Amazon Fine Food Reviews Analysis

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

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

[1]. Reading Data

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


import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from tqdm import tqdm
from bs4 import BeautifulSoup

import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import PorterStemmer
from nltk.stem import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer,TfidfTransformer
from sklearn.metrics import confusion_matrix,accuracy_score,roc_auc_score,auc,roc_curve,classification_report,precision_score,rec
all_score,f1_score, hamming_loss

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from prettytable import PrettyTable

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

from tqdm import tqdm
import os
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('D:\Study_materials\Applied_AI\Assignments\database.sqlite')

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 negative rating(0).
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)
```

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

In [3]:
```
sample_data = filtered_data.head(100000)
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed 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 [4]:
```
#Sorting data according to ProductId in ascending order
sorted_data=sample_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [5]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[5]:
```
(87775, 10)
```

In [6]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(sample_data['Id'].size*1.0)*100
```

Out[6]:
```
87.775
```

In [7]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [8]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(87773, 10)
```

Out[8]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

In [9]:
```
final.head()
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1	0	1192060800	made in china	My dogs loves this chicken but its a product f...
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0	1	1195948800	Dog Lover Delites	Our dogs just love them. I saw them in a pet ...
70677	76870	B00002N8SM	A19Q006CSFT011	Arlielle	0	0	0	1288396800	only one fruitfly stuck	I had an infestation of fruitflies, they were ...
70676	76869	B00002N8SM	A1FYH4S02BW7FN	wonderer	0	0	0	1290038400	Doesn't work!! Don't waste your money!!	Worst product I have gotten in long time. Woul...
70675	76868	B00002N8SM	AUE8TB5VHS6ZV	eyeofthestorm	0	0	0	1306972800	A big rip off	I wish I'd read the reviews before making this...

[3] Preprocessing

In [10]:
```
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
4
I wish I'd read the reviews before making this purchase. It's basically a cardsotck box that is sticky all over the OUTSIDE. Thos
e pink-ish things that look like entrances "into" the trap? They're just pictures. There *is no* inside of the trap. All the flie
s will be stuck to the OUTSIDE. It's basically fly paper, just horribly, horribly HORRIBLY overpriced.<br /><br />Do yourself a f
avor and just get fly paper or fly strips. Same yuck factor, but much cheaper.
```

In [11]:
```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('************************************')
print(sno.stem('tasty'))
```

```
{"doesn't", 'before', 'into', 'being', 'didn', 'off', "won't", 'more', "mustn't", 'had', 're', 'an', 'wouldn', "didn't", 'throug
h', 'haven', 'mustn', 'this', 'own', "couldn't", "wouldn't", 'd', 'the', 'she', "aren't", 'whom', 'ourselves', 'doesn', 'out', 'h
ers', 'now', 'hasn', 'of', "shan't", 'if', 'than', 'having', 'ours', 't', 'how', 'will', 'what', 'am', 'until', 'between', 'it',
'be', "you're", 'most', 'theirs', 'yourself', 'again', "it's", 'do', 'doing', 'couldn', 'does', 'but', 'll', 'in', 'few', 'such',
'who', 'himself', "don't", "she's", 'any', 'its', "hasn't", 'against', 'aren', 'were', "you'll", 'can', 'because', 'about', 'shou
ld', 'isn', 'her', 'by', 'themselves', 'for', 'have', 'on', 'a', 'each', 'm', 'not', 'further', 'they', 'me', 'is', 'only', 'bot
h', 'and', 'with', 'herself', "haven't", 'don', 'their', 'we', 'did', 'under', 'nor', 've', 'myself', 'here', 'as', "should've",
's', 'after', 'i', 'all', 'yourselves', 'over', 'been', 'him', 'below', 'your', 'while', "wasn't", 'our', 'ain', 'once', 'weren',
'has', 'them', "you've", 'at', 'hadn', 'where', 'some', 'was', 'when', 'wasn', 'o', 'just', 'from', "mightn't", 'yours', 'why',
'won', 'his', "shouldn't", "isn't", "that'll", "you'd", 'to', 'are', 'you', 'mightn', 'needn', "needn't", 'he', "hadn't", 'or',
'up', 'these', 'my', 'then', 'other', 'shouldn', 'there', 'which', 'ma', 'those', 'above', 'during', 'itself', 'no', 'down', 'ver
y', 'shan', 'that', 'y', 'so', 'same', 'too', "weren't"}
************************************
tasti
```

In [12]:
```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.

final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
for i, sent in enumerate(tqdm(final['Text'].values)):
    filtered_sentence=[]
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
        # we have used cleanpunc(w).split(), one more split function here because consider w="abc.def", cleanpunc(w) will return
 "abc def"
        # if we dont use .split() function then we will be considring "abc def" as a single word, but if you use .split() function
we will get "abc", "def"
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8') #snoball stemmer
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 1:
                        all_positive_words.append(s) #list of all words used to describe positive reviews
                    if(final['Score'].values)[i] == 0:
                        all_negative_words.append(s) #list of all words used to describe negative reviews reviews
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")
    final_string.append(str1)

    #############---- storing the data into .sqlite file ------########################
final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-processing of the review
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
```

```
100%|██████████████████████████████████████████████████████████████████████████| 87773/87773 [05:14<00:00, 279.37it/s]
```

In [13]:
```
final = final.sort_values('Time',axis = 0,ascending = True, inplace = False, kind = 'quicksort', na_position='last')
```

In [14]:
```
final.columns
```

Out[14]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
       'CleanedText'],
      dtype='object')
```

In [15]:
```
X  = final['CleanedText'].values
y = final['Score']
```

In [16]:
```
# Creating training, test and cross validation set

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size= 0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_train,y_train, test_size = 0.3, random_state=0)
```

In [17]:
```
print("Size of X_train and y_train:", X_train.shape,y_train.shape)
print("Size of X_test and y_test:", X_test.shape,y_test.shape)
print("Size of X_tr and y_tr:", X_tr.shape,y_tr.shape)
print("Size of X_cv and y_cv:", X_cv.shape,y_cv.shape)
```

```
Size of X_train and y_train: (61441,) (61441,)
Size of X_test and y_test: (26332,) (26332,)
Size of X_tr and y_tr: (43008,) (43008,)
Size of X_cv and y_cv: (18433,) (18433,)
```

RandomForest Classifier

In [29]:
```
from sklearn.ensemble import RandomForestClassifier

def RF_Classifier(X_train,X_cv,y_train,y_cv):
    pred_train = []
    pred_cv = []
    t_depth = [2,3,5,8,10,20]
    estimators = [100,200,300,400,500]
    for i in t_depth:
        for j in estimators:
            clf = RandomForestClassifier(n_estimators = j, max_depth = i, n_jobs = -1, class_weight='balanced')
            clf.fit(X_train,y_train)
            prob_train = clf.predict_proba(X_train)[:,1]
            prob_cv = clf.predict_proba(X_cv)[:,1]
            auc_score_train = roc_auc_score(y_train,prob_train)
            auc_score_cv = roc_auc_score(y_cv,prob_cv)
            pred_train.append(auc_score_train)
            pred_cv.append(auc_score_cv)
    cmap=sns.light_palette("green")
    # representing heat map for auc score
    print("-"*40, "AUC Score for train data", "-"*40)
    pred_train = np.array(pred_train)
    pred_train = pred_train.reshape(len(t_depth),len(estimators))
    plt.figure(figsize=(10,5))
    sns.heatmap(pred_train,annot=True, cmap=cmap, fmt=".3f", xticklabels=estimators,yticklabels=t_depth)
    plt.xlabel('Estimator')
    plt.ylabel('Depth')
    plt.show()
    print("-"*40, "AUC Score for CV data", "-"*40)
    pred_cv = np.array(pred_cv)
    pred_cv = pred_cv.reshape(len(t_depth),len(estimators))
    plt.figure(figsize=(10,5))
    sns.heatmap(pred_cv, annot=True, cmap=cmap, fmt=".3f", xticklabels=estimators, yticklabels=t_depth)
    plt.xlabel('Estimator')
    plt.ylabel('Depth')
    plt.show()
```

Testing model:

In [33]:
```
import scikitplot.metrics as skplt

def testing(X_train,y_train,X_test,y_test,optimal_depth,optimal_estimator):
    clf = RandomForestClassifier(n_estimators = optimal_estimator, max_depth = optimal_depth,class_weight='balanced')
    clf.fit(X_train,y_train)
    prob_train = clf.predict_proba(X_train)[:,1]
    prob_test = clf.predict_proba(X_test)[:,1]

    print("AUC Score for train data",roc_auc_score(y_train,prob_train))
    print("AUC Score for test data",roc_auc_score(y_test,prob_test))
    # calculate roc curve
    fpr_train, tpr_train, threshold_tr = roc_curve(y_train,prob_train)
    fpr_test, tpr_test, threshold_te = roc_curve(y_test,prob_test)

