Amazon Fine Food Review - Random Forest and GBDT

1. Objective

To find a review whether positive or negative

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import warnings
import sqlite3
warnings.filterwarnings("ignore")
```

2. Data Cleaning

In [2]:
```
#connecting database

con=sqlite3.connect("database.sqlite")

# Read data from database

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

# Removal of Duplicates

pre_data=raw_data.drop_duplicates(['UserId','ProfileName','Time','Text'],keep="f

# Removal of Unconditioning data (denominator>numerator)

pre_data=pre_data[pre_data.HelpfulnessNumerator<=pre_data.HelpfulnessDenominator


# Finding NaN values in dataframe

# Reference
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.isnull.html

# Findind NaN values

if pre_data.isnull().values.any() == False:
    print("There is No NaN values in the DataFrame")
else:
    print(" There is NaN values present in the DataFrame")
```

```
There is No NaN values in the DataFrame
```

In [3]:
```
# sort data based on Time

filter_data=pre_data.sort_values(by=["Time"],axis=0)

# Class Label changing
# positive class label = 1
# negative class label = 0
a=[]
for i in filter_data["Score"]:
    if i > 3:
        a.append(1)
    else:
        a.append(0)
filter_data["Score"]=a
```

In [4]:
```
filter_data.shape
```

Out[4]:
```
(364171, 10)
```

In [5]:
```
filter_data["Score"].value_counts()
```

Out[5]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

3. Text Preprocessing

- We took the Text column for the further review idendification task, because text is the most important feature compared to other features.

In [6]:
```
# References
# https://medium.com/@jorlugaqui/how-to-strip-html-tags-from-a-string-in-python
# https://stackoverflow.com/a/40823105/4084039
# https://stackoverflow.com/questions/19790188/expanding-english-language-contra
# https://stackoverflow.com/questions/18082130/python-regex-to-remove-all-words
# https://stackoverflow.com/questions/5843518/remove-all-special-characters-punc
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://gist.github.com/sebleier/554280
# stemming tutorial: https://www.geeksforgeeks.org/python-stemming-words-with-nl
# Lemmatisation tutorial: https://www.geeksforgeeks.org/python-lemmatization-wit
# NLTK Stemming package list: https://www.nltk.org/api/nltk.stem.html

from nltk.stem.snowball import EnglishStemmer
import re
from tqdm import tqdm
stemmer=EnglishStemmer()
```

In [7]:
```
raw_text_data=filter_data["Text"].values
```

In [8]:
```
# Stopwords

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

# expanding contractions

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 [9]:
```
preprocessed_text_data=[]
for i in tqdm(raw_text_data):
# removing of HTML tags
    a=re.sub("<.*?>"," ",i)
# removing url
    b=re.sub(r"http\S+"," ",a)
# expanding contractions
    c=decontracted(b)
# removing alpha_numeric
    d=re.sub("\S*\d\S*", " ",c)
# removing Special characters
    e=re.sub('[^A-Za-z0-9]+', ' ',d)
# removing stopwords
    k=[]
    for w in e.split():
        if w.lower() not in stopwords:
            s=(stemmer.stem(w.lower())).encode('utf8')
            k.append(s)
    preprocessed_text_data.append(b' '.join(k).decode())
```

```
100%|██████████| 364171/364171 [06:51<00:00, 884.81it/s]
```

In [10]:
```
filter_data["Text"]=preprocessed_text_data
```

In [11]:
```
filter_data.shape
```

Out[11]:
```
(364171, 10)
```

In [12]:
```
# we took the sample data size as 100k

final_data=filter_data[:100000]
final_data.shape
```

Out[12]: (100000, 10)

4. Data Splitting

In [13]:
```
# References
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tra

from sklearn.model_selection import train_test_split
```

In [14]:
```
X=final_data.Text
Y=final_data.Score
```

In [15]:
```
x_1,x_test,y_1,y_test=train_test_split(X,Y,test_size=0.2,random_state=40)
x_train,x_cv,y_train,y_cv=train_test_split(x_1,y_1,test_size=0.25,random_state=4
print(" Train data Size")
print(x_train.shape,y_train.shape)

print("cv data size")
print(x_cv.shape,y_cv.shape)
print("Test data size")
print(x_test.shape,y_test.shape)
```

```
 Train data Size
(60000,) (60000,)
cv data size
(20000,) (20000,)
Test data size
(20000,) (20000,)
```

5. Featurization

5.1 Bag of Words (BOW)

In [16]:
```
# Reference
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.t

from sklearn.feature_extraction.text import CountVectorizer
```

In [17]:
```
bow_model=CountVectorizer(ngram_range=(1,2),min_df=5,max_features=500)

# BOW on Train data

bow_train_vec1=bow_model.fit_transform(x_train)

# BOW on cv data

bow_cv_vec1=bow_model.transform(x_cv)

# BOW on Test data

bow_test_vec1=bow_model.transform(x_test)
```

In [18]:
```
# the number of words in BOW or Vector size

print("The size of BOW vectorizer")
print(bow_train_vec1.get_shape()[1])
```

```
The size of BOW vectorizer
500
```

5.2 TFIDF

In [19]:
```
# References
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.t

from sklearn.feature_extraction.text import TfidfVectorizer
```

In [20]:
```
tfidf_model=TfidfVectorizer(ngram_range=(1,2),min_df=5,max_features=500)

# TFIDF on Train data

tfidf_train_vec1=tfidf_model.fit_transform(x_train)

# TFIDF on cv data

tfidf_cv_vec1=tfidf_model.transform(x_cv)

# TFIDF on Test data

tfidf_test_vec1=tfidf_model.transform(x_test)
```

In [21]:
```
# the number of words in BOW or Vector size

print("The size of TFIDF vectorizer")
print(tfidf_train_vec1.get_shape()[1])
```

```
The size of TFIDF vectorizer
500
```

5.3 W2V

In [22]:
```
# References
# https://radimrehurek.com/gensim/models/word2vec.html
# https://machinelearningmastery.com/develop-word-embeddings-python-gensim/
# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY

from gensim.models import Word2Vec
```

In [23]:
```
list_sentences_train=[]
for i in tqdm(list(x_train)):
    list_sentences_train.append(i.split())
```

```
100%|██████████| 60000/60000 [00:00<00:00, 156353.17it/s]
```

In [24]:
```
word2vec_model=Word2Vec(list_sentences_train,min_count=5,size=50,workers=4)
```

In [25]:
```
word2vec_words_train=list(word2vec_model.wv.vocab)
print(" Number of words")
print("_________________")
print(" ")
print(len(word2vec_words_train))
print("="*125)
print(" sample words")
print("_________________")
print(" ")
print(word2vec_words_train[100:150])
```

```
 Number of words
_________________

10407
===============================================================================
==============================================
 sample words
_________________

['told', 'carri', 'lot', 'use', 'product', 'mani', 'dish', 'marinad', 'flavor',
'beat', 'pungent', 'yet', 'smooth', 'bring', 'meat', 'imagin', 'prefer', 'cold'
, 'press', 'great', 'way', 'nice', 'abl', 'pour', 'spray', 'bottom', 'line', 'l
over', 'beefeat', 'went', 'profit', 'health', 'pet', 'sad', 'pro', 'treat', 'st
ill', 'made', 'usa', 'bottl', 'help', 'tremend', 'adjust', 'daycar', 'pump', 'm
other', 'end', 'day', 'babi', 'hungri']
```

In [26]:
```
# list of sentences cv data

list_sentences_cv=[]
for i in tqdm(list(x_cv)):
    list_sentences_cv.append(i.split())

# list of sentences test data

list_sentences_test=[]
for i in tqdm(list(x_test)):
    list_sentences_test.append(i.split())
```

```
100%|██████████| 20000/20000 [00:00<00:00, 53909.87it/s]
100%|██████████| 20000/20000 [00:00<00:00, 162475.77it/s]
```

5.4 Avg W2V

In [27]:
```
# Reference
# formula of Avg word2vec = sum of all (wi)[i=0 to n]/n

# avg word2vec on training data

avg_word2vec_train=[]
for i in tqdm(list_sentences_train):
    vector=np.zeros(50)
    no_of_words=0
    for k in i:
        try:
            w2v_data=word2vec_model.wv[k]
            vector=vector+w2v_data
            no_of_words=no_of_words+1
        except:
            pass
    if no_of_words != 0:
        vector=vector/no_of_words
    avg_word2vec_train.append(vector)
avg_w2v_train=np.asmatrix(avg_word2vec_train)
print("shape of Avg Word2vec train")
print(avg_w2v_train.shape)
```

```
100%|██████████| 60000/60000 [00:16<00:00, 3648.94it/s]

shape of Avg Word2vec train
(60000, 50)
```

In [28]:
```
# avg word2vec on cv data

avg_word2vec_cv=[]
for i in tqdm(list_sentences_cv):
    vector=np.zeros(50)
    no_of_words=0
    for k in i:
        try:
            w2v_data=word2vec_model.wv[k]
            vector=vector+w2v_data
            no_of_words=no_of_words+1
        except:
            pass
    if no_of_words != 0:
        vector=vector/no_of_words
    avg_word2vec_cv.append(vector)
avg_w2v_cv=np.asmatrix(avg_word2vec_cv)
print("shape of Avg Word2vec cv")
print(avg_w2v_cv.shape)
```

```
100%|██████████| 20000/20000 [00:05<00:00, 3644.05it/s]

shape of Avg Word2vec cv
(20000, 50)
```

In [29]:
```
# avg word2vec on test data

avg_word2vec_test=[]
for i in tqdm(list_sentences_test):
    vector=np.zeros(50)
    no_of_words=0
    for k in i:
        try:
            w2v_data=word2vec_model.wv[k]
            vector=vector+w2v_data
            no_of_words=no_of_words+1
        except:
            pass
    if no_of_words != 0:
        vector=vector/no_of_words
    avg_word2vec_test.append(vector)
avg_w2v_test=np.asmatrix(avg_word2vec_test)
print("shape of Avg Word2vec test")
print(avg_w2v_test.shape)
```

```
100%|██████████| 20000/20000 [00:05<00:00, 3472.69it/s]

shape of Avg Word2vec test
(20000, 50)
```

5.5 TFIDF W2V

In [30]:
```
# References
# https://stackoverflow.com/questions/21553327
# https://github.com/devBOX03


# tfidf word2vec on training data

model=TfidfVectorizer()
tfidf_w2v_model=model.fit_transform(x_train)
tfidf_w2v=model.get_feature_names()
tfidf_word2vec_train=[]
row=0
for i in tqdm(list_sentences_train):
    vec=np.zeros(50)
    weight_sum=0
    for w in i:
        try:
            w2v_freq=word2vec_model.wv[w]
            tfidf_freq=tfidf_w2v_model[row,tfidf_w2v.index(w)]
            vec=vec+(w2v_freq*tfidf_freq)
            weight_sum=weight_sum+tfidf_freq
        except:
            pass
    vec=vec/weight_sum
    tfidf_word2vec_train.append(vec)
    row=row+1
tfidf_w2v_train=np.asmatrix(tfidf_word2vec_train)
print("Shape of TFIDF word2vec train")
print(tfidf_w2v_train.shape)
```

```
100%|██████████| 60000/60000 [23:49<00:00, 41.96it/s]

Shape of TFIDF word2vec train
(60000, 50)
```

In [31]:
```
# tfidf word2vec on cv data

tfidf_w2v_model=model.transform(x_cv)
tfidf_word2vec_cv=[]
row=0
for i in tqdm(list_sentences_cv):
    vec=np.zeros(50)
    weight_sum=0
    for w in i:
        try:
            w2v_freq=word2vec_model.wv[w]
            tfidf_freq=tfidf_w2v_model[row,tfidf_w2v.index(w)]
            vec=vec+(w2v_freq*tfidf_freq)
            weight_sum=weight_sum+tfidf_freq
        except:
            pass
    vec=vec/weight_sum
    tfidf_word2vec_cv.append(vec)
    row=row+1
tfidf_w2v_cv=np.asmatrix(tfidf_word2vec_cv)
print("Shape of TFIDF word2vec cv")
print(tfidf_w2v_cv.shape)
```

```
100%|██████████| 20000/20000 [07:51<00:00, 42.41it/s]

Shape of TFIDF word2vec cv
(20000, 50)
```

In [32]:
```
# tfidf word2vec on test data

tfidf_w2v_model=model.transform(x_test)
tfidf_word2vec_test=[]
row=0
for i in tqdm(list_sentences_test):
    vec=np.zeros(50)
    weight_sum=0
    for w in i:
        try:
            w2v_freq=word2vec_model.wv[w]
            tfidf_freq=tfidf_w2v_model[row,tfidf_w2v.index(w)]
            vec=vec+(w2v_freq*tfidf_freq)
            weight_sum=weight_sum+tfidf_freq
        except:
            pass
    vec=vec/weight_sum
    tfidf_word2vec_test.append(vec)
    row=row+1
tfidf_w2v_test=np.asmatrix(tfidf_word2vec_test)
print("Shape of TFIDF word2vec test")
print(tfidf_w2v_test.shape)
```

```
100%|██████████| 20000/20000 [08:12<00:00, 40.65it/s]

Shape of TFIDF word2vec test
(20000, 50)
```

6. Random Forest Model

6.1 Creating function for Random Forest

In [64]:
```
# References
# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFores
# ROC_CURVE:https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ro
# ROC_AUC_CURVE: https://scikit-learn.org/stable/modules/generated/sklearn.metri
# AUC_CURVE:https://scikit-learn.org/stable/modules/generated/sklearn.metrics.au
# CONFUSION_MATRIX:https://scikit-learn.org/stable/modules/generated/sklearn.met

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,roc_auc_score,roc_curve
import math
```

In [65]:
```
# References for Python Functions:
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/functi
# https://www.geeksforgeeks.org/functions-in-python/
# https://www.geeksforgeeks.org/g-fact-41-multiple-return-values-in-python/

# Fuction for Hyper parameter Tuning

def Random_Forest(**para):

    auc_train=[]
    auc_cv=[]

    for i,j in tqdm(zip(para["no_tree"],para["depth"])):
        model=RandomForestClassifier(n_estimators=i,max_depth=j,class_weight="ba
        model.fit(para["train_vector"],para['train_label'])

    # Prediction of training data

        train_proba=model.predict_proba(para["train_vector"])
        train=roc_auc_score(para["train_label"],train_proba[:,1])
        auc_train.append(train)

    # Prediction of cv data

        cv_proba=model.predict_proba(para["cv_vector"])
        cv=roc_auc_score(para["cv_label"],cv_proba[:,1])
        auc_cv.append(cv)

    return auc_train,auc_cv
```

In [66]:
```
def best_RF (**para):

    # Model training

    model=RandomForestClassifier(n_estimators=para["best_tree"],max_depth=para['
    model.fit(para["train_vector"],para['train_label'])

    # training data

    DT_train_proba=model.predict_proba(para["train_vector"])
    train_proba=DT_train_proba
    fpr_train,tpr_train,thres_train=roc_curve(para["train_label"],DT_train_proba
    auc_train=roc_auc_score(para["train_label"],DT_train_proba[:,1])

    # test data

    DT_test_proba=model.predict_proba(para["test_vector"])
    test_proba=DT_test_proba
    fpr_test,tpr_test,thres_test=roc_curve(para["test_label"],DT_test_proba[:,1]
    auc_test=roc_auc_score(para["test_label"],DT_test_proba[:,1])

    return train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_trai
```

In [67]:
```
# References
# https://pythonprogramming.net/matplotlib-3d-scatterplot-tutorial/

from mpl_toolkits.mplot3d import Axes3D
```

