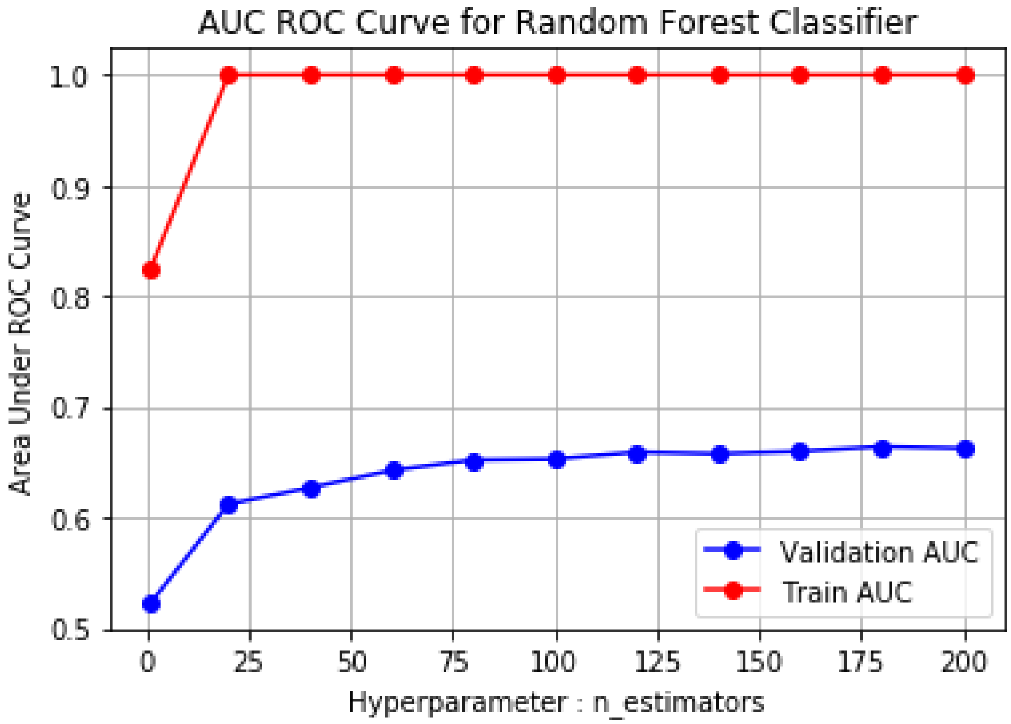
6.2 Preparing/Training Google Word2Vec

```
In [107]: 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,
```

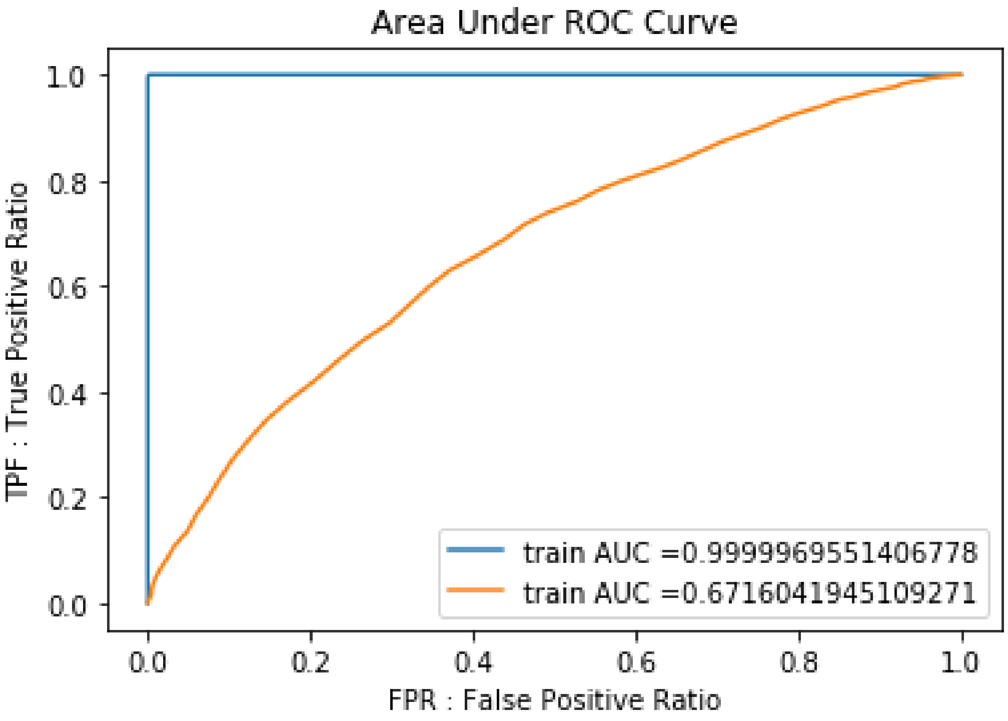
```
Preparing to load pre-trained Word2Vec model !
Successfully loaded model into memory !!
Words similar to "similar" :  [('terrific', 0.798933207988739), ('fantastic', 0.79352122545242
Words similar to "worst" :  [('Worst', 0.6146091222763062), ('weakest', 0.6143776774406433), (
```

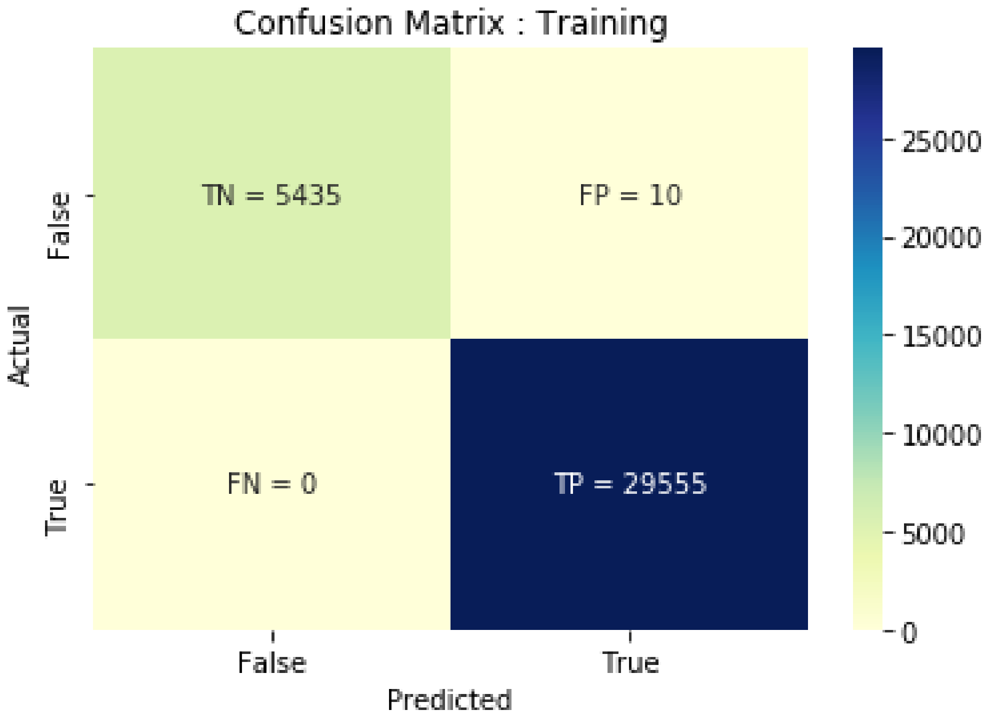
6.2.1 [5.1.5] Applying Random Forests on AVG W2V, SET 3