    # plot the roc curve for the model

    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.plot(fpr_train, tpr_train, marker='.',color= 'r',label='Train Data')
    plt.plot(fpr_test, tpr_test, marker='.',color ='b',label='Test Data')
    plt.title("Line Plot of ROC Curve on Train Data and Test Data")
    plt.legend(loc='upper left')
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

    #plot confusion matrix

    prediction_train=clf.predict(X_train)
    prediction_test=clf.predict(X_test)

    print("macro f1 score for train data :",f1_score(y_train, prediction_train, average = 'macro'))
    print("macro f1 score for test data :",f1_score(y_test, prediction_test, average = 'macro'))
    print("micro f1 score for train data:",f1_score(y_train, prediction_train, average = 'micro'))
    print("micro f1 score for test data:",f1_score(y_test, prediction_test, average = 'micro'))
    print("hamming loss for train data:",hamming_loss(y_train,prediction_train))
    print("hamming loss for test data:",hamming_loss(y_test,prediction_test))
    print("Precision recall report for train data:\n",classification_report(y_train, prediction_train))
    print("Precision recall report for test data:\n",classification_report(y_test, prediction_test))
    skplt.plot_confusion_matrix(y_train,prediction_train,title='Confusion Matrix - Train Data')
    skplt.plot_confusion_matrix(y_test,prediction_test,title='Confusion Matrix - Test Data')
```

XGBoost Classifier

In [37]:
```
from xgboost import XGBClassifier

def xgb_Classifier(X_train,X_cv,y_train,y_cv):
    pred_cv = []
    pred_train = []
    depths = [2, 3, 5, 8, 10, 20]
    estimators = [100, 200, 300, 400, 500]
    for i in depths:
        for j in estimators:
            clf = XGBClassifier(n_estimators=j, max_depth=i, scale_pos_weight=1, objective='binary:logistic')
            clf.fit(X_train,y_train)
            prob_train = clf.predict_proba(X_train)[:,1]
            prob_cv = clf.predict_proba(X_cv)[:,1]
            auc_score_train = roc_auc_score(y_train,prob_train)
            auc_score_cv = roc_auc_score(y_cv,prob_cv)
            pred_train.append(auc_score_train)
            pred_cv.append(auc_score_cv)
    cmap=sns.light_palette("green")
    # representing heat map for auc score
    print("-"*40, "AUC Score for train data", "-"*40)
    pred_train = np.array(pred_train)
    pred_train = pred_train.reshape(len(depths),len(estimators))
    plt.figure(figsize=(10,5))
    sns.heatmap(pred_train,annot=True, cmap=cmap, fmt=".3f", xticklabels=estimators,yticklabels=depths)
    plt.xlabel('Estimators')
    plt.ylabel('Depths')
    plt.show()
    print("-"*40, "AUC Score for CV data", "-"*40)
    pred_cv = np.array(pred_cv)
    pred_cv = pred_cv.reshape(len(depths),len(estimators))
    plt.figure(figsize=(10,5))
    sns.heatmap(pred_cv, annot=True, cmap=cmap, fmt=".3f", xticklabels=estimators, yticklabels=depths)
    plt.xlabel('Estimators')
    plt.ylabel('Depths')
    plt.show()
```

Testing Model

In [41]:
```
import scikitplot.metrics as skplt
def testing1(X_train,y_train,X_test,y_test,optimal_depth,optimal_estimator):
    clf = XGBClassifier(n_estimators = optimal_estimator, max_depth = optimal_depth)
    clf.fit(X_train,y_train)
    prob_train1 = clf.predict_proba(X_train)[:,1]
    prob_test1 = clf.predict_proba(X_test)[:,1]

    print("AUC Score for train data",roc_auc_score(y_train,prob_train1))
    print("AUC Score for test data",roc_auc_score(y_test,prob_test1))

    fpr_train, tpr_train, thresholds = roc_curve(y_train,prob_train1)
    fpr_test, tpr_test, thresholds = roc_curve(y_test,prob_test1)

    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.plot(fpr_train, tpr_train, marker='.',color= 'r',label='Train Data')
    plt.plot(fpr_test, tpr_test, marker='.',color ='b',label='Test Data')
    plt.title("Line Plot of ROC Curve on Train Data and Test Data")
    plt.legend(loc='upper left')
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
    #plot confusion matrix
    prediction_train=clf.predict(X_train)
    prediction_test=clf.predict(X_test)
    print("macro f1 score for train data :",f1_score(y_train, prediction_train, average = 'macro'))
    print("macro f1 score for test data :",f1_score(y_test, prediction_test, average = 'macro'))
    print("micro f1 score for train data:",f1_score(y_train, prediction_train, average = 'micro'))
    print("micro f1 score for test data:",f1_score(y_test, prediction_test, average = 'micro'))
    print("hamming loss for train data:",hamming_loss(y_train,prediction_train))
    print("hamming loss for test data:",hamming_loss(y_test,prediction_test))
    print("Precision recall report for train data:\n",classification_report(y_train, prediction_train))
    print("Precision recall report for test data:\n",classification_report(y_test, prediction_test))
    skplt.plot_confusion_matrix(y_train,prediction_train,title='Confusion Matrix for Train Data')
    skplt.plot_confusion_matrix(y_test,prediction_test,title='Confusion Matrix for Test Data')
```

Top 20 features:

In [23]:
```
from wordcloud import WordCloud

def imp_feature(vectorizer,classifier, n =20):
    features = []
    feature_names = vectorizer.get_feature_names()
    coefs = sorted(zip(classifier.feature_importances_, feature_names))
    top = coefs[:-(n + 1):-1]
    print('\033[1m' + "feature_importances\tfeatures" + '\033[0m')
    print("="*35)
    for (coef1, feat1) in top:
        print("%.4f\t\t\t%-15s" % (coef1, feat1))
        features.append(feat1)
    wordcloud = WordCloud(background_color='black',width=1600,height=800).generate(" ".join(features))    #top 20 features in wor
d cloud
    fig = plt.figure(figsize=(30,20))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.tight_layout(pad=0)
    #fig.savefig("features.png")
    plt.show()
```

Techniques for vectorization :--

1. Bag of Words (BoW)

In [24]:
```
count_vec = CountVectorizer()
BOW_X_train = count_vec.fit_transform(X_tr)
BOW_X_cv = count_vec.transform(X_cv)
BOW_X_test = count_vec.transform(X_test)
```

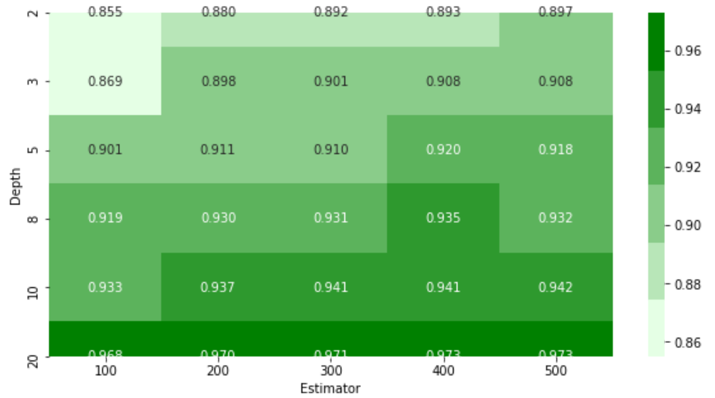
In [25]:
```
#Standardizing data using StandardScaler

sc = StandardScaler(with_mean=False)
BOW_X_train_sc = sc.fit_transform(BOW_X_train)
BOW_X_cv_sc = sc.transform(BOW_X_cv)
BOW_X_test_sc = sc.transform(BOW_X_test)

print("The shape of out text BOW vectorizer ",BOW_X_train_sc.get_shape())
print("CV Data Size: ",BOW_X_cv_sc.shape)
print("Test Data Size: ",BOW_X_test_sc.shape)
```

