09 Amazon Fine Food Reviews Analysis_RF

April 28, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

from sklearn.metrics import roc_auc_score
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
!pip install -q PTable
from prettytable import PrettyTable
!pip3 install xgboost
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import ParameterGrid
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
!pip3 install wordcloud
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
!pip install -q scikit-plot
import scikitplot as skplt
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
#for finding nonzero elements in sparse matrix
from scipy.sparse import find
#for f1_Score
from sklearn.metrics import f1_score
#for displaying time
from datetime import datetime
```

```
#for roc curve
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy.sparse import coo_matrix, hstack
from scipy import interp
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from tqdm import tqdm
import os
import xgboost as xgb
```

```
You are using pip version 19.0.3, however version 19.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.


Requirement already satisfied: xgboost in c:\users\shubh\anaconda3\lib\site-packages (0.82)
Requirement already satisfied: scipy in c:\users\shubh\anaconda3\lib\site-packages (from xgboos
Requirement already satisfied: numpy in c:\users\shubh\anaconda3\lib\site-packages (from xgboos


You are using pip version 19.0.3, however version 19.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.


Requirement already satisfied: wordcloud in c:\users\shubh\anaconda3\lib\site-packages (1.5.0)
Requirement already satisfied: numpy>=1.6.1 in c:\users\shubh\anaconda3\lib\site-packages (fro
Requirement already satisfied: pillow in c:\users\shubh\anaconda3\lib\site-packages (from word


You are using pip version 19.0.3, however version 19.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
You are using pip version 19.0.3, however version 19.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
C:\Users\shubh\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
# using SQLite Table to read data.
#os.chdir("/content/drive/Colab Notebooks") #changing directory
con = sqlite3.connect('database.sqlite')
```

```
# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# you can change the number to any other number based on your computing power

#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con)

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

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

```
Number of data points in our data (525814, 10)
```

Out[2]:

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

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

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

In [3]:

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

In [4]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

```
Out[4]:                 UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R1105J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

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

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

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

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

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

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

```
In [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

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

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

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

```
Out[9]: (364173, 10)
```

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

Out[10]: 69.25890143662969

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

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

display.head()
```

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

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

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

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

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

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

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

```
(364171, 10)
```

Out[13]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
```

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
```

```
In [17]: # https://stackoverflow.com/a/47091490/4084039
         import re

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

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

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

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only th
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alon
```

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

```
Great ingredients although chicken should have been 1st rather than chicken broth the only thin
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

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

```
"hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: # 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%|| 364171/364171 [05:40<00:00, 1069.10it/s]
```

```
In [23]: preprocessed_reviews[1500]
```

```
Out[23]: 'great ingredients although chicken rather chicken broth thing not think belongs cano
```

[3.2] Splitting the data

```
In [24]: final['Text'] = preprocessed_reviews
         finalp = final[final.Score==1].sample(30000,random_state=2)
         finaln = final[final.Score==0].sample(30000,random_state=2)
         finalx = pd.concat([finalp,finaln],ignore_index=True)
         finalx = finalx.sort_values('Time')
         y = finalx.Score.values
         X = finalx.Text.values
         Xtr,Xtest,ytr,ytest = train_test_split(X,y,test_size = 0.3)
         print(finalx.Score.value_counts())
         print(Xtr.shape,Xtest.shape,ytr.shape,ytest.shape)
```

```
1    30000
0    30000
Name: Score, dtype: int64
(42000,) (18000,) (42000,) (18000,)
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [25]: #BoW
         count_bow = CountVectorizer(ngram_range=(1,2),min_df=10) #in scikit-learn
```

```
bow_vec_tr = count_bow.fit_transform(Xtr)
print("some feature names ", count_bow.get_feature_names()[:10])
print('='*50)
bow_vec_test = count_bow.transform(Xtest)
```

```
some feature names  ['aa', 'ability', 'able', 'able buy', 'able chew', 'able drink', 'able eat
==================================================
```

5.2 [4.3] TF-IDF

```
In [26]: #fidf
         tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
         tfidf_tr = tf_idf_vect.fit_transform(Xtr)
         print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
         print('='*50)

         tfidf_test = tf_idf_vect.transform(Xtest)
         print("the type of count vectorizer ",type(tfidf_tr))
         print("the shape of out text TFIDF vectorizer ",tfidf_tr.get_shape())
         print("the number of unique words including both unigrams and bigrams ", tfidf_tr.get_
```

```
some sample features(unique words in the corpus) ['aa', 'ability', 'able', 'able buy', 'able c
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (42000, 25044)
the number of unique words including both unigrams and bigrams  25044
```

5.3 [4.4] Word2Vec

```
In [27]: # Train your own Word2Vec model using your own text corpus
         i=0
         list_of_sentance=[]
         for sentance in Xtr:
             list_of_sentance.append(sentance.split())
```

```
In [28]: # 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.bi
        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, t
```

```
[('awesome', 0.8741124868392944), ('fantastic', 0.8215399384498596), ('good', 0.805439114570617
==================================================
[('nastiest', 0.8355489373207092), ('best', 0.7728778123855591), ('weakest', 0.708370089530944
```

In [29]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  13054
sample words  ['really', 'wanted', 'like', 'chocolate', 'cookies', 'reading', 'reviews', 'not'
```

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

[4.4.1.1] Avg W2v

In [30]:
```
# average Word2Vec for training data
i=0
list_of_sent_intr=[]
for sent in Xtr:
    list_of_sent_intr.append(sent.split())

# compute average word2vec for each review.
sent_vectors_intr = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_intr): # for each review/sentence
```

```
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors_intr.append(sent_vec)
    print(len(sent_vectors_intr))
    print(len(sent_vectors_intr[0]))

    # average Word2Vec for test data
    i=0
    list_of_sent_intest=[]
    for sent in Xtest:
        list_of_sent_intest.append(sent.split())

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

```
100%|| 42000/42000 [02:37<00:00, 266.68it/s]


42000
50


100%|| 18000/18000 [01:09<00:00, 260.16it/s]


18000
50
```

[4.4.1.2] TFIDF weighted W2v

In [31]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10)
tf_idf_matrix = model.fit_transform(Xtr)
model.transform(Xtest)
# 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 [32]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_intr = []; # the tfidf-w2v for each sentence/review is stored in th
row=0;
for sent in tqdm(list_of_sent_intr): # 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_intr.append(sent_vec)
    row += 1

tfidf_sent_vectors_intest = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in tqdm(list_of_sent_intest): # 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_intest.append(sent_vec)
            row += 1
```

```
100%|| 42000/42000 [06:15<00:00, 111.78it/s]
100%|| 18000/18000 [01:48<00:00, 166.48it/s]
```

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 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>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

Function for RF as well as XGBoost

```
In [33]: def rf(ft_train,ft_test,query):
             start = datetime.now()
             #Giving Parameters for tuning
             parameters = {'max_depth':[1, 5, 10, 50, 100, 500, 1000], 'n_estimators':[100, 500,
             rf = RandomForestClassifier(n_jobs=-1)
             clf = GridSearchCV(rf, param_grid = parameters, scoring='roc_auc', cv=2,return_trai
             clf.fit(ft_train,ytr)

             results = clf.cv_results_
             train_score = results['mean_train_score']
             train_score_reshaped = train_score.reshape(7,4)
             test_score = results['mean_test_score']
             test_score_reshaped = test_score.reshape(7,4)
             max_depth=[1, 5, 10, 50, 100, 500, 1000]
             n_estimators=[100, 500, 700, 1000]
```

```
#Making into a Dataframe for Heatmaps
df_trainscore = pd.DataFrame(train_score_reshaped,columns=n_estimators,index=max_dep
df_testscore = pd.DataFrame(test_score_reshaped,columns=n_estimators,index=max_depth

#Getting Max Values
train_max_value = df_trainscore.values.max()
test_max_value = df_testscore.values.max()

#Finding location of the max values (row,column)
i1,j1=  np.where(df_trainscore.values == train_max_value)
i2,j2 = np.where(df_testscore.values == test_max_value)
max_depth_train = list(df_trainscore.index[i1])[0]
n_est_train = list(df_trainscore.columns[j1])[0]
max_depth_test = list(df_testscore.index[i2])[0]
n_est_test = list(df_testscore.columns[j2])[0]


#Calculating Optimal Values
max_depth_optimal = int(np.median((max_depth_train,max_depth_test)))
n_est_optimal = int(np.median((n_est_train,n_est_test)))

#Plotting Heat Maps
fig, (ax1, ax2) =plt.subplots(1,2)
sns.heatmap(df_trainscore, annot = True, ax=ax1)
sns.heatmap(df_testscore, annot = True, ax=ax2)
ax1.set_title('Training plot')
ax1.set_xlabel('n_estimators')
ax1.set_ylabel('max_depth')
ax2.set_title('Validation plot')
ax2.set_xlabel('n_estimators')
ax2.set_ylabel('max_depth')
fig.show()


print('The maximum Train AUC is {} for  {},{} . The max Validation AUC is {} for {}
print('Optimal parameters are max_depth = {} and n_estimators={} ' .format(max_depth
print("="*50)

#Training model with optimal parameters
model = RandomForestClassifier(max_depth=max_depth_optimal,n_estimators=n_est_optima
model.fit(ft_train,ytr)
pred_train = model.predict_proba(ft_train)
pred_test = model.predict_proba(ft_test)
p_train = model.predict(ft_train)
p_test = model.predict(ft_test)
f = model.feature_importances_
```

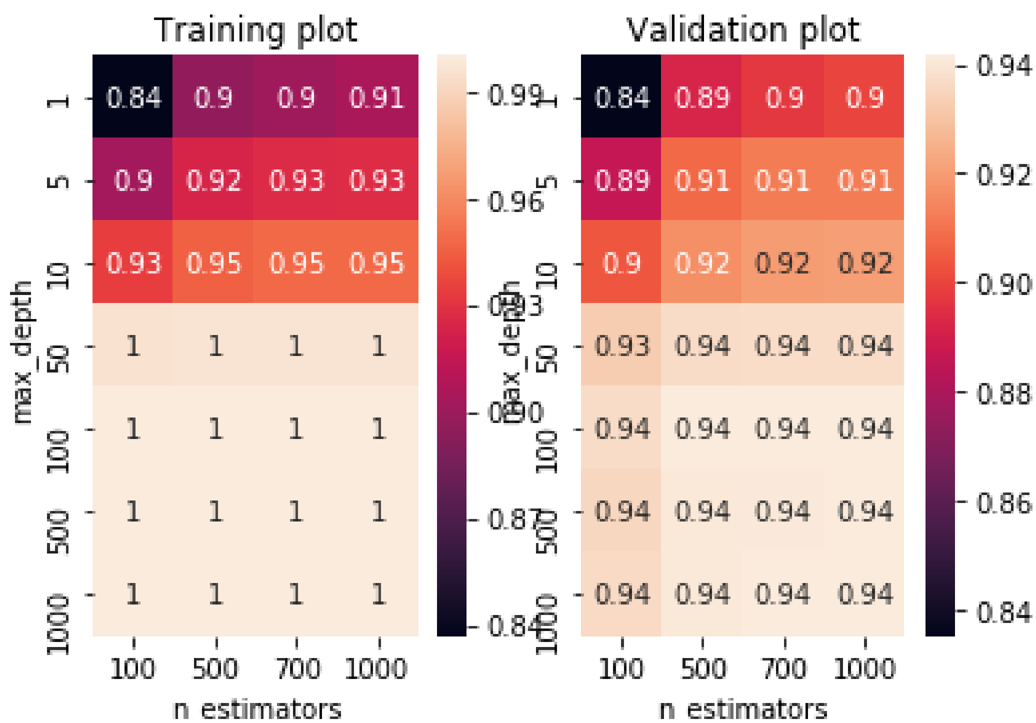
```
    #Getting FPR AND TPR values for ROC Curve for train and test data