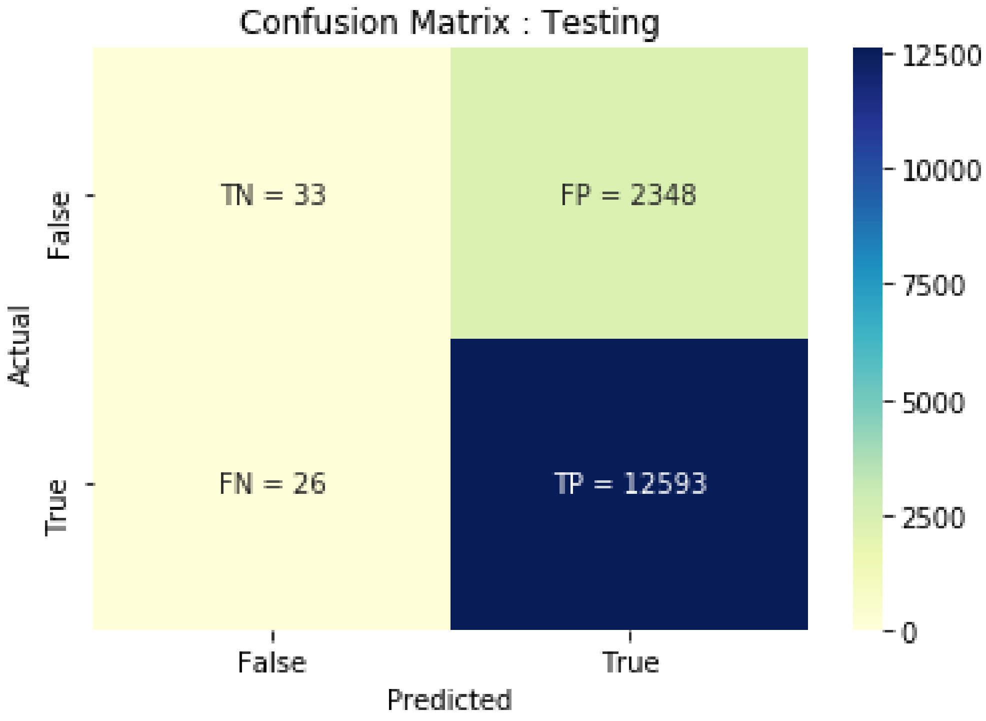
```
In [108]: # Please write all the code with proper documentation
          csv_path = 'saved_models/Assignment9/Avg-W2Vec_rf_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='Avg-W2
                                                     results_path=csv_path, 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/Assignment9/{0}_rf.pkl'.format('Avg-W2Vec')
          clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-W2Vec', mode

          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(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

          # appending the data results
          prettytable_data.append(['Avg-W2Vec', 'Random Forests', best_parameters['n_estimator
```



```
{'n_estimators': 180}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9999969551406778
Area Under the Curve for Test :  0.6716041945109271
```

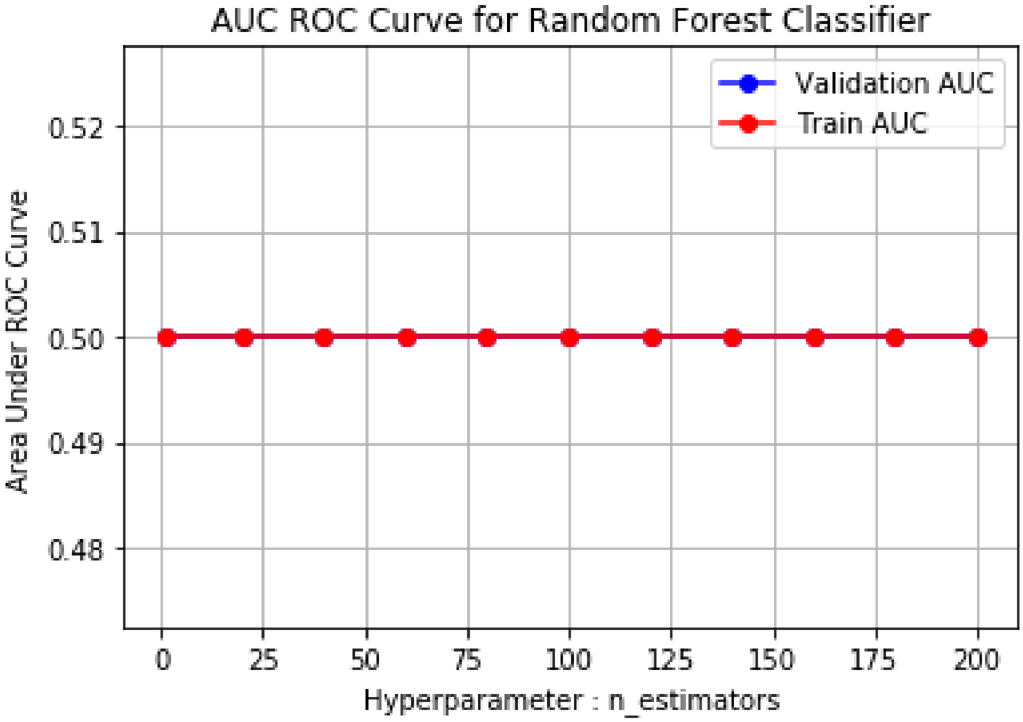

Area Under ROC Curve
TPF : True Positive Ratio
FPR : False Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
train AUC =0.9999969551406778
train AUC =0.6716041945109271


Confusion Matrix : Training
TN = 5435
FP = 10
FN = 0
TP = 29555
Actual
False
True
Predicted
False
True
25000
20000
15000
10000
5000
0

