Asignment_8__Decision_Trees

April 9, 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
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
# 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
!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.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
!apt-get install -y -qq software-properties-common python-software-properties module-i
!add-apt-repository -y ppa:alessandro-strada/ppa 2>&1 > /dev/null
!apt-get update -qq 2>&1 > /dev/null
!apt-get -y install -qq google-drive-ocamlfuse fuse
from google.colab import auth
auth.authenticate_user()
from oauth2client.client import GoogleCredentials
creds = GoogleCredentials.get_application_default()
import getpass
!google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.client_secret} 
vcode = getpass.getpass()
!echo {vcode} | google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.
!mkdir drive
!google-drive-ocamlfuse drive
```

```
E: Package 'python-software-properties' has no installation candidate
ůůůůůůůůůů
mkdir: cannot create directory drive: File exists
fuse: mountpoint is not empty
fuse: if you are sure this is safe, use the 'nonempty' mount option
```

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 [0]:

```
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-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
```

```
4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

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

```
In [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 [0]:
```
#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 [0]:
```
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 [0]:
```
# 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 t
==================================================
```

```
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 alo
```

```
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 thi
```

```
In [0]: # 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', "t
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", '
                    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [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 stopwo
            preprocessed_reviews.append(sentance.strip())
```

```
100%|| 364171/364171 [03:22<00:00, 1796.50it/s]
```

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

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

[3.2] Splitting the data

```
In [24]: final['Text'] = preprocessed_reviews
         finalp = final[final.Score==1].sample(50000,random_state=2)
         finaln = final[final.Score==0].sample(50000,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    50000
0    50000
Name: Score, dtype: int64
(70000,) (30000,) (70000,) (30000,)
```

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', 'abandoned', 'abc', 'abdominal', 'abdominal pain', 'ability', 'able
==================================================
```

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', 'abandoned', 'abc', 'abdominal', 'abdon
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (70000, 41545)
the number of unique words including both unigrams and bigrams  41545
```

5.3 [4.4] Word2Vec

```
In [0]: # 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
```

```
[('terrific', 0.8704612255096436), ('fantastic', 0.8425412178039551), ('awesome', 0.8412139415
==================================================
[('nastiest', 0.8318163752555847), ('best', 0.7561851143836975), ('greatest', 0.68949103355407
```

```
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  16627
sample words  ['always', 'like', 'spaghetti', 'sauce', 'chunky', 'thick', 'not', 'one', 'added
```

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%|| 70000/70000 [02:35<00:00, 449.44it/s]
  0%|          | 0/30000 [00:00<?, ?it/s]

70000
50


100%|| 30000/30000 [01:07<00:00, 446.31it/s]

30000
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: # 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%|| 70000/70000 [09:43<00:00, 120.06it/s]
100%|| 30000/30000 [04:09<00:00, 120.03it/s]
```

6 [5] Assignment 8: Decision Trees

```
<li><strong>Apply Decision Trees 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 (best `depth` in range [1, 5, 10, 50, 100, 500, 100], and
    <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>Graphviz</strong>
    <ul>
<li>Visualize your decision tree with Graphviz. It helps you to understand how a decision is be
<li>Since feature names are not obtained from word2vec related models, visualize only BOW & TFI
<li>Make sure to print the words in each node of the decision tree instead of printing its inde
<li>Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated in
    </ul>
</li>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Find the top 20 important features from both feature sets <font color='red'>Set 1</font> an
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
```

```
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

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

7 Applying Decision Trees

[5.0.1] Decision Trees Function

In [0]:
```
def DT_tuning(ft_train,ft_test,query):
    start = datetime.now()
    #Giving Parameters for tuning
    parameters = {'max_depth':[1, 5, 10, 50, 100, 500, 100], 'min_samples_split':[5, 10,
    dt = DecisionTreeClassifier()
    clf = GridSearchCV(dt, param_grid = parameters, scoring='roc_auc', cv=5,n_jobs=4,retu
    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, 100]
    min_samples_split=[5, 10, 100, 500]

    #Making into a Dataframe for Heatmaps
    df_trainscore = pd.DataFrame(train_score_reshaped,columns=min_samples_split,index=max
    df_testscore = pd.DataFrame(test_score_reshaped,columns=min_samples_split,index=max_d
```

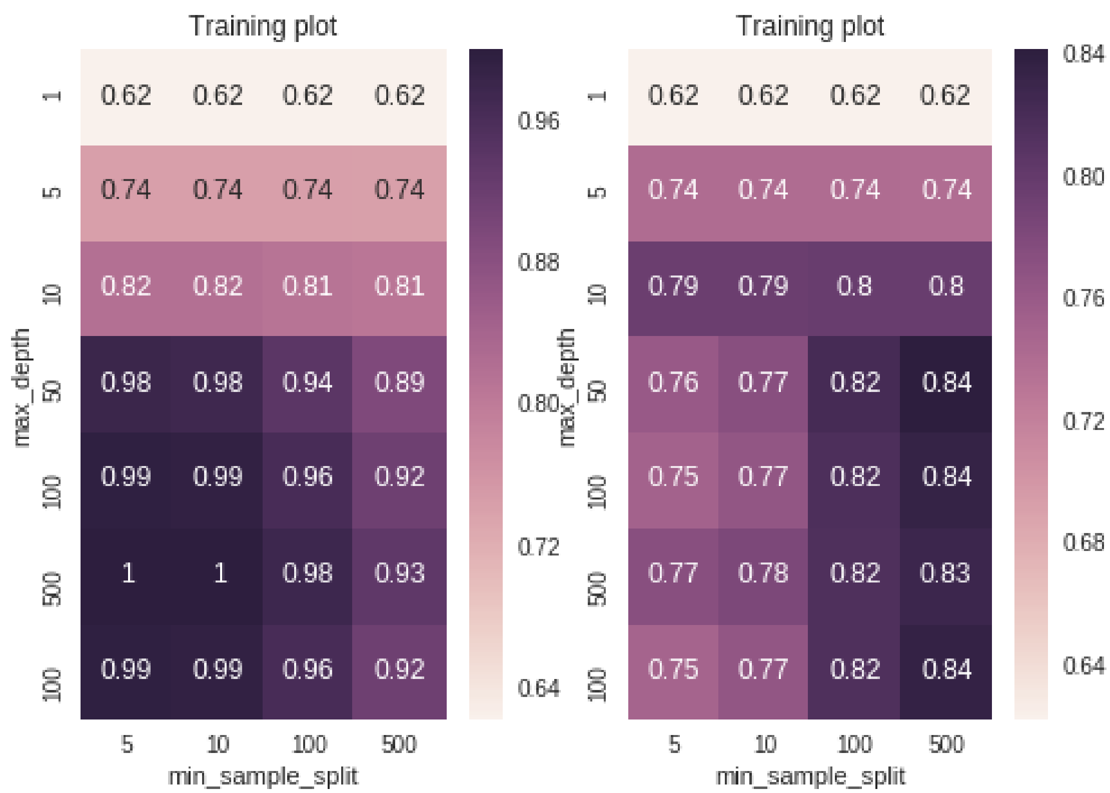
```
#Getting Max Values
train_max_value = df_trainscore.values.max()
test_max_value = df_testscore.values.max()

#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]
min_split_train = list(df_trainscore.columns[j1])[0]
max_depth_test = list(df_testscore.index[i2])[0]
min_split_test = list(df_testscore.columns[j2])[0]

#Calculating Optimal Values
max_depth_optimal = int(np.median((max_depth_train,max_depth_test)))
min_split_optimal = int(np.median((min_split_train,min_split_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('min_sample_split')
ax1.set_ylabel('max_depth')
ax2.set_title('Training plot')
ax2.set_xlabel('min_sample_split')
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 min_sample_split={} ' .format(max_de
print("="*50)