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    fpr,tpr,_ = roc_curve(ytr,pred_train[:,1])
    roc_auc_train = roc_auc_score(ytr,pred_train[:,1])
    fpr2 = dict()
    tpr2 = dict()
    roc_auc2 = dict()
    fpr2,tpr2,_ = roc_curve(ytest,pred_test[:,1])
    roc_auc_test = roc_auc_score(ytest,pred_test[:,1])
    plt.figure()
    plt.title(" ROC Curve")
    plt.plot(fpr,tpr,'b',label='ROC curve for train data(area = %0.2f)' % roc_auc_train
    plt.plot(fpr2,tpr2,'r',label='ROC curve for test data(area = %0.2f)' % roc_auc_test
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc="lower right")
    plt.show()
    #return max_depth_optimal,n_estimators_optimal
    print('This is the ROC_AUC curve using optimal parameters with ROC_AUC of %0.2f for
    print("="*50)

    #For confusion matrix
    print("Confusion Matrix for Train data")
    skplt.metrics.plot_confusion_matrix(ytr,p_train)
    print(classification_report(ytr,p_train))
    print("="*50)
    print("Confusion matrix for Test data")
    skplt.metrics.plot_confusion_matrix(ytest,p_test)
    print(classification_report(ytest,p_test))

    print("Time taken to run this cell :", datetime.now() - start)
    if query == 1:
      return f
```

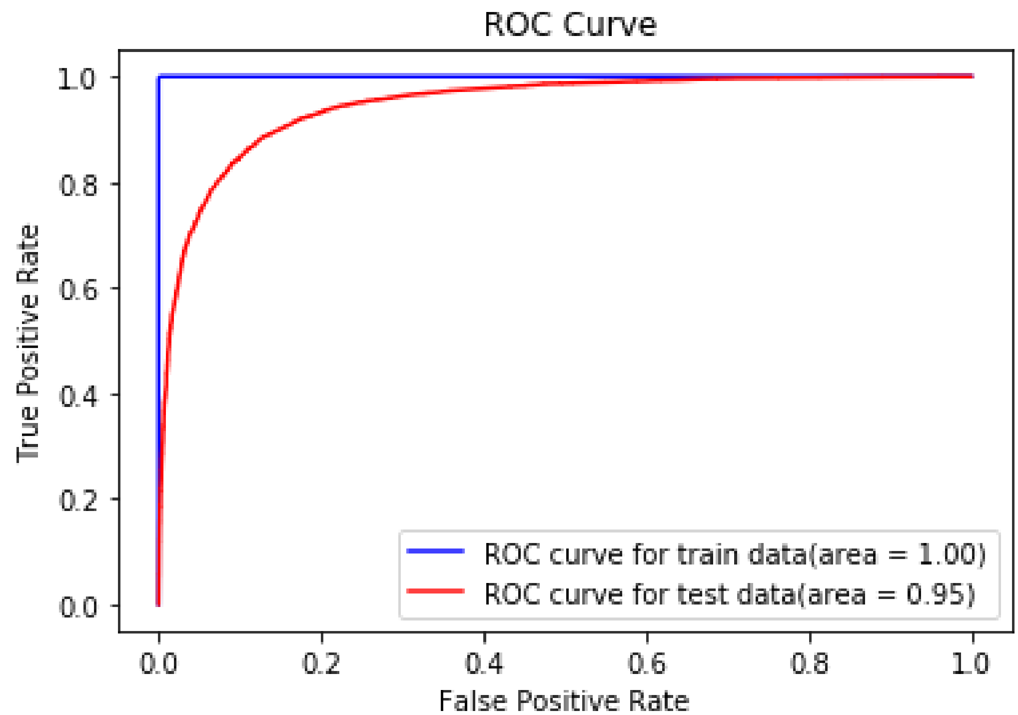
```
In [34]: #wordcloud
         def wcd(vector,w):
           features = vector.get_feature_names()
           top_features = pd.DataFrame(w,features)
           text = str(top_features[0].sort_values(ascending=False)[0:20])
           wordcloud = WordCloud(colormap="Oranges_r").generate(text)
           plt.imshow(wordcloud)
           plt.axis("off")
           plt.show()
```

6.1.1 [5.1.1] Applying Random Forests on BOW, SET 1

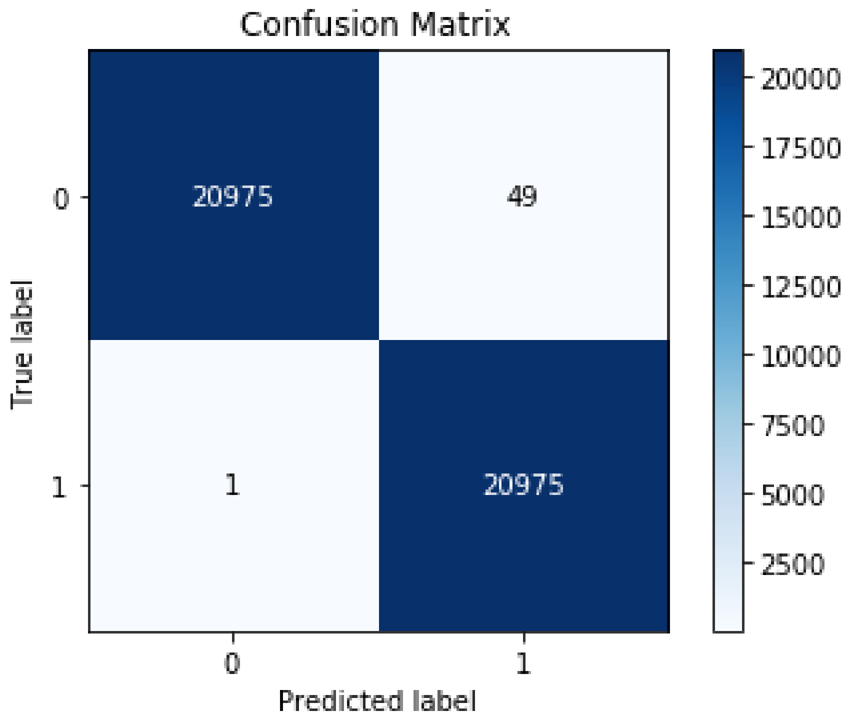
```
In [35]: w1 = rf(bow_vec_tr,bow_vec_test,1)
```

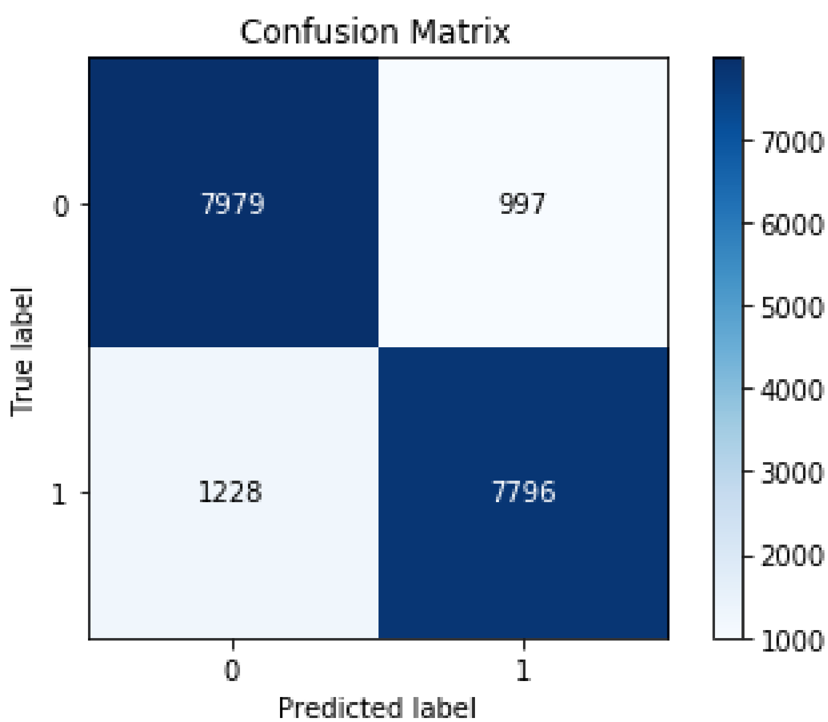
```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999970702909581 for  500,700 . The max Validation AUC is 0.94156359
Optimal parameters are max_depth = 300 and n_estimators=600
====================================================
```






```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.95 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.87      0.89      0.88      8976
          1       0.89      0.86      0.88      9024

avg / total       0.88      0.88      0.88     18000

Time taken to run this cell : 1:01:15.098462
```


Confusion Matrix
20975
49
1
20975
True label
Predicted label
0
1
20000
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix
7979
997
1228
7796
True label
Predicted label
0
1
7000
6000
5000
4000
3000
2000
1000

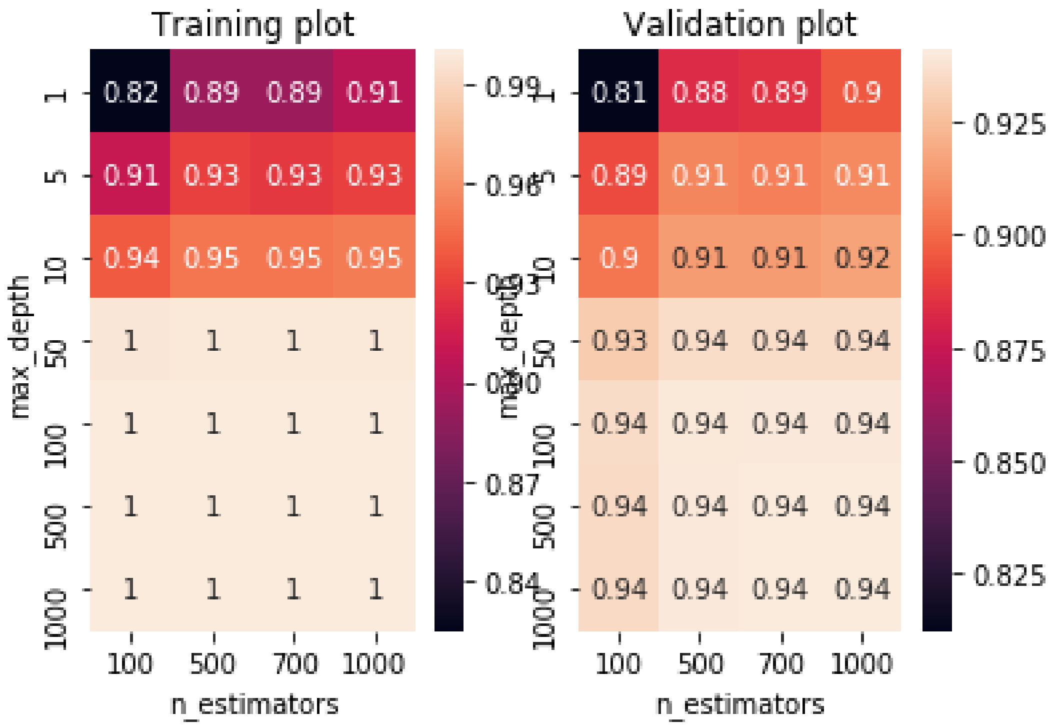
6.1.2 [5.1.2] Wordcloud of top 20 important features from SET 1