In [77]:
```
# References
# https://stackoverflow.com/questions/6282058/writing-numerical-values-on-the-pl
#https://matplotlib.org/api/_as_gen/matplotlib.pyplot.annotate.html
# https://pythonprogramming.net/matplotlib-3d-scatterplot-tutorial/

# Fuction for plotting AUC values

def auc_score(**para):

    plt.close()
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(para["tree"],para["depth"],para["auc_train"], c='b', marker='o',labe
    ax.plot(para["tree"],para["depth"],para["auc_cv"],c="r",marker='o',label="AU
    ax.set_xlabel('Tree')
    ax.set_ylabel('Depth')
    ax.set_zlabel('Auc_ score')
    plt.title("Hyperparameter Tuning")
    plt.legend()
    plt.show()
```

In [76]:
```
def roc_model(**para):
    plt.close()
    plt.plot(para["fpr_train"],para["tpr_train"],"green",label="ROC curve of Tra
    plt.plot(para["fpr_test"],para["tpr_test"],"red",label="ROC curve of Test da
    plt.plot([0, 1], [0, 1], color='blue',linestyle='--',label="Center of ROC Cu
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC curve")
    plt.legend()
    plt.show()
```

In [70]:
```
# References
# confusion matrix of Train and Test data
# https://stackoverflow.com/questions/47264597/confusion-matrix-from-probabilit
# plotting confusion matrix: https://seaborn.pydata.org/generated/seaborn.heatma


# Function for confusion matrix

def cm_plot(**para):
    #  confusion matrix of training data
    train_pred_cm=np.argmax(para["train_proba"],axis=1)
    train_confusion_matrix=confusion_matrix(para["train_label"],train_pred_cm,la
    train_cm=pd.DataFrame(train_confusion_matrix,index=["Negative","Positive"],c

    # confusion matrix of test data

    test_pred_cm=np.argmax(para["test_proba"],axis=1)
    test_confusion_matrix=confusion_matrix(para["test_label"],test_pred_cm,label
    test_cm=pd.DataFrame(test_confusion_matrix,index=["Negative","Positive"],col

    plt.close()
    plt.figure(1,figsize=(10,10))
    plt.subplot(211)
    sns.heatmap(train_cm,annot=True,fmt='d')
    plt.title("Confusion matrix of Train Data")
    plt.subplot(212)
    sns.heatmap(test_cm,annot=True,fmt='d')
    plt.title("Confusion matrix of Test Data")
    plt.show()
```

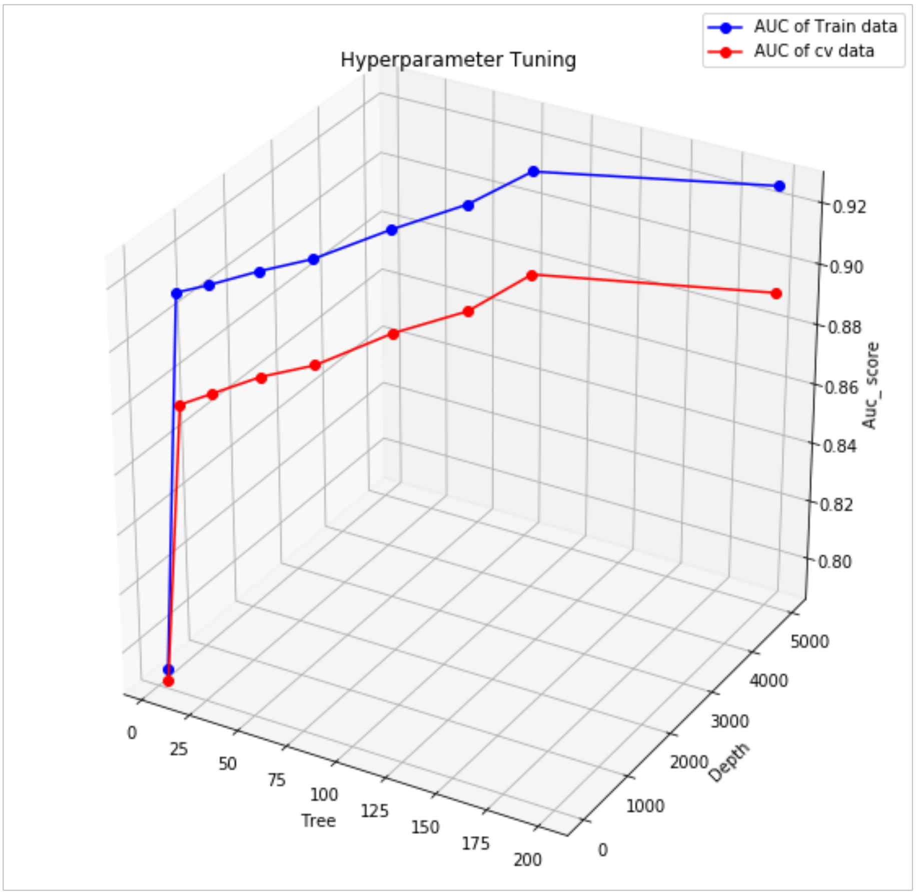
6.2 Random Forest using BOW

In [43]:
```
tree=[5,15,30,45,60,75,90,100,200]
depth=[5,50,100,500,1000,2000,3000,4000,5000]
```

In [44]:
```
# Hyperparameter tuning

auc_train,auc_cv=Random_Forest(no_tree=tree,depth=depth,train_vector=bow_train_v
                                                            cv_vector=bow_cv_vec1,cv_label=y_cv
```

```
9it [05:32, 58.90s/it]
```

In [45]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

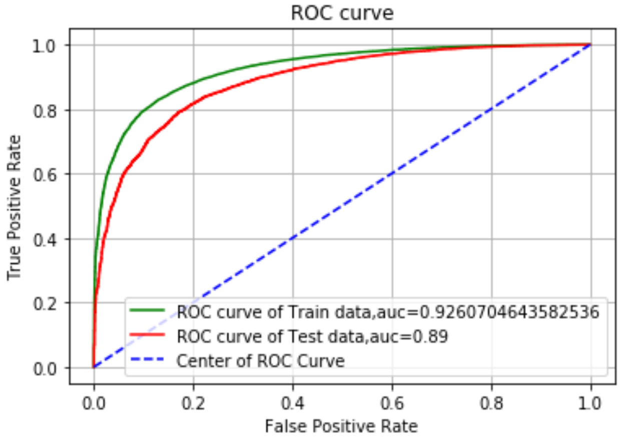


Observation:

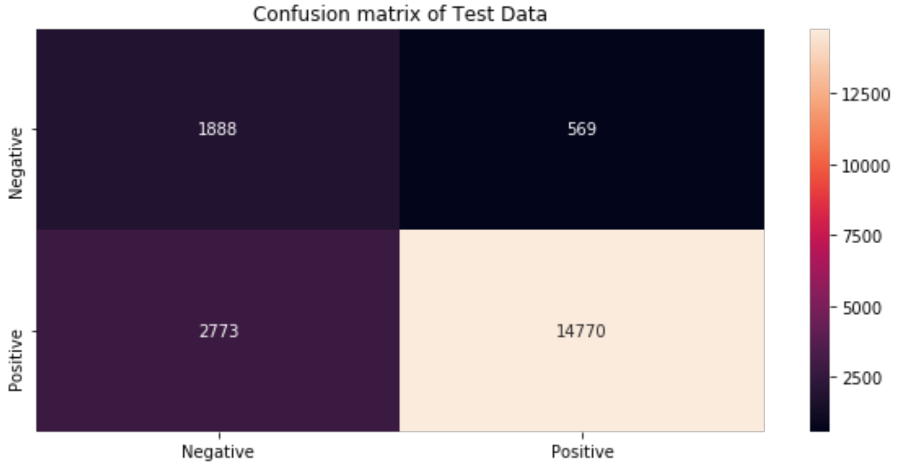
- To avoid overfitting and underfitting,choose (no of base learners=90,depth=3000), we get auc_score=0.85

In [46]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_RF(best_tree=90,best_depth=3000,train_vector=bow_train_vec1,train_label=y_
                                          test_vector=bow_test_vec1,test_label=y_test)
```

In [47]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
        text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [48]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```



Observation:

- When we applying best hyperparameter (no of base learners=90,depth =3000) on model, we get auc score of future unseen data is 0.89

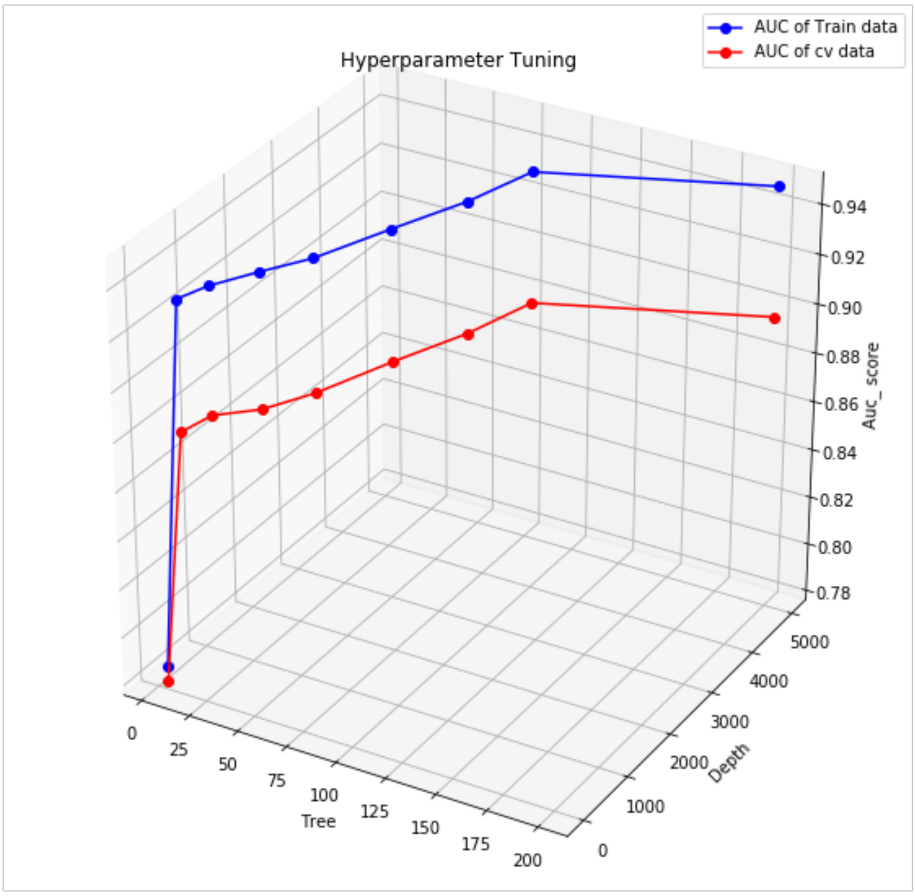
6.3 Random Forest using TFIDF

In [49]:
```
tree=[5,15,30,45,60,75,90,100,200]
depth=[5,50,100,500,1000,2000,3000,4000,5000]
```

In [50]:
```
# Hyperparameter tuning

auc_train,auc_cv=Random_Forest(no_tree=tree,depth=depth,train_vector=tfidf_trair
                                                cv_vector=tfidf_cv_vec1,cv_label=y_
```

```
9it [06:28, 67.93s/it]
```

In [51]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

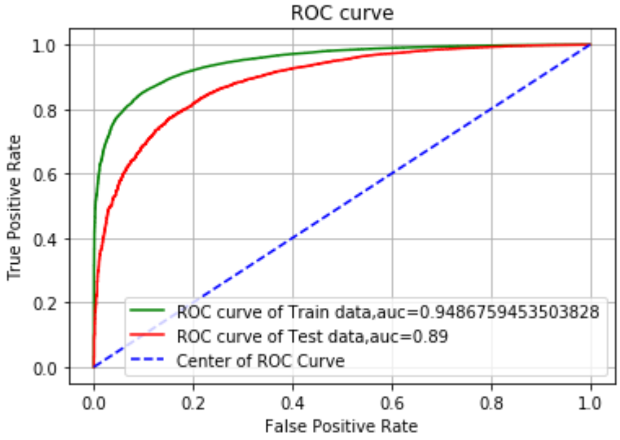


Observation:

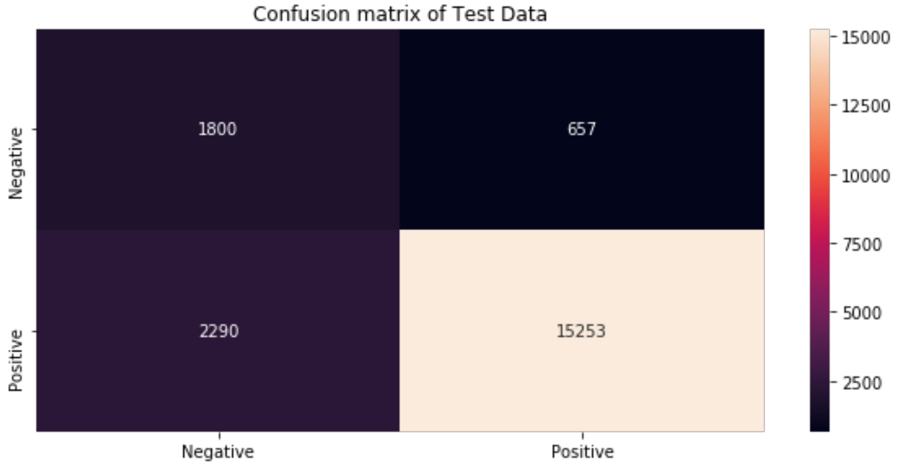
- To avoid overfitting and underfitting,choose (no of base learners=90,depth=3000), we get auc_score=0.86

In [52]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_RF(best_tree=90,best_depth=3000,train_vector=tfidf_train_vec1,train_label=
                              test_vector=tfidf_test_vec1,test_label=y_test)
```

In [53]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi
# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [54]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```



Observation:

- When we applying best hyperparameter (no of base learners=90,depth =3000) on model, we get auc score of future unseen data is 0.89

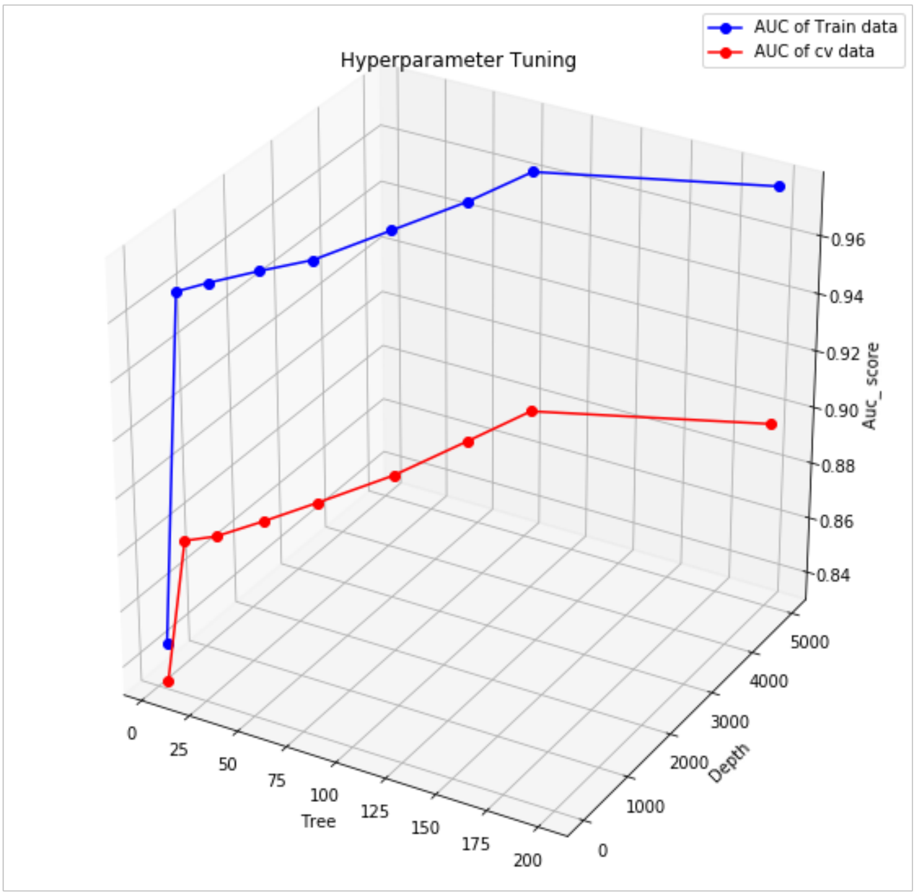
6.4 Random Forest using Avg W2V

In [55]:
```
tree=[5,15,30,45,60,75,90,100,200]
depth=[5,50,100,500,1000,2000,3000,4000,5000]
```