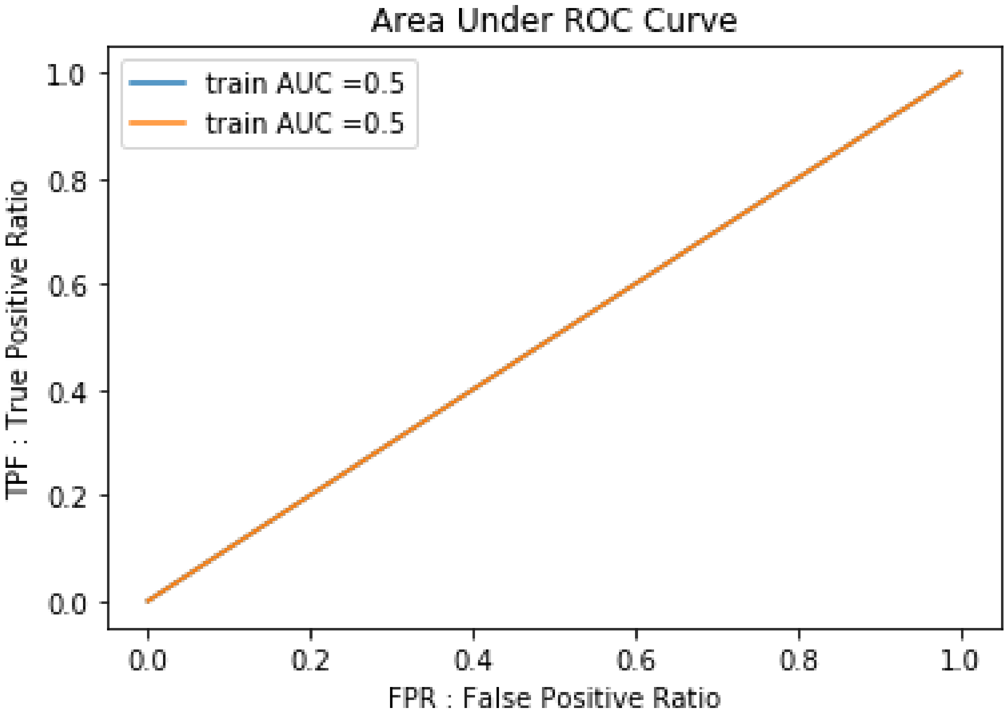


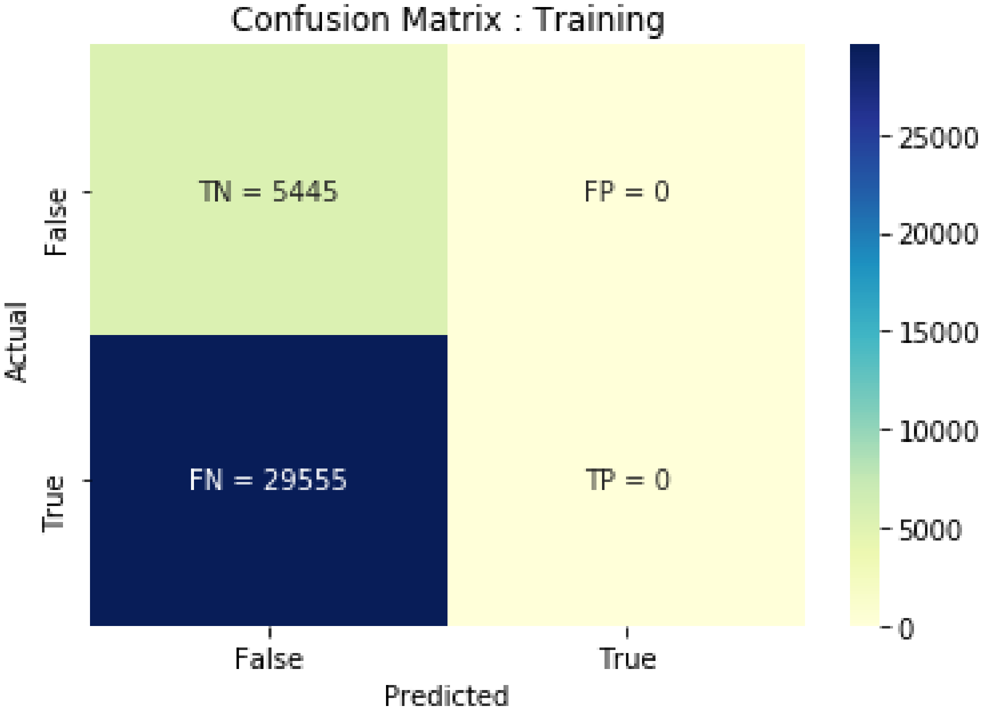
6.2.2 [5.1.6] Applying Random Forests on TFIDF W2V, SET 4

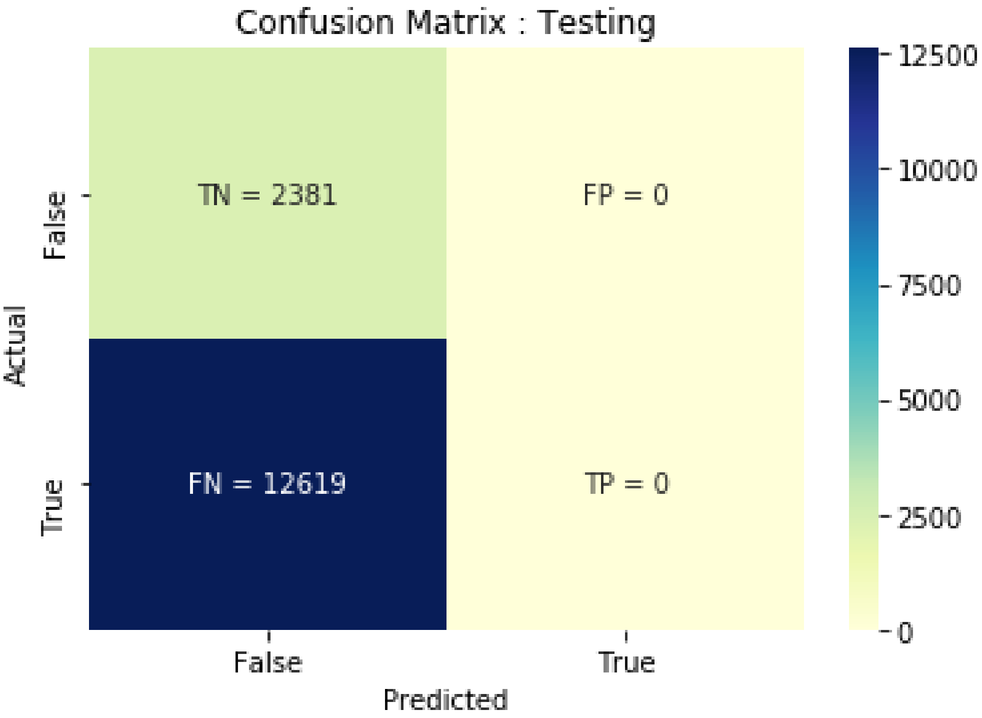
```
In [109]: # Please write all the code with proper documentation
          csv_path = 'saved_models/Assignment9/TFIDF-W2Vec-W2Vec_rf_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF-
                                                      results_path=csv_path, 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/Assignment9/{0}_rf.pkl'.format('TFIDF-W2Vec')
          clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF-W2Vec', mo

          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(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

          # appending the data results
          prettytable_data.append(['TFIDF-W2Vec', 'Random Forests', best_parameters['n_estimato
```



```
{'n_estimators': 1}
Loading 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 = 5445
FP = 0
FN = 29555
TP = 0
Actual
False
True
False
True
Predicted
25000
20000
15000
10000
5000
0



6.3 [5.2] Applying GBDT using XGBOOST

```
In [110]: '''Perform Simple Cross Validation'''
          def perform_hyperparameter_tuning(X, Y, vectorizer, results_path, retrain=False, W2V_
              #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 = {
                      'n_estimators' : [1, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200]
                  }

                  n_estimators = []
```

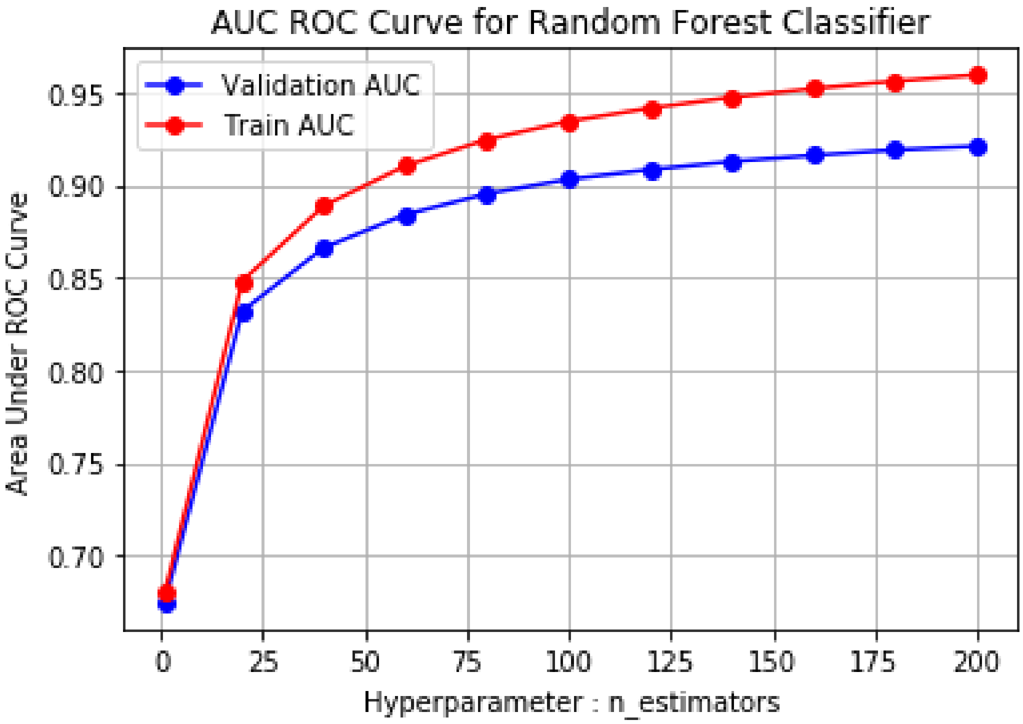
```
train_scores = []
test_scores = []

train_mean_score = []
test_mean_score = []

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

for estimators in hyperparameters['n_estimators']:

    #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
        clf = xgb.XGBClassifier(max_depth=5, n_estimators=estimators, colsamp

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

        #Prediction
        train_results = clf.predict_proba(x_train)
        cv_results = clf.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))]
```

```
                print('CV iteration : n_estimators={0}, train_score={1}, test_score=
                  .format(estimators, train_score, test_score))

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

            n_estimators.append(estimators)

            print('CV : n_estimators={0}, train_score={1}, test_score={2}'
                  .format(estimators, sum(train_scores)/len(train_scores), sum(test_

            train_scores = []
            test_scores = []