```
In [36]: wcd(count_bow,w1)
```

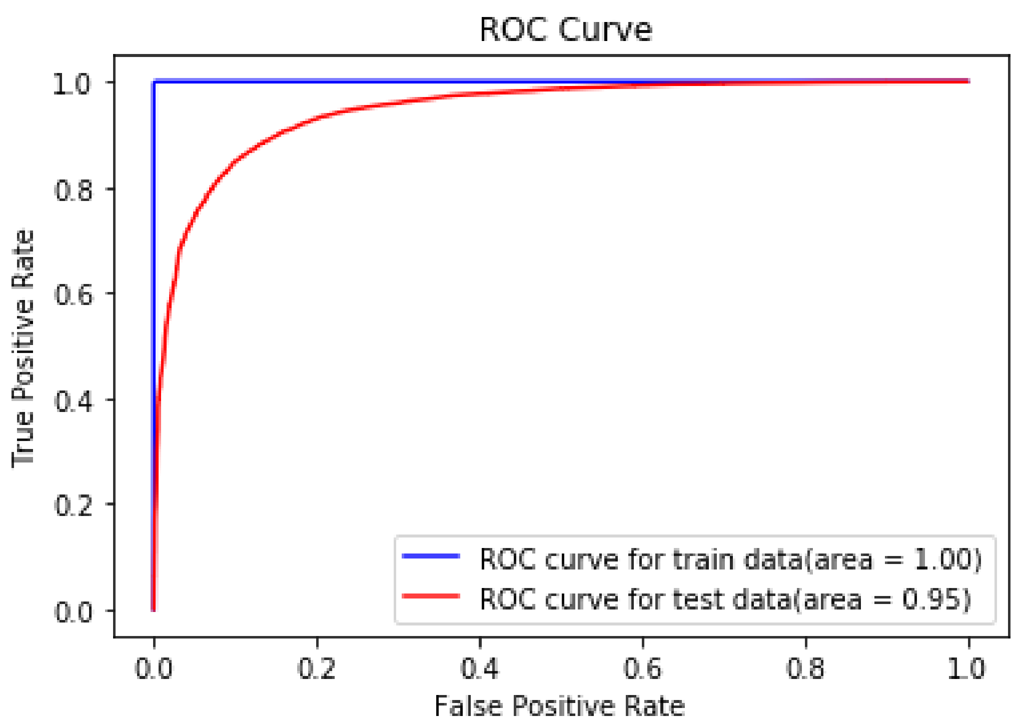
6.1.3 [5.1.3] Applying Random Forests on TFIDF, SET 2

```
In [37]: w2 = rf(tfidf_tr,tfidf_test,1)
```

```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999970793612647 for  500,500 . The max Validation AUC is 0.94080701
Optimal parameters are max_depth = 500 and n_estimators=750
==================================================
```

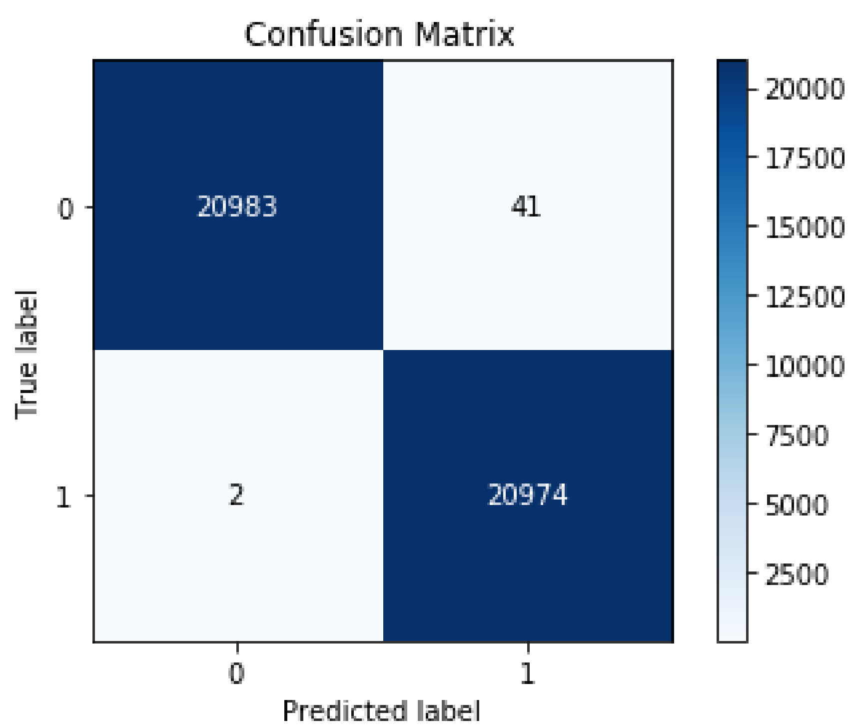


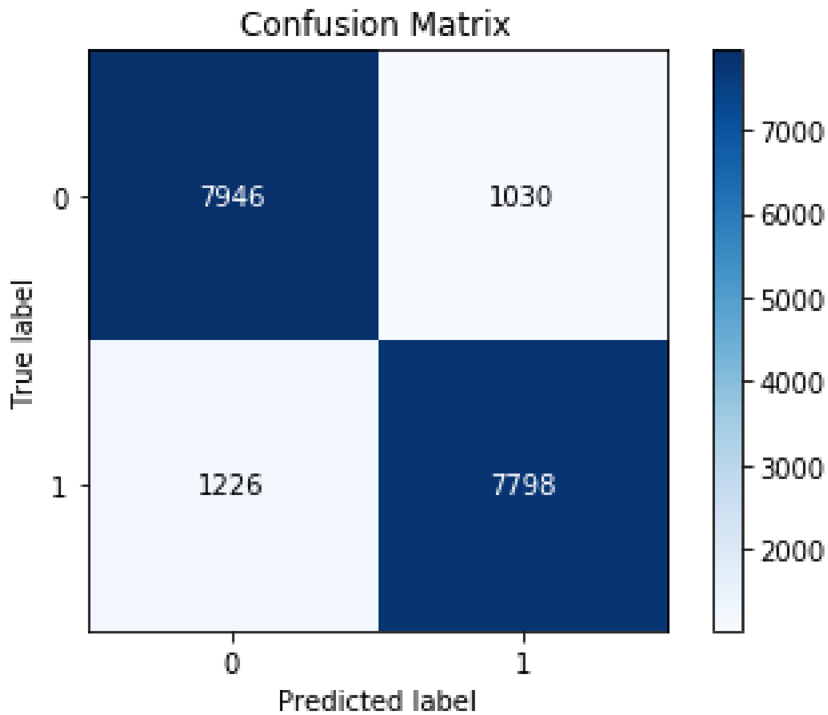
Training plot
Validation plot
max_depth
n_estimators



ROC Curve
True Positive Rate
False Positive Rate
ROC curve for train data(area = 1.00)
ROC curve for test data(area = 0.95)

```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.95 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.87      0.89      0.88      8976
          1       0.88      0.86      0.87      9024

avg / total       0.87      0.87      0.87     18000

Time taken to run this cell : 1:03:10.659817
```





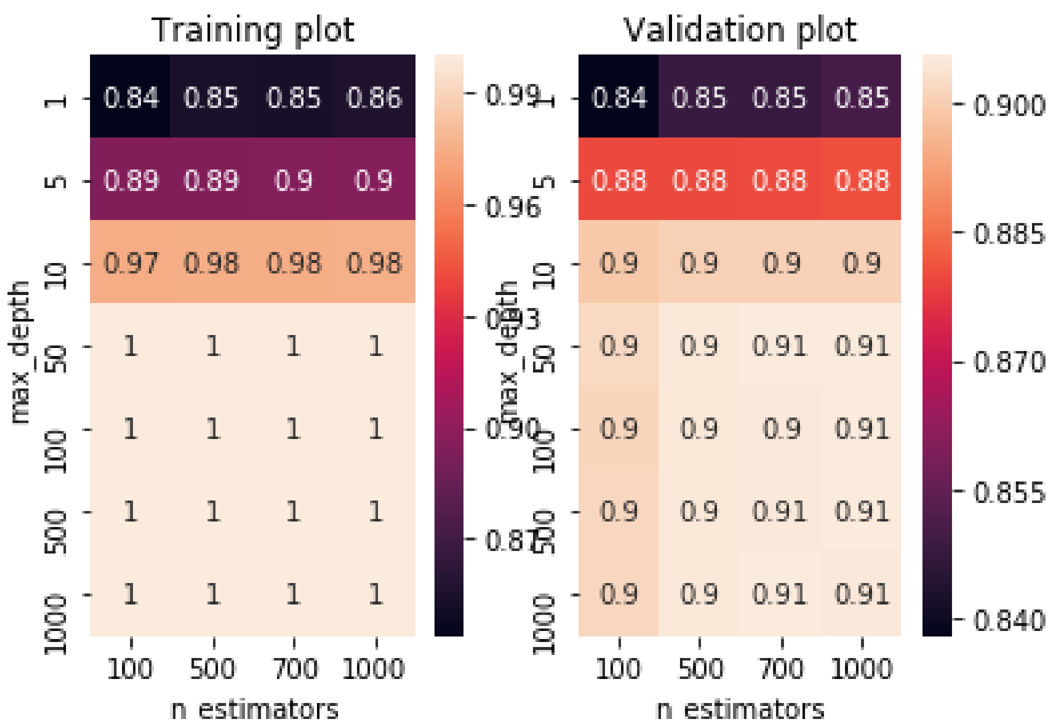
6.1.4 [5.1.4] Wordcloud of top 20 important features from SET 2

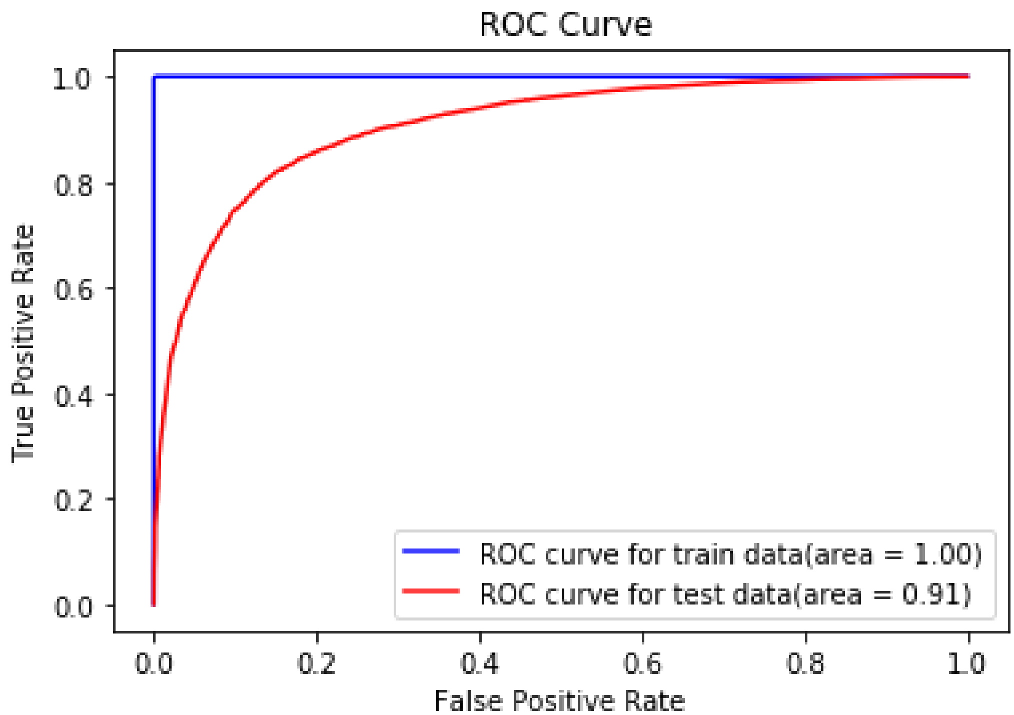
```
In [38]: wcd(tf_idf_vect,w2)
```

6.1.5 [5.1.5] Applying Random Forests on AVG W2V, SET 3

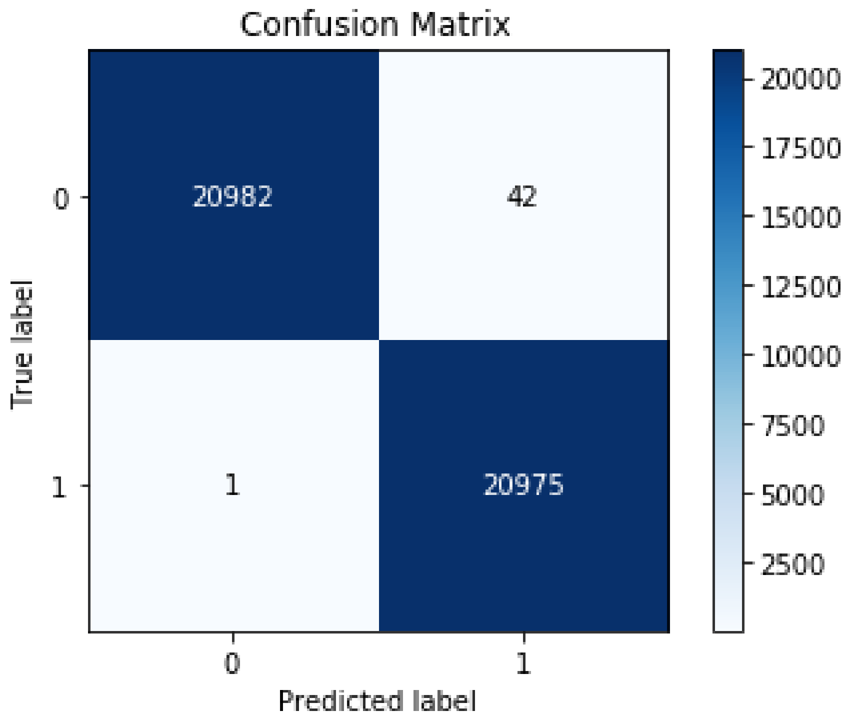
```
In [39]: rf(sent_vectors_intr,sent_vectors_intest,0)
```

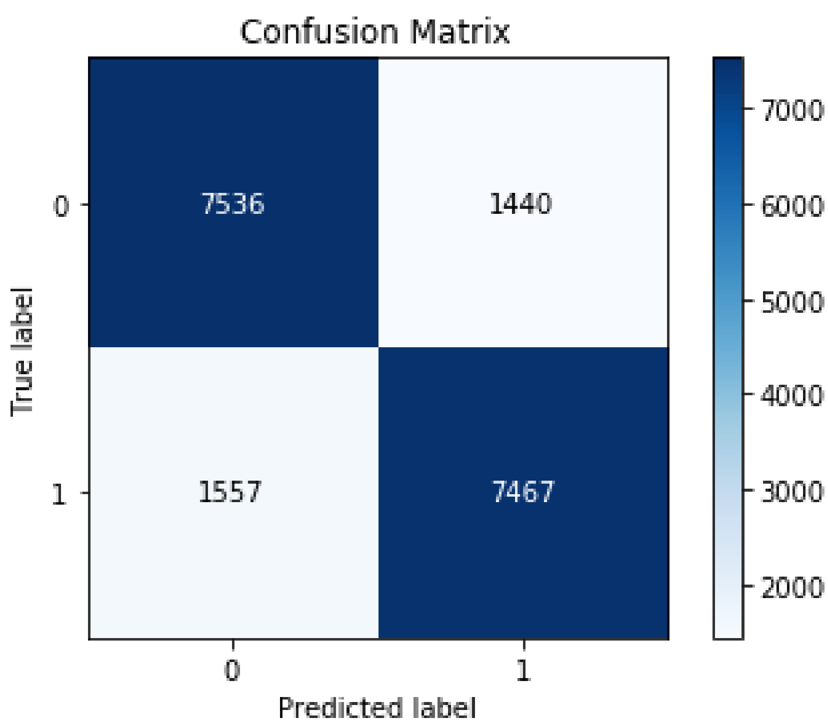
```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.999997090699148 for  1000,700 . The max Validation AUC is 0.90541322
Optimal parameters are max_depth = 750 and n_estimators=850
==================================================
```