In [56]:
```
# Hyperparameter tuning

auc_train,auc_cv=Random_Forest(no_tree=tree,depth=depth,train_vector=avg_w2v_tra
                                                  cv_vector=avg_w2v_cv,cv_label=y_cv)
```

```
9it [06:00, 63.41s/it]
```

In [57]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

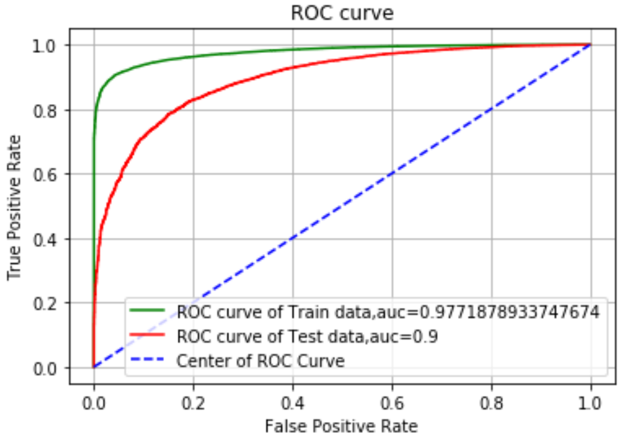


Observation:

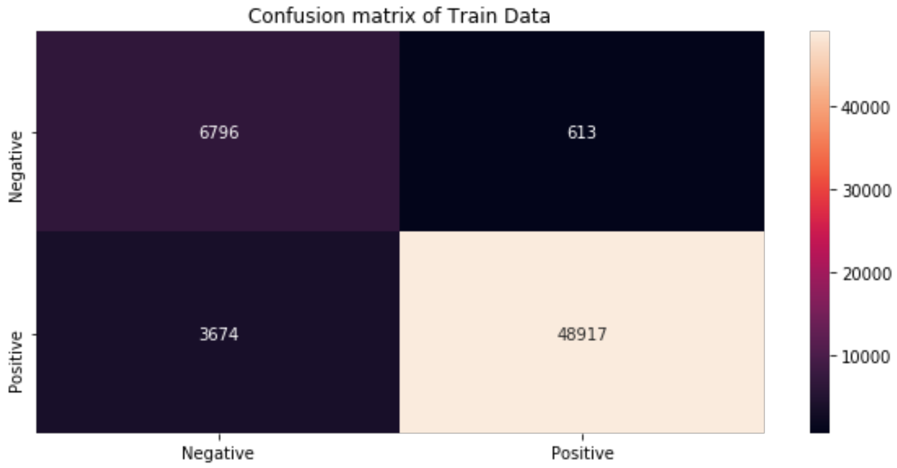
- To avoid overfitting and underfitting,choose (no of base learners=90,depth=3000), we get auc_score=0.86

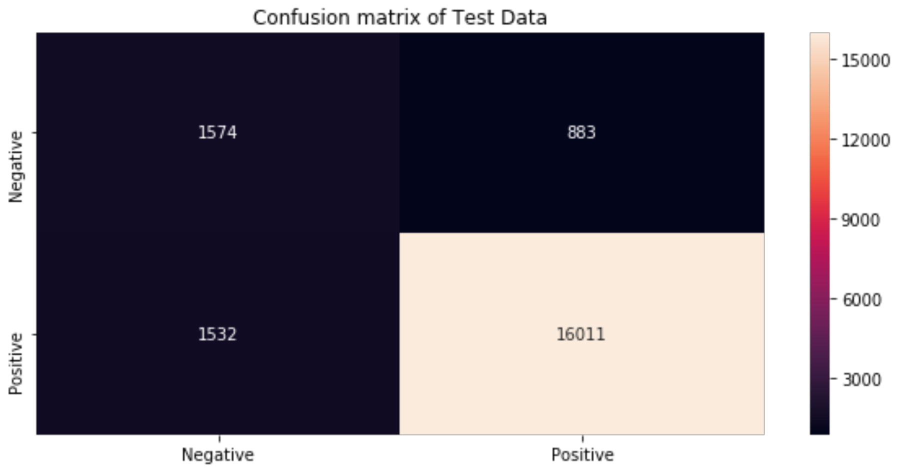
In [58]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_RF(best_tree=90,best_depth=3000,train_vector=avg_w2v_train,train_label=y_t
                               test_vector=avg_w2v_test,test_label=y_test)
```

In [59]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [60]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=90,depth =3000) on model, we get auc score of future unseen data is 0.90

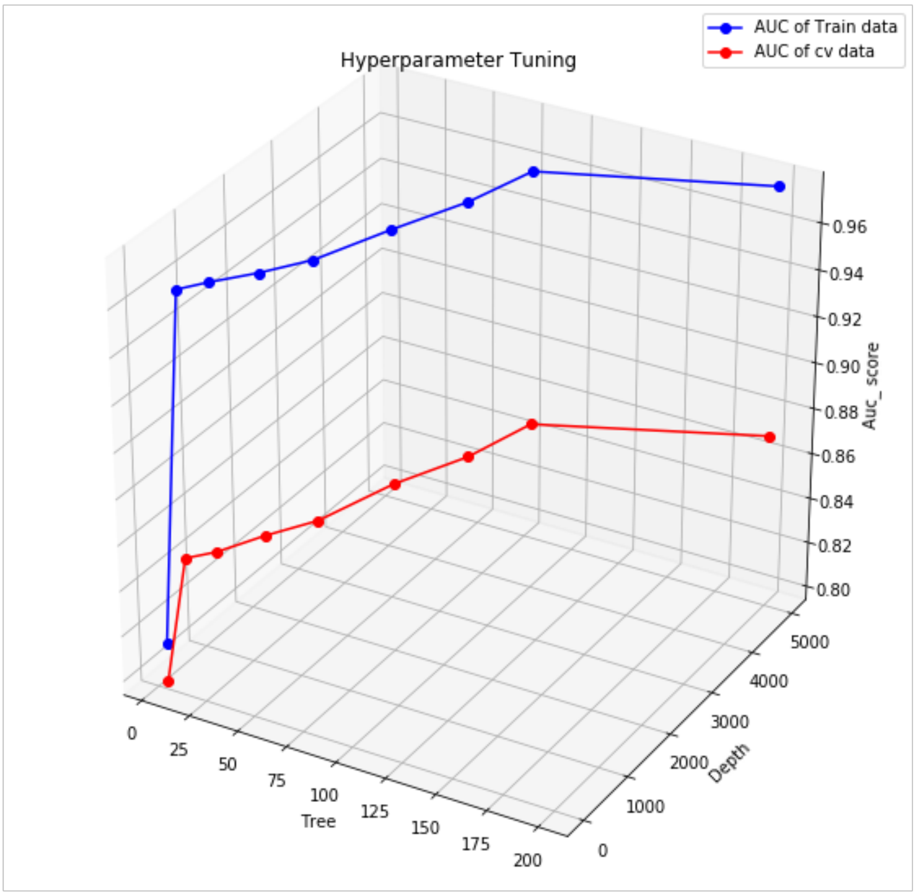
6.5 Random Forest using TFIDF W2V

In [61]:
```
tree=[5,15,30,45,60,75,90,100,200]
depth=[5,50,100,500,1000,2000,3000,4000,5000]
```

In [62]:
```
# Hyperparameter tuning

auc_train,auc_cv=Random_Forest(no_tree=tree,depth=depth,train_vector=tfidf_w2v_t
                                                          cv_vector=tfidf_w2v_cv,cv_label=y_c
```

```
9it [06:17, 66.40s/it]
```

In [63]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

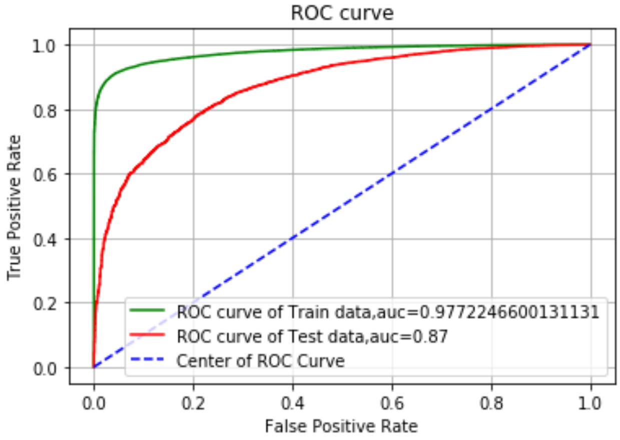


Observation:

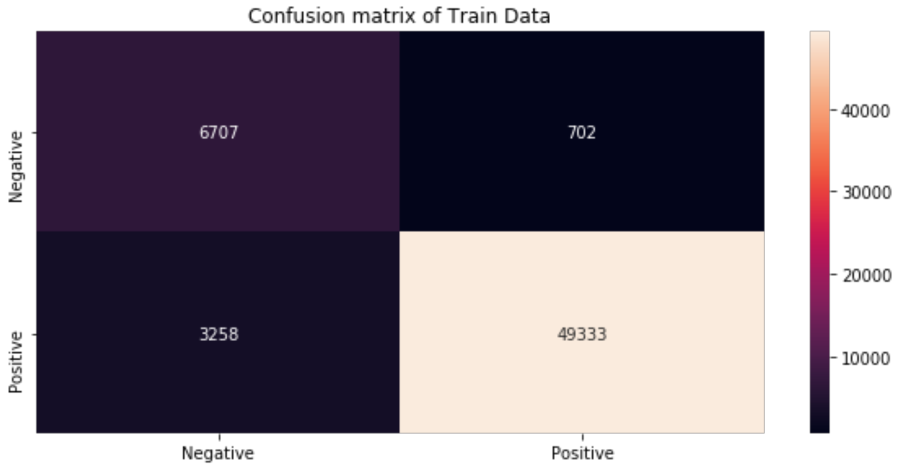
- To avoid overfitting and underfitting,choose (no of base learners=90,depth=3000), we get auc_score=0.82

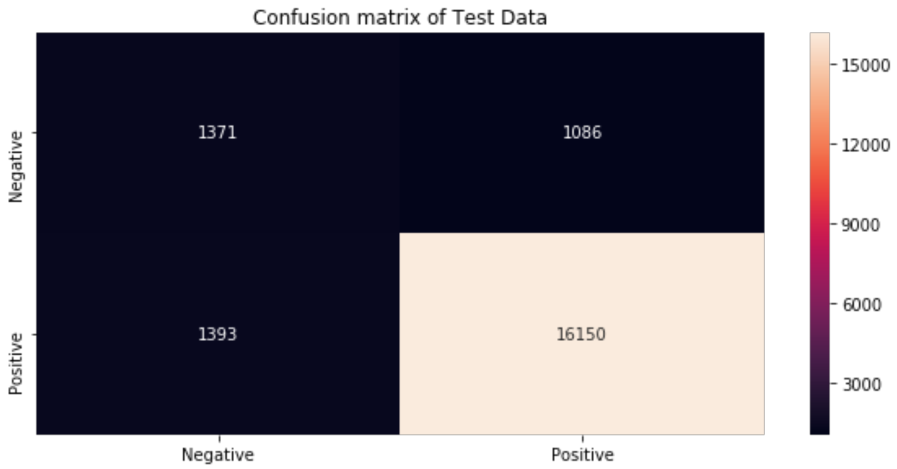
In [64]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_RF(best_tree=90,best_depth=3000,train_vector=tfidf_w2v_train,train_label=y
                                        test_vector=tfidf_w2v_test,test_label=y_test)
```

In [65]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [66]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=90,depth =3000) on model, we get auc score of future unseen data is 0.87

6.6 Model Observations

In [79]:
```
# References
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
```

In [80]:
```
x = PrettyTable()

x.field_names = ["Vectorizer","Model", "Number of Base Learners", "Max_depth", '

x.add_row(["BOW","Random Forest",90,3000,0.89])
x.add_row(["TFIDF","Random Forest",90,3000,0.89])
x.add_row(["Avg W2V","Random Forest",90,3000,0.90])
x.add_row(["TFIDF W2V","Random Forest",90,3000,0.87])

print(x)
```

```
+------------+---------------+-------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+-------------------------+-----------+------+
|    BOW     | Random Forest |            90           |    3000   | 0.89 |
|   TFIDF    | Random Forest |            90           |    3000   | 0.89 |
|  Avg W2V   | Random Forest |            90           |    3000   | 0.9  |
| TFIDF W2V  | Random Forest |            90           |    3000   | 0.87 |
+------------+---------------+-------------------------+-----------+------+
```

- Random Forest using Avg W2V gives slightly Better result compared to other Vectorizers of the Random Forest Model.

6.7. Visualizing Random Forest

6.7.1 Visualizing Random Forest using BoW

In [69]:
```
# Refernces

# https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz
# https://stackoverflow.com/questions/27817994/visualizing-decision-tree-in-scik
# https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-
```

Getting Tree

In [70]:
```
from sklearn import tree
```

In [71]:
```
model=RandomForestClassifier(n_estimators=90,max_depth=3000,class_weight="balanc
model.fit(bow_train_vec1,y_train)
```

Out[71]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=3000, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=20,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=90, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

In [72]:
```
feature=bow_model.get_feature_names()
```

In [73]:
```
tree.export_graphviz(model.estimators_[2],max_depth=2,out_file="BoW_RF.dot",clas
```

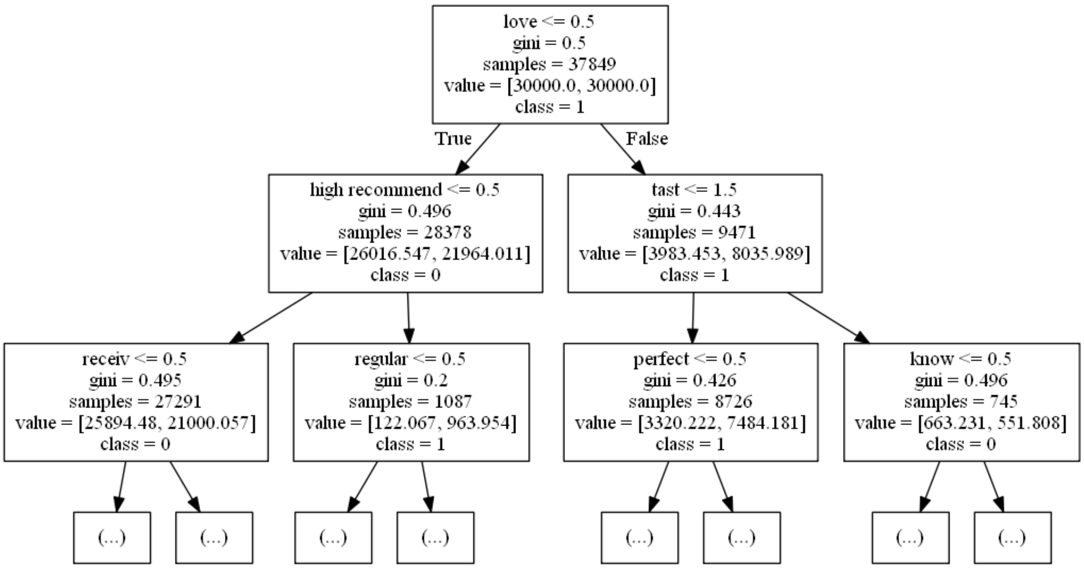
Image of the Second Estimator of Random Forest