        # Creating a DataFrame from the saved data for visualization
        results_df = pd.DataFrame({
            'n_estimators' : n_estimators,
            '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 
        return results_df
```

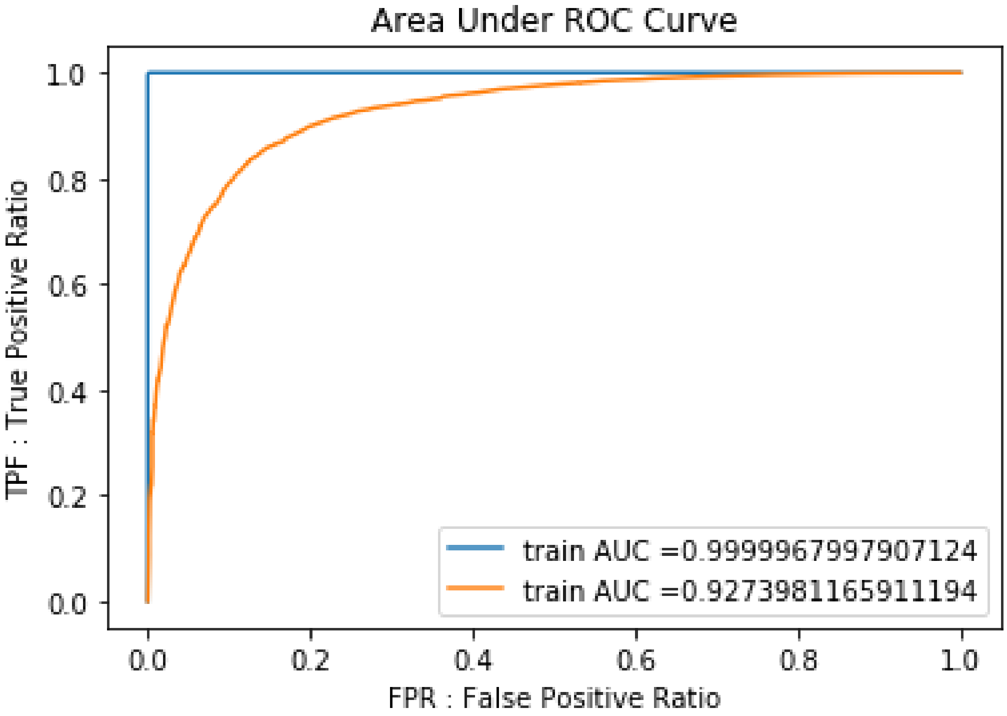
6.3.1 [5.2.1] Applying XGBOOST on BOW, SET 1

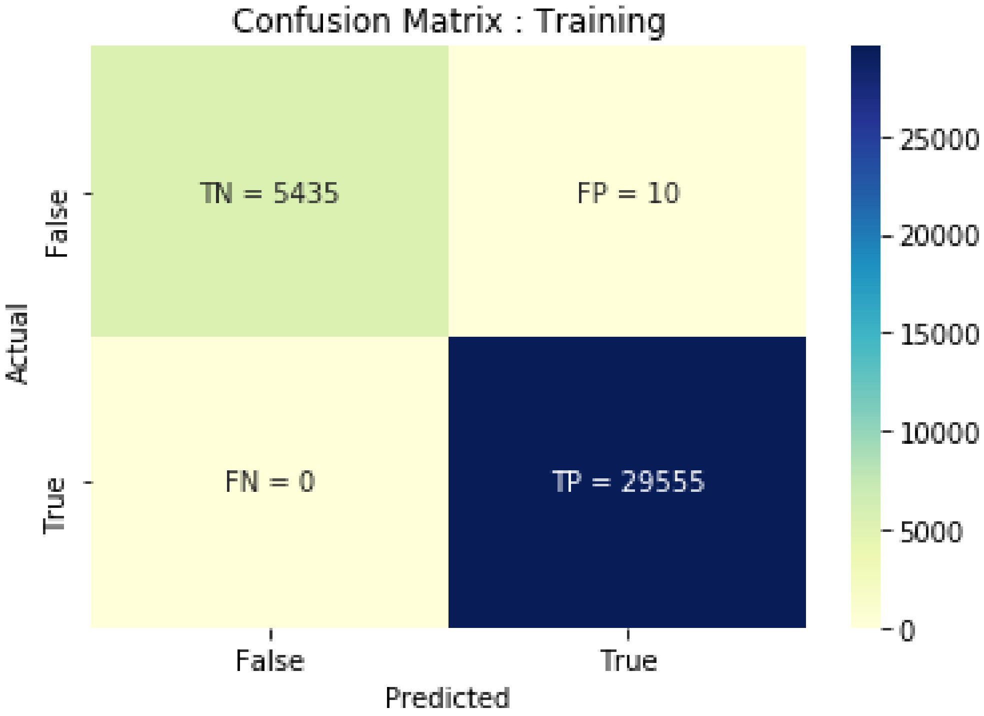
In [111]:

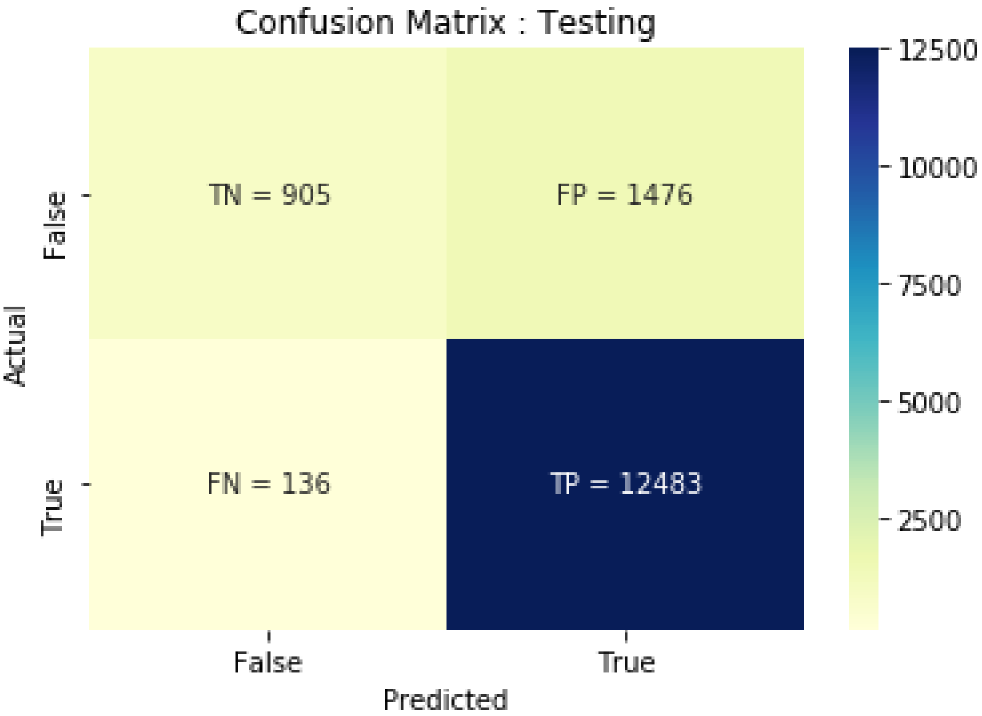
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment9/BOW_rf_xgb_results.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='BOW',
                                           results_path=csv_path, retrain=False, W2V_
# Analysing best parameters
best_parameters = analyse_results(cv_results)
pprint.pprint(best_parameters)

# retraining the model with best parameters
model_path = 'saved_models/Assignment9/{0}_rf_xgb.pkl'.format('BOW')
clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'BOW', model_pat

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(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra
# appending the data results
prettytable_data.append(['BOW', 'Random Forests XGB', best_parameters['n_estimators']
```



```
{'n_estimators': 200}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9999967997907124
Area Under the Curve for Test :  0.9273981165911194
```