```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.91 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.83      0.84      0.83      8976
          1       0.84      0.83      0.83      9024

avg / total       0.83      0.83      0.83     18000

Time taken to run this cell : 0:27:17.943669
```

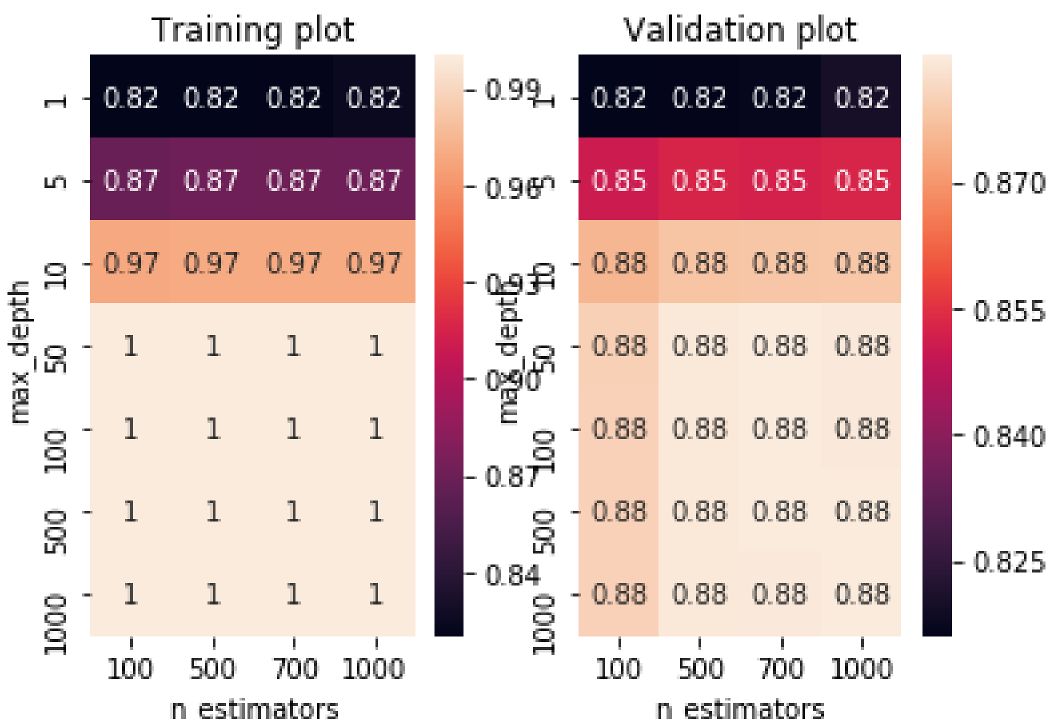

Confusion Matrix
20982
42
1
20975
True label
0
1
Predicted label
20000
17500
15000
12500
10000
7500
5000
2500

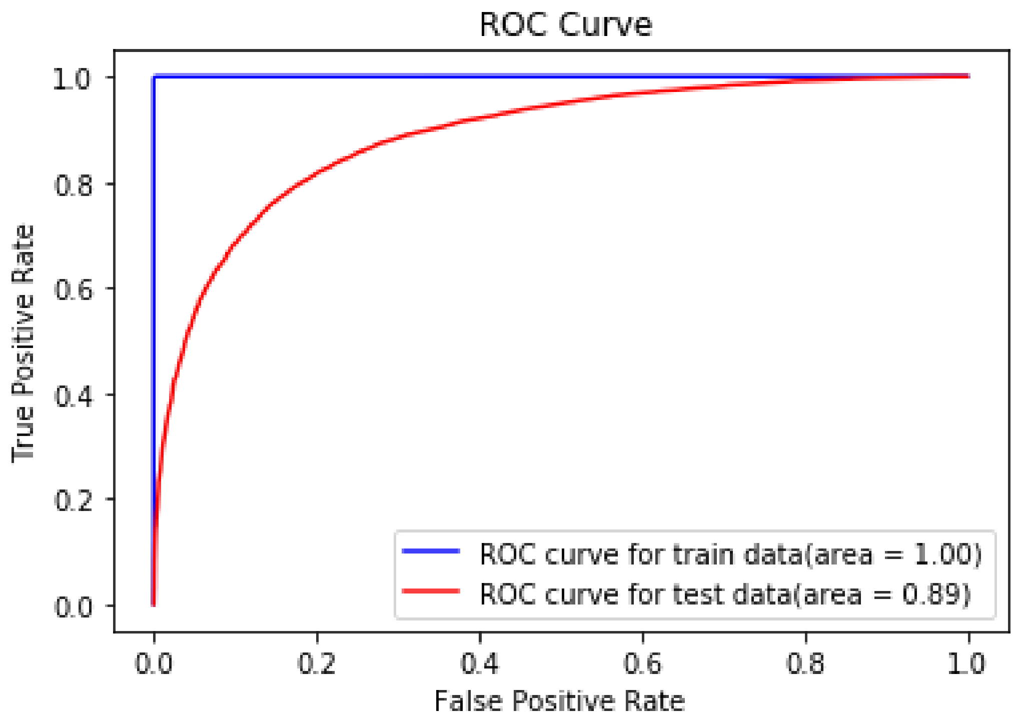

Confusion Matrix
7536
1440
1557
7467
True label
0
1
Predicted label
7000
6000
5000
4000
3000
2000

6.1.6 [5.1.6] Applying Random Forests on TFIDF W2V, SET 4

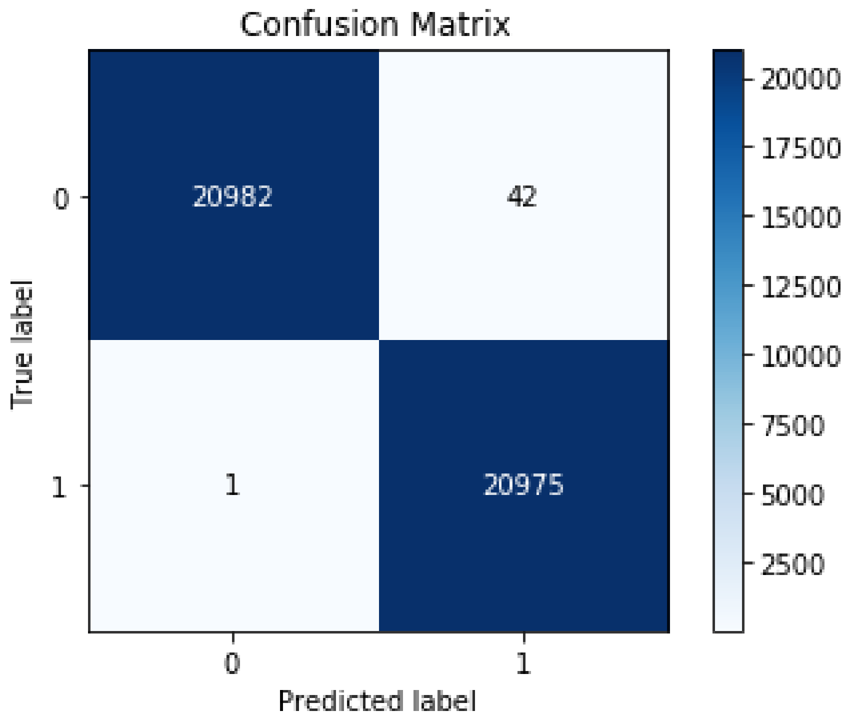
```
In [40]: rf(tfidf_sent_vectors_intr,tfidf_sent_vectors_intest,0)
```

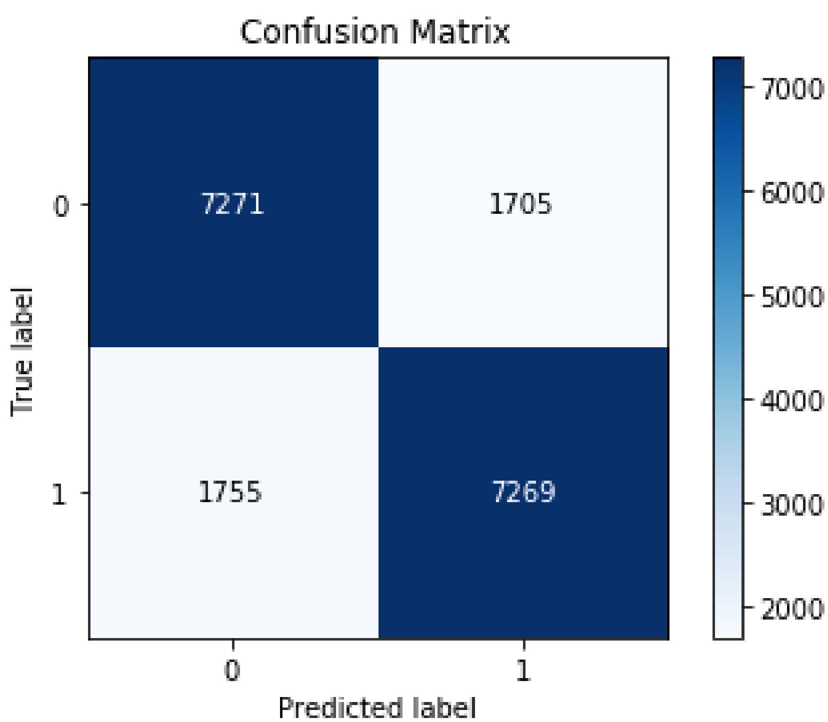
```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999970657558046 for  1000,700 . The max Validation AUC is 0.8849992
Optimal parameters are max_depth = 1000 and n_estimators=850
==================================================
```