In [74]:
```
# References

# https://stackoverflow.com/questions/11854847/how-can-i-display-an-image-from-a
```

In [75]:
```
from IPython.display import Image
```

In [104]:
```
Image(filename="BoW_RF.png")
```

Out[104]:



6.7.2 Visualizing Random Forest using TFIDF

Getting Tree

In [76]:
```
model=RandomForestClassifier(n_estimators=90,max_depth=3000,class_weight="balanc
model.fit(tfidf_train_vec1,y_train)
```

Out[76]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=3000, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=20,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=90, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

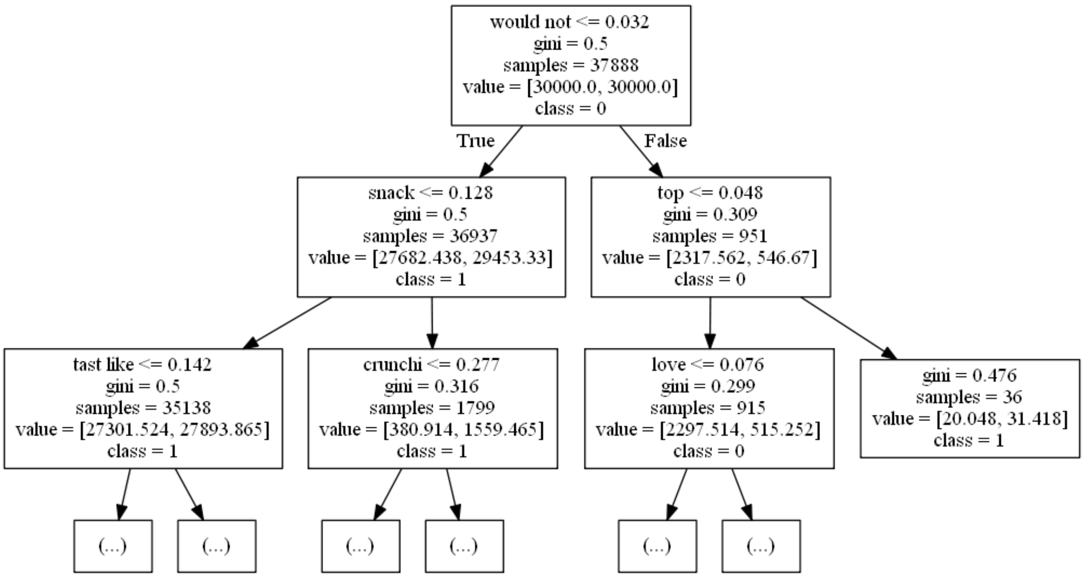
In [77]:
```
feature=tfidf_model.get_feature_names()
```

In [78]:
```
tree.export_graphviz(model.estimators_[2],max_depth=2,out_file="Tfidf_RF.dot",cl
```

Image of the Second Estimator of Random Forest

In [109]:
```
Image(filename="Tfidf_RF.png")
```

Out[109]:



7. Gradient Boosting Decision Tree (GBDT)

7.1 Creating function for GBDT

In [34]:
```
# References
# https://xgboost.readthedocs.io/en/latest/parameter.html#
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-x
# ROC_CURVE:https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ro
# ROC_AUC_CURVE: https://scikit-learn.org/stable/modules/generated/sklearn.metri
# AUC_CURVE:https://scikit-learn.org/stable/modules/generated/sklearn.metrics.au
# CONFUSION_MATRIX:https://scikit-learn.org/stable/modules/generated/sklearn.met

from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix,roc_auc_score,roc_curve
import math
```

In [134]:
```
# References for Python Functions:
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/functi
# https://www.geeksforgeeks.org/functions-in-python/
# https://www.geeksforgeeks.org/g-fact-41-multiple-return-values-in-python/

# Fuction for Hyper parameter Tuning

def Gradient_Boosting(**para):

    auc_train=[]
    auc_cv=[]

    for i,j in tqdm(zip(para["no_tree"],para["depth"])):
        model=XGBClassifier(n_estimators=i,max_depth=j,learning_rate=0.05,subsam
        model.fit(para["train_vector"],para['train_label'])

    # Prediction of training data

        train_proba=model.predict_proba(para["train_vector"])
        train=roc_auc_score(para["train_label"],train_proba[:,1])
        auc_train.append(train)

    # Prediction of cv data

        cv_proba=model.predict_proba(para["cv_vector"])
        cv=roc_auc_score(para["cv_label"],cv_proba[:,1])
        auc_cv.append(cv)

    return auc_train,auc_cv
```

In [156]:
```
def best_GBDT (**para):

    # Model training

    model=XGBClassifier(n_estimators=para["best_tree"],max_depth=para["best_dept
    model.fit(para["train_vector"],para['train_label'])

    # training data

    DT_train_proba=model.predict_proba(para["train_vector"])
    train_proba=DT_train_proba
    fpr_train,tpr_train,thres_train=roc_curve(para["train_label"],DT_train_proba
    auc_train=roc_auc_score(para["train_label"],DT_train_proba[:,1])

    # test data

    DT_test_proba=model.predict_proba(para["test_vector"])
    test_proba=DT_test_proba
    fpr_test,tpr_test,thres_test=roc_curve(para["test_label"],DT_test_proba[:,1]
    auc_test=roc_auc_score(para["test_label"],DT_test_proba[:,1])

    return train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_trai
```

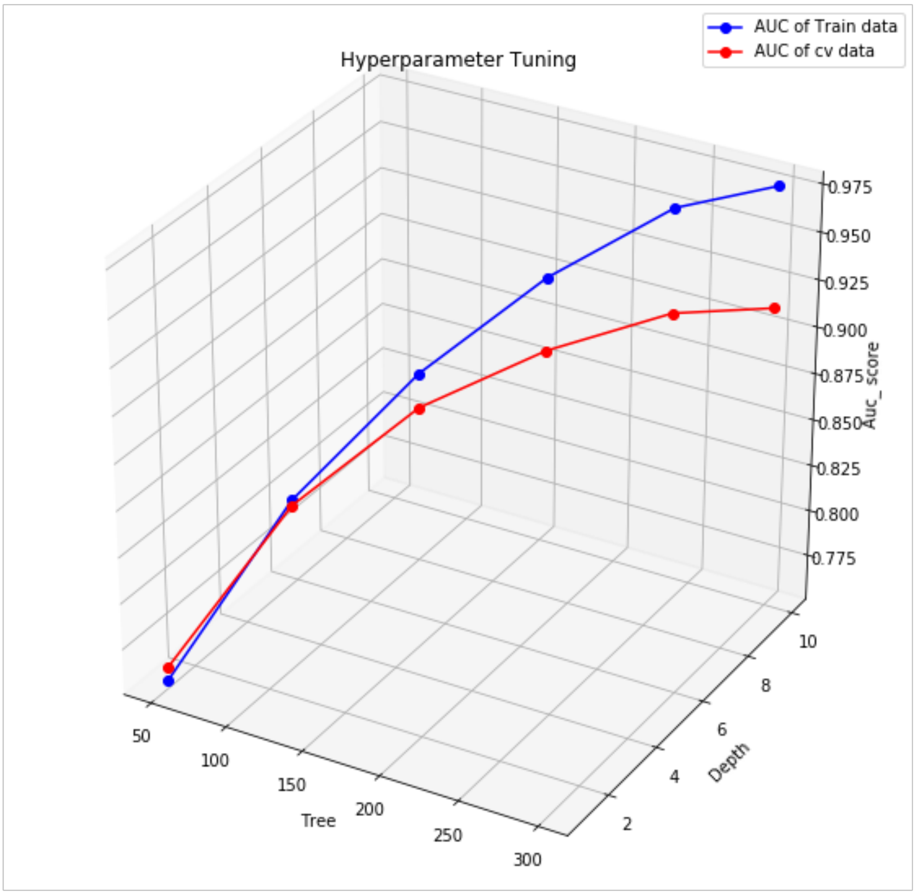
7.2 GBDT using BOW

In [150]:
```
tree=[50,100,150,200,250,300]
depth=[1,3,5,7,9,10]
```

In [151]:
```
# Hyperparameter tuning

auc_train,auc_cv=Gradient_Boosting(no_tree=tree,depth=depth,train_vector=bow_tra
                                             cv_vector=bow_cv_vec1,cv_label=y_cv
```

```
6it [05:29, 70.49s/it]
```

In [152]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

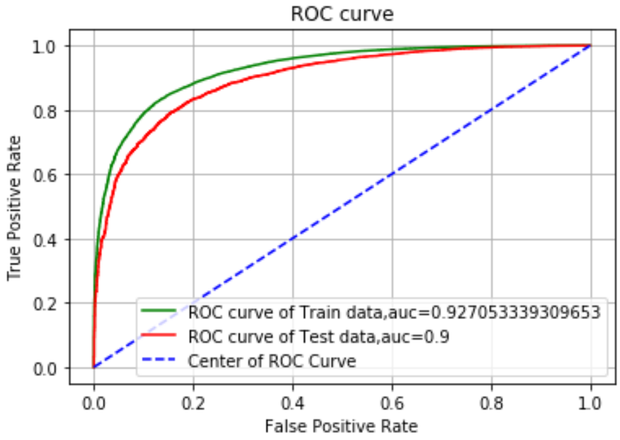


Observation:

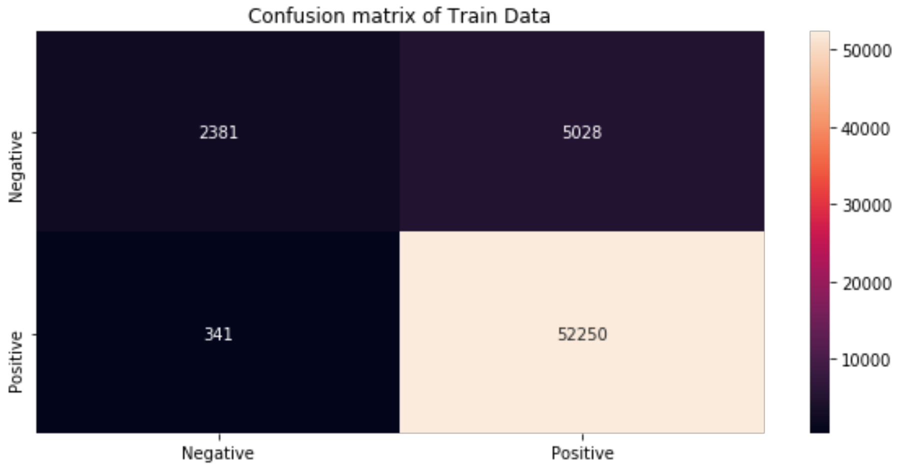
- To avoid overfitting and underfitting,choose (no of base learners=200,depth=7), we get auc_score=0.85

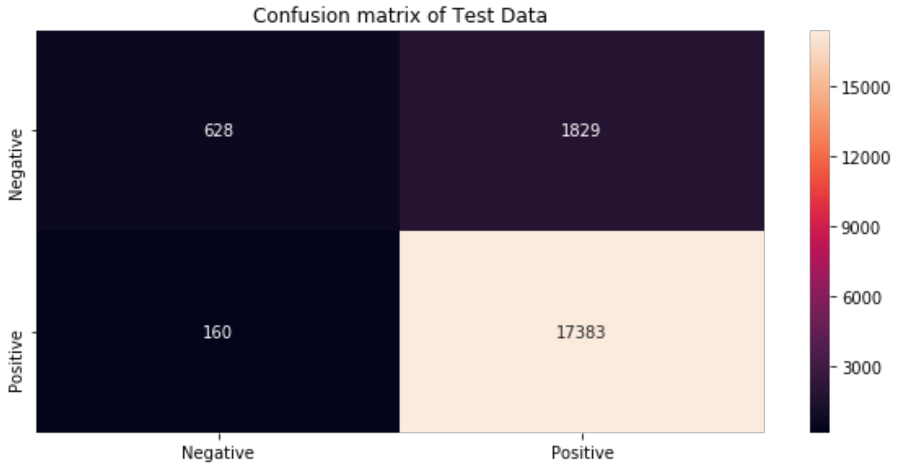
In [157]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_GBDT(best_tree=200,best_depth=7,train_vector=bow_train_vec1,train_label=y_
                              test_vector=bow_test_vec1,test_label=y_test)
```

In [158]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [159]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=200,depth =7) on model, we get auc score of future unseen data is 0.90

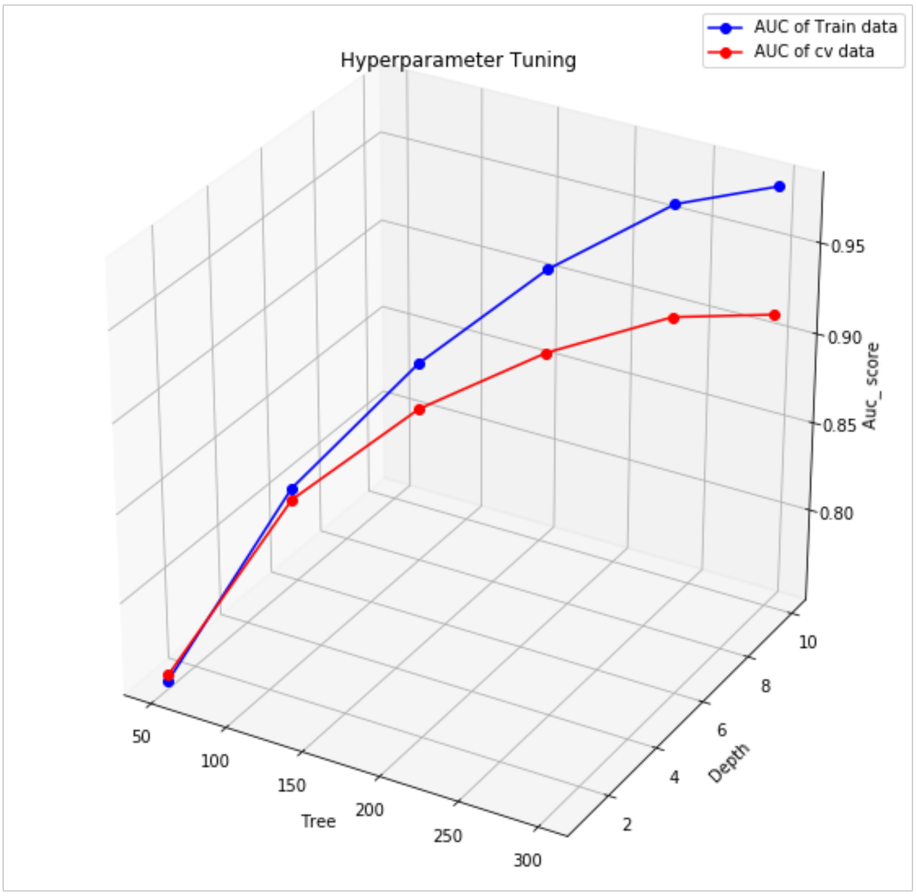
7.3 GBDT using TFIDF

In [160]:
```
tree=[50,100,150,200,250,300]
depth=[1,3,5,7,9,10]
```

In [161]:
```
# Hyperparameter tuning

auc_train,auc_cv=Gradient_Boosting(no_tree=tree,depth=depth,train_vector=tfidf_t
                                   cv_vector=tfidf_cv_vec1,cv_label=y_
```