```
The shape of out text BOW vectorizer  (43008, 24467)
CV Data Size:  (18433, 24467)
Test Data Size:  (26332, 24467)
```

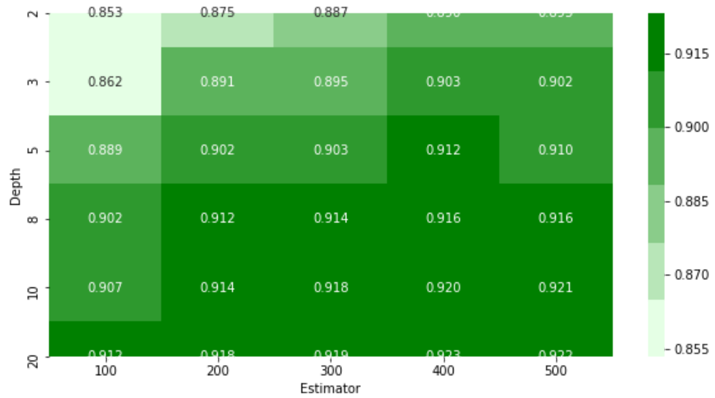
Training RandomForest Classifier:

In [31]:
```
RF_Classifier(BOW_X_train_sc,BOW_X_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



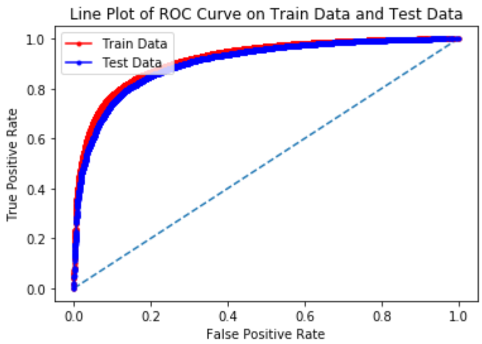
```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



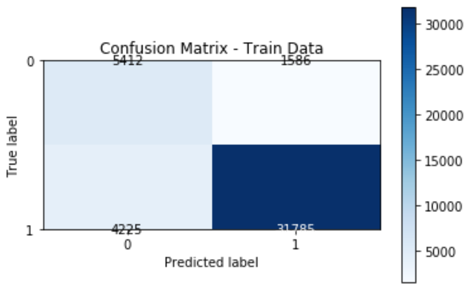
Testing RandomForest Classifier:

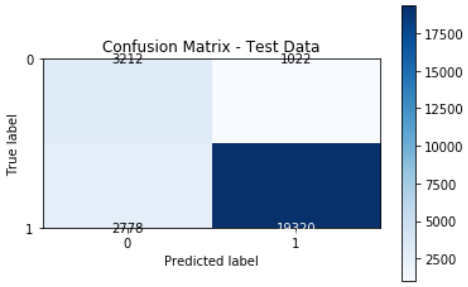
In [34]:
```
import scikitplot

testing(BOW_X_train_sc,y_tr,BOW_X_test_sc,y_test,optimal_depth=5,optimal_estimator=400)
```

```
AUC Score for train data 0.9180368489461701
AUC Score for test data 0.9046712911904043
```




```
macro f1 score for train data : 0.7834606832239264
macro f1 score for test data : 0.7693936685354561
micro f1 score for train data: 0.8648856026785714
micro f1 score for test data: 0.8556888956402856
hamming loss for train data: 0.13511439732142858
hamming loss for test data: 0.14431110435971442
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.56      0.77      0.65      6998
           1       0.95      0.88      0.92     36010

    accuracy                           0.86     43008
   macro avg       0.76      0.83      0.78     43008
weighted avg       0.89      0.86      0.87     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.54      0.76      0.63      4234
           1       0.95      0.87      0.91     22098

    accuracy                           0.86     26332
   macro avg       0.74      0.82      0.77     26332
weighted avg       0.88      0.86      0.87     26332
```

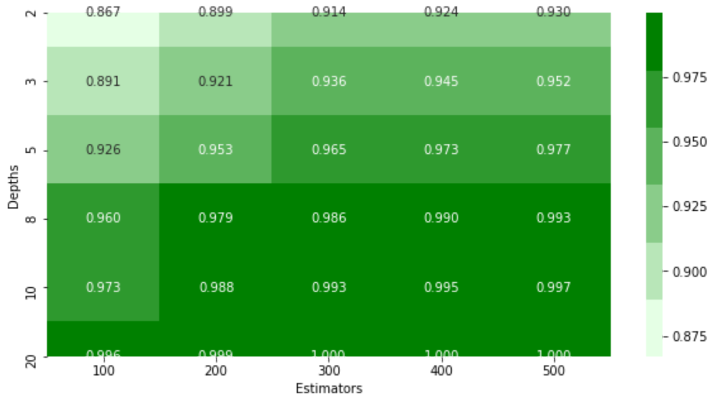




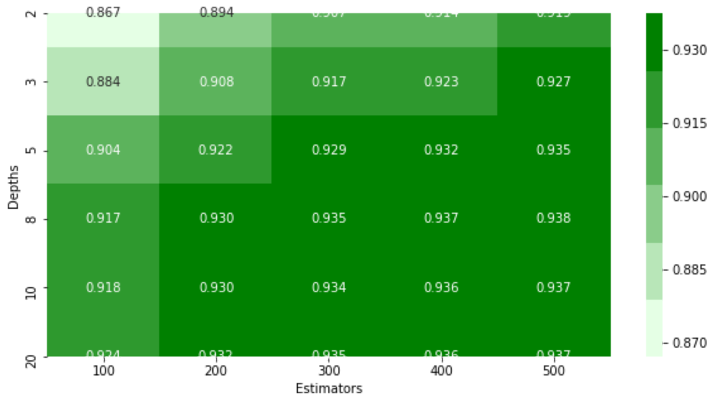
Training XGBoost Classifier:

In [39]:
```
xgb_Classifier(BOW_X_train_sc,BOW_X_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



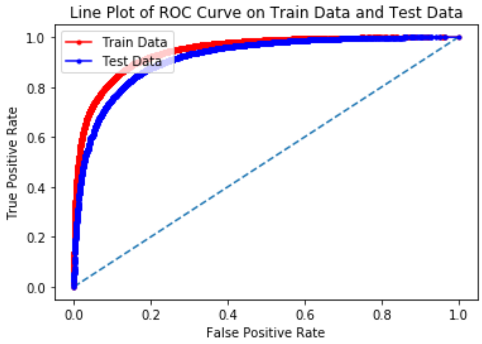

```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



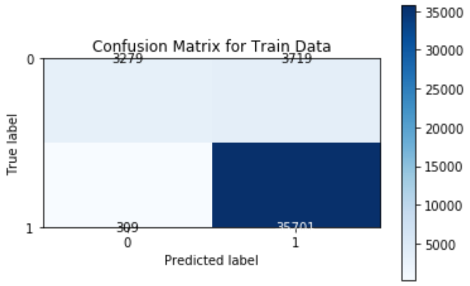
Testing XGBoost Classifier:

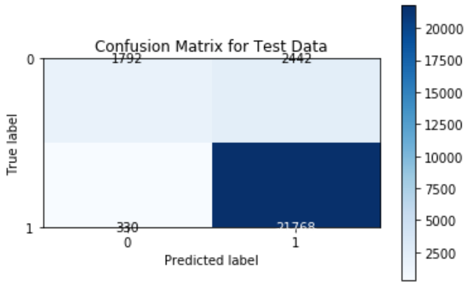
In [42]:
```
testing1(BOW_X_train_sc,y_tr,BOW_X_test_sc,y_test,optimal_depth=3,optimal_estimator=400)
```