#Training model with optimal parameters
model = DecisionTreeClassifier(max_depth=max_depth_optimal,min_samples_split=min_spli
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()
```

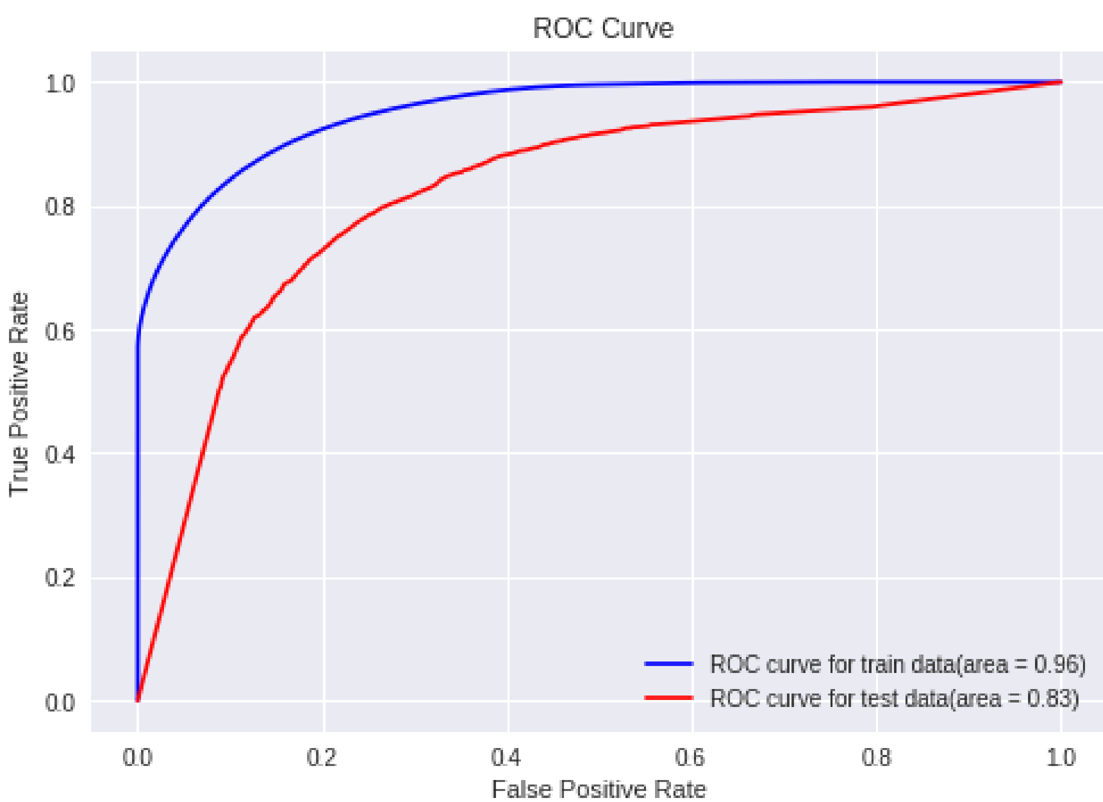
```
    fpr,tpr,_ = roc_curve(ytr,pred_train[:,1])
    roc_auc_train = roc_auc_score(ytr,pred_train[:,1])
    fpr2 = dict()
    tpr2 = dict()
    roc_auc2 = dict()
    fpr2,tpr2,_ = roc_curve(ytest,pred_test[:,1])
    roc_auc_test = roc_auc_score(ytest,pred_test[:,1])
    plt.figure()
    plt.title(" 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,min_split_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,model
```

7.1 [5.1] Applying Decision Trees on BOW, SET 1

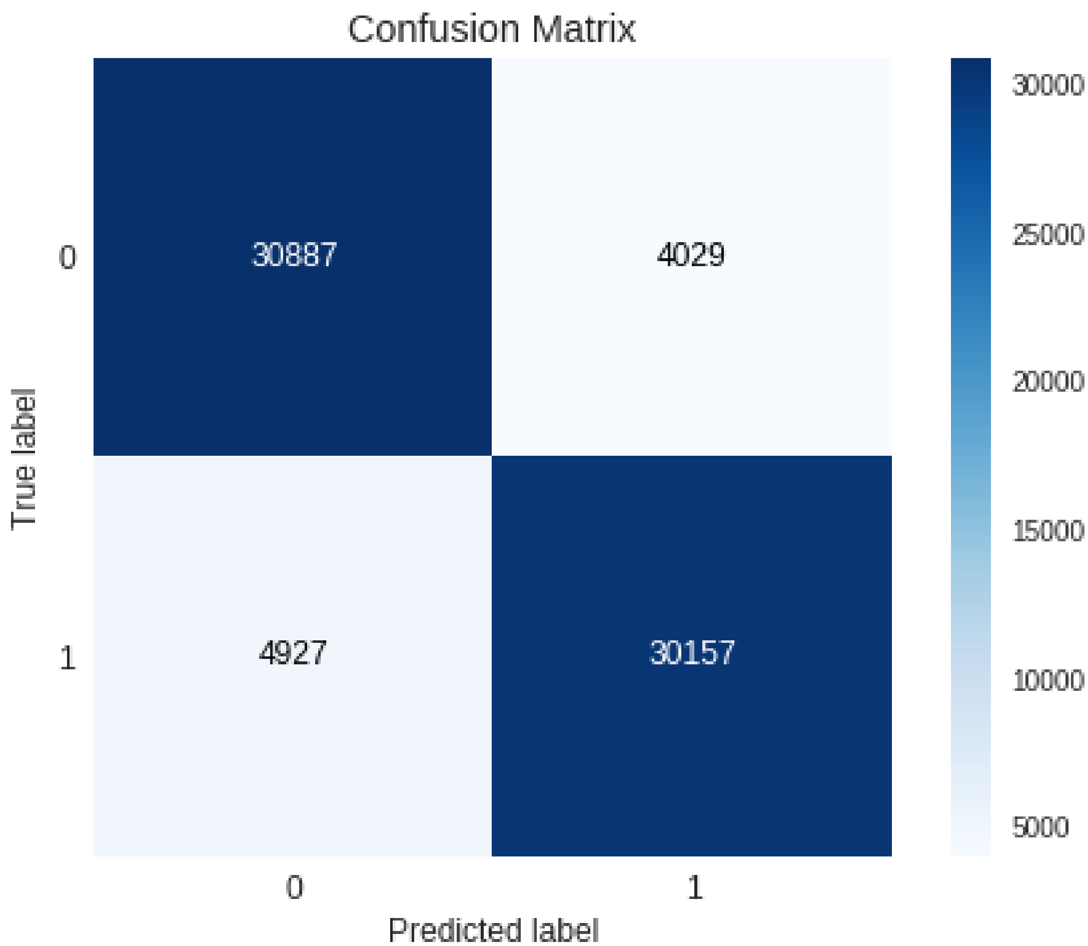
```
In [34]: w1,model1 = DT_tuning(bow_vec_tr,bow_vec_test,1)
```

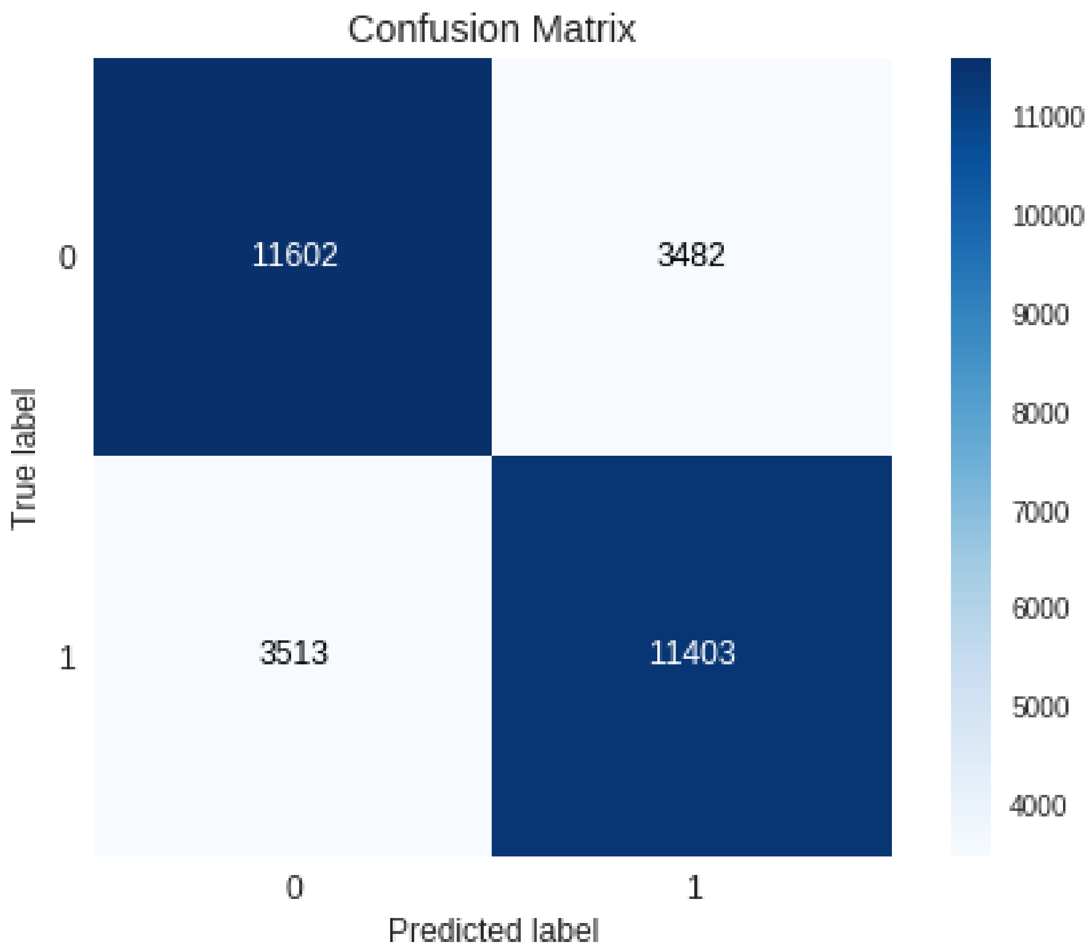
```
The maximum Train AUC is 0.999661695690313 for  500,5 . The max Validation AUC is 0.84139373932
Optimal parameters are max_depth = 275 and min_sample_split=252
==================================================
```


Training plot
Training plot
max_depth
min_sample_split
1
5
10
50
100
500
100
5
10
100
500
0.62 0.62 0.62 0.62
0.74 0.74 0.74 0.74
0.82 0.82 0.81 0.81
0.98 0.98 0.94 0.89
0.99 0.99 0.96 0.92
1 1 0.98 0.93
0.99 0.99 0.96 0.92
0.96
0.88
0.80
0.72
0.64
0.62 0.62 0.62 0.62
0.74 0.74 0.74 0.74
0.79 0.79 0.8 0.8
0.76 0.77 0.82 0.84
0.75 0.77 0.82 0.84
0.77 0.78 0.82 0.83
0.75 0.77 0.82 0.84
0.84
0.80
0.76
0.72
0.68
0.64


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve for train data(area = 0.96)
ROC curve for test data(area = 0.83)