```
6it [11:19, 147.90s/it]
```

In [162]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

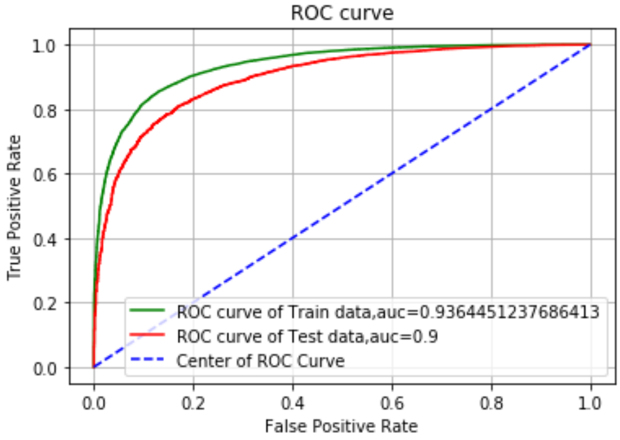


Observation:

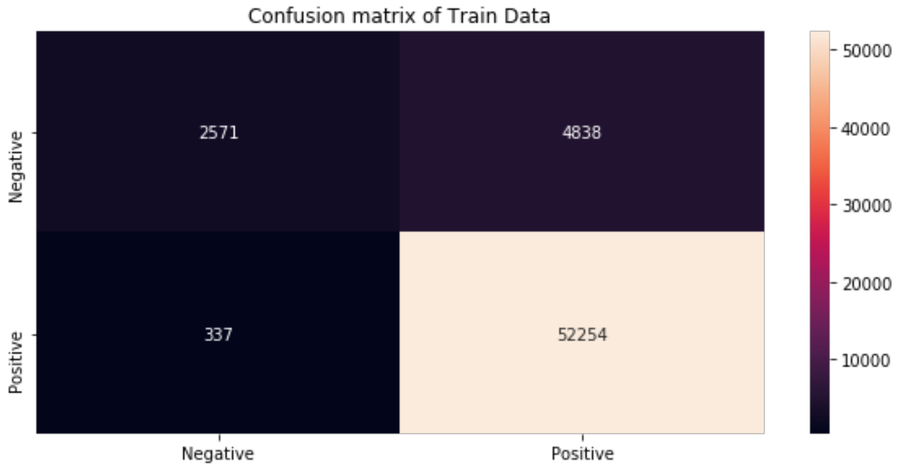
- To avoid overfitting and underfitting,choose (no of base learners=200,depth=7), we get auc_score=0.85

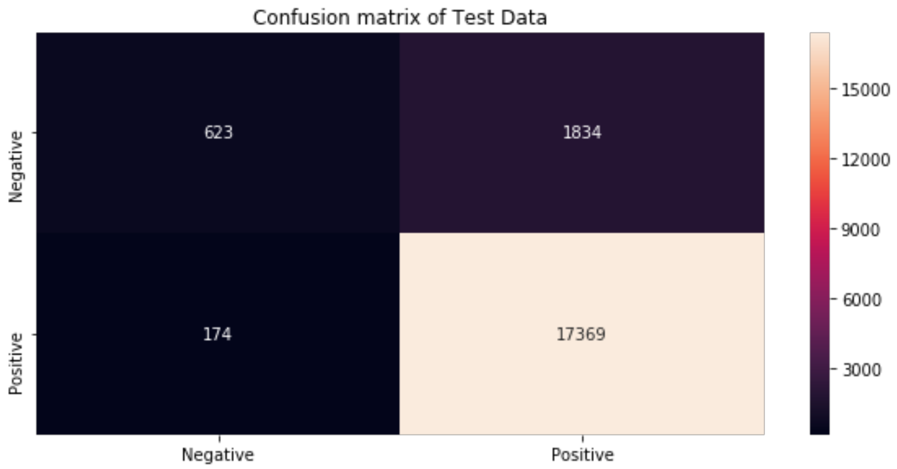
In [163]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_GBDT(best_tree=200,best_depth=7,train_vector=tfidf_train_vec1,train_label=
                               test_vector=tfidf_test_vec1,test_label=y_test)
```

In [164]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [165]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=200,depth =7) on model, we get auc score of future unseen data is 0.90

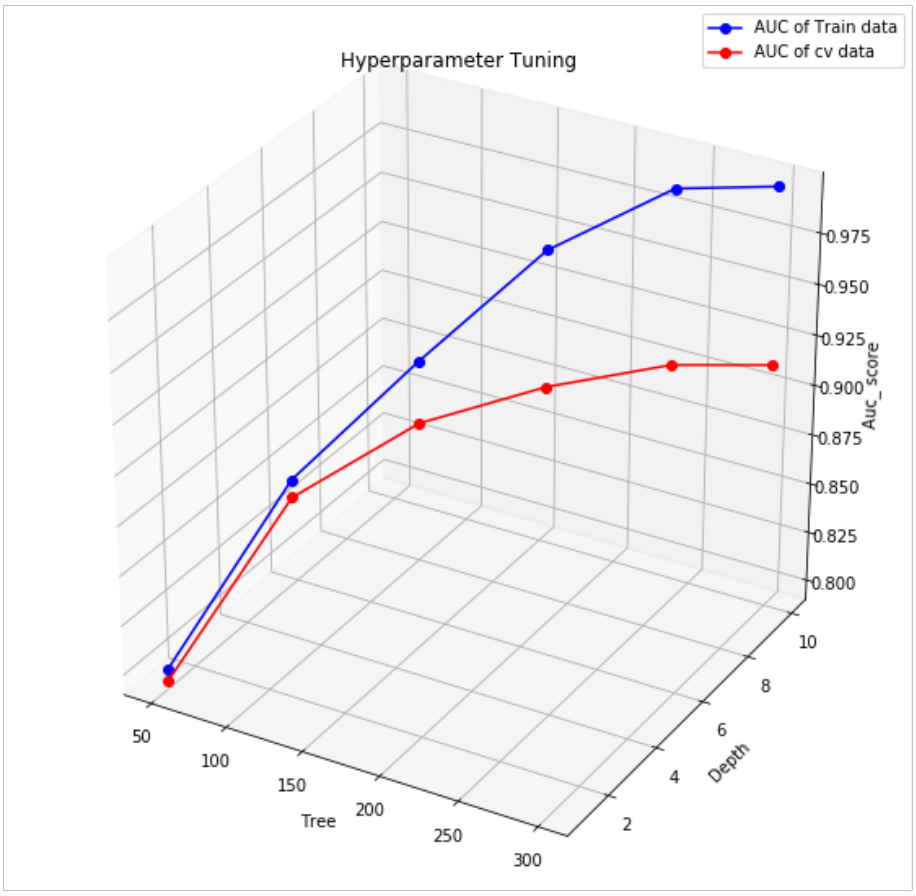
7.4 GBDT using Avg W2V

In [166]:
```
tree=[50,100,150,200,250,300]
depth=[1,3,5,7,9,10]
```

In [167]:
```
# Hyperparameter tuning

auc_train,auc_cv=Gradient_Boosting(no_tree=tree,depth=depth,train_vector=avg_w2v
                                    cv_vector=avg_w2v_cv,cv_label=y_cv)
```

```
6it [17:23, 229.49s/it]
```

In [168]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

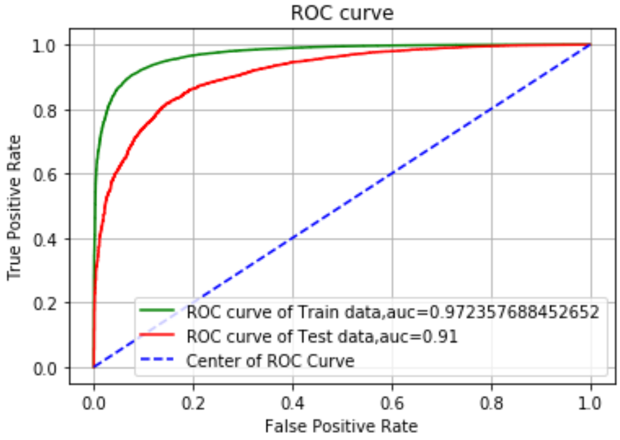


Observation:

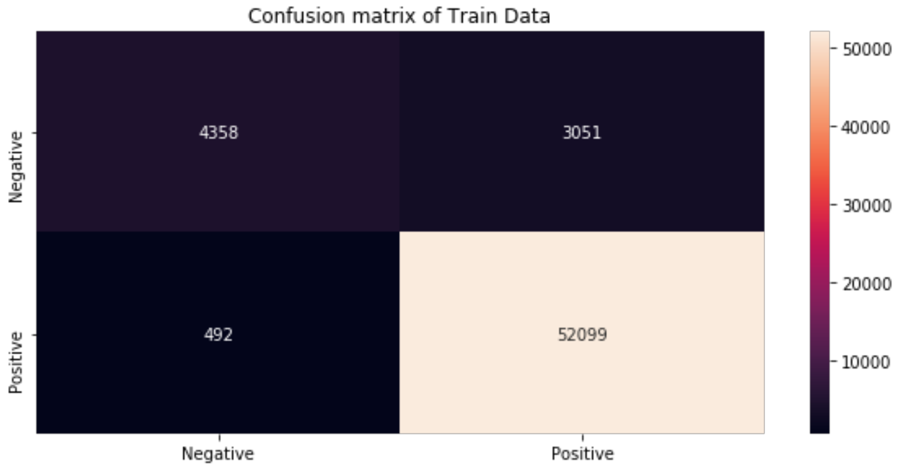
- To avoid overfitting and underfitting,choose (no of base learners=200,depth=7, we get auc_score=0.86

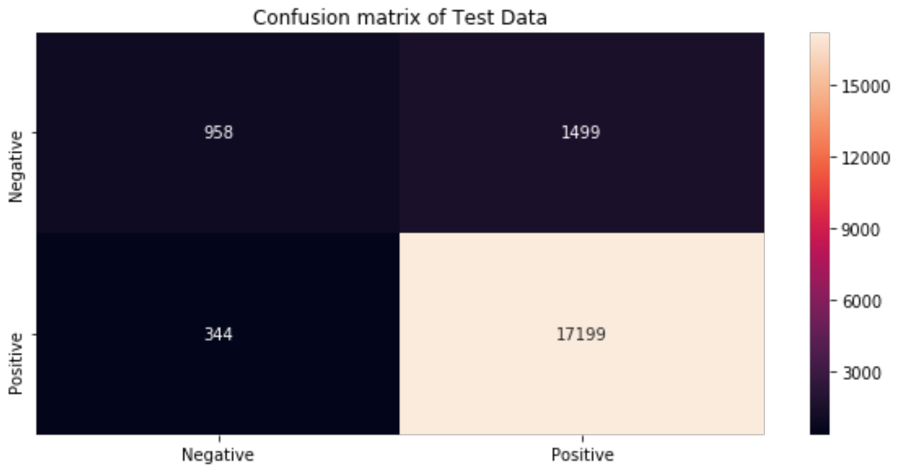
In [169]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_GBDT(best_tree=200,best_depth=7,train_vector=avg_w2v_train,train_label=y_t
                              test_vector=avg_w2v_test,test_label=y_test)
```

In [170]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
          text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [171]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=200,depth =7) on model, we get auc score of future unseen data is 0.91

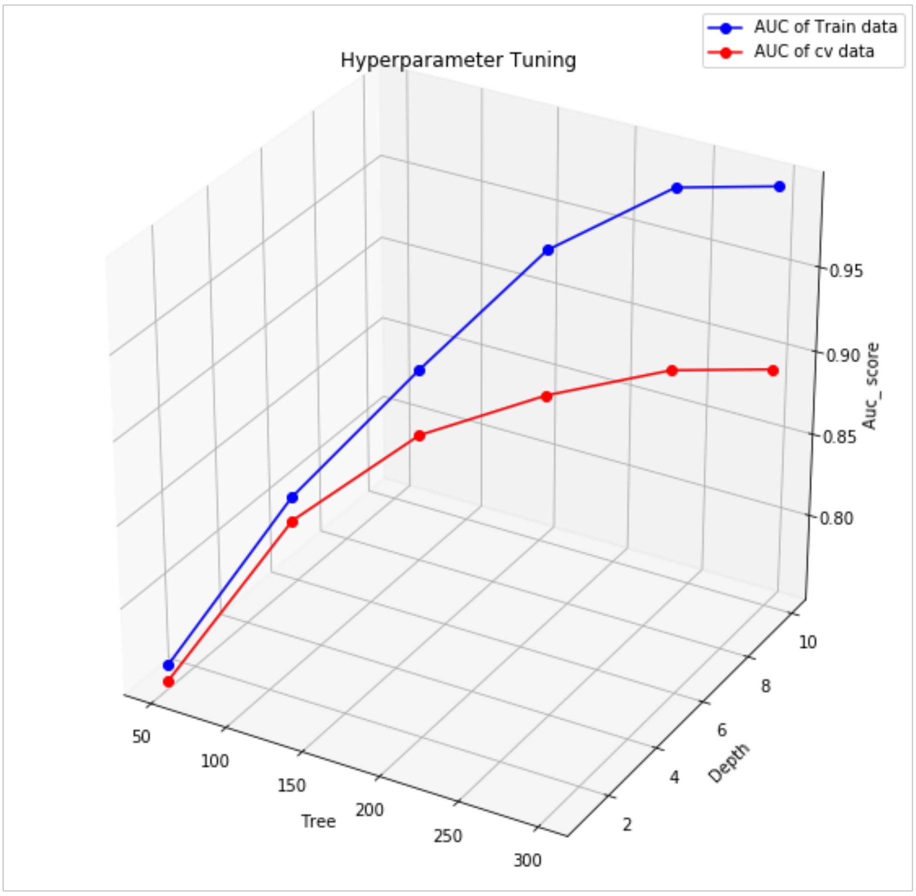
7.5 GBDT using TFIDF W2V

In [172]:
```
tree=[50,100,150,200,250,300]
depth=[1,3,5,7,9,10]
```

In [174]:
```
# Hyperparameter tuning

auc_train,auc_cv=Gradient_Boosting(no_tree=tree,depth=depth,train_vector=tfidf_w
                                             cv_vector=tfidf_w2v_cv,cv_label=y_c
```