```
AUC Score for train data 0.9447968987687916
AUC Score for test data 0.9181528695573584
```




```
macro f1 score for train data : 0.7830484728416077
macro f1 score for test data : 0.7520082923037057
micro f1 score for train data: 0.9063430059523809
micro f1 score for test data: 0.8947288470302294
hamming loss for train data: 0.09365699404761904
hamming loss for test data: 0.10527115296977062
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.91      0.47      0.62      6998
           1       0.91      0.99      0.95     36010

    accuracy                           0.91     43008
   macro avg       0.91      0.73      0.78     43008
weighted avg       0.91      0.91      0.89     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.84      0.42      0.56      4234
           1       0.90      0.99      0.94     22098

    accuracy                           0.89     26332
   macro avg       0.87      0.70      0.75     26332
weighted avg       0.89      0.89      0.88     26332
```



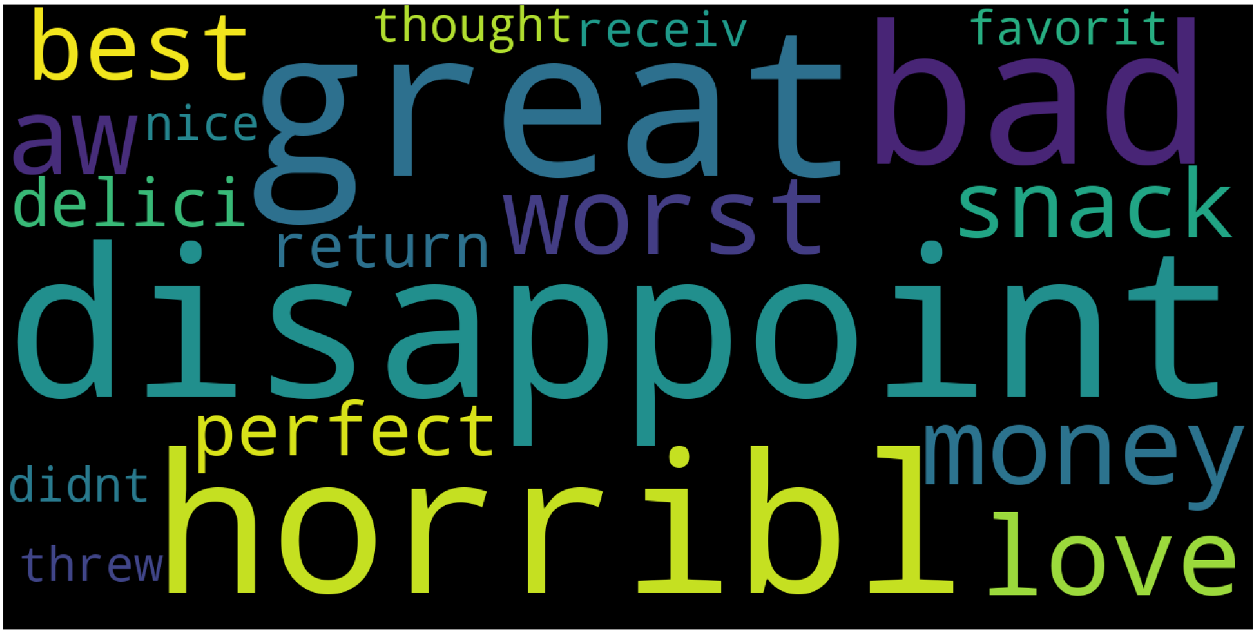


Top 20 Features:

For RandomForest:

In [43]:
```
clf = RandomForestClassifier(max_depth =5, n_estimators = 400,class_weight='balanced')
clf.fit(BOW_X_train_sc,y_tr)
features = imp_feature(count_vec,clf)
```

```
feature_importances      features
===================================
0.0212                   disappoint
0.0197                   great
0.0192                   horribl
0.0183                   bad
0.0174                   money
0.0155                   love
0.0143                   worst
0.0140                   aw
0.0139                   snack
0.0130                   best
0.0124                   perfect
0.0124                   delici
0.0120                   return
0.0117                   thought
0.0112                   favorit
0.0111                   would
0.0109                   didnt
0.0109                   threw
0.0099                   nice
0.0088                   receiv
```



For XGBoost:

In [44]:
```
clf = XGBClassifier(max_depth =3, n_estimators = 400,class_weight='balanced')
clf.fit(BOW_X_train_sc,y_tr)
features = imp_feature(count_vec,clf)
```

```
feature_importances      features
===================================
0.0251                   disappoint
0.0162                   return
0.0148                   great
0.0145                   horribl
0.0142                   love
0.0127                   wast
0.0117                   money
0.0116                   worst
0.0109                   terribl
0.0108                   aw
0.0097                   best
0.0094                   bad
0.0093                   delici
0.0093                   threw
0.0086                   refund
0.0083                   thought
0.0082                   favorit
0.0079                   perfect
0.0077                   mayb
0.0077                   disgust
```

2. TF-IDF

In [45]:
```
tf_idf_vec = TfidfVectorizer(ngram_range=(1,2))
tfidf_train = tf_idf_vec.fit_transform(X_tr)
tfidf_cv = tf_idf_vec.transform(X_cv)
tfidf_test = tf_idf_vec.transform(X_test)

print("The type of count vectorizer ",type(tfidf_train))
print("The shape of out text TFIDF vectorizer ",tfidf_train.get_shape())
print("Size of CV dataset:", tfidf_cv.shape)
print("Size of test dataset:", tfidf_test.shape)
```

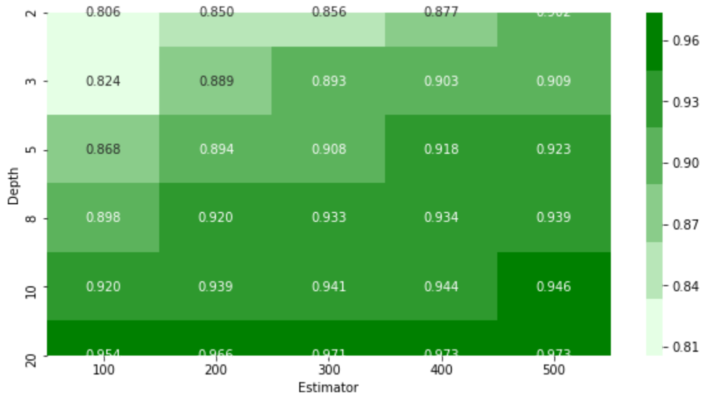
```
The type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of out text TFIDF vectorizer  (43008, 683623)
Size of CV dataset: (18433, 683623)
Size of test dataset: (26332, 683623)
```

In [46]:
```
#Standardizing data using StandardScaler

sc = StandardScaler(with_mean=False)
tfidf_train_sc = sc.fit_transform(tfidf_train)
tfidf_cv_sc = sc.transform(tfidf_cv)
tfidf_test_sc = sc.transform(tfidf_test)
```

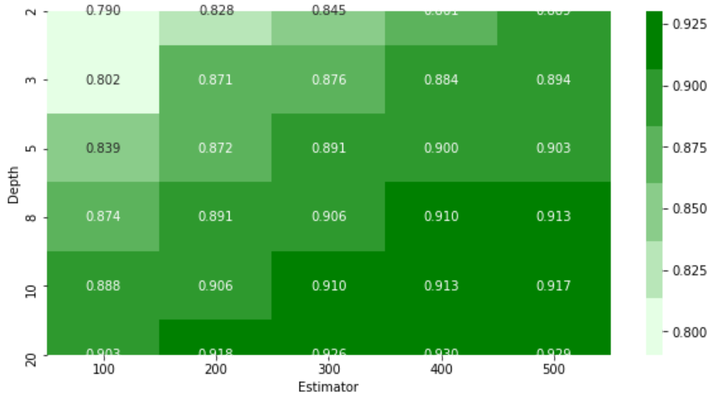
Training RandomForest Classifier

In [47]:
```
RF_Classifier(tfidf_train_sc,tfidf_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