```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.83 for test data
==================================================
Confusion Matrix for Train data
              precision    recall  f1-score   support

           0       0.86      0.88      0.87     34916
           1       0.88      0.86      0.87     35084

   micro avg       0.87      0.87      0.87     70000
   macro avg       0.87      0.87      0.87     70000
weighted avg       0.87      0.87      0.87     70000

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

           0       0.77      0.77      0.77     15084
           1       0.77      0.76      0.77     14916

   micro avg       0.77      0.77      0.77     30000
   macro avg       0.77      0.77      0.77     30000
weighted avg       0.77      0.77      0.77     30000

Time taken to run this cell : 0:51:26.700109
```


Confusion Matrix
True label
Predicted label
0
1
0
1
30887
4029
4927
30157
30000
25000
20000
15000
10000
5000



7.1.1 [5.1.1] Top 20 important features from SET 1

In [35]:
```
#Merging them into a dataframe.
features_BoW = count_bow.get_feature_names()
top_features = pd.DataFrame(w1,features_BoW)
print('Top 20 important features are:')
print(top_features[0].sort_values(ascending=False)[0:20])
```

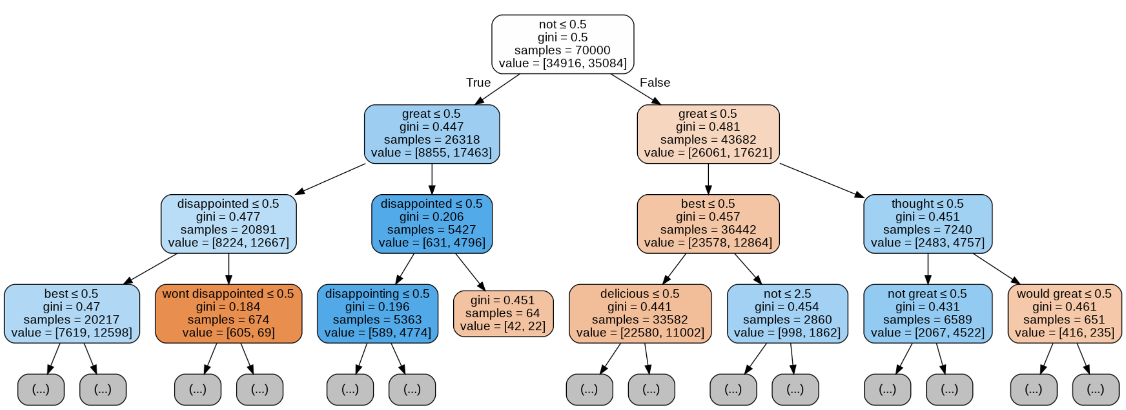
```
Top 20 important features are:
not             0.116971
great           0.078145
best            0.036701
delicious       0.032702
love            0.025702
disappointed    0.023851
good            0.021133
perfect         0.021090
```

```
loves             0.014494
favorite          0.013234
excellent         0.012873
money             0.011469
thought           0.011384
bad               0.010969
worst             0.008971
easy              0.008386
unfortunately     0.008356
not good          0.008006
wonderful         0.007909
nice              0.007163
Name: 0, dtype: float64
```

7.1.2 [5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

```
In [38]: #For Graphviz #referenced from STACKOVERFLOW
         from graphviz import Source
         from sklearn import tree
         graph = Source( tree.export_graphviz(model1, out_file=None, feature_names=features_BoW
         png_bow = graph.pipe(format='png')
         with open('dtreeBoW.png','wb') as f:
             f.write(png_bow)