```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.89 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.81      0.81      0.81      8976
          1       0.81      0.81      0.81      9024

avg / total       0.81      0.81      0.81     18000

Time taken to run this cell : 0:27:11.411400
```


Confusion Matrix
20982
42
1
20975
True label
Predicted label
0
1
0
1
20000
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix
7271
1705
1755
7269
True label
Predicted label
0
1
0
1
7000
6000
5000
4000
3000
2000

6.2 [5.2] Applying GBDT using XGBOOST

In [41]:
```
def xg(ft_train,ft_test):
    start = datetime.now()
    #Giving Parameters for tuning
    parameters = {'max_depth':[1, 5, 10, 50, 100, 500, 1000], 'n_estimators':[100, 500,
    xgboost = xgb.XGBClassifier(subsample=0.8, colsample_bytree=0.5, colsample_bylevel=
    clf = GridSearchCV(xgboost, param_grid = parameters, scoring='roc_auc', cv=2,return_
    clf.fit(ft_train,ytr)

    results = clf.cv_results_
    train_score = results['mean_train_score']
    train_score_reshaped = train_score.reshape(7,4)
    test_score = results['mean_test_score']
    test_score_reshaped = test_score.reshape(7,4)
    max_depth=[1, 5, 10, 50, 100, 500, 1000]
    n_estimators=[100, 500, 700, 1000]


    #Making into a Dataframe for Heatmaps
    df_trainscore = pd.DataFrame(train_score_reshaped,columns=n_estimators,index=max_de
    df_testscore = pd.DataFrame(test_score_reshaped,columns=n_estimators,index=max_dept

    #Getting Max Values
    train_max_value = df_trainscore.values.max()
    test_max_value = df_testscore.values.max()

    #Finding location of the max values (row,column)
    i1,j1=  np.where(df_trainscore.values == train_max_value)
    i2,j2 = np.where(df_testscore.values == test_max_value)
    max_depth_train = list(df_trainscore.index[i1])[0]
    n_est_train = list(df_trainscore.columns[j1])[0]
    max_depth_test = list(df_testscore.index[i2])[0]
    n_est_test = list(df_testscore.columns[j2])[0]


    #Calculating Optimal Values
    max_depth_optimal = int(np.median((max_depth_train,max_depth_test)))
    n_est_optimal = int(np.median((n_est_train,n_est_test)))

    #Plotting Heat Maps
    fig, (ax1, ax2) =plt.subplots(1,2)
    sns.heatmap(df_trainscore, annot = True, ax=ax1)
    sns.heatmap(df_testscore, annot = True, ax=ax2)
    ax1.set_title('Training plot')
    ax1.set_xlabel('n_estimators')
    ax1.set_ylabel('max_depth')
    ax2.set_title('Validation plot')
```

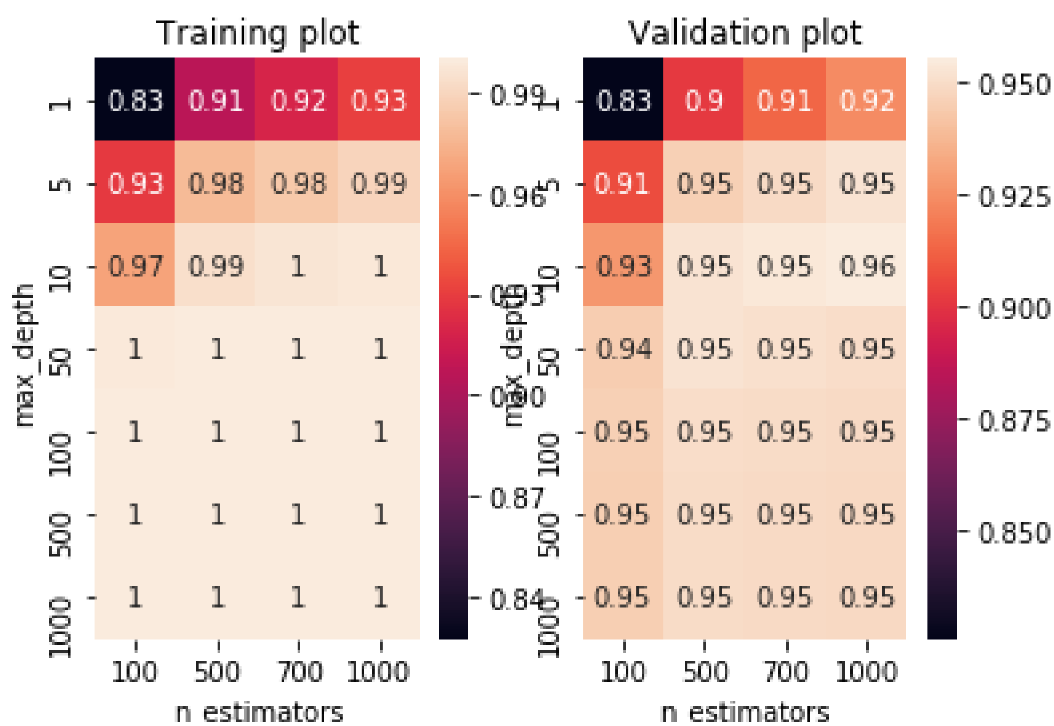
```
ax2.set_xlabel('n_estimators')
ax2.set_ylabel('max_depth')
fig.show()


print('The maximum Train AUC is {} for  {},{} . The max Validation AUC is {} for {}
print('Optimal parameters are max_depth = {} and n_estimators={} ' .format(max_dept
print("="*50)

#Training model with optimal parameters
model = xgb.XGBClassifier(n_jobs=-1,max_depth=max_depth_optimal,n_estimators=n_est_
model.fit(ft_train,ytr)
pred_train = model.predict_proba(ft_train)
pred_test = model.predict_proba(ft_test)
p_train = model.predict(ft_train)
p_test = model.predict(ft_test)
f = model.feature_importances_

#Getting FPR AND TPR values for ROC Curve for train and test data

fpr = dict()
tpr = dict()
roc_auc = dict()
fpr,tpr,_ = roc_curve(ytr,pred_train[:,1])
roc_auc_train = roc_auc_score(ytr,pred_train[:,1])
fpr2 = dict()
tpr2 = dict()
roc_auc2 = dict()
fpr2,tpr2,_ = roc_curve(ytest,pred_test[:,1])
roc_auc_test = roc_auc_score(ytest,pred_test[:,1])
plt.figure()
plt.title(" ROC Curve")
plt.plot(fpr,tpr,'b',label='ROC curve for train data(area = %0.2f)' % roc_auc_train
plt.plot(fpr2,tpr2,'r',label='ROC curve for test data(area = %0.2f)' % roc_auc_test
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
#return max_depth_optimal,n_estimators_optimal
print('This is the ROC_AUC curve using optimal parameters with ROC_AUC of %0.2f for
print("="*50)

#For confusion matrix
print("Confusion Matrix for Train data")
skplt.metrics.plot_confusion_matrix(ytr,p_train)
print(classification_report(ytr,p_train))
print("="*50)
print("Confusion matrix for Test data")
```

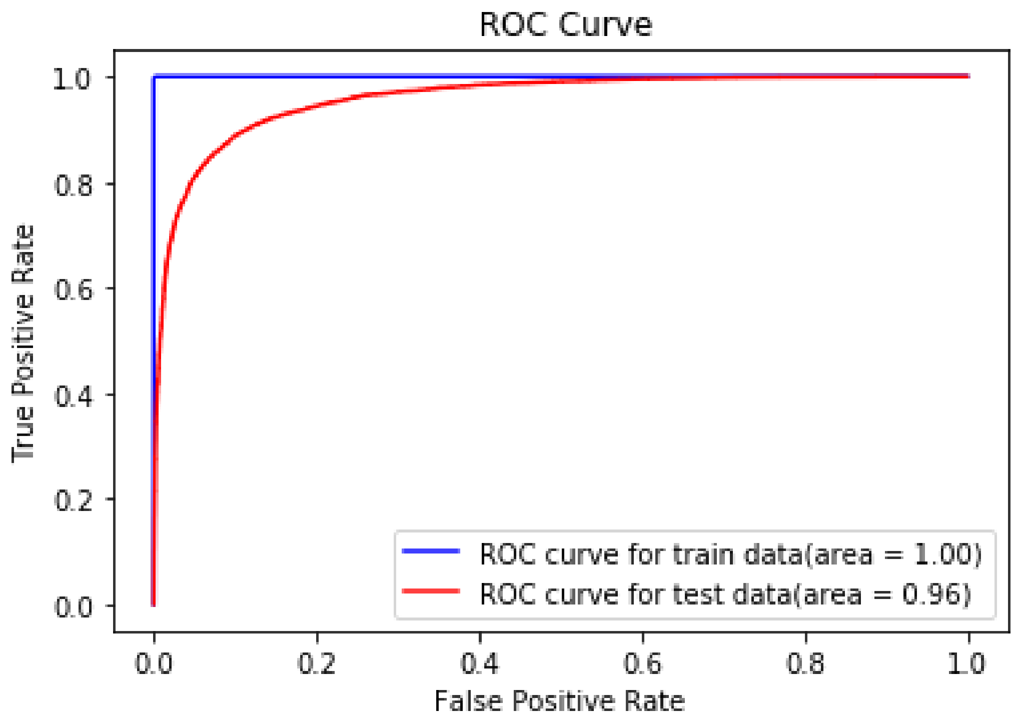
```
skplt.metrics.plot_confusion_matrix(ytest,p_test)
print(classification_report(ytest,p_test))