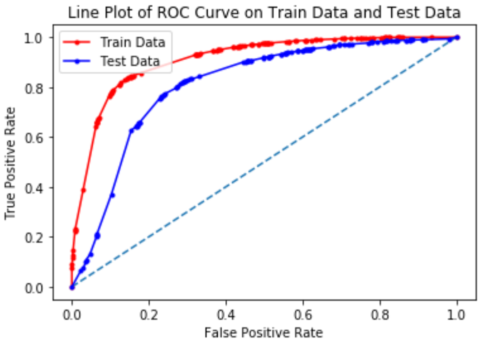
```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



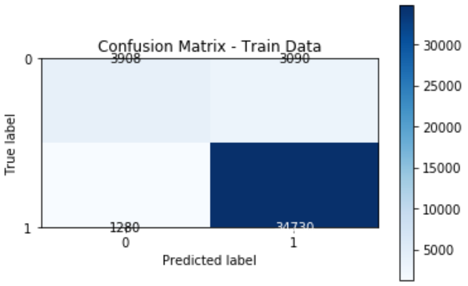
Testing RandomForest Classifier:

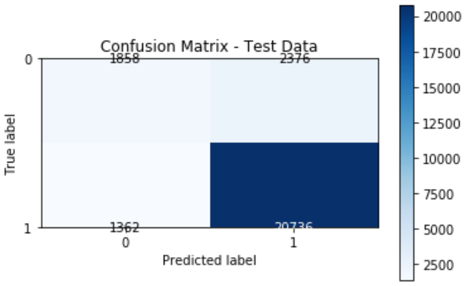
In [35]:
```
testing(tfidf_train_sc,y_tr,tfidf_test_sc,y_test,optimal_depth=5,optimal_estimator=400)
```

```
AUC Score for train data 0.911302287026269
AUC Score for test data 0.8101831770299801
```




```
macro f1 score for train data : 0.7911008649423039
macro f1 score for test data : 0.7079217297188739
micro f1 score for train data: 0.8983909970238095
micro f1 score for test data: 0.8580434452377336
hamming loss for train data: 0.10160900297619048
hamming loss for test data: 0.14195655476226646
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.75      0.56      0.64      6998
           1       0.92      0.96      0.94     36010

    accuracy                           0.90     43008
   macro avg       0.84      0.76      0.79     43008
weighted avg       0.89      0.90      0.89     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.58      0.44      0.50      4234
           1       0.90      0.94      0.92     22098

    accuracy                           0.86     26332
   macro avg       0.74      0.69      0.71     26332
weighted avg       0.85      0.86      0.85     26332
```

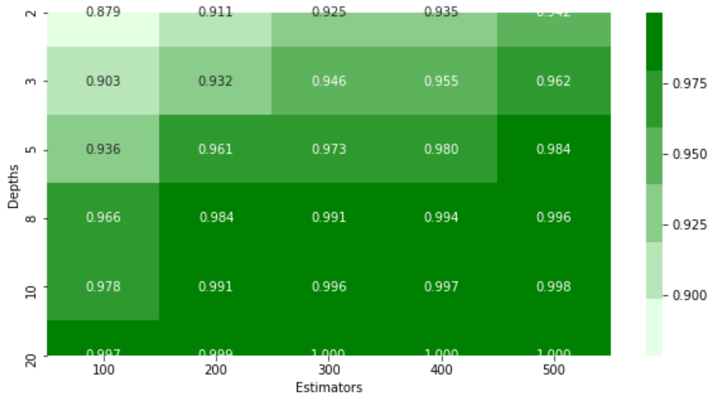




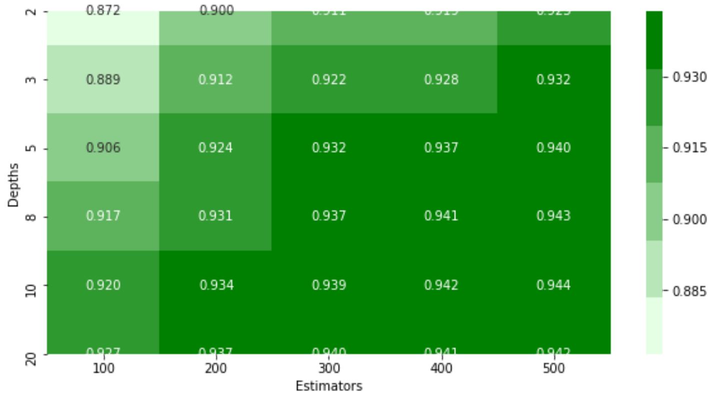
Training XGBoost Classifier

```
In [49]: xgb_Classifier(tfidf_train_sc,tfidf_cv_sc,y_tr,y_cv)
```

```
-------------------------------------- AUC Score for train data --------------------------------------
```



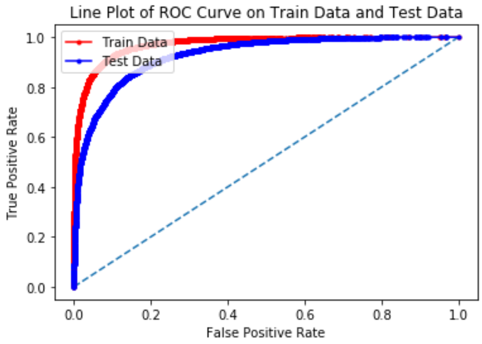

```
-------------------------------------- AUC Score for CV data --------------------------------------
```



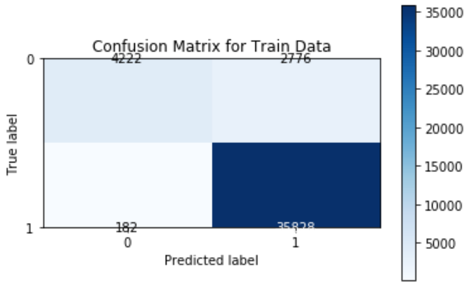
Testing XGBoost Classifier:

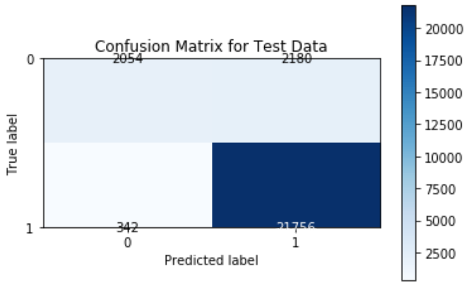
In [50]:
```
testing1(tfidf_train_sc,y_tr,tfidf_test_sc,y_test,optimal_depth=5,optimal_estimator=300)
```

```
AUC Score for train data 0.9726735388910658
AUC Score for test data 0.9286331845607403
```