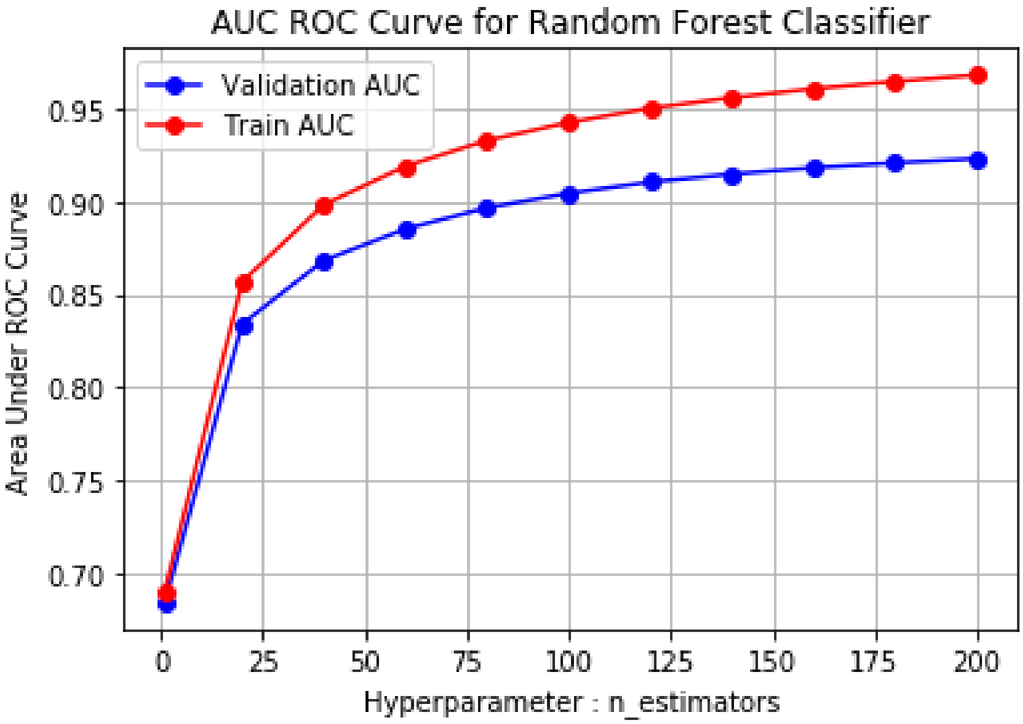

Area Under ROC Curve
TPF : True Positive Ratio
FPR : False Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
train AUC =0.9999967997907124
train AUC =0.9273981165911194


Confusion Matrix : Training
TN = 5435
FP = 10
FN = 0
TP = 29555
Actual
False
True
Predicted
False
True
25000
20000
15000
10000
5000
0

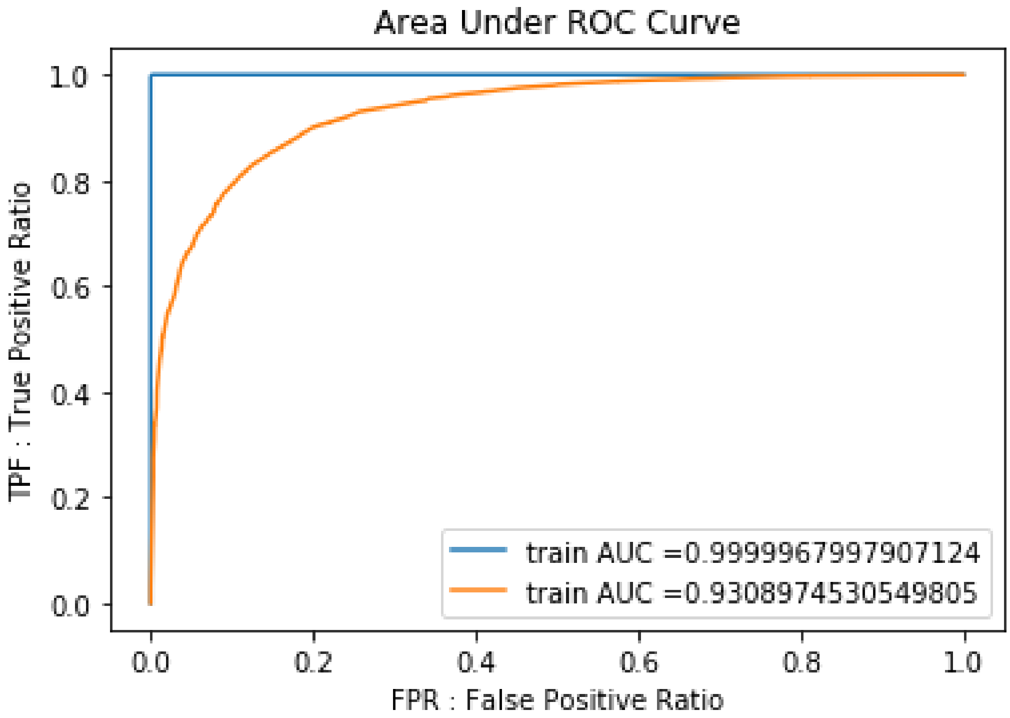


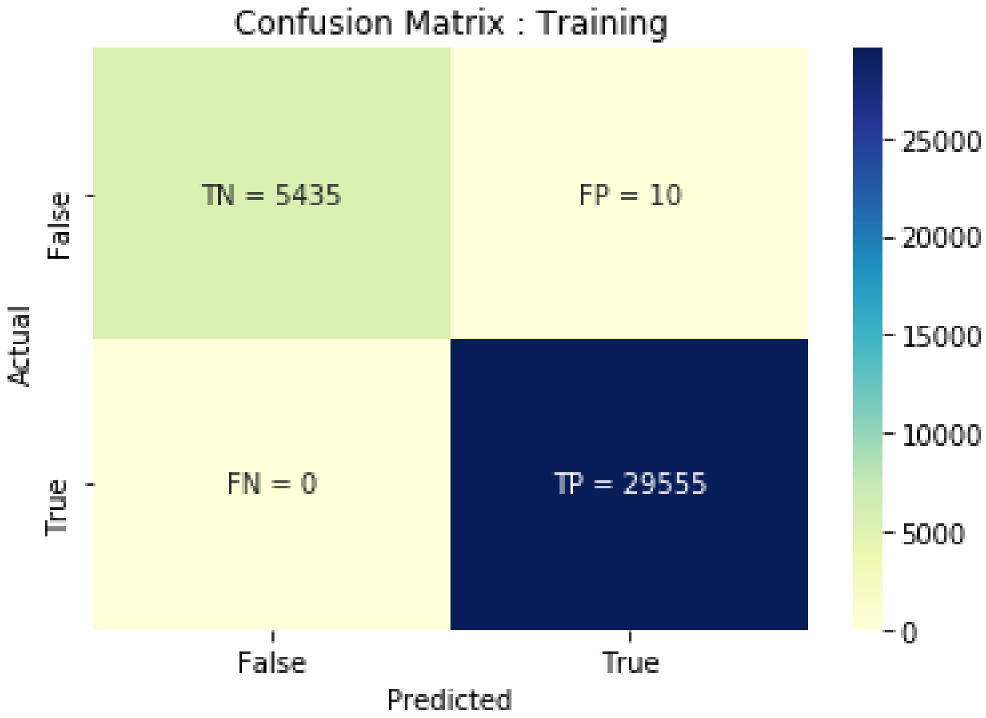
6.3.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