```
6it [17:28, 228.39s/it]
```

In [175]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

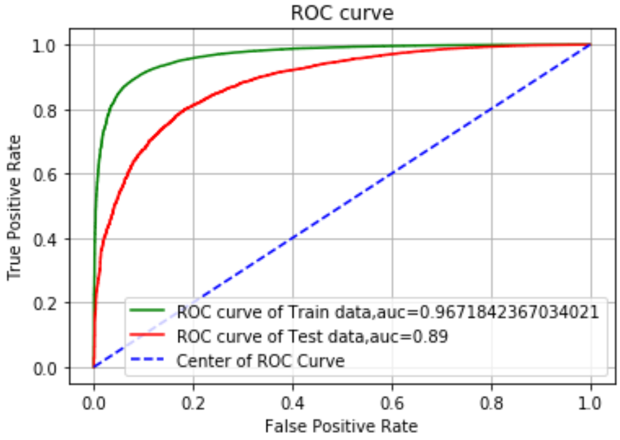


Observation:

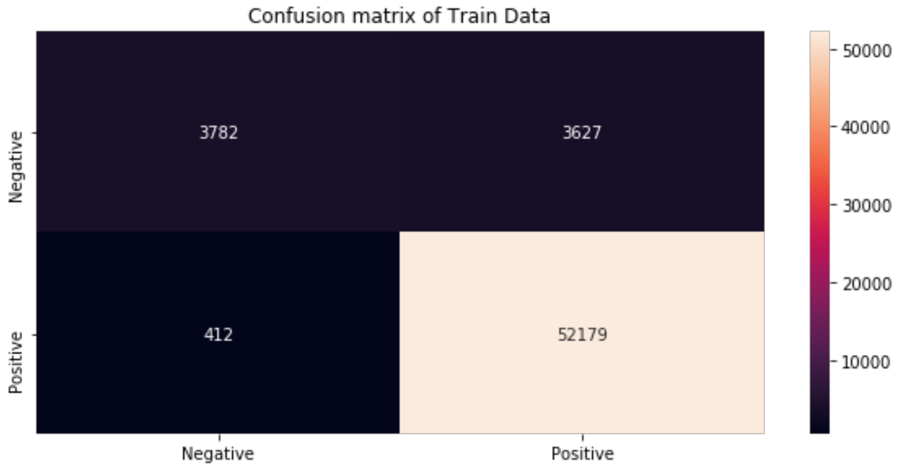
- To avoid overfitting and underfitting,choose (no of base learners=200,depth=7), we get auc_score=0.83

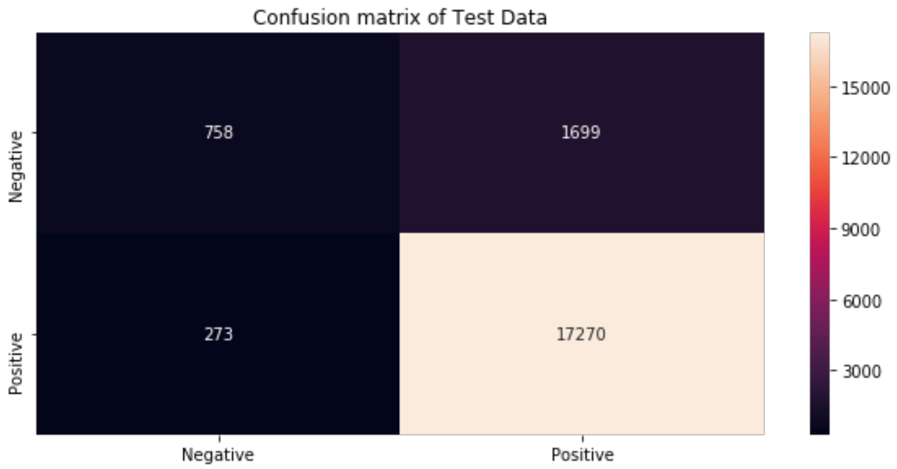
In [176]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_GBDT(best_tree=200,best_depth=7,train_vector=tfidf_w2v_train,train_label=y
                               test_vector=tfidf_w2v_test,test_label=y_test)
```

In [177]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [178]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=200,depth =7) on model, we get auc score of future unseen data is 0.89

7.6 Model Observations

In [67]:
```
# References
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
```

In [81]:
```
y = PrettyTable()

y.field_names = ["Vectorizer","Model", "Number of Base Learners", "Max_depth", '

y.add_row(["BOW","GBDT",200,7,0.90])
y.add_row(["TFIDF","GBDT",200,7,0.90])
y.add_row(["Avg W2V","GBDT",200,7,0.91])
y.add_row(["TFIDF W2V","GBDT",200,7,0.89])

print(y)
```

```
+------------+-------+-------------------------+-----------+------+
| Vectorizer | Model | Number of Base Learners | Max_depth | AUC  |
+------------+-------+-------------------------+-----------+------+
|    BOW     |  GBDT |           200           |     7     | 0.9  |
|   TFIDF    |  GBDT |           200           |     7     | 0.9  |
|  Avg W2V   |  GBDT |           200           |     7     | 0.91 |
| TFIDF W2V  |  GBDT |           200           |     7     | 0.89 |
+------------+-------+-------------------------+-----------+------+
```

- GBDT using Avg W2V gives slightly Better result compared to other Vectorizers of the GBDT Model.

7.7. Visualizing GBDT

7.7.1 Visualizing GBDT using BoW

In [181]:
```
# Refernces

# https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz
# https://stackoverflow.com/questions/27817994/visualizing-decision-tree-in-scik
# https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-
```

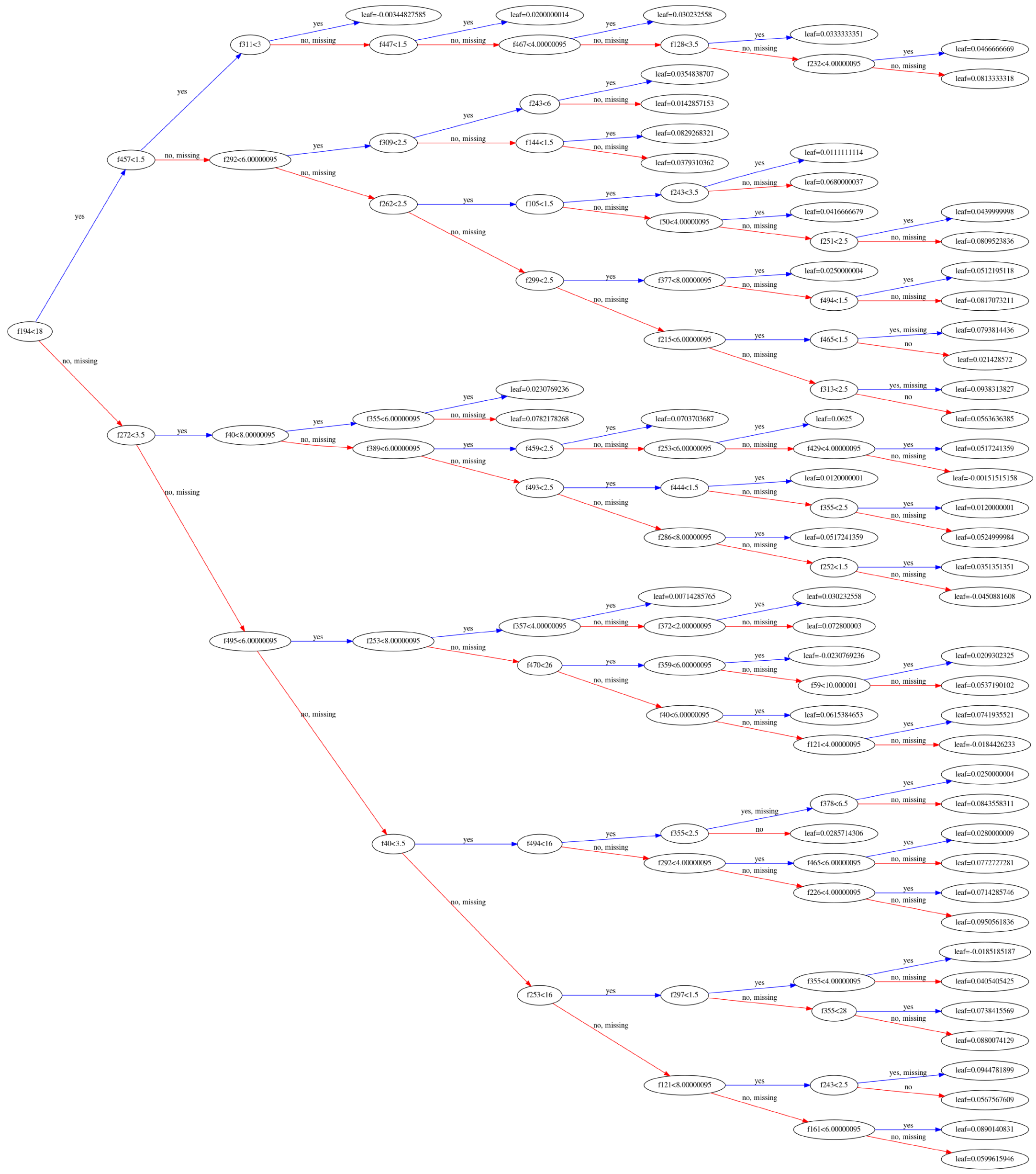
In [127]:
```
model=XGBClassifier(n_estimators=200,max_depth=7,learning_rate=0.05,subsample=0.
model.fit(bow_train_vec1,y_train)
```

Out[127]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.05, max_delta_step=0,
       max_depth=7, min_child_weight=5, missing=None, n_estimators=200,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
```

In [128]:
```
feature=bow_model.get_feature_names()
```

In [129]:
```
# References
# https://machinelearningmastery.com/visualize-gradient-boosting-decision-trees-

import xgboost as xgb
from xgboost import plot_tree
```

In [138]:
```
# References
# https://machinelearningmastery.com/visualize-gradient-boosting-decision-trees-
# https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboos

plt.close()
plot_tree(model,num_trees=0,rankdir="LR")
fig = plt.gcf()
fig.set_size_inches(150, 100)

plt.show()
```



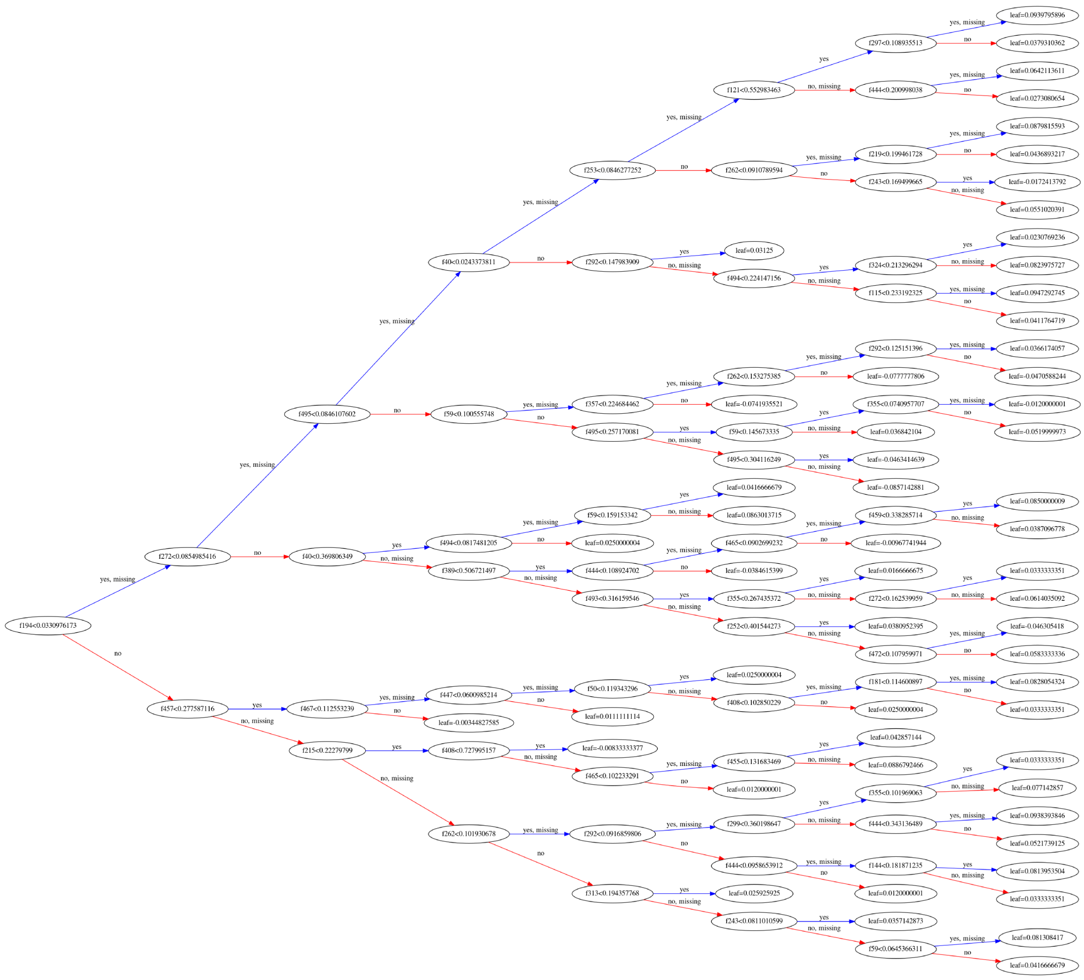
7.7.2 Visualizing GBDT using TFIDF

In [139]:
```
model=XGBClassifier(n_estimators=200,max_depth=7,learning_rate=0.05,subsample=0.
model.fit(tfidf_train_vec1,y_train)
```

Out[139]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.05, max_delta_step=0,
       max_depth=7, min_child_weight=5, missing=None, n_estimators=200,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
```

In [140]:
```
feature=tfidf_model.get_feature_names()
```

In [141]:
```
# References
# https://machinelearningmastery.com/visualize-gradient-boosting-decision-trees-
# https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost

plt.close()
plot_tree(model,num_trees=0,rankdir="LR")
fig = plt.gcf()
fig.set_size_inches(150, 100)
plt.show()
```



8. Feature Importance

8.1 Feature Importance of Random Forest and Wordcloud visualization

8.1.1 Feature Importance on BoW

In [79]:
```
model=RandomForestClassifier(n_estimators=90,max_depth=3000,class_weight="balanc
model.fit(bow_train_vec1,y_train)
```

Out[79]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=3000, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=20,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=90, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

In [80]:
```
fi=model.feature_importances_
```

In [81]:
```
fi=np.argsort(fi)[::-1]
```

In [83]:
```
important_features_bow_RF=np.take(bow_model.get_feature_names(),fi[0:20])
```

In [84]:
```
print("Top 20 Important Features of Random Forest (BOW)")
print("="*125)
print(important_features_bow_RF)
```

```
Top 20 Important Features of Random Forest (BOW)
===============================================================================
==============================================
['not' 'great' 'best' 'love' 'disappoint' 'delici' 'would' 'perfect'
 'favorit' 'good' 'money' 'high recommend' 'would not' 'bad' 'tast'
 'excel' 'product' 'nice' 'find' 'easi']
```

In [85]:
```
# References
# https://www.geeksforgeeks.org/generating-word-cloud-python/

from wordcloud import WordCloud
```

In [87]:
```
words_bow =" ".join(important_features_bow_RF)
```

In [104]:
```
wordcloud_bow_RF = WordCloud(width=720, height=720, max_words=20).generate(words
```

In [107]:
```
plt.close()
plt.figure(figsize = (5,5))
plt.imshow(wordcloud_bow_RF)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

8.1.2 Feature Importance on TFIDF

In [106]:
```
model=RandomForestClassifier(n_estimators=90,max_depth=3000,class_weight="balanc
model.fit(tfidf_train_vec1,y_train)
```

Out[106]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=3000, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=20,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=90, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

In [108]:
```
fi=model.feature_importances_
```

In [109]:
```
fi=np.argsort(fi)[::-1]
```

In [110]:
```
important_features_tfidf_RF=np.take(tfidf_model.get_feature_names(),fi[0:20])
```

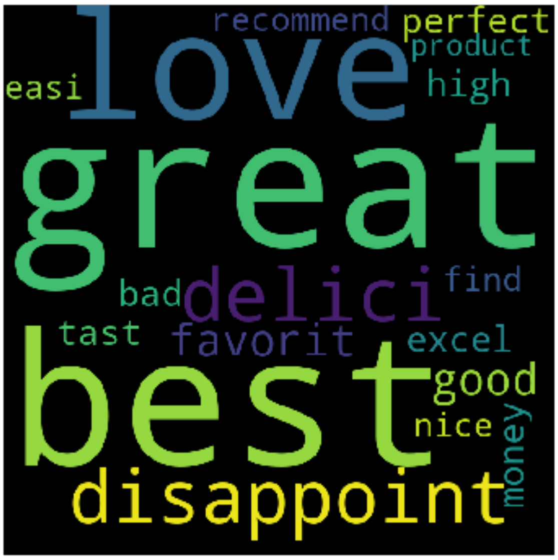
In [111]:
```
print("Top 20 Important Features of Random Forest (TFIDF)")
print("="*125)
print(important_features_tfidf_RF)
```

```
Top 20 Important Features of Random Forest (TFIDF)
================================================================================
==============================================
['not' 'great' 'best' 'love' 'disappoint' 'delici' 'would' 'favorit'
 'good' 'perfect' 'money' 'tast' 'bad' 'excel' 'high recommend'
 'would not' 'find' 'easi' 'nice' 'product']
```

In [112]:
```
words_tfidf =" ".join(important_features_tfidf_RF)
```

In [113]:
```
wordcloud_tfidf_RF = WordCloud(width=720, height=720, max_words=20).generate(wo
```

In [114]:
```
plt.close()
plt.figure(figsize = (5,5))
plt.imshow(wordcloud_tfidf_RF)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