```
macro f1 score for train data : 0.8504638974360126
macro f1 score for test data : 0.7824111249400486
micro f1 score for train data: 0.9312220982142857
micro f1 score for test data: 0.9042229986328422
hamming loss for train data: 0.06877790178571429
hamming loss for test data: 0.09577700136715783
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.96      0.60      0.74      6998
           1       0.93      0.99      0.96     36010

    accuracy                           0.93     43008
   macro avg       0.94      0.80      0.85     43008
weighted avg       0.93      0.93      0.92     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.86      0.49      0.62      4234
           1       0.91      0.98      0.95     22098

    accuracy                           0.90     26332
   macro avg       0.88      0.73      0.78     26332
weighted avg       0.90      0.90      0.89     26332
```



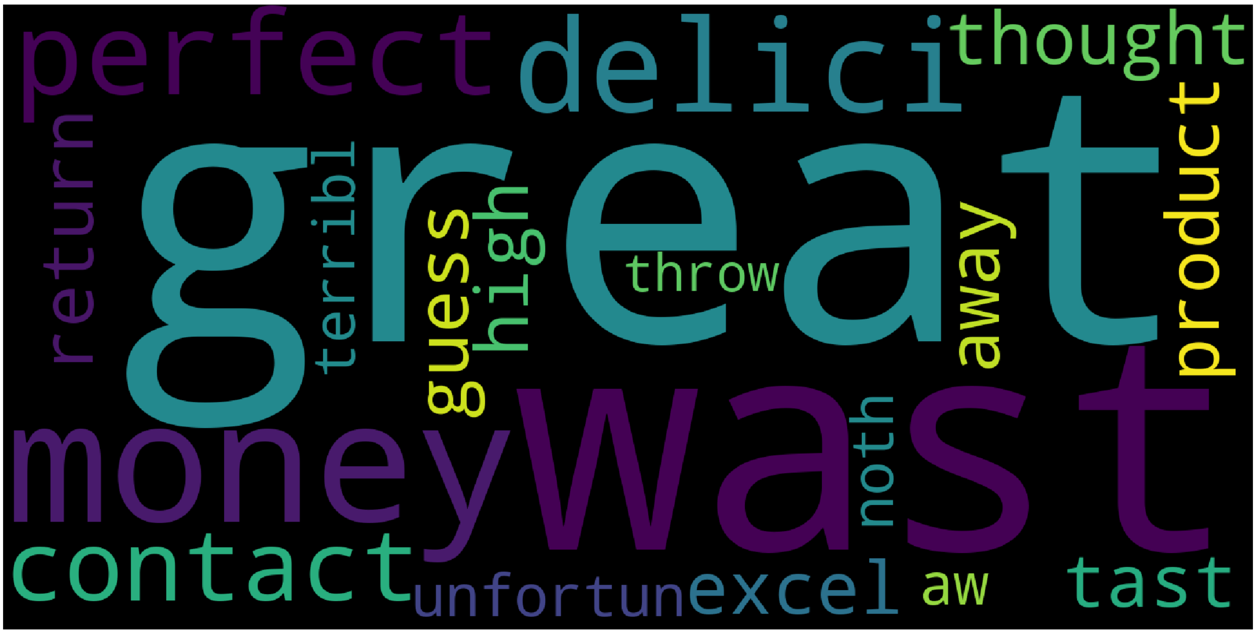


Top 20 Features:

RandomForest

In [51]:
```
clf = RandomForestClassifier(max_depth =5, n_estimators = 400,class_weight='balanced')
clf.fit(tfidf_train_sc,y_tr)
features = imp_feature(tf_idf_vec,clf)
```

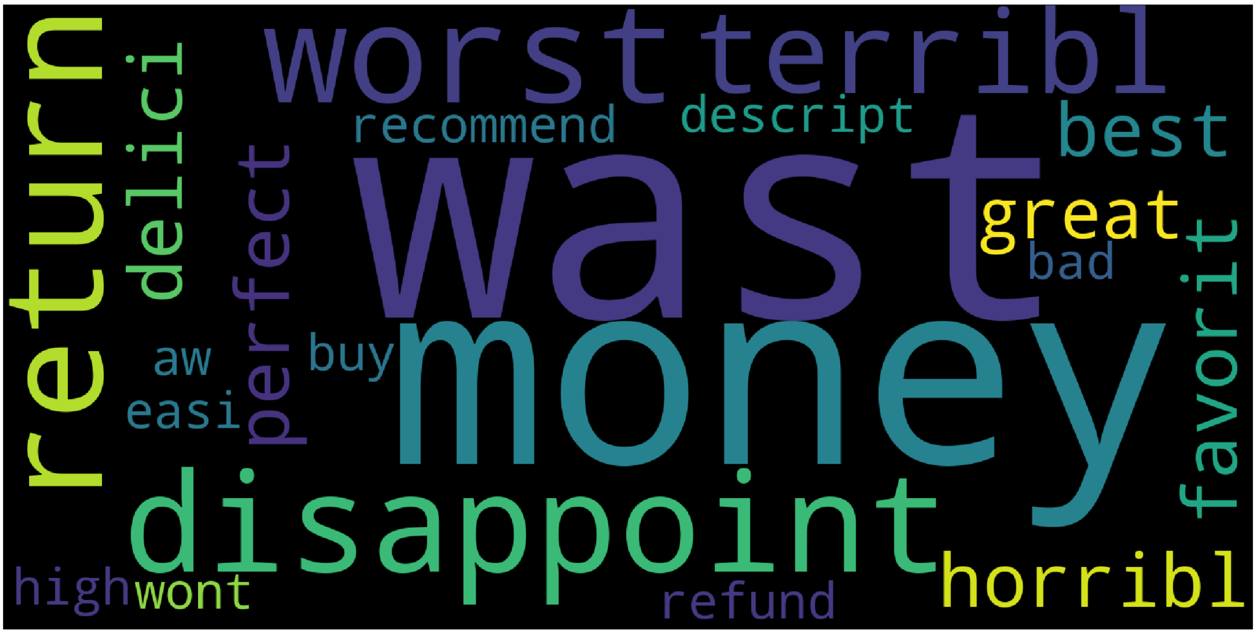
```
feature_importances      features
===================================
0.0174                   delici
0.0091                   great
0.0089                   perfect
0.0073                   wast money
0.0070                   contact
0.0066                   thought
0.0065                   return
0.0062                   tast great
0.0060                   excel
0.0058                   guess
0.0058                   product
0.0057                   high
0.0054                   away
0.0053                   unfortun
0.0048                   terribl
0.0047                   money
0.0046                   wast
0.0044                   aw
0.0044                   noth
0.0044                   throw
```



XGBoost

In [52]:
```
clf = XGBClassifier(max_depth =5, n_estimators = 300,class_weight='balanced')
clf.fit(tfidf_train_sc,y_tr)
features = imp_feature(tf_idf_vec,clf)
```

```
feature_importances      features
===================================
0.0126                   disappoint
0.0124                   wast money
0.0112                   worst
0.0098                   return
0.0089                   terribl
0.0084                   horribl
0.0080                   best
0.0080                   money
0.0077                   delici
0.0076                   perfect
0.0073                   favorit
0.0069                   great
0.0068                   descript
0.0066                   aw
0.0066                   wast
0.0066                   bad
0.0065                   high recommend
0.0065                   refund
0.0063                   easi
0.0063                   wont buy
```



3. Avg-W2V

In [53]:
```
i=0
list_sent_train=[]
for sent in X_tr:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_sent_train.append(filtered_sentence)
```

In [54]:
```
i=0
list_sent_train1=[]
for sent in X_tr:
    filtered_sentence=[]
    sent=sent
    for w in sent.split():
        for cleaned_words in w.split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_sent_train1.append(filtered_sentence)
```

In [55]:
```
i=0
list_sent_CV=[]
for sent in X_cv:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_sent_CV.append(filtered_sentence)
```

In [56]:
```
i=0
list_sent_test=[]
for sent in X_test:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_sent_test.append(filtered_sentence)
```

In [57]:
```
import gensim
w2v_model = gensim.models.Word2Vec(list_sent_train,min_count=5,size=50,workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [58]:
```
def avg_w2v(list_of_sent):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in list_of_sent: # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
    return sent_vectors
```

In [59]:
```
train_avgw2v = avg_w2v(list_sent_train)
```

```
43008
50
```

In [60]:
```
cv_avgw2v = avg_w2v(list_sent_CV)
```

```
18433
50
```

In [61]:
```
test_avgw2v = avg_w2v(list_sent_test)
```

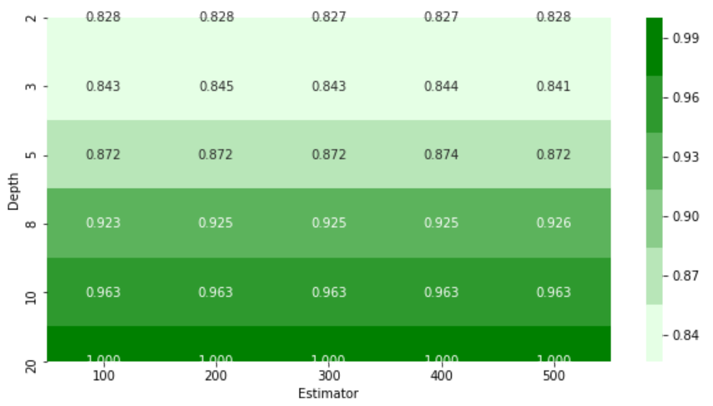
```
26332
50
```

In [62]:
```
#Standardizing data using StandardScaler

sc = StandardScaler(with_mean=False)
aw2v_X_train_sc = sc.fit_transform(train_avgw2v)
aw2v_X_cv_sc = sc.transform(cv_avgw2v)
aw2v_X_test_sc = sc.transform(test_avgw2v)
```