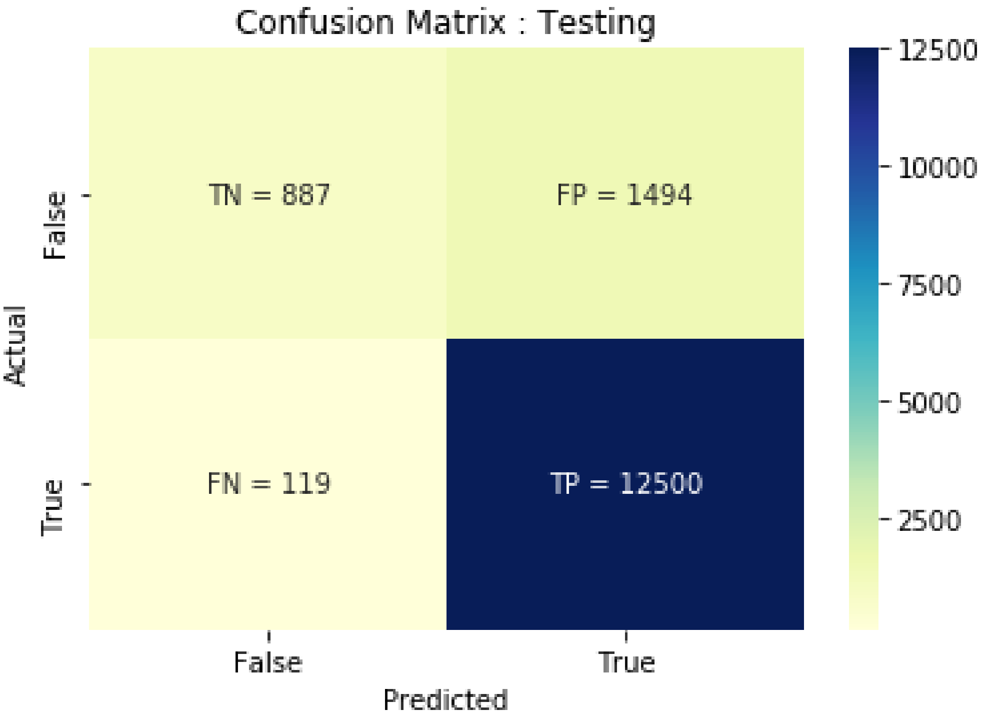
```
In [112]: # Please write all the code with proper documentation
          csv_path = 'saved_models/Assignment9/TFIDF_rf_xgb_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF
                                                     results_path=csv_path, 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/Assignment9/{0}_rf_xgb.pkl'.format('TFIDF')
          clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF', model_pa

          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(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

          # appending the data results
          prettytable_data.append(['TFIDF', 'Random Forests XGB', best_parameters['n_estimators
```



```
{'n_estimators': 200}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9999967997907124
Area Under the Curve for Test :  0.9308974530549805
```

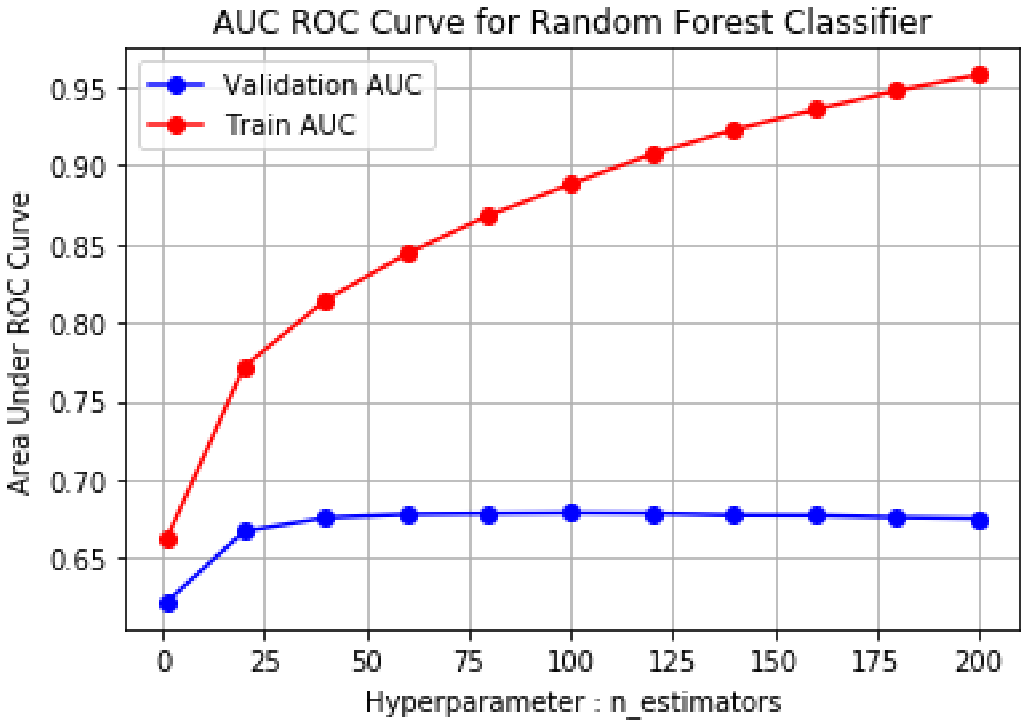

Area Under ROC Curve
TPF : True Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0
train AUC =0.9999967997907124
train AUC =0.9308974530549805
0.0
0.2
0.4
0.6
0.8
1.0
FPR : False Positive Ratio


Confusion Matrix : Training
Actual
False
True
TN = 5435
FP = 10
FN = 0
TP = 29555
False
True
Predicted
25000
20000
15000
10000
5000
0