8.2 Feature Importance of GBDT and Wordcloud visualization

8.2.1 Feature Importance on BoW

In [35]:
```
model=XGBClassifier(n_estimators=200,max_depth=7,learning_rate=0.05,subsample=0.
model.fit(bow_train_vec1,y_train)
```

Out[35]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.05, max_delta_step=0,
       max_depth=7, min_child_weight=5, missing=None, n_estimators=200,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
```

In [190]:
```
fi=model.feature_importances_
```

In [191]:
```
fi=np.argsort(fi)[::-1]
```

In [192]:
```
important_features_bow_GBDT=np.take(bow_model.get_feature_names(),fi[0:20])
```

In [193]:
```
print("Top 20 Important Features of GBDT (BOW)")
print("="*125)
print(important_features_bow_GBDT)
```

```
Top 20 Important Features of GBDT (BOW)
================================================================================
==============================================
['disappoint' 'great' 'money' 'perfect' 'would not' 'delici' 'best' 'easi'
 'high recommend' 'excel' 'wonder' 'favorit' 'nice' 'addict' 'add' 'enjoy'
 'amaz' 'yummi' 'happi' 'satisfi']
```

In [194]:
```
# References
# https://www.geeksforgeeks.org/generating-word-cloud-python/

from wordcloud import WordCloud
```

In [195]:
```
words_bow =" ".join(important_features_bow_GBDT)
```

In [196]:
```
wordcloud_bow_GBDT = WordCloud(width=720, height=720, max_words=20).generate(wo
```

In [197]:
```
plt.close()
plt.figure(figsize = (5,5))
plt.imshow(wordcloud_bow_GBDT)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

8.2.2 Feature Importance on TFIDF

In [198]:
```
model=XGBClassifier(n_estimators=200,max_depth=7,learning_rate=0.05,subsample=0.
model.fit(tfidf_train_vec1,y_train)
```

Out[198]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.05, max_delta_step=0,
       max_depth=7, min_child_weight=5, missing=None, n_estimators=200,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
```

In [199]:
```
fi=model.feature_importances_
```

In [200]:
```
fi=np.argsort(fi)[::-1]
```

In [201]:
```
important_features_tfidf_GBDT=np.take(tfidf_model.get_feature_names(),fi[0:20])
```

In [202]:
```
print("Top 20 Important Features of GBDT (TFIDF)")
print("="*125)
print(important_features_tfidf_GBDT)
```

```
Top 20 Important Features of GBDT (TFIDF)
==============================================================================================================================
['money' 'disappoint' 'high recommend' 'delici' 'great' 'perfect' 'best'
 'easi' 'would not' 'favorit' 'wonder' 'snack' 'amaz' 'tasti' 'add'
 'excel' 'nice' 'addict' 'enjoy' 'meal']
```

In [203]:
```
words_tfidf =" ".join(important_features_tfidf_GBDT)
```

In [204]:
```
wordcloud_tfidf_GBDT = WordCloud(width=720, height=720, max_words=20).generate(w
```

In [206]:
```
plt.close()
plt.figure(figsize = (5,5))
plt.imshow(wordcloud_tfidf_GBDT)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

9. Feature Engineering

- We do feature engineering on Random Forest Model using TFIDF-W2V. Because this gives slightly less performance result compared to others.

9.1 Adding Summary Text as a Feature with Review Text

- We consider summary text as a feature,we do preprocessing and featurization on the summary text and then we horizontally stack the summary text to the review text. so finally we get the extra word vector to improve our model.

9.1.1 Summary Text Preprocessing

In [42]:
```
raw_summary_text_data=filter_data.Summary.values
```

In [43]:
```
# Preprocessing

preprocessed_summary_text_data=[]
for i in tqdm(raw_summary_text_data):
# removing of HTML tags
    a=re.sub("<.*?>"," ",i)
# removing url
    b=re.sub(r"http\S+"," ",a)
# expanding contractions
    c=decontracted(b)
# removing alphA_numeric
    d=re.sub("\S*\d\S*", " ",c)
# removing Special characters
    e=re.sub('[^A-Za-z0-9]+', ' ',d)
# removing stopwords
    k=[]
    for w in e.split():
        if w.lower() not in stopwords:
            s=(stemmer.stem(w.lower())).encode('utf8')
            k.append(s)
    preprocessed_summary_text_data.append(b' '.join(k).decode())
```

```
100%|██████████| 364171/364171 [00:41<00:00, 8709.32it/s]
```

In [44]:
```
filter_data["Summary"]=preprocessed_summary_text_data
```

In [45]:
```
filter_data.shape
```

Out[45]:
```
(364171, 10)
```

In [46]:
```
# we took the sample data size as 100k

final_data=filter_data[:100000]
final_data.shape
```

Out[46]:
```
(100000, 10)
```

9.1.2. Data Splitting

In [47]:
```
# References
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tra

from sklearn.model_selection import train_test_split
```

In [48]:
```
X=final_data.Summary
Y=final_data.Score
```

In [49]:
```
x_1,x_test_2,y_1,y_test_2=train_test_split(X,Y,test_size=0.2,random_state=40)
x_train_2,x_cv_2,y_train_2,y_cv_2=train_test_split(x_1,y_1,test_size=0.25,random
print(" Train data Size")
print(x_train_2.shape,y_train_2.shape)

print("cv data size")
print(x_cv_2.shape,y_cv_2.shape)
print("Test data size")
print(x_test_2.shape,y_test_2.shape)
```

```
 Train data Size
(60000,) (60000,)
cv data size
(20000,) (20000,)
Test data size
(20000,) (20000,)
```

9.1.3. Featurization

In [50]:
```
list_sentences_train_2=[]
for i in tqdm(list(x_train_2)):
    list_sentences_train_2.append(i.split())
```

```
100%|██████████| 60000/60000 [00:00<00:00, 116498.68it/s]
```

In [51]:
```
word2vec_model_fe=Word2Vec(list_sentences_train_2,min_count=5,size=50,workers=4)
```

In [52]:
```
word2vec_words_train_2=list(word2vec_model_fe.wv.vocab)
print(" Number of words")
print("_________________")
print(" ")
print(len(word2vec_words_train_2))
print("="*125)
print(" sample words")
print("_________________")
print(" ")
print(word2vec_words_train_2[100:150])
```

```
 Number of words
_________________

2757
===============================================================================
==============================================
 sample words
_________________

['strong', 'yummmmmm', 'nectar', 'nice', 'select', 'confus', 'keurig', 'organ',
'black', 'cherri', 'concentr', 'must', 'work', 'food', 'make', 'go', 'yeah', 'm
ove', 'rice', 'krispi', 'treat', 'barbequ', 'chip', 'green', 'bowl', 'edibl', '
pet', 'health', 'risk', 'get', 'unexpect', 'guest', 'super', 'deal', 'anyon', '
need', 'gluten', 'favorit', 'no', 'raspberri', 'celesti', 'season', 'garden', '
refresh', 'tasti', 'light', 'kiwi', 'low', 'caffein', 'hand']
```

In [53]:
```
# list of sentences cv data

list_sentences_cv_2=[]
for i in tqdm(list(x_cv_2)):
    list_sentences_cv_2.append(i.split())

# list of sentences test data

list_sentences_test_2=[]
for i in tqdm(list(x_test_2)):
    list_sentences_test_2.append(i.split())
```

```
100%|██████████| 20000/20000 [00:00<00:00, 494663.82it/s]
100%|██████████| 20000/20000 [00:00<00:00, 465431.30it/s]
```

In [54]:
```
# References
# https://stackoverflow.com/questions/21553327
# https://github.com/devBOX03


# tfidf word2vec on training data

model_2=TfidfVectorizer()
tfidf_w2v_model_2=model_2.fit_transform(x_train_2)
tfidf_w2v_2=model_2.get_feature_names()
tfidf_word2vec_train_2=[]
row=0
for i in tqdm(list_sentences_train_2):
    vec=np.zeros(50)
    weight_sum=0
    for w in i:
        try:
            w2v_freq=word2vec_model_fe.wv[w]
            tfidf_freq=tfidf_w2v_model_2[row,tfidf_w2v_2.index(w)]
            vec=vec+(w2v_freq*tfidf_freq)
            weight_sum=weight_sum+tfidf_freq
        except:
            pass
    vec=vec/weight_sum
    tfidf_word2vec_train_2.append(vec)
    row=row+1
tfidf_w2v_train_2=np.asmatrix(tfidf_word2vec_train_2)
print("Shape of TFIDF word2vec train")
print(tfidf_w2v_train_2.shape)
```

```
100%|██████████| 60000/60000 [00:44<00:00, 1345.42it/s]

Shape of TFIDF word2vec train
(60000, 50)
```

In [55]:
```
# tfidf word2vec on cv data

tfidf_w2v_model_2=model_2.transform(x_cv_2)
tfidf_word2vec_cv_2=[]
row=0
for i in tqdm(list_sentences_cv_2):
    vec=np.zeros(50)
    weight_sum=0
    for w in i:
        try:
            w2v_freq=word2vec_model_fe.wv[w]
            tfidf_freq=tfidf_w2v_model_2[row,tfidf_w2v_2.index(w)]
            vec=vec+(w2v_freq*tfidf_freq)
            weight_sum=weight_sum+tfidf_freq
        except:
            pass
    vec=vec/weight_sum
    tfidf_word2vec_cv_2.append(vec)
    row=row+1
tfidf_w2v_cv_2=np.asmatrix(tfidf_word2vec_cv_2)
print("Shape of TFIDF word2vec cv")
print(tfidf_w2v_cv_2.shape)
```

```
100%|██████████| 20000/20000 [00:15<00:00, 1313.06it/s]

Shape of TFIDF word2vec cv
(20000, 50)
```

In [56]:
```
# tfidf word2vec on test data

tfidf_w2v_model_2=model_2.transform(x_test_2)
tfidf_word2vec_test_2=[]
row=0
for i in tqdm(list_sentences_test_2):
    vec=np.zeros(50)
    weight_sum=0
    for w in i:
        try:
            w2v_freq=word2vec_model_fe.wv[w]
            tfidf_freq=tfidf_w2v_model_2[row,tfidf_w2v_2.index(w)]
            vec=vec+(w2v_freq*tfidf_freq)
            weight_sum=weight_sum+tfidf_freq
        except:
            pass
    vec=vec/weight_sum
    tfidf_word2vec_test_2.append(vec)
    row=row+1
tfidf_w2v_test_2=np.asmatrix(tfidf_word2vec_test_2)
print("Shape of TFIDF word2vec test")
print(tfidf_w2v_test_2.shape)
```

```
100%|██████████| 20000/20000 [00:14<00:00, 1364.36it/s]

Shape of TFIDF word2vec test
(20000, 50)
```

9.1.4 Horizontally stacking

In [57]:
```
# References
# https://docs.scipy.org/doc/numpy/reference/generated/numpy.hstack.html
```

In [58]:
```
# For training data

tfidf_w2v_train_fe=np.hstack((tfidf_w2v_train,tfidf_w2v_train_2))

# For cv data

tfidf_w2v_cv_fe=np.hstack((tfidf_w2v_cv,tfidf_w2v_cv_2))

# For test data

tfidf_w2v_test_fe=np.hstack((tfidf_w2v_test,tfidf_w2v_test_2))
```

In [59]:
```
print(tfidf_w2v_train_fe.shape)
print(tfidf_w2v_cv_fe.shape)
print(tfidf_w2v_test_fe.shape)
```

```
(60000, 100)
(20000, 100)
(20000, 100)
```

9.1.5 Feature Engineering on Random Forest (TFIDF-W2V)

In [60]:
```
tree=[5,15,30,45,60,75,90,100,200]
depth=[5,50,100,500,1000,2000,3000,4000,5000]
```

In [61]:
```
# To eliminate NaN values produced in the TFIDF W2V vectorizer
# https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer
# https://stackoverflow.com/questions/44727793/imputer-mean-strategy-removes-nan

from sklearn.impute import SimpleImputer
```

In [62]:
```
imp=SimpleImputer(missing_values=np.nan,strategy='mean')
tfidf_w2v_train_fe_im=imp.fit_transform(tfidf_w2v_train_fe)
tfidf_w2v_cv_fe_im=imp.fit_transform(tfidf_w2v_cv_fe)
tfidf_w2v_test_fe_im=imp.fit_transform(tfidf_w2v_test_fe)
```

In [71]:
```
# Hyperparameter tuning

auc_train,auc_cv=Random_Forest(no_tree=tree,depth=depth,train_vector=tfidf_w2v_t
                                             cv_vector=tfidf_w2v_cv_fe_im,cv_lab
```

```
9it [08:21, 90.87s/it]
```

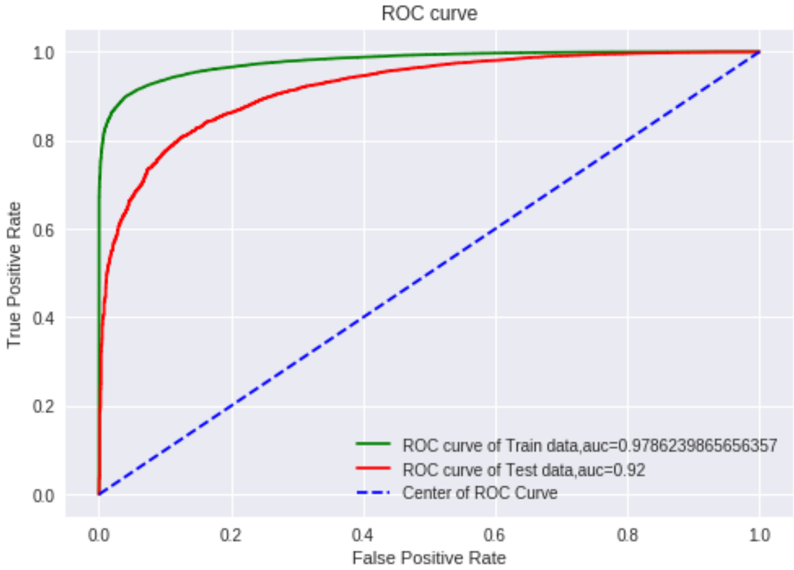
In [72]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

Observation:

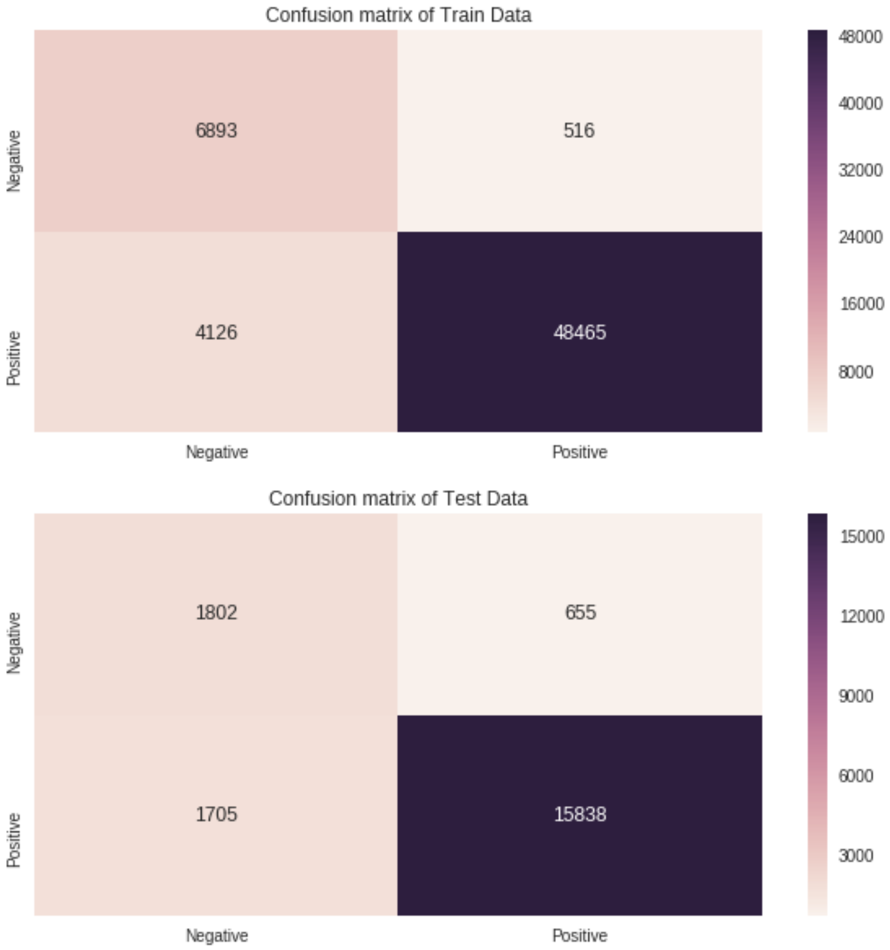
- To avoid overfitting and underfitting,choose (no of base learners=90,depth=3000), we get auc_score=0.88