         from IPython.display import Image
         Image(png_bow)
```

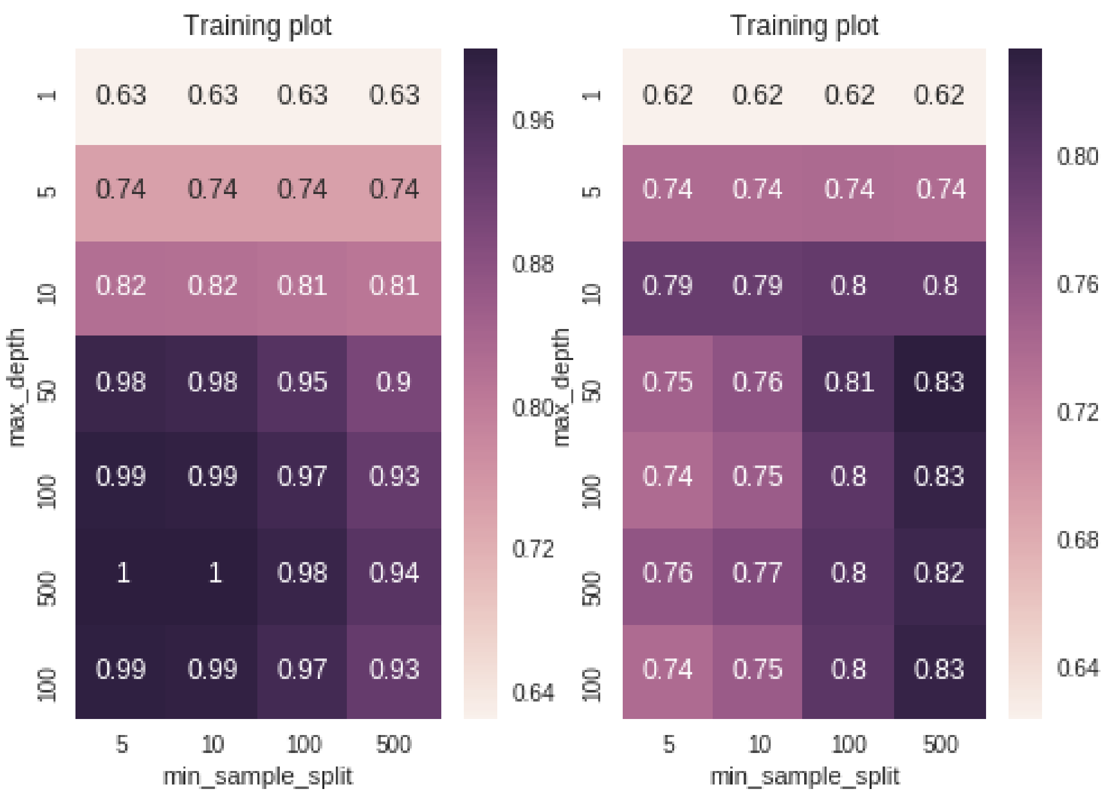
Out[38]:

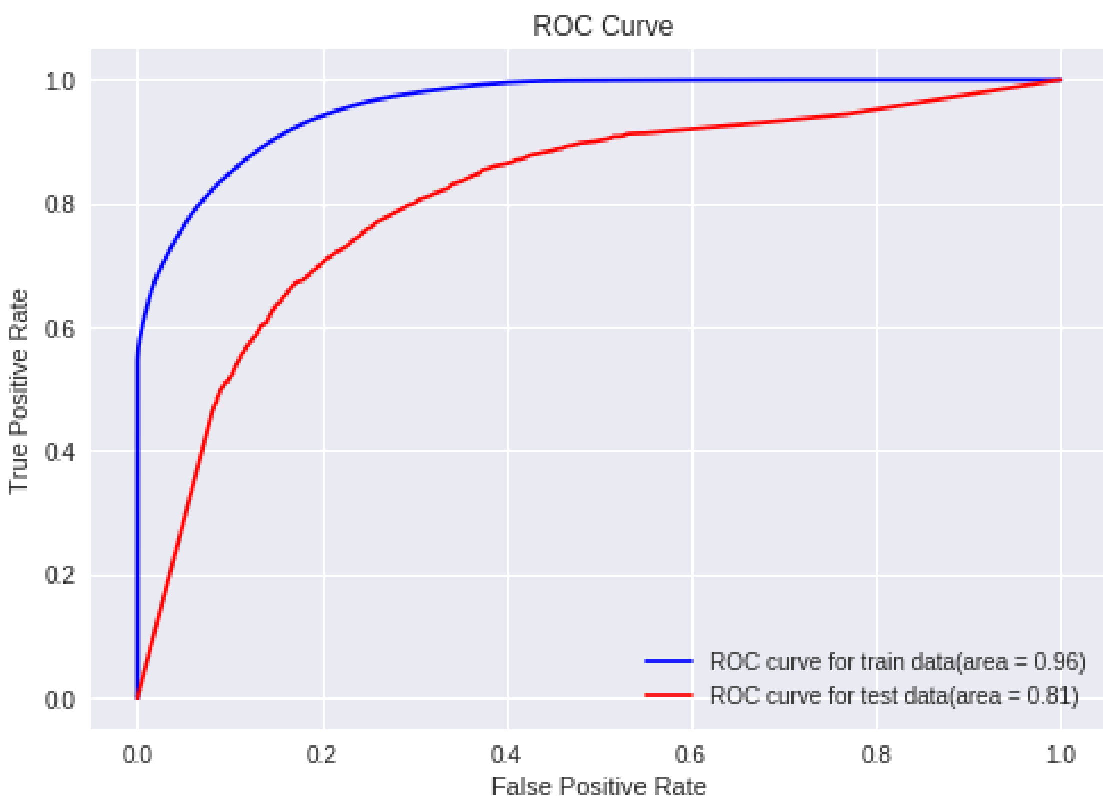


7.2 [5.2] Applying Decision Trees on TFIDF, SET 2

```
In [39]: w2,model2 = DT_tuning(tfidf_tr,tfidf_test,1)
```

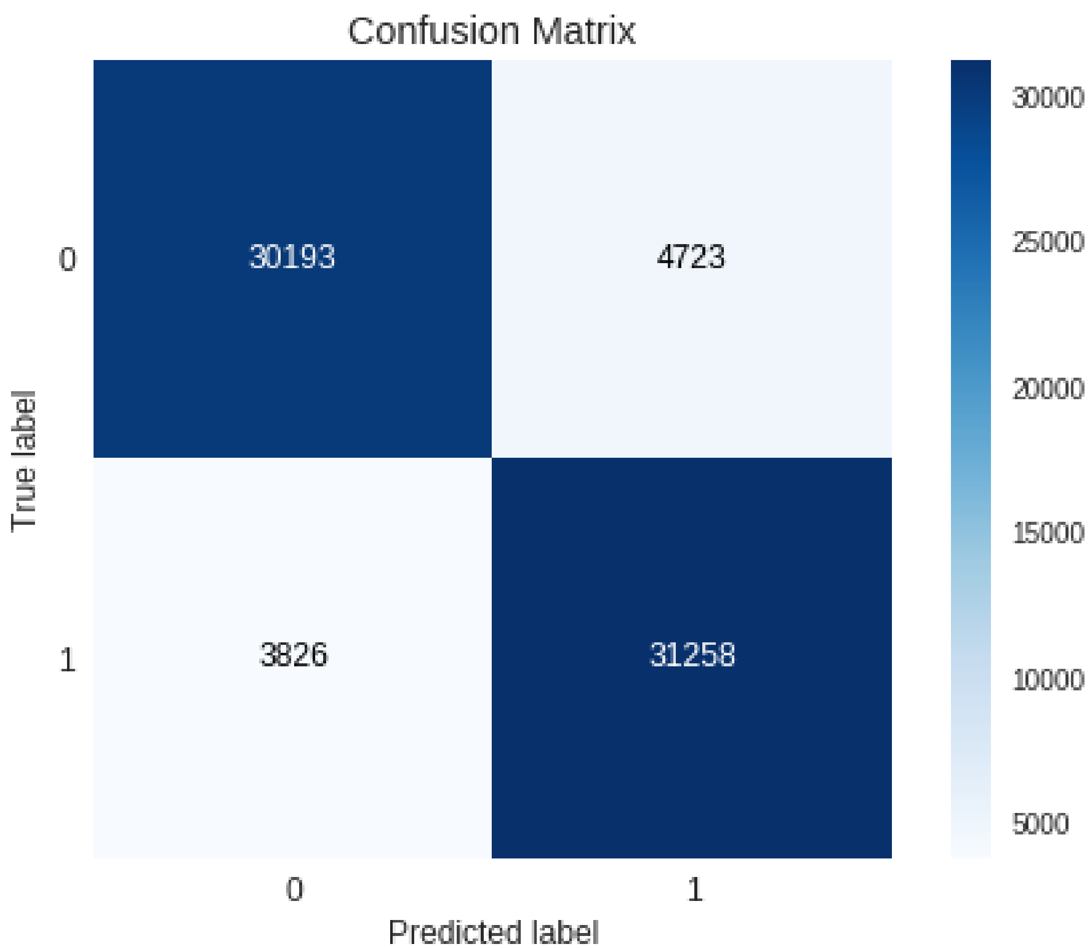
```
The maximum Train AUC is 0.999724788158891 for  500,5 . The max Validation AUC is 0.83379989345
Optimal parameters are max_depth = 275 and min_sample_split=252
==================================================
```

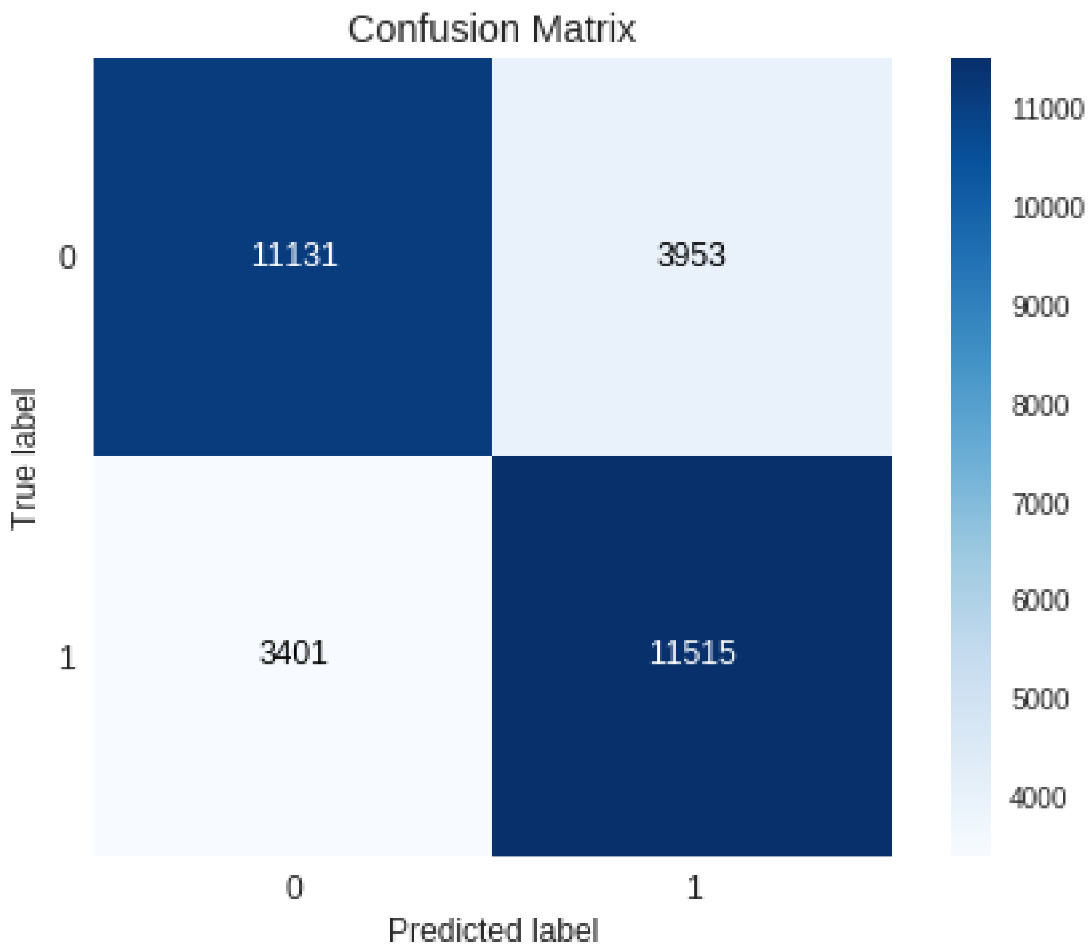