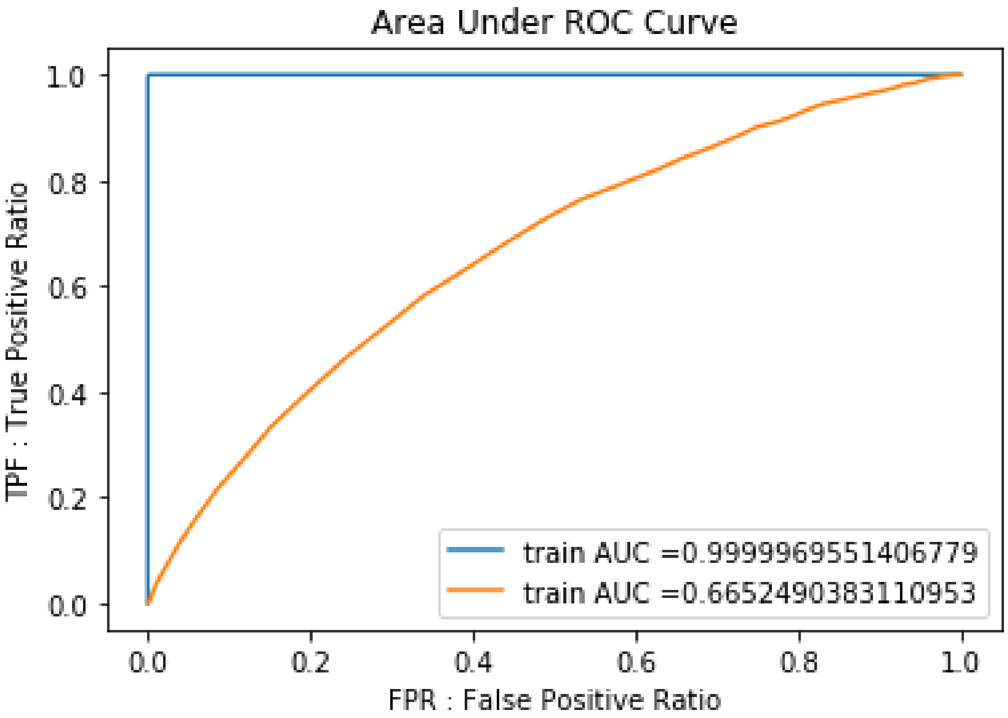


6.3.3 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

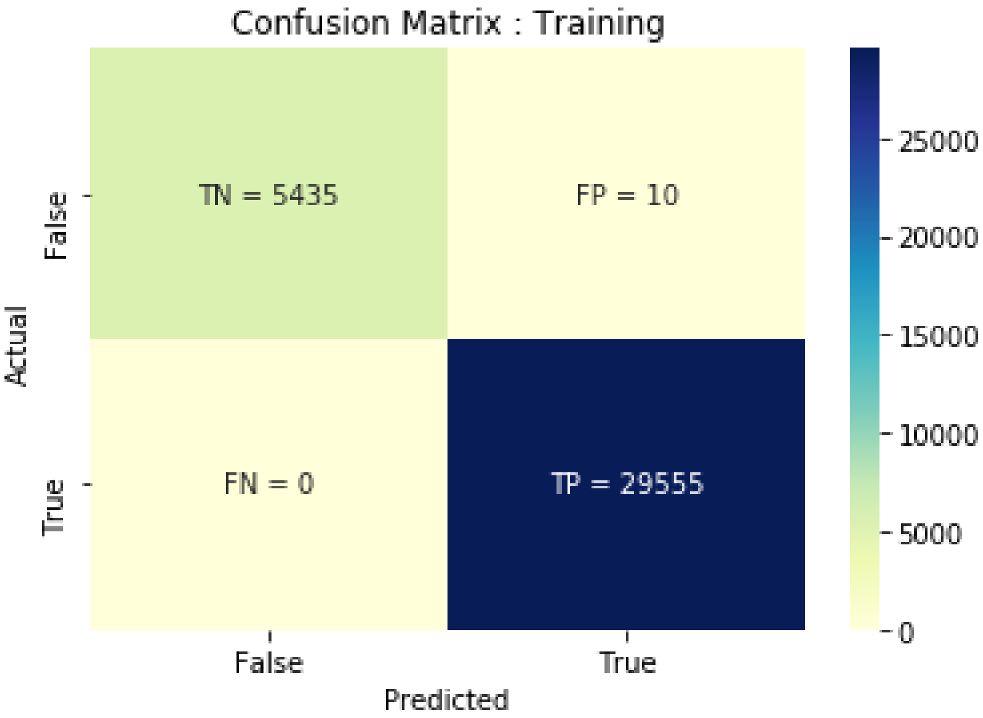
```
In [113]: # Please write all the code with proper documentation
          csv_path = 'saved_models/Assignment9/Avg-W2Vec_rf_xgb_results.csv'
          cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='Avg-W
                                                     results_path=csv_path, 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/Assignment9/{0}_rf_xgb.pkl'.format('Avg-W2Vec')
          clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-W2Vec', mod

          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(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

          # appending the data results
          prettytable_data.append(['Avg-W2Vec', 'Random Forests XGB', best_parameters['n_estima
```



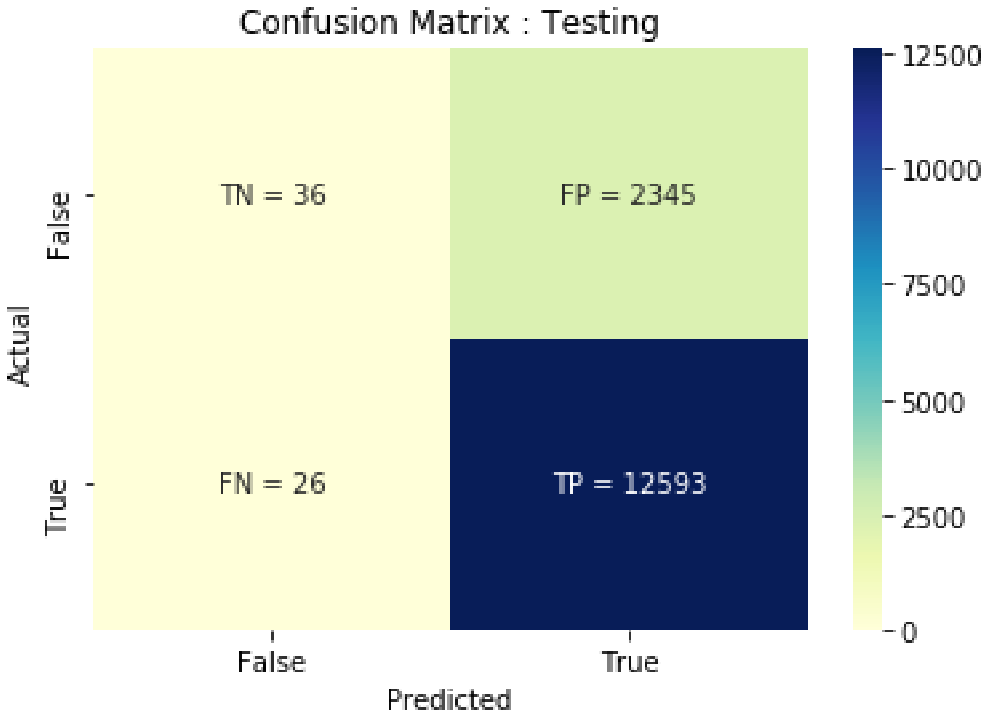
```
{'n_estimators': 100}
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9999969551406779
Area Under the Curve for Test :  0.6652490383110953
```



Area Under ROC Curve
TPF : True Positive Ratio
FPR : False Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0
train AUC =0.9999969551406779
train AUC =0.6652490383110953

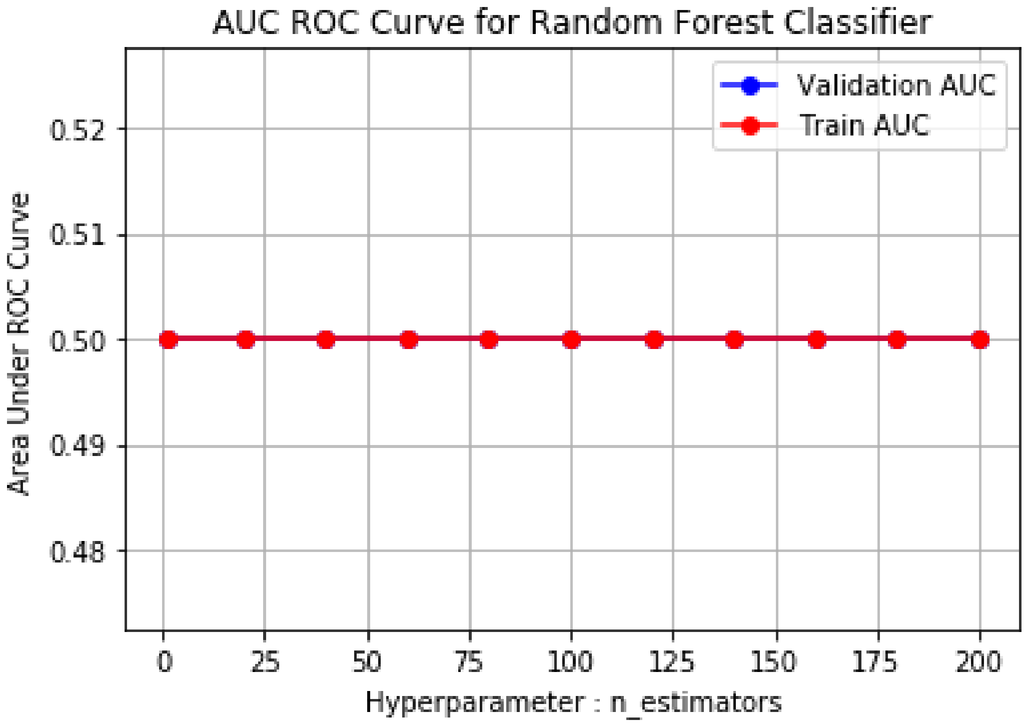


Confusion Matrix : Training
TN = 5435
FP = 10
FN = 0
TP = 29555
Actual
False
True
Predicted
False
True
25000
20000
15000
10000
5000
0