print("Time taken to run this cell :", datetime.now() - start)
```

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

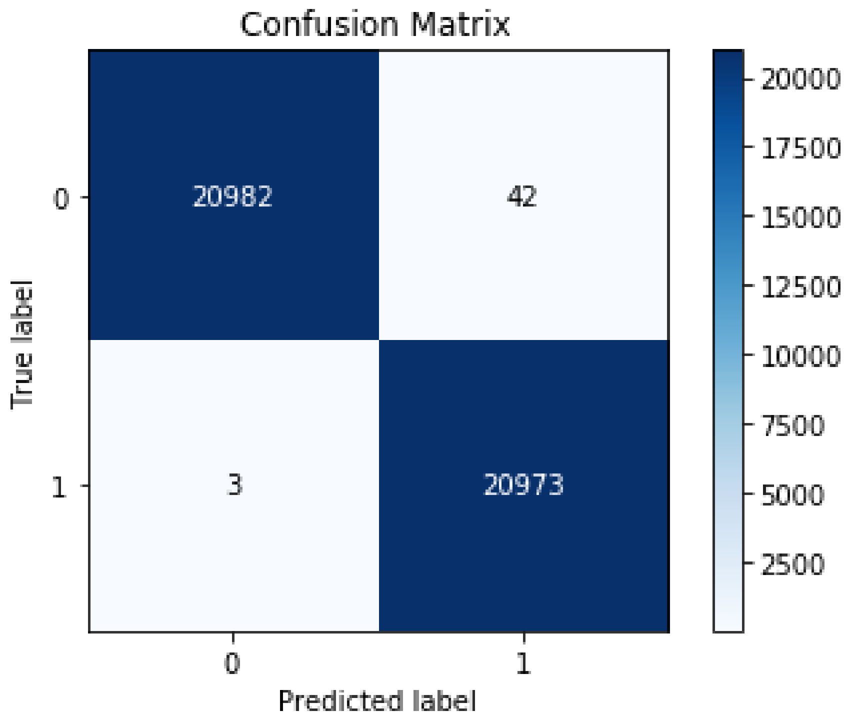
```
In [42]: xg(bow_vec_tr,bow_vec_test)
```

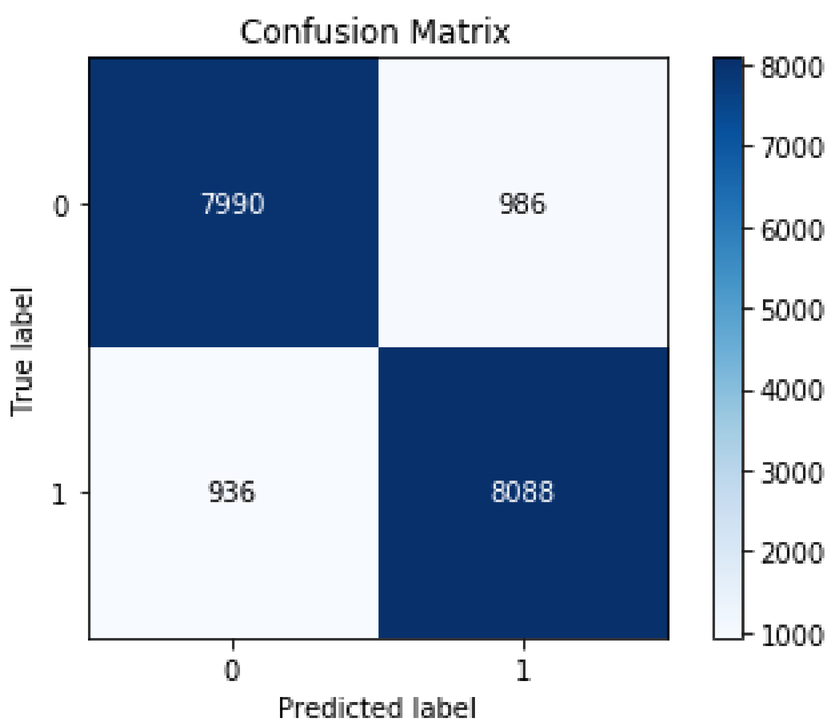
```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999968594063289 for  500,500 . The max Validation AUC is 0.95510750
Optimal parameters are max_depth = 255 and n_estimators=750
====================================================


C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
  if diff:
C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
  if diff:
```






```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.96 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.90      0.89      0.89      8976
          1       0.89      0.90      0.89      9024

avg / total       0.89      0.89      0.89     18000

Time taken to run this cell : 3:22:02.227670
```

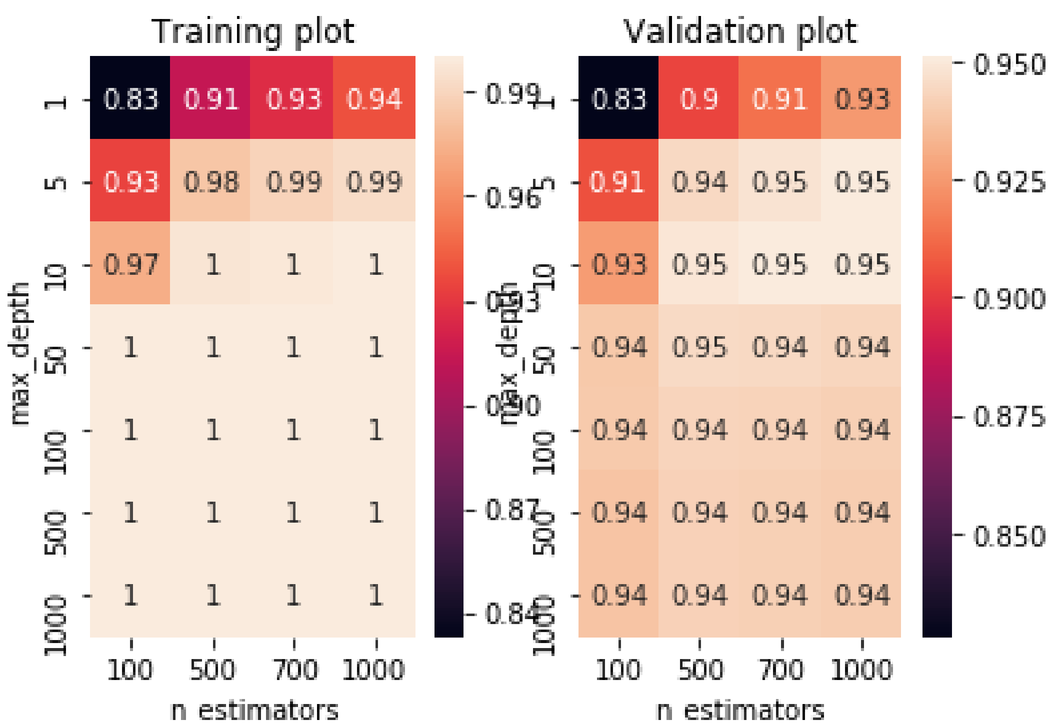

Confusion Matrix
20982
42
3
20973
0
1
0
1
True label
Predicted label
20000
17500
15000
12500
10000
7500
5000
2500

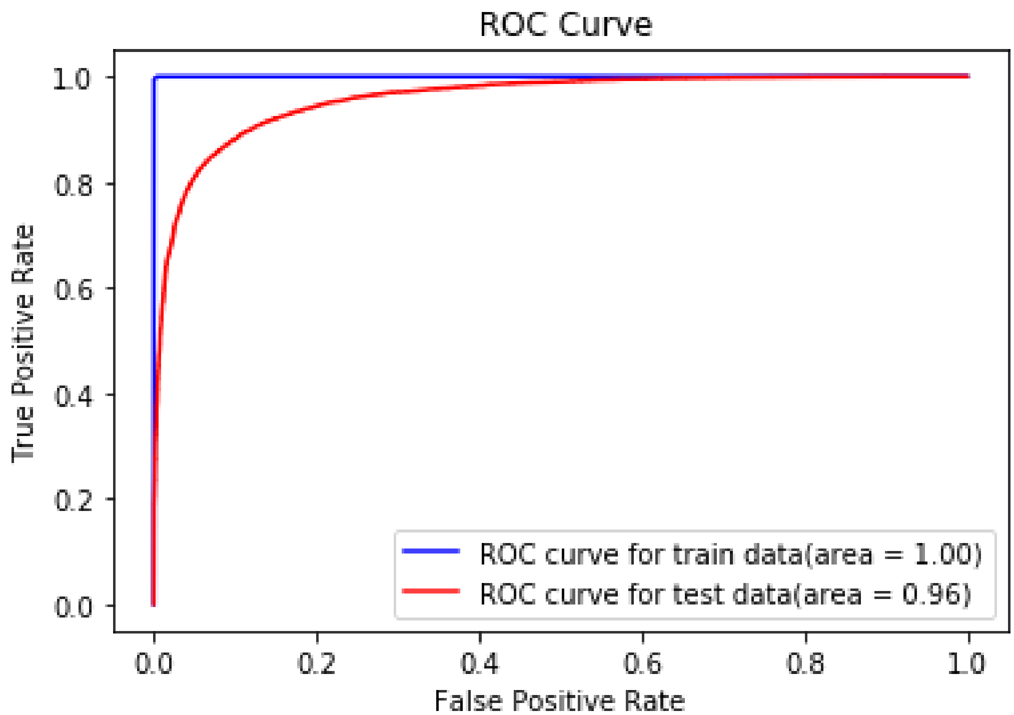

Confusion Matrix
7990
986
936
8088
0
1
0
1
True label
Predicted label
8000
7000
6000
5000
4000
3000
2000
1000

6.2.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

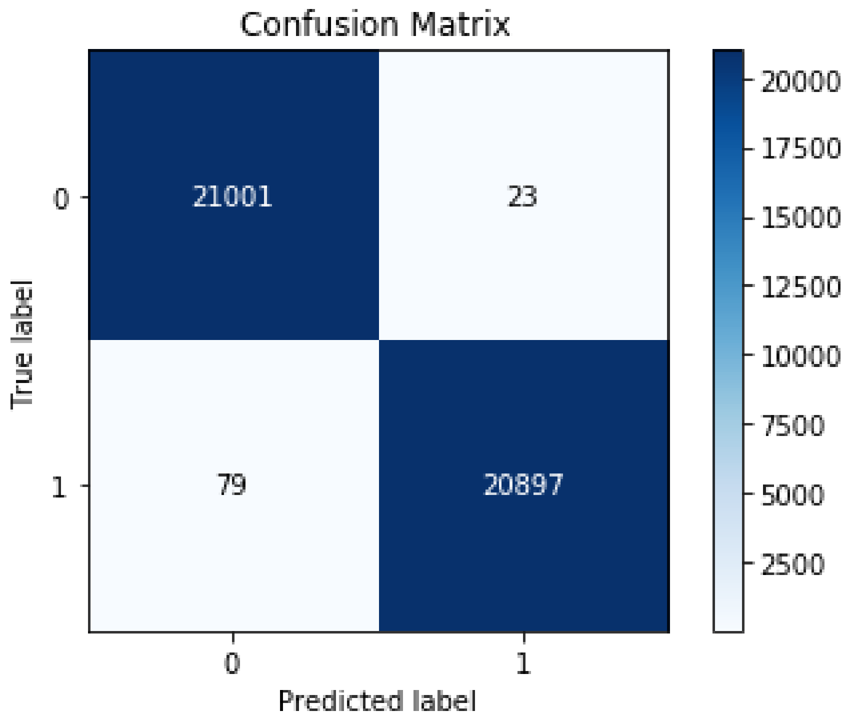
In [43]:
```
xg(tfidf_tr,tfidf_test)
```

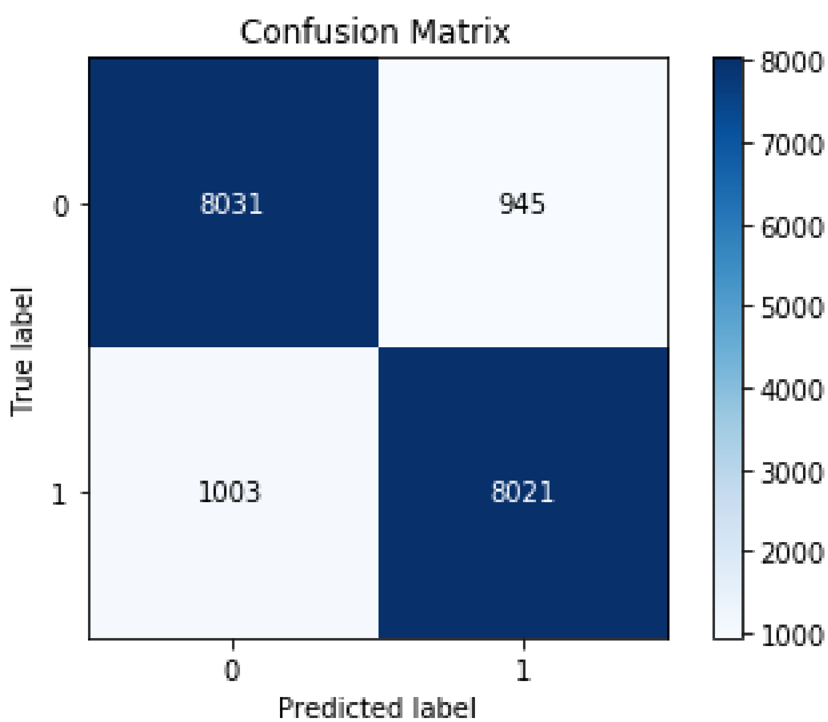
```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999969410390885 for  50,700 . The max Validation AUC is 0.95093143
Optimal parameters are max_depth = 27 and n_estimators=850
==================================================