```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.81 for test data
==================================================
Confusion Matrix for Train data
              precision    recall  f1-score   support

           0       0.89      0.86      0.88     34916
           1       0.87      0.89      0.88     35084

   micro avg       0.88      0.88      0.88     70000
   macro avg       0.88      0.88      0.88     70000
weighted avg       0.88      0.88      0.88     70000

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

           0       0.77      0.74      0.75     15084
           1       0.74      0.77      0.76     14916

   micro avg       0.75      0.75      0.75     30000
   macro avg       0.76      0.75      0.75     30000
weighted avg       0.76      0.75      0.75     30000
```

```
Time taken to run this cell : 0:59:32.593036
```





7.2.1 [5.2.1] Top 20 important features from SET 2

```
In [40]: #Merging them into a dataframe.
         features_tfidf = tf_idf_vect.get_feature_names()
         top_features2 = pd.DataFrame(w2,features_tfidf)
         print('Top 20 important features are:')
         print(top_features2[0].sort_values(ascending=False)[0:20])
```

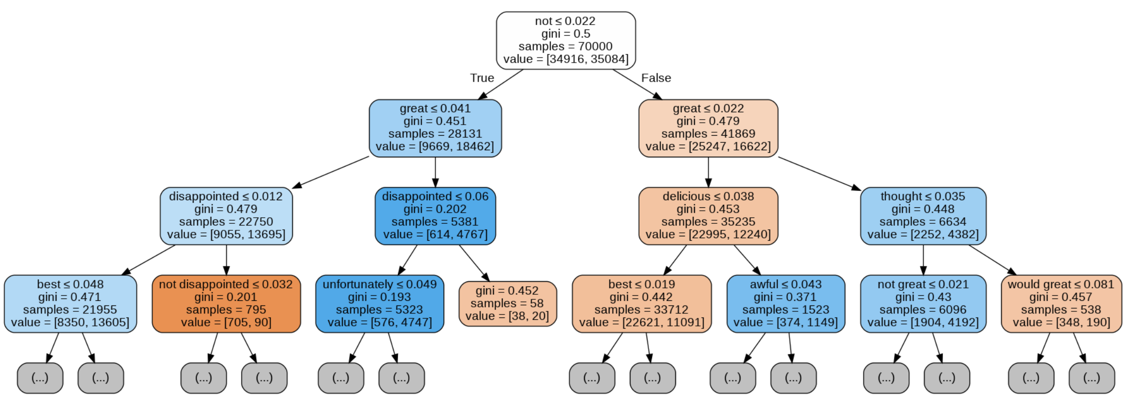
```
Top 20 important features are:
not                0.112687
great              0.076376
best               0.034427
delicious          0.033351
love               0.028606
disappointed       0.025647
good               0.023622
perfect            0.020485
```

```
loves            0.014477
bad              0.013540
favorite         0.013046
excellent        0.012737
money            0.011831
easy             0.009245
thought          0.009189
worst            0.008782
unfortunately    0.008473
nice             0.008045
wonderful        0.007828
awful            0.007327
Name: 0, dtype: float64
```

7.2.2 [5.2.2] Graphviz visualization of Decision Tree on TFIDF, SET 2

In [41]:
```
#For Graphviz #referenced from STACKOVERFLOW
graph = Source( tree.export_graphviz(model2, out_file=None, feature_names=features_tf
png_tfidf = graph.pipe(format='png')
with open('dtreetfidf.png','wb') as f:
    f.write(png_tfidf)

from IPython.display import Image
Image(png_tfidf)
```

Out[41]:

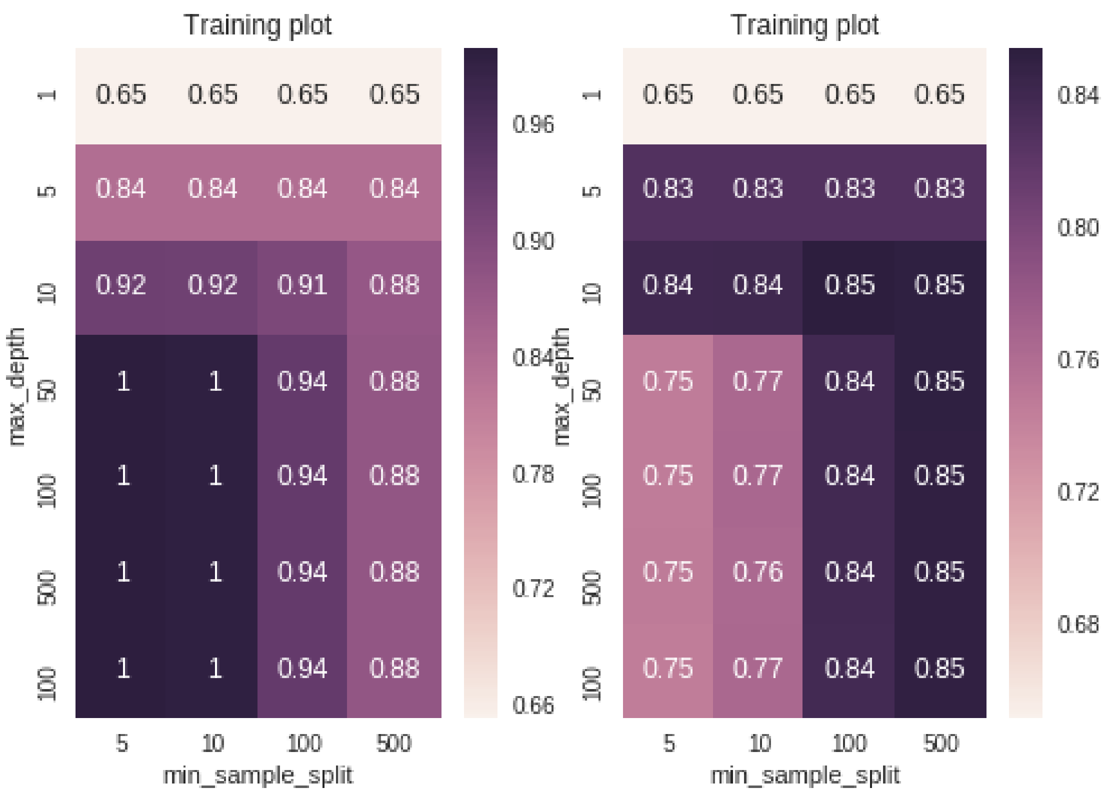


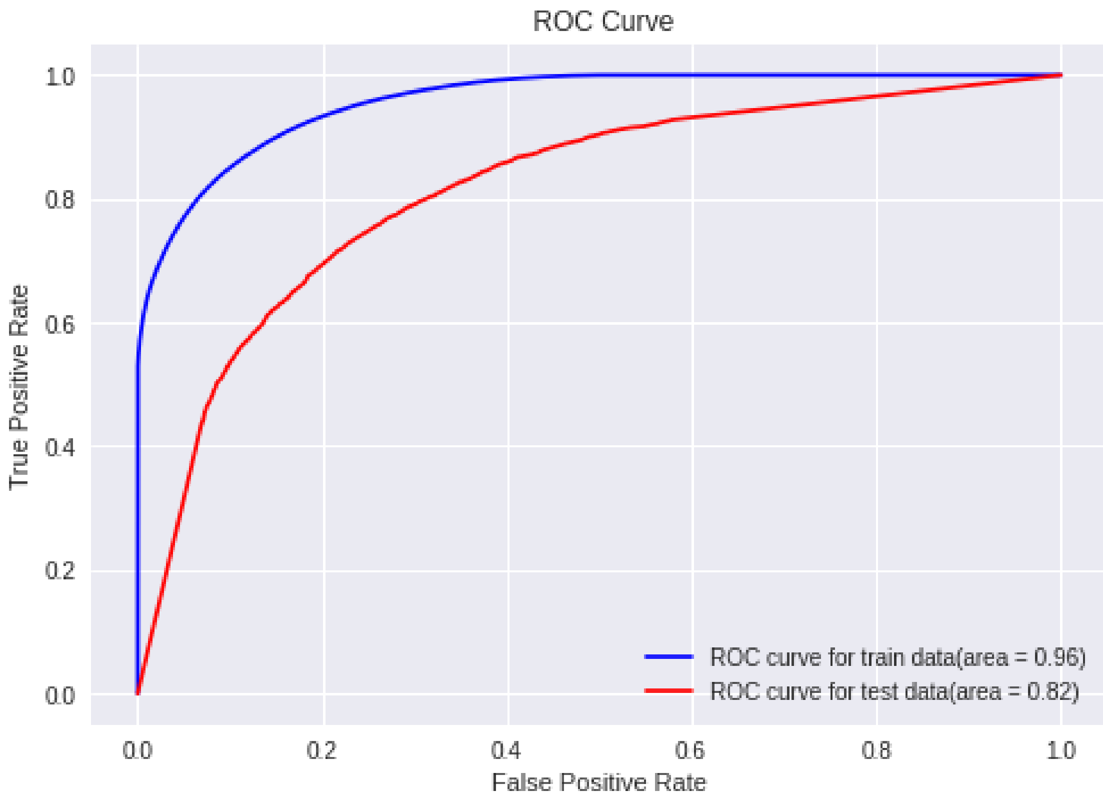
7.3 [5.3] Applying Decision Trees on AVG W2V, SET 3

In [42]:
```
DT_tuning(sent_vectors_intr,sent_vectors_intest,0)
```