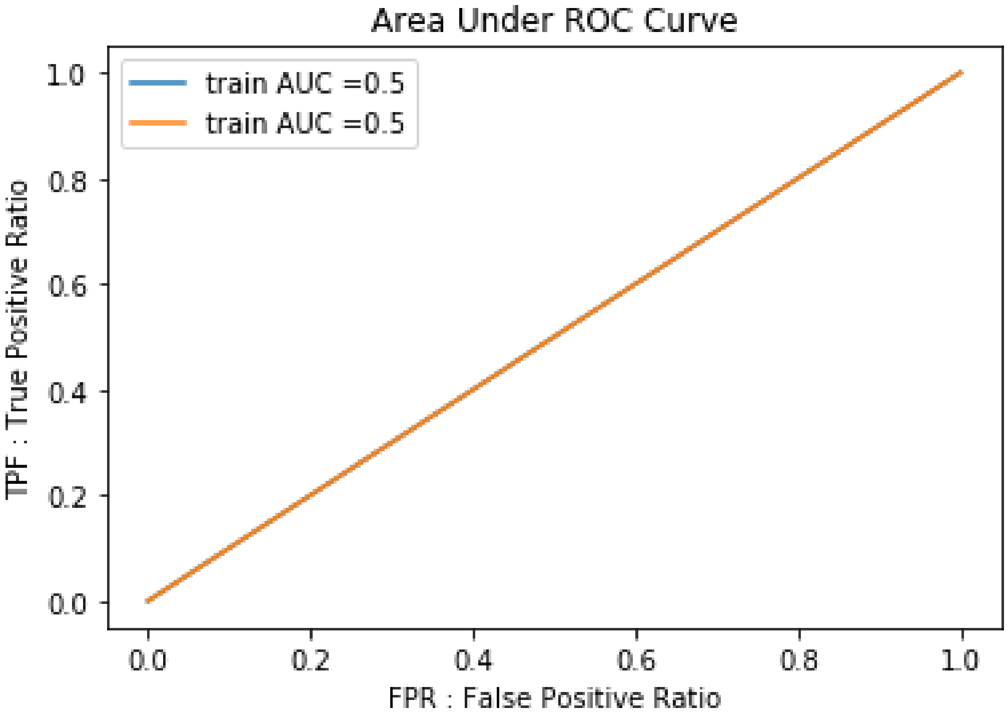


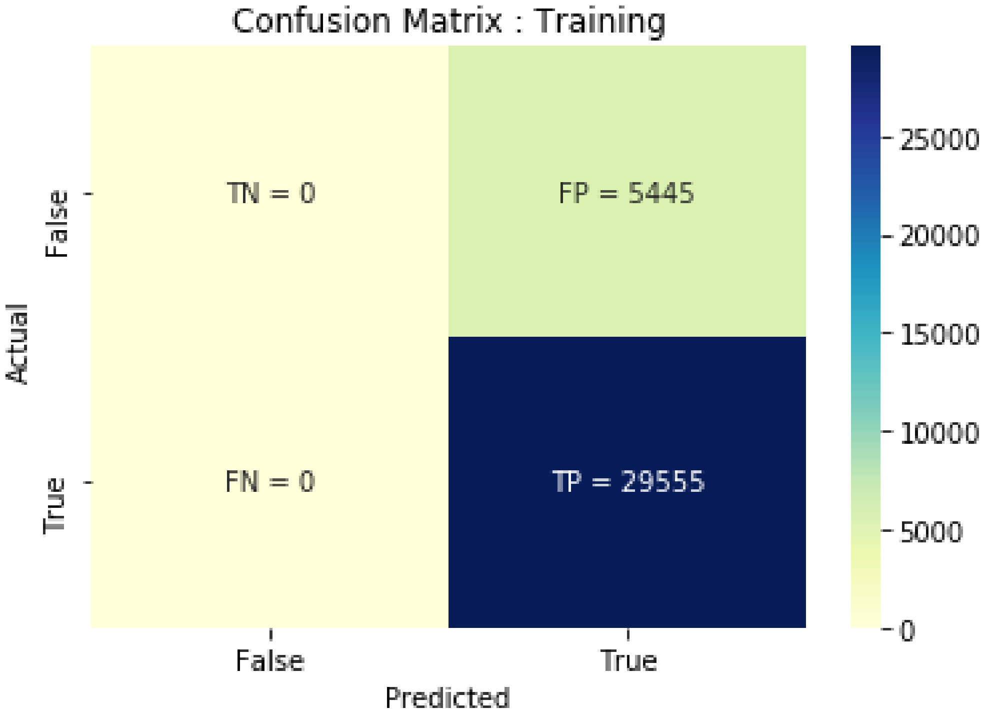
6.3.4 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

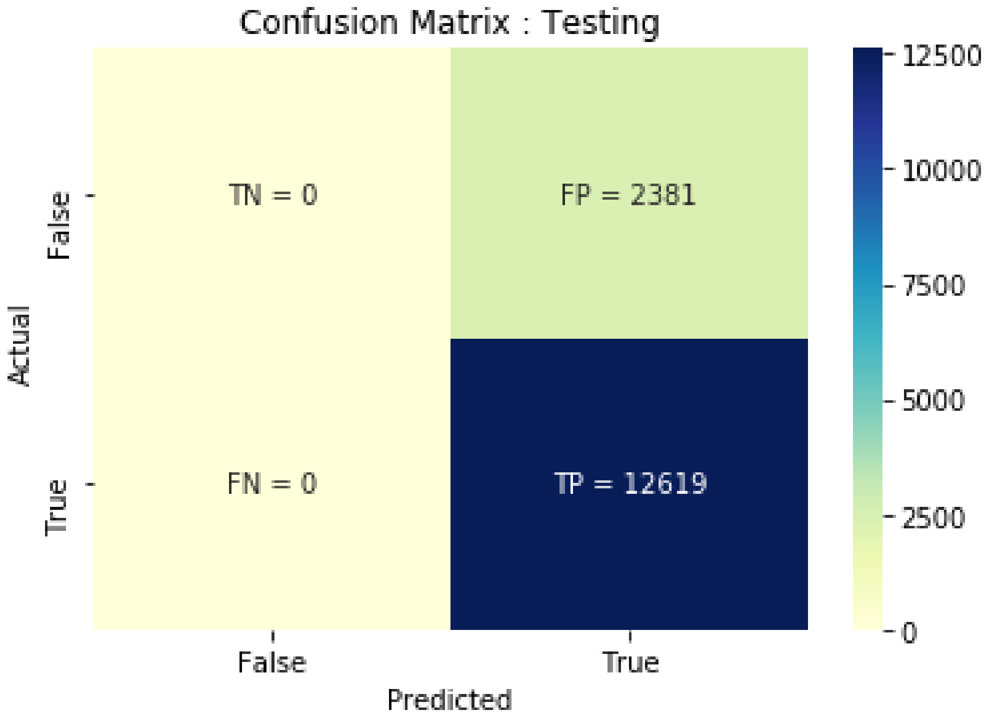
In [114]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment9/TFIDF-W2Vec_rf_xgb_results.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF-
                                           results_path=csv_path, retrain=False, W2V_
# Analysing best parameters
best_parameters = analyse_results(cv_results)
pprint.pprint(best_parameters)

# retraining the model with best parameters
model_path = 'saved_models/Assignment9/{0}_rf_xgb.pkl'.format('TFIDF-W2Vec')
clf = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF-W2Vec', mo

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(clf, vectorizer_obj, Dx_train, Dx_test, Dy_tra

# appending the data results
prettytable_data.append(['TFIDF-W2Vec', 'Random Forests XGB', best_parameters['n_esti
```



```
{'n_estimators': 1}
Loading 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 = 0
FP = 5445
FN = 0
TP = 29555
Actual
False
True
Predicted
False
True
25000
20000
15000
10000
5000
0



7 [6] Conclusions

```
In [119]: from prettytable import PrettyTable
          # Please compare all your models using Prettytable library
          x = PrettyTable()

          x.field_names = ["Vectorizer", "Model", "n_estimators", "Train AUC", "Test AUC"]
          [x.add_row(i) for i in prettytable_data]
          print(x)
```

```
+-------------+--------------------+--------------+--------------------+--------------------+
|  Vectorizer |       Model        | n_estimators |     Train AUC      |      Test AUC      |
+-------------+--------------------+--------------+--------------------+--------------------+
|     BOW     |   Random Forests   |     180      | 0.9999967997907125 | 0.927260543464937  |
|    TFIDF    |   Random Forests   |     180      | 0.9999967997907125 | 0.9294060485380355 |
|  Avg-W2Vec  |   Random Forests   |     180      | 0.9999969551406778 | 0.6716041945109271 |
| TFIDF-W2Vec |   Random Forests   |      1       |        0.5         |        0.5         |
|     BOW     | Random Forests XGB |     200      | 0.9999967997907124 | 0.9273981165911194 |
|    TFIDF    | Random Forests XGB |     200      | 0.9999967997907124 | 0.9308974530549805 |
|  Avg-W2Vec  | Random Forests XGB |     100      | 0.9999969551406779 | 0.6652490383110953 |
| TFIDF-W2Vec | Random Forests XGB |      1       |        0.5         |        0.5         |
+-------------+--------------------+--------------+--------------------+--------------------+
```