C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
  if diff:
C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
  if diff:
```






```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.96 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.89      0.89      0.89      8976
          1       0.89      0.89      0.89      9024

avg / total       0.89      0.89      0.89     18000

Time taken to run this cell : 7:27:16.417076
```

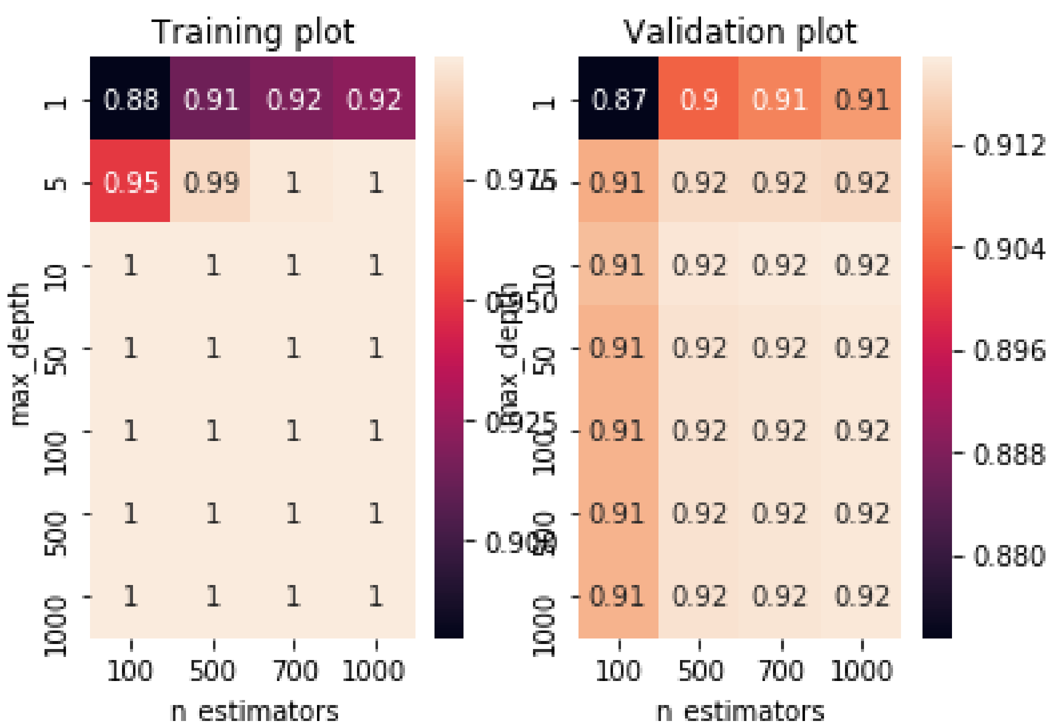

Confusion Matrix
21001
23
79
20897
True label
Predicted label
0
1
20000
17500
15000
12500
10000
7500
5000
2500

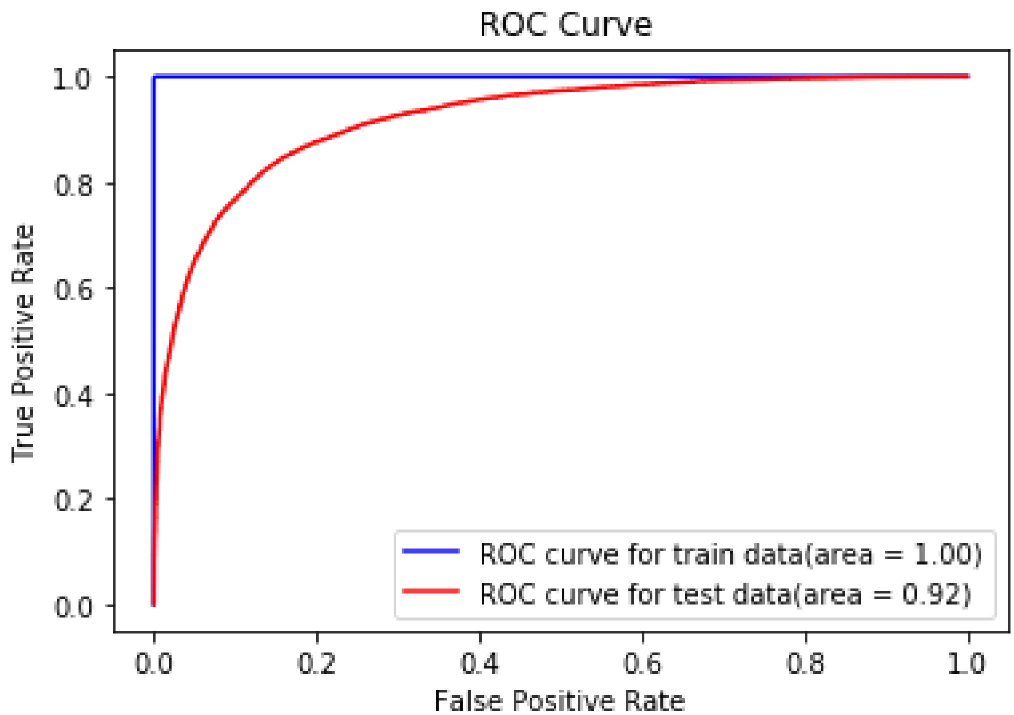

Confusion Matrix
8031
945
1003
8021
True label
Predicted label
0
1
8000
7000
6000
5000
4000
3000
2000
1000

6.2.3 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

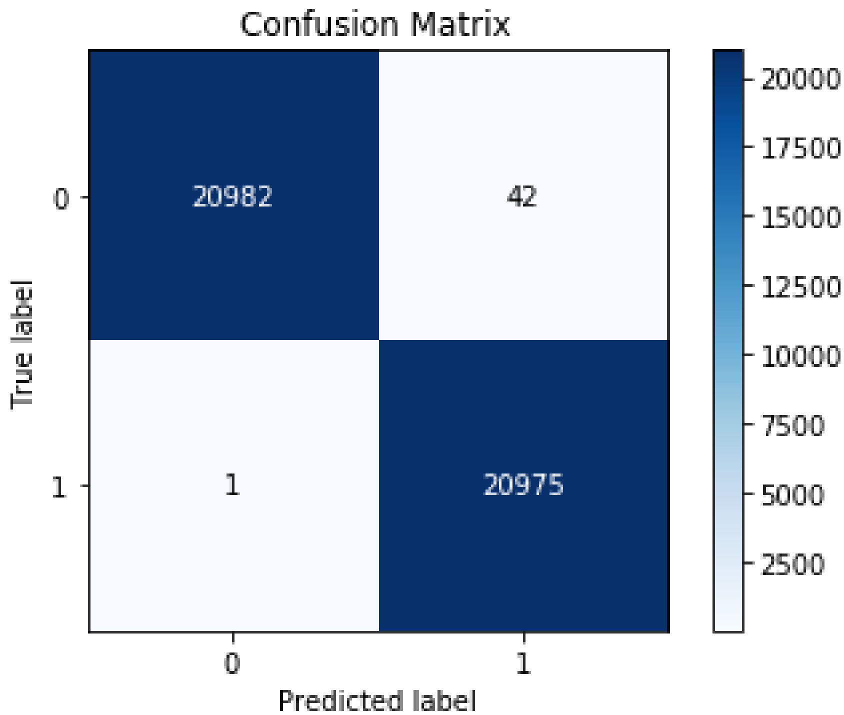
```
In [44]: w2vtrain = np.array(sent_vectors_intr)
         w2vtest = np.array(sent_vectors_intest)
         xg(w2vtrain,w2vtest)
```

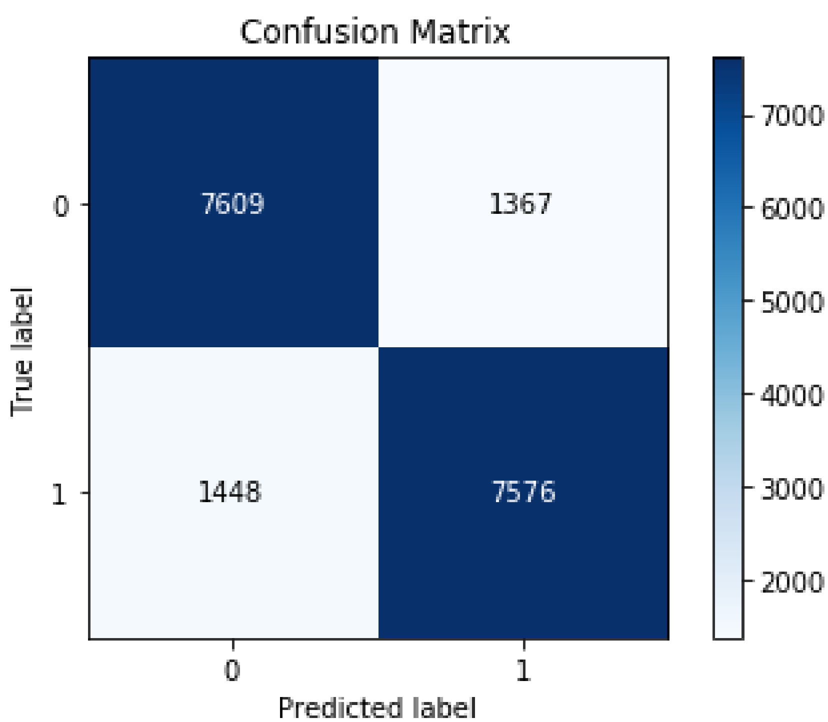
```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999971315155278 for  10,500 . The max Validation AUC is 0.91868332
Optimal parameters are max_depth = 10 and n_estimators=750
==================================================


C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarn
  if diff:
C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarn
  if diff:
```