```
The maximum Train AUC is 0.99956890106638 for  500,5 . The max Validation AUC is 0.85384889249
Optimal parameters are max_depth = 255 and min_sample_split=52
```

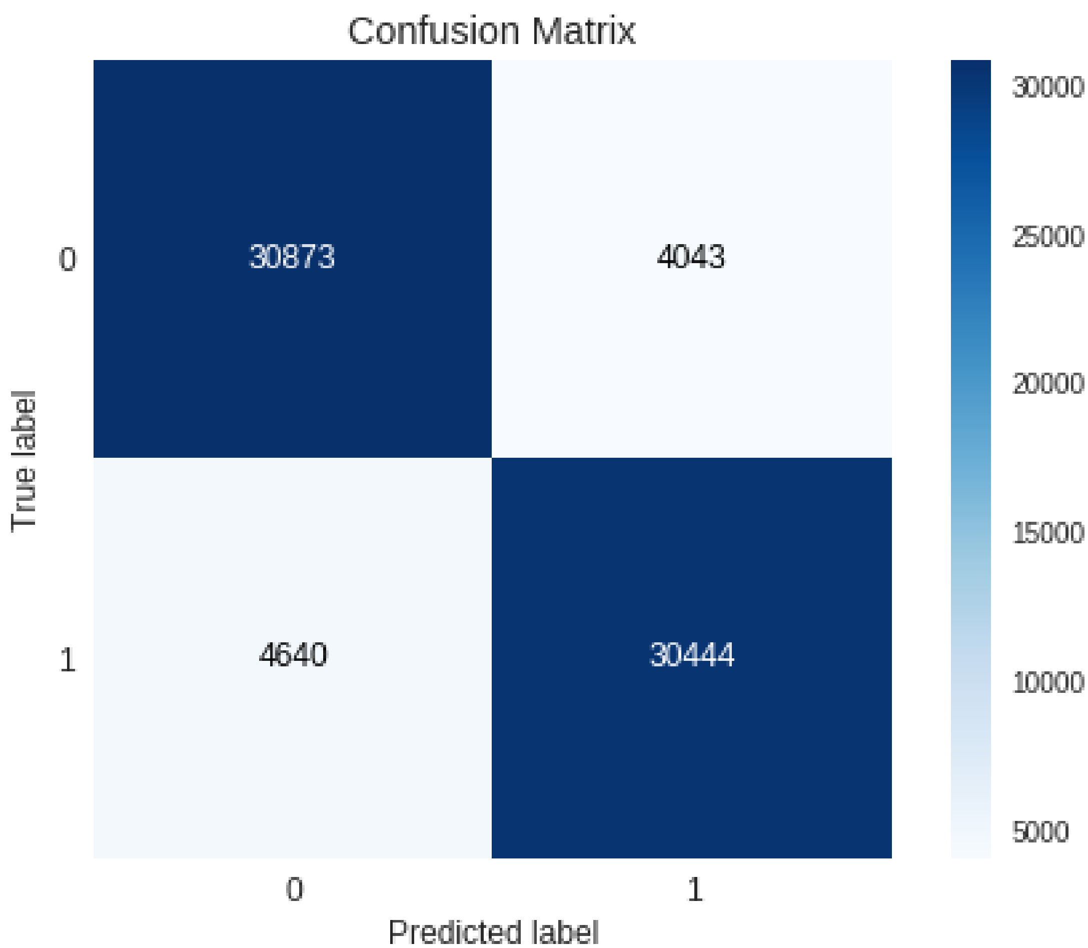
==

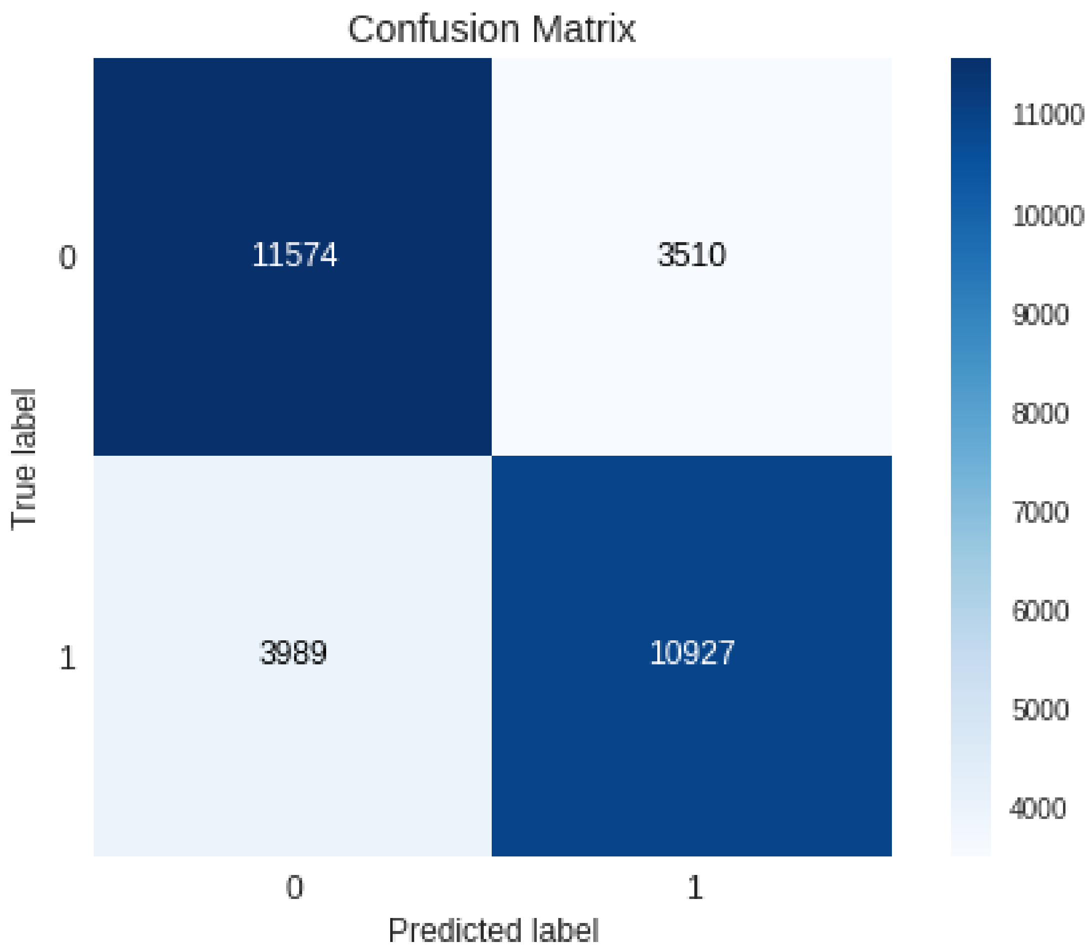

Training plot
max_depth
1
5
10
50
100
500
100
min_sample_split
5
10
100
500
0.65 0.65 0.65 0.65
0.84 0.84 0.84 0.84
0.92 0.92 0.91 0.88
1 1 0.94 0.88
1 1 0.94 0.88
1 1 0.94 0.88
1 1 0.94 0.88
0.96
0.90
0.84
0.78
0.72
0.66
Training plot
max_depth
1
5
10
50
100
500
100
min_sample_split
5
10
100
500
0.65 0.65 0.65 0.65
0.83 0.83 0.83 0.83
0.84 0.84 0.85 0.85
0.75 0.77 0.84 0.85
0.75 0.77 0.84 0.85
0.75 0.76 0.84 0.85
0.75 0.77 0.84 0.85
0.84
0.80
0.76
0.72
0.68