In [73]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_RF(best_tree=90,best_depth=3000,train_vector=tfidf_w2v_train_fe_im,train_l
                               test_vector=tfidf_w2v_test_fe_im,test_label=y_tes
```

In [78]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
         text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [75]:
```
# confusion matrix
cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```



Observation:

- When we applying best hyperparameter (no of base learners=90,depth =3000) on model, we get auc score of future unseen data is 0.92

Model Observations

In [83]:
```
z = PrettyTable()

z.field_names = ["Vectorizer","Model", "Number of Base Learners", "Max_depth", '

z.add_row(["TFIDF","Random Forest",90,3000,0.92])
print(z)
```

```
+------------+--------------+-------------------------+-----------+------+
| Vectorizer |     Model    | Number of Base Learners | Max_depth | AUC  |
+------------+--------------+-------------------------+-----------+------+
|   TFIDF    | Random Forest |           90           |    3000   | 0.92 |
+------------+--------------+-------------------------+-----------+------+
```

9.2 Adding Review Text length as a feature with Review and Summary Text vector

In [84]:
```
# Lengh of the Words in Each Review document

a=[]
for i in preprocessed_text_data:
    a.append(len(i.split()))
```

In [85]:
```
# Adding Length as a new Feature in DataFrame

filter_data["Length"]=a
```

9.2.1Column Standardization using Standardization Formula:

- $(Xi - mean)/std$

In [86]:
```
mean1=filter_data.Length.mean()
std1=filter_data.Length.std()
```

In [87]:
```
b=a
c=[]
for i in b:
    stand=(i-mean1)/std1
    c.append(abs(stand))
```

In [88]:
```
filter_data.Length=c
```

9.2.2. Data Splitting

In [89]:
```
# we took the sample data size as 100k

final_data=filter_data[:100000]
final_data.shape
```

Out[89]:
```
(100000, 11)
```

In [90]:
```
X=final_data.Length
Y=final_data.Score
```

In [91]:
```
x_1,x_test_3,y_1,y_test_3=train_test_split(X,Y,test_size=0.2,random_state=40)
x_train_3,x_cv_3,y_train_3,y_cv_3=train_test_split(x_1,y_1,test_size=0.25,randon
print(" Train data Size")
print(x_train_3.shape,y_train_3.shape)

print("cv data size")
print(x_cv_3.shape,y_cv_3.shape)
print("Test data size")
print(x_test_3.shape,y_test_3.shape)
```

```
 Train data Size
(60000,) (60000,)
cv data size
(20000,) (20000,)
Test data size
(20000,) (20000,)
```

9.2.3 Horizontally stacking

Feature Engineering on TFIDF-W2V

In [93]:
```
# hstack takes list of list values. so we convert list to list of list

# For BOW
a_train=[]
for i in x_train_3.values:
    b=[]
    b.append(i)
    a_train.append(b)

a_cv=[]
for i in x_cv_3.values:
    b=[]
    b.append(i)
    a_cv.append(b)

a_test=[]
for i in x_test_3.values:
    b=[]
    b.append(i)
    a_test.append(b)
```

In [94]:
```
# For Training Data
tfidf_w2v_train_fe_im1=np.hstack((tfidf_w2v_train_fe_im,a_train))


# For cv Data

tfidf_w2v_cv_fe_im1=np.hstack((tfidf_w2v_cv_fe_im,a_cv))


# For test Data

tfidf_w2v_test_fe_im1=np.hstack((tfidf_w2v_test_fe_im,a_test))
```

In [95]:
```
tfidf_w2v_train_fe_im1.shape
```

Out[95]:
```
(60000, 101)
```

9.2.4 Feature engineering on Random Forest (TFIDF W2V)

In [96]:
```
tree=[5,15,30,45,60,75,90,100,200]
depth=[5,50,100,500,1000,2000,3000,4000,5000]
```

In [98]:
```
# Hyperparameter tuning

auc_train,auc_cv=Random_Forest(no_tree=tree,depth=depth,train_vector=tfidf_w2v_t
                                             cv_vector=tfidf_w2v_cv_fe_im1,cv_la
```

```
9it [07:53, 84.89s/it]
```

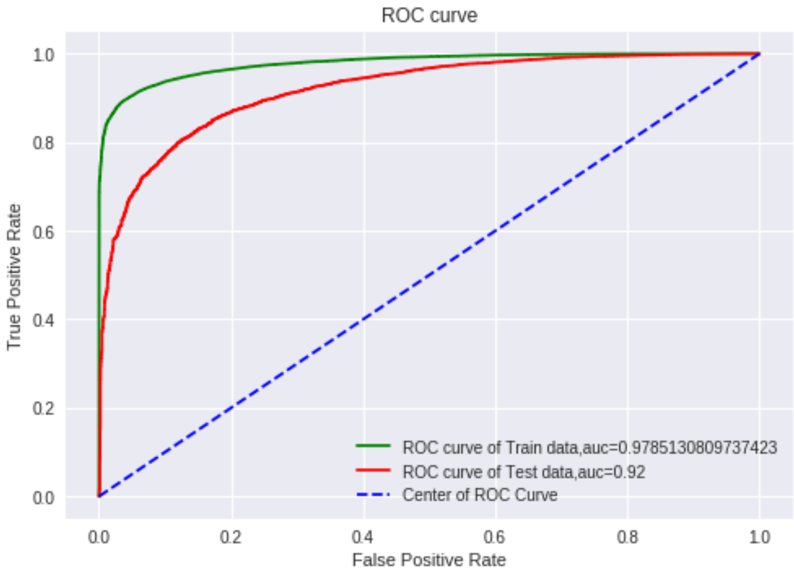
In [99]:
```
# auc_score plotting

auc_score(tree=tree,depth=depth,auc_train=auc_train,auc_cv=auc_cv)
```

Observation:

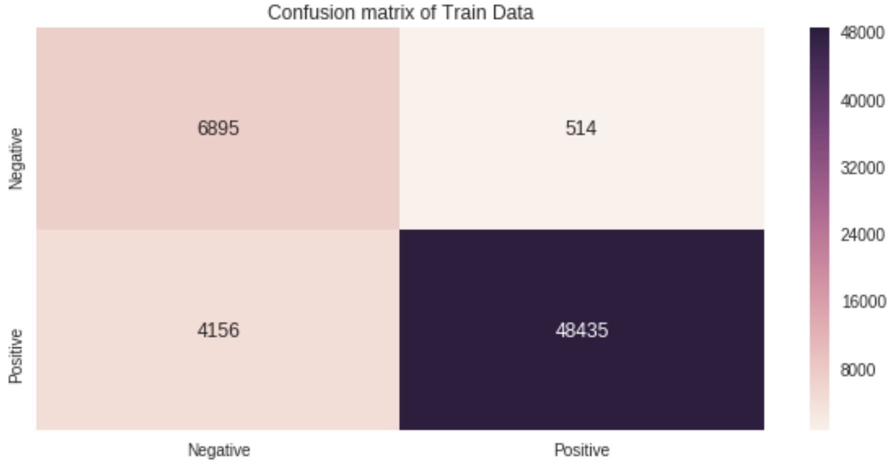
- To avoid overfitting and underfitting,choose (no of base learners=90,depth=3000), we get auc_score=0.88

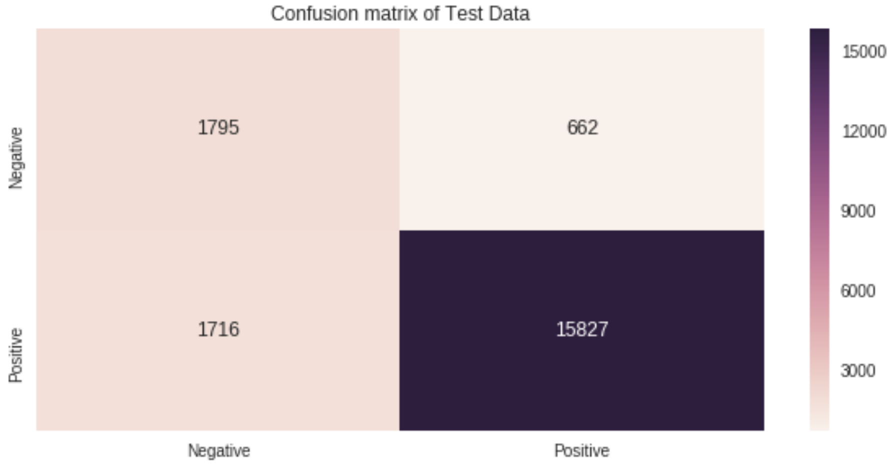
In [100]:
```
# Apply best hyperparameter

train_proba,test_proba,fpr_train,tpr_train,fpr_test,tpr_test,auc_train,auc_test,
=best_RF(best_tree=90,best_depth=3000,train_vector=tfidf_w2v_train_fe_im1,train_
                                test_vector=tfidf_w2v_test_fe_im1,test_label=y_te
```

In [101]:
```
# References
# https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-poi

# plotting ROC graph

roc_model(fpr_train=fpr_train,tpr_train=tpr_train,fpr_test=fpr_test,tpr_test=tp
        text1=str(auc_train),text2=str(round(auc_test,2)))
```



In [102]:
```
# confusion matrix

cm_plot(train_proba=train_proba,train_label=y_train,test_proba=test_proba,test_l
```





Observation:

- When we applying best hyperparameter (no of base learners=90,depth =3000) on model, we get auc score of future unseen data is 0.92

Model Observations

In [103]:
```
f = PrettyTable()

f.field_names = ["Vectorizer","Model", "Number of Base Learners", "Max_depth", '

f.add_row(["TFIDF","Random Forest",90,3000,0.92])
print(f)
```

```
+------------+---------------+-------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+-------------------------+-----------+------+
|   TFIDF    | Random Forest |            90           |    3000   | 0.92 |
+------------+---------------+-------------------------+-----------+------+
```

9.3 Model Observations

In [104]:
```
print ("After Applying Feature Engineering on Model")
print(' ')
print("Feature Engineering( Review Text + Summary)")
print(' ')
print(z)
print(' ')
print("Feature Engineering (Review Text + Summary + Length)")
print(' ')
print(f)
```

```
After Applying Feature Engineering on Model

Feature Engineering( Review Text + Summary)

+------------+---------------+------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+------------------------+-----------+------+
|   TFIDF    | Random Forest |           90           |    3000   | 0.92 |
+------------+---------------+------------------------+-----------+------+

Feature Engineering (Review Text + Summary + Length)

+------------+---------------+------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+------------------------+-----------+------+
|   TFIDF    | Random Forest |           90           |    3000   | 0.92 |
+------------+---------------+------------------------+-----------+------+
```

- After applying Feature Engineering on the Random Forest (TFIDF W2V), The Summary Text is used to improve the model performance. But the length does not make any impact on the model. So we just ignore the length feature. Therefore we will use Summary Text as a feature for further model performance improvement.

10. Conclusion

In [105]:
```
print ("1. Before Applying Feature Engineering on Model(Review Text)")
print(' ')
print(x)
print(y)
print(' ')
print ("2. After Applying Feature Engineering on Model")
print(' ')
print("Feature Engineering( Review Text + Summary)")
print(' ')
print(z)
print("Feature Engineering (Review Text + Summary + Length)")
print(' ')
print(f)
```

```
1. Before Applying Feature Engineering on Model(Review Text)

+------------+---------------+------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+------------------------+-----------+------+
|    BOW     | Random Forest |           90           |    3000   | 0.89 |
|   TFIDF    | Random Forest |           90           |    3000   | 0.89 |
|  Avg W2V   | Random Forest |           90           |    3000   | 0.9  |
| TFIDF W2V  | Random Forest |           90           |    3000   | 0.87 |
+------------+---------------+------------------------+-----------+------+
+------------+-------+------------------------+-----------+------+
| Vectorizer | Model | Number of Base Learners | Max_depth | AUC  |
+------------+-------+------------------------+-----------+------+
|    BOW     |  GBDT |          200           |     7     | 0.9  |
|   TFIDF    |  GBDT |          200           |     7     | 0.9  |
|  Avg W2V   |  GBDT |          200           |     7     | 0.91 |
| TFIDF W2V  |  GBDT |          200           |     7     | 0.89 |
+------------+-------+------------------------+-----------+------+

2. After Applying Feature Engineering on Model

Feature Engineering( Review Text + Summary)

+------------+---------------+------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+------------------------+-----------+------+
|   TFIDF    | Random Forest |           90           |    3000   | 0.92 |
+------------+---------------+------------------------+-----------+------+
Feature Engineering (Review Text + Summary + Length)

+------------+---------------+------------------------+-----------+------+
| Vectorizer |     Model     | Number of Base Learners | Max_depth | AUC  |
+------------+---------------+------------------------+-----------+------+
|   TFIDF    | Random Forest |           90           |    3000   | 0.92 |
+------------+---------------+------------------------+-----------+------+
```

Data Cleaning ,Preprocessing and splitting:

- In the Data Cleaning process, we clean the duplicate datapoints and unconditioning data points. After the data cleaning process we get 364171 data points and sort based on timestamp.
- Then select the Review Text Feature as a important feature, then do data preprocessing on all the data points.
- Then select top 100k sample data points for further process. and then split the 100k data points using simple cross validation technique. Train= 60000, CV=20000, Test=20000.

Featurization:

- Then apply the data points on BOW,TFIDF,Avg W2V and TFIDF W2V for converting text to vector.

Random Forest Model:

- Then apply these featurization vector on Random Forest model . There are two hyperparameter one is depth and another one is Number of base learners.
- Random Forest model (Avg-W2V) gives slightly better result compared to other vectorizers.

GBDT Model:

- Then apply these featurization vector on GBDT model . There are two hyperparameter one is depth and another one is Number of base learners.
- GBDT model (Avg-W2V) gives slightly better result compared to other vectorizers.

Graph visualization:

- The BoW and Tfidf models of Random Forest and GBDT were visualized by using graphviz tool.

Feature Importance and Wordcloud Visualization:

- Then took the top 20 important features both BOW and TFIDF and these features are displayed by using wordcloud tool.

Feature Engineering:

- we took TFIDF- W2V for feature engineering, because its result is less compared to other vectors.
- We will apply feature engineering for improve the Random Forest and GBDT Model performance. For consider Summary and Review Text Length as a feature.
- After applying Feature Engineering on the Random Forest Model, The Summary Text feature is improve model performance. But the length does not make any impact on the model. So we just ignore the length feature for future improvement.
- We consider the Summary Text feature for further Model performance improvement.