```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.92 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.84      0.85      0.84      8976
          1       0.85      0.84      0.84      9024

avg / total       0.84      0.84      0.84     18000

Time taken to run this cell : 0:29:05.095060
```

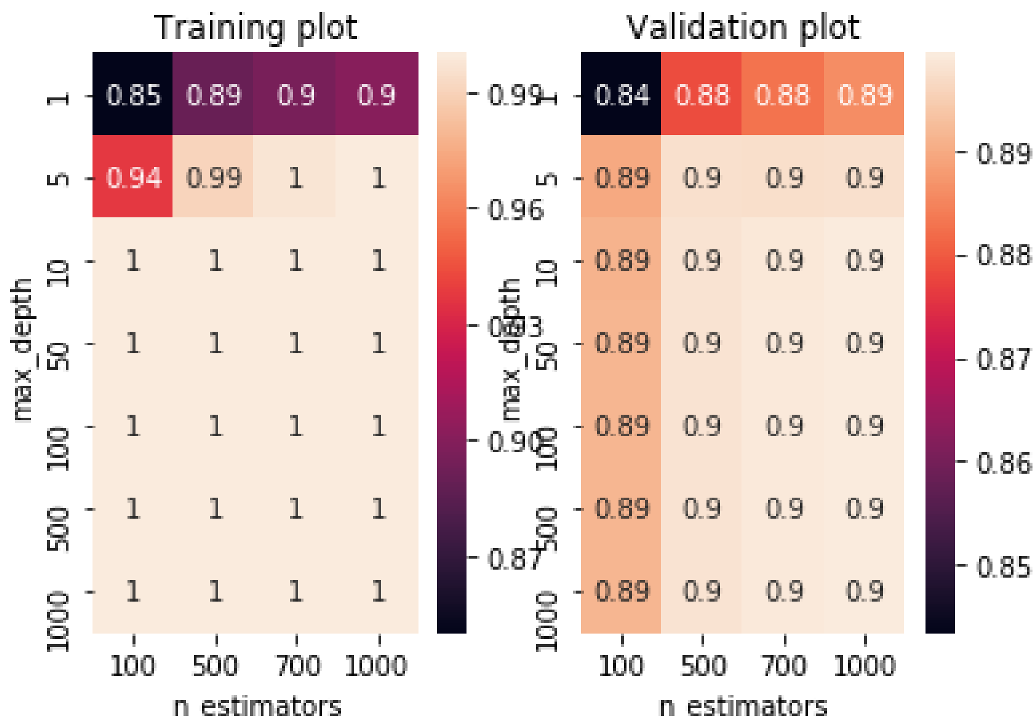

Confusion Matrix
20982
42
1
20975
True label
Predicted label
0
1
20000
17500
15000
12500
10000
7500
5000
2500

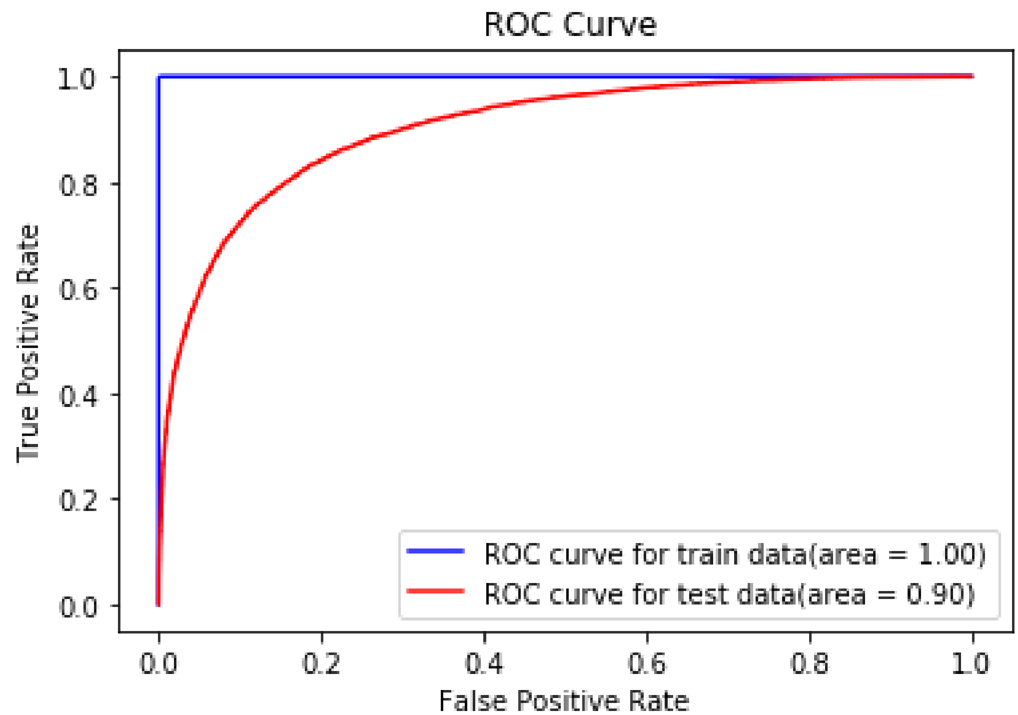

Confusion Matrix
7609
1367
1448
7576
True label
Predicted label
0
1
7000
6000
5000
4000
3000
2000

6.2.4 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

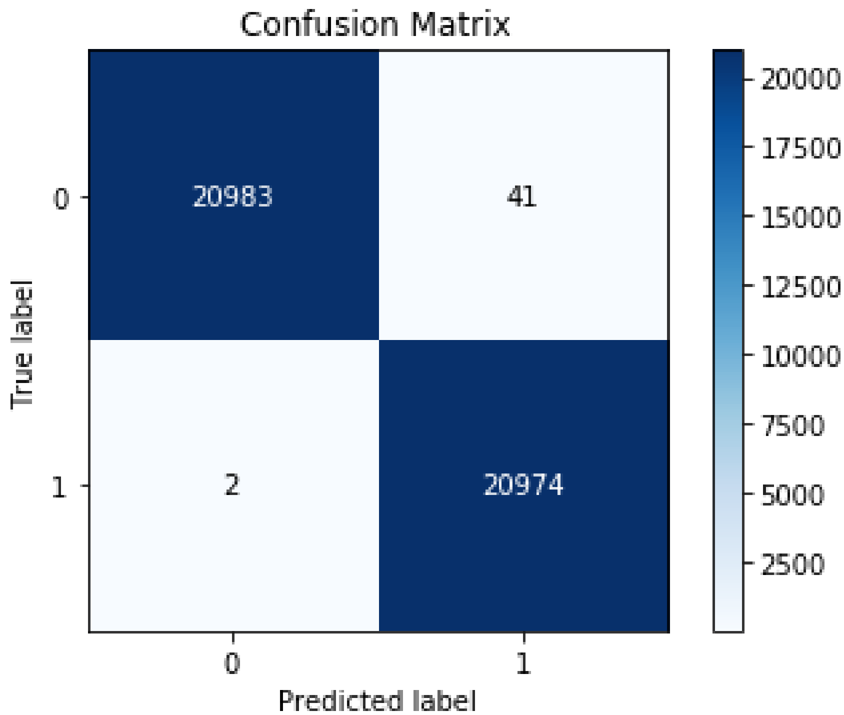
```
In [45]: tfw2vtrain = np.array(tfidf_sent_vectors_intr)
         tfw2vtest = np.array(tfidf_sent_vectors_intest)
         xg(tfw2vtrain,tfw2vtest)
```

```
C:\Users\shubh\Anaconda3\lib\site-packages\matplotlib\figure.py:459: UserWarning: matplotlib is
  "matplotlib is currently using a non-GUI backend, "


The maximum Train AUC is 0.9999970793612647 for  10,500 . The max Validation AUC is 0.89935394
Optimal parameters are max_depth = 30 and n_estimators=750
==================================================


C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
  if diff:
C:\Users\shubh\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
  if diff:
```

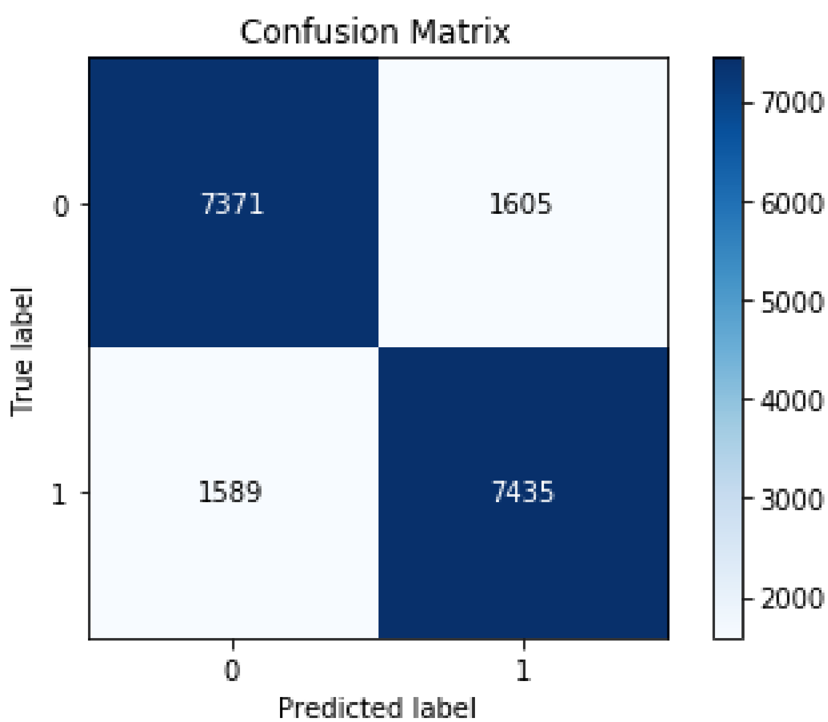





```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.90 for test data
==================================================
Confusion Matrix for Train data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     21024
          1       1.00      1.00      1.00     20976

avg / total       1.00      1.00      1.00     42000

==================================================
Confusion matrix for Test data
             precision    recall  f1-score   support

          0       0.82      0.82      0.82      8976
          1       0.82      0.82      0.82      9024

avg / total       0.82      0.82      0.82     18000

Time taken to run this cell : 0:31:43.692104
```



Confusion Matrix
0
1
True label
20983
41
2
20974
0
1
Predicted label
20000
17500
15000
12500
10000
7500
5000
2500



Confusion Matrix
0
1
True label
7371
1605
1589
7435
0
1
Predicted label
7000
6000
5000
4000
3000
2000

7 [6] Conclusions

```
In [47]: x = PrettyTable()

         x.field_names = ["Algorithm", "Vectorizer", "max_depth","n_estimators", "AUC"]
         x.add_row(["RandomForest", "BoW", 300, 600, 0.95])
         x.add_row(["RandomForest", "Tfidf", 500, 750, 0.95])
         x.add_row(["RandomForest", "Avg W2V", 750, 850, 0.91])
         x.add_row(["RandomForest", "Tfidf weighted W2V", 1000, 850, 0.89])
         x.add_row(["XGBoost", "BoW", 255, 750, 0.96])
         x.add_row(["XGBoost","Tfidf", 27, 850, 0.96])
         x.add_row(["XGBoost","Avg W2V", 10, 750, 0.92])
         x.add_row(["XGBoost","Tfidf weighted W2V", 30, 750, 0.90])
         print(x)
```

```
+--------------+--------------------+-----------+--------------+------+
|  Algorithm   |     Vectorizer     | max_depth | n_estimators | AUC  |
+--------------+--------------------+-----------+--------------+------+
| RandomForest |        BoW         |    300    |     600      | 0.95 |
| RandomForest |       Tfidf        |    500    |     750      | 0.95 |
| RandomForest |      Avg W2V       |    750    |     850      | 0.91 |
| RandomForest | Tfidf weighted W2V |    1000   |     850      | 0.89 |
|   XGBoost    |        BoW         |    255    |     750      | 0.96 |
|   XGBoost    |       Tfidf        |     27    |     850      | 0.96 |
|   XGBoost    |      Avg W2V       |     10    |     750      | 0.92 |
|   XGBoost    | Tfidf weighted W2V |     30    |     750      | 0.9  |
+--------------+--------------------+-----------+--------------+------+
```

Conclusions: 1) Both Random Forest and XGBoost have a good accuracy but also have high time complexity. 2) These models can be trusted since they have high AUC. 3) Random Forest is based on bagging which tends to reduce variance without affecting Bias.GBDT is based on boosting which tends to reduce Bias without affecting Variance. However, GBDT based on XGBoost takes the best of both using GBDT and Random Forest using row sampling/col. sampling.Thus, GBDT using XGBoost has better AUC than RandomForest as it is evident from the above table.