```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.82 for test data
==================================================
Confusion Matrix for Train data
              precision    recall  f1-score   support

           0       0.87      0.88      0.88     34916
           1       0.88      0.87      0.88     35084

   micro avg       0.88      0.88      0.88     70000
   macro avg       0.88      0.88      0.88     70000
weighted avg       0.88      0.88      0.88     70000

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

           0       0.74      0.77      0.76     15084
           1       0.76      0.73      0.74     14916

   micro avg       0.75      0.75      0.75     30000
   macro avg       0.75      0.75      0.75     30000
weighted avg       0.75      0.75      0.75     30000
```

```
Time taken to run this cell : 0:12:07.193103
```

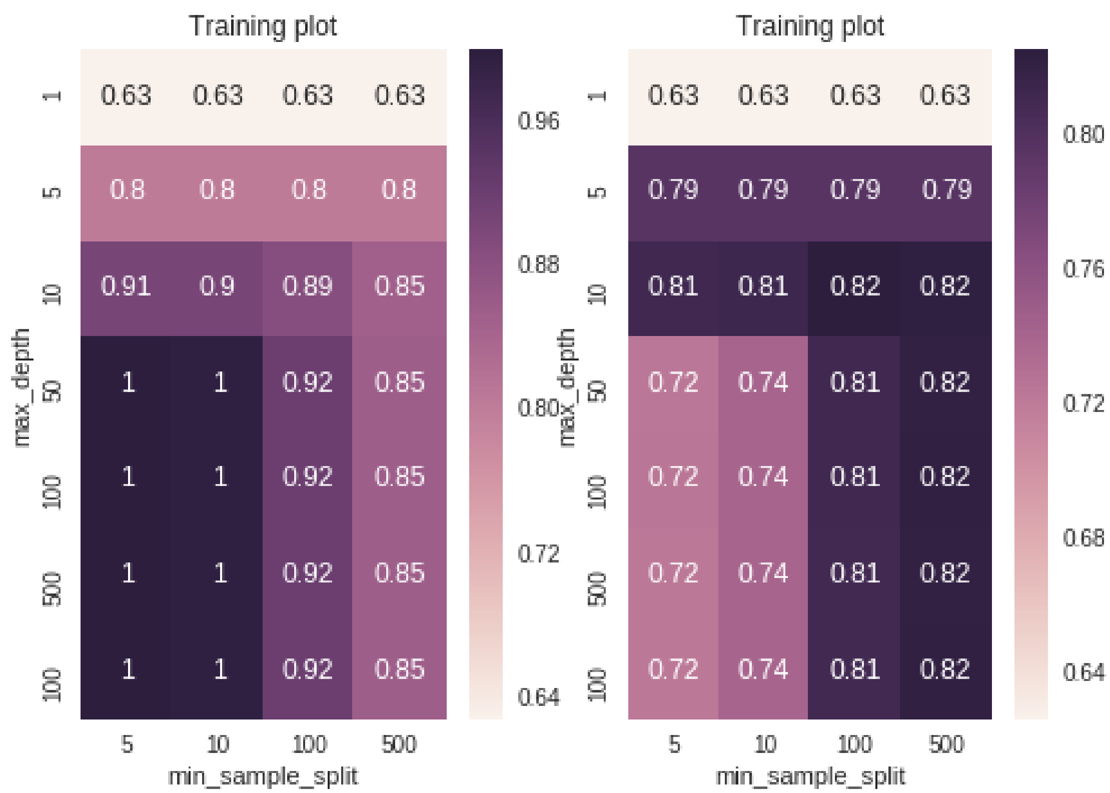


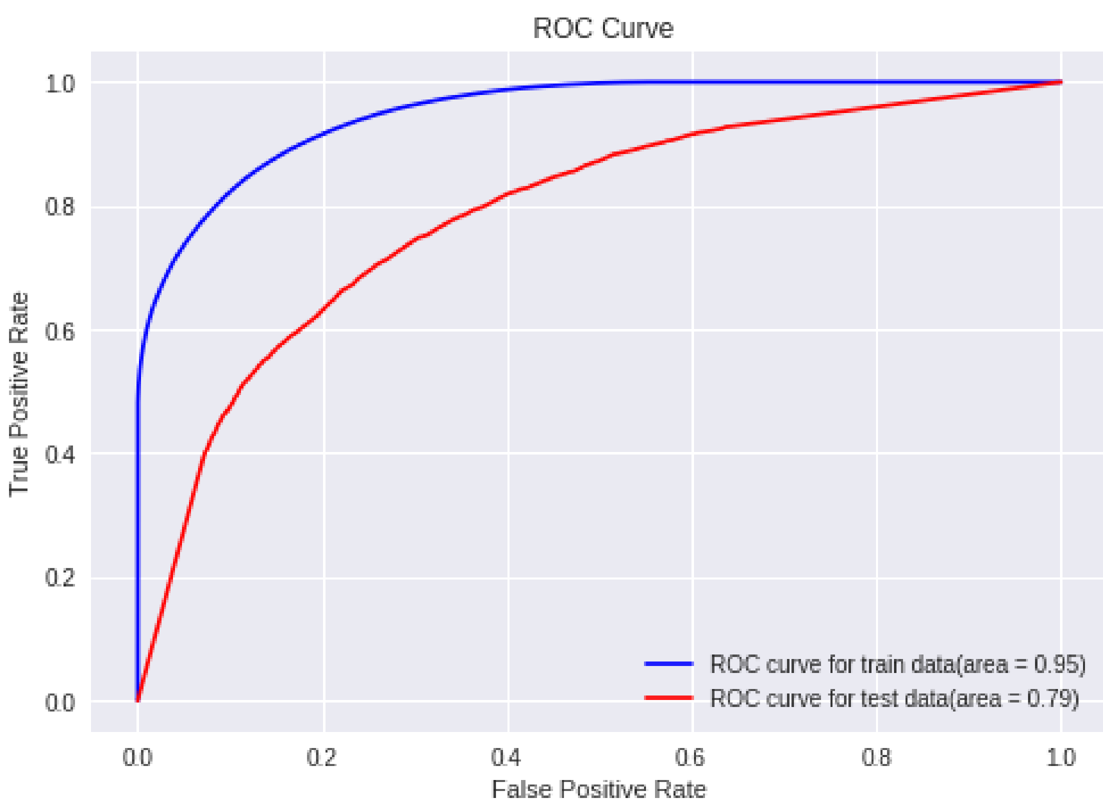


7.4 [5.4] Applying Decision Trees on TFIDF W2V, SET 4

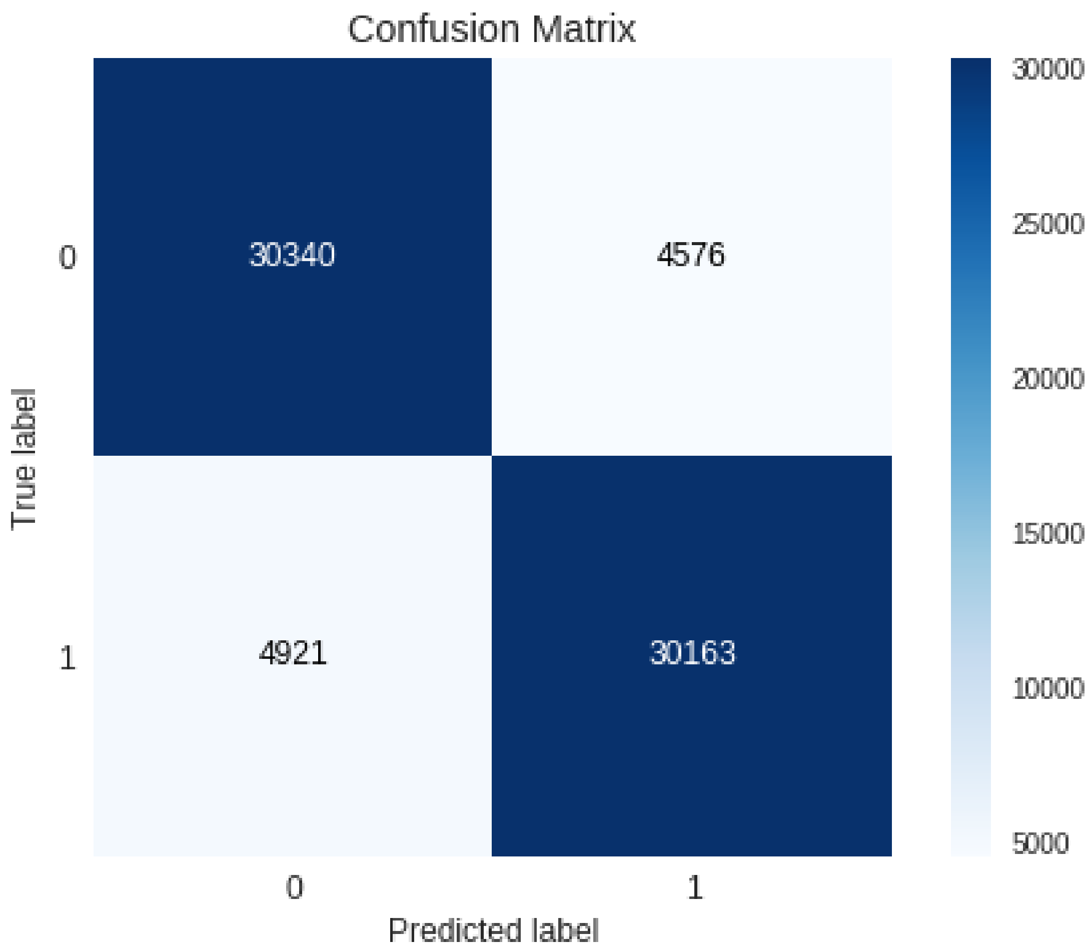
```
In [43]: DT_tuning(tfidf_sent_vectors_intr,tfidf_sent_vectors_intest,0)
```

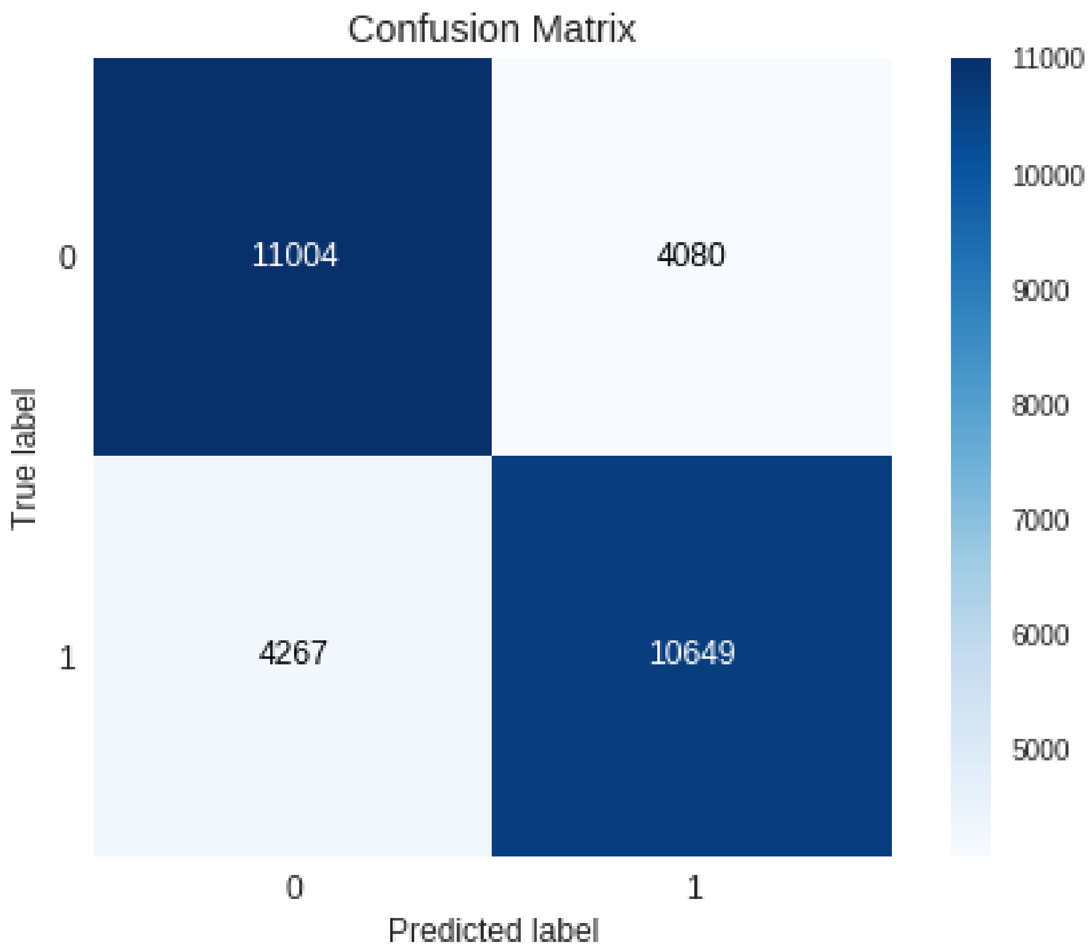
```
The maximum Train AUC is 0.9994835354235768 for  50,5 . The max Validation AUC is 0.82490993760
Optimal parameters are max_depth = 30 and min_sample_split=52
==================================================
```


Training plot
max_depth
min_sample_split
Training plot
max_depth
min_sample_split


ROC Curve
True Positive Rate
False Positive Rate
ROC curve for train data(area = 0.95)
ROC curve for test data(area = 0.79)

```
This is the ROC_AUC curve using optimal parameters with ROC_AUC of 0.79 for test data
==================================================
Confusion Matrix for Train data
              precision    recall  f1-score   support

           0       0.86      0.87      0.86     34916
           1       0.87      0.86      0.86     35084

   micro avg       0.86      0.86      0.86     70000
   macro avg       0.86      0.86      0.86     70000
weighted avg       0.86      0.86      0.86     70000

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

           0       0.72      0.73      0.73     15084
           1       0.72      0.71      0.72     14916

   micro avg       0.72      0.72      0.72     30000
   macro avg       0.72      0.72      0.72     30000
weighted avg       0.72      0.72      0.72     30000

Time taken to run this cell : 0:11:54.372935
```


Confusion Matrix
True label
0
1
30340
4576
4921
30163
0
1
Predicted label
30000
25000
20000
15000
10000
5000



8 [6] Conclusions

```
In [3]: x = PrettyTable()

        x.field_names = ["Vectorizer", "max_depth", " min_samples_split", "AUC"]
        x.add_row(["BoW", 275, 252, 0.83])
        x.add_row(["Tfidf", 275, 252, 0.81])
        x.add_row(["Avg W2V", 255, 52, 0.82])
        x.add_row(["Tfidf weighted W2V", 30, 52, 0.79])
        print(x)
```

```
+--------------------+-----------+-------------------+------+
|     Vectorizer     | max_depth |  min_samples_split | AUC  |
+--------------------+-----------+-------------------+------+
|        BoW         |    275    |        252        | 0.83 |
|       Tfidf        |    275    |        252        | 0.81 |
|      Avg W2V       |    255    |         52        | 0.82 |
```

Tfidf weighted W2V	30	52	0.79

Observations: 1) All the models have a high AUC, which means the hyper parameter tuning gave optimal parameters and trusted models.

2) Decision Trees are highly interpretable as can be seen from the graphviz.As the depth increases, interpretability increases.