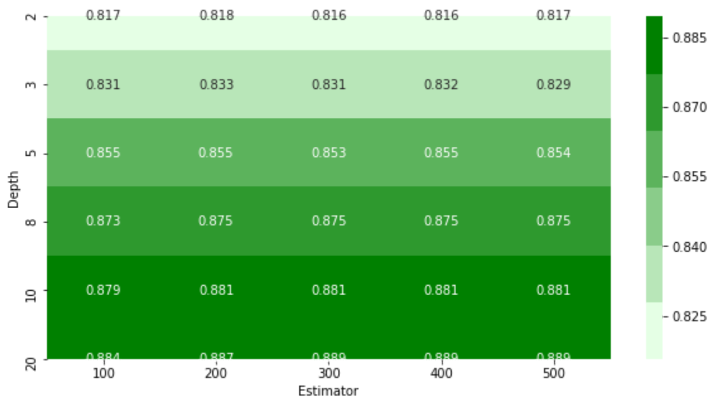
Training RandomForest Classifier:

In [64]:
```
RF_Classifier(aw2v_X_train_sc,aw2v_X_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



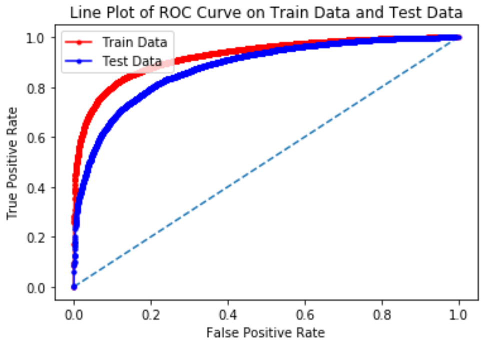

```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



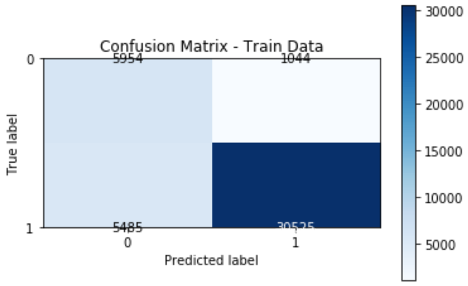
Testing RandomForest Classifier:

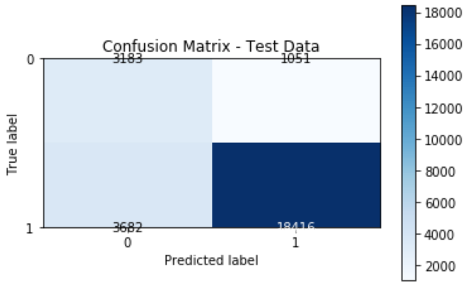
In [65]:
```
testing(aw2v_X_train_sc,y_tr,aw2v_X_test_sc,y_test,optimal_depth=8,optimal_estimator=300)
```

```
AUC Score for train data 0.925272660915774
AUC Score for test data 0.8792803970700706
```




```
macro f1 score for train data : 0.7746311446357439
macro f1 score for test data : 0.7298476718186518
micro f1 score for train data: 0.8481910342261905
micro f1 score for test data: 0.8202567218593346
hamming loss for train data: 0.15180896577380953
hamming loss for test data: 0.17974327814066535
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.52      0.85      0.65      6998
           1       0.97      0.85      0.90     36010

    accuracy                           0.85     43008
   macro avg       0.74      0.85      0.77     43008
weighted avg       0.89      0.85      0.86     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.46      0.75      0.57      4234
           1       0.95      0.83      0.89     22098

    accuracy                           0.82     26332
   macro avg       0.70      0.79      0.73     26332
weighted avg       0.87      0.82      0.84     26332
```

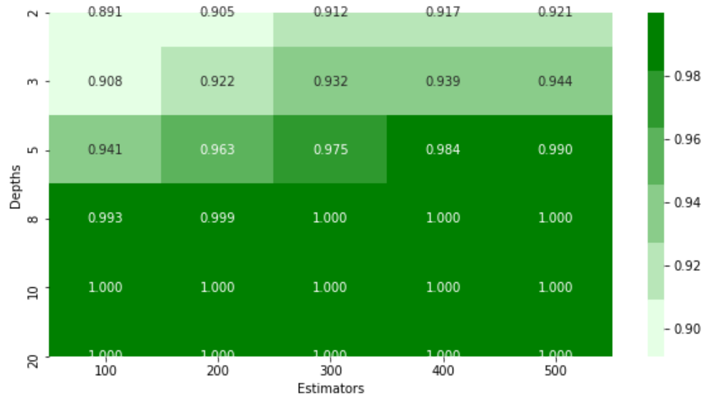




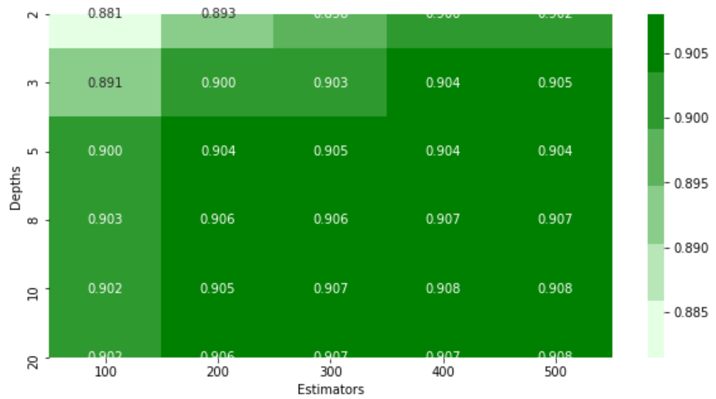
Training XGBoost Classifier:

In [66]:
```
xgb_Classifier(aw2v_X_train_sc,aw2v_X_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



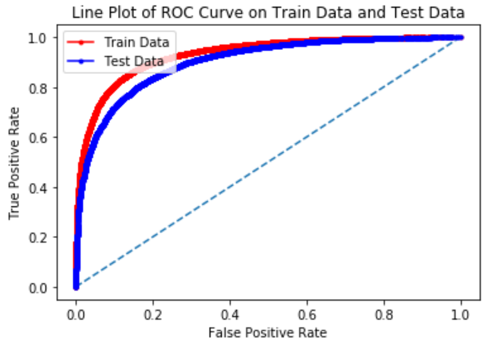

```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



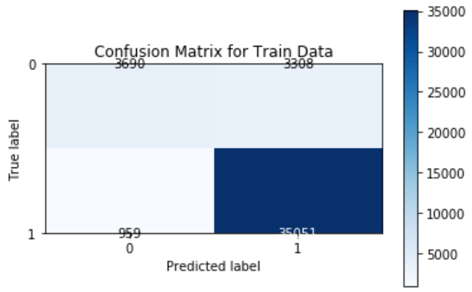
Testing XGBoost Classifier:

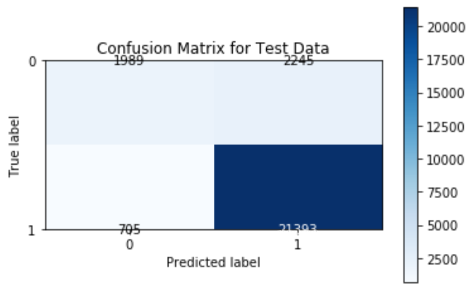
In [67]:
```
testing1(aw2v_X_train_sc,y_tr,aw2v_X_test_sc,y_test,optimal_depth=3,optimal_estimator=300)
```

```
AUC Score for train data 0.9317263217744841
AUC Score for test data 0.9024958837331006
```




```
macro f1 score for train data : 0.7881317533040173
macro f1 score for test data : 0.7548455368515199
micro f1 score for train data: 0.9007859002976191
micro f1 score for test data: 0.887969011089169
hamming loss for train data: 0.09921409970238096
hamming loss for test data: 0.11203098891083092
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.79      0.53      0.63      6998
           1       0.91      0.97      0.94     36010

    accuracy                           0.90     43008
   macro avg       0.85      0.75      0.79     43008
weighted avg       0.89      0.90      0.89     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.74      0.47      0.57      4234
           1       0.91      0.97      0.94     22098

    accuracy                           0.89     26332
   macro avg       0.82      0.72      0.75     26332
weighted avg       0.88      0.89      0.88     26332
```





4. TF_IDF-W2V

In [68]:
```
tf_idf_vect = TfidfVectorizer()
tfidf_train = tf_idf_vect.fit_transform(X_tr)
print("The type of count vectorizer ",type(tfidf_train))
print("The shape of out text TFIDF vectorizer ",tfidf_train.get_shape())
tfidf_cv = tf_idf_vect.transform(X_cv)
tfidf_test = tf_idf_vect.transform(X_test)
print("CV Data Size: ",tfidf_cv.shape)
print("Test Data Size: ",tfidf_test.shape)
```

```
The type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of out text TFIDF vectorizer  (43008, 24467)
CV Data Size:  (18433, 24467)
Test Data Size:  (26332, 24467)
```

In [69]:
```
t = tf_idf_vect.get_feature_names()
tfidf_sent_vectors_train = [] # the tfidf-w2v for each sentence/review is stored in this list
row=0
for sent in tqdm(list_sent_train):
    sent_vec = np.zeros(50)
    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]
            tfidf = tfidf_train[row,t.index(word)]
            sent_vec += (vec * tfidf)
            cnt_words += tfidf
    if cnt_words != 0:
        sent_vec /= cnt_words
    tfidf_sent_vectors_train.append(sent_vec)
    row += 1
print(len(tfidf_sent_vectors_train))
print(len(tfidf_sent_vectors_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 43008/43008 [16:37<00:00, 43.13it/s]

43008
50
```

In [70]:
```
import time
start1 = time.clock()
t = tf_idf_vect.get_feature_names()
tfidf_sent_vectors_CV = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_sent_CV):
    sent_vec = np.zeros(50)
    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]
            tfidf = tfidf_cv[row,t.index(word)]
            sent_vec += (vec * tfidf)
            cnt_words += tfidf
    if cnt_words != 0:
        sent_vec /= cnt_words
    tfidf_sent_vectors_CV.append(sent_vec)
    row += 1
print(len(tfidf_sent_vectors_CV))
print(len(tfidf_sent_vectors_CV[0]))
print((time.clock()-start1)/60)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 18433/18433 [06:29<00:00, 40.64it/s]

18433
50
6.487487379533317
```

In [71]:
```
start2 = time.clock()
t = tf_idf_vect.get_feature_names()
tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_sent_test):
    sent_vec = np.zeros(50)
    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]
            tfidf = tfidf_test[row,t.index(word)]
            sent_vec += (vec * tfidf)
            cnt_words += tfidf
    if cnt_words != 0:
        sent_vec /= cnt_words
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
print(len(tfidf_sent_vectors_test))
print(len(tfidf_sent_vectors_test[0]))
print((time.clock()-start1)/60)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 26332/26332 [10:31<00:00, 55.55it/s]

26332
50
17.020035282533595
```

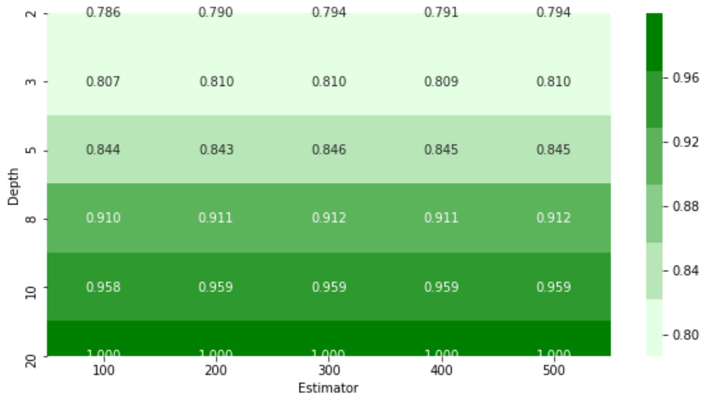
In [72]:
```
train_tfidfw2v = tfidf_sent_vectors_train
cv_tfidfw2v = tfidf_sent_vectors_CV
test_tfidfw2v = tfidf_sent_vectors_test
```

In [73]:
```
#Standardizing data using StandardScaler

sc = StandardScaler(with_mean=False)
tfidfw2v_X_train_sc = sc.fit_transform(train_tfidfw2v)
tfidfw2v_X_cv_sc = sc.transform(cv_tfidfw2v)
tfidfw2v_X_test_sc = sc.transform(test_tfidfw2v)
```

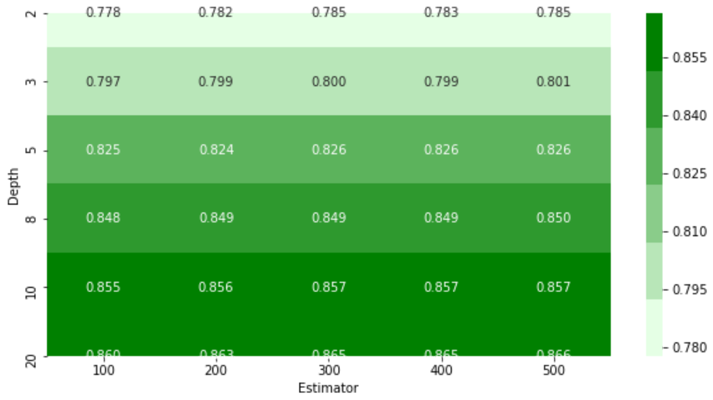
Training RandomForest Classifier

```
In [74]: RF_Classifier(tfidfw2v_X_train_sc,tfidfw2v_X_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



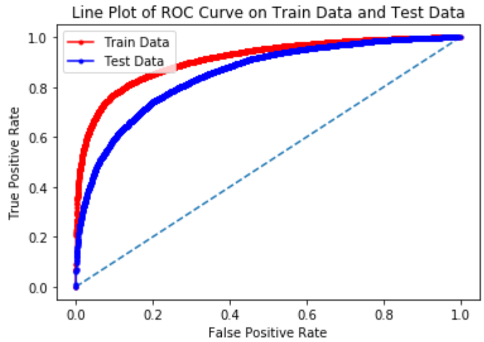

```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



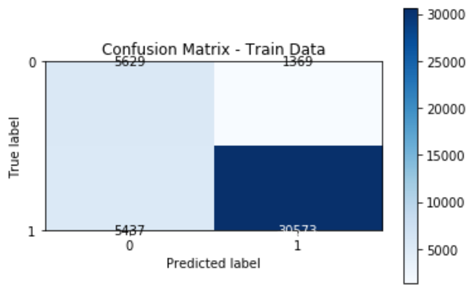
Testing RandomForest Classifier

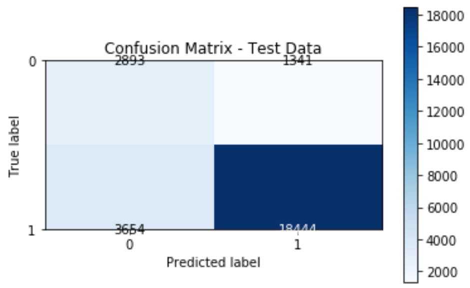
In [75]:
```
testing(tfidfw2v_X_train_sc,y_tr,tfidfw2v_X_test_sc,y_test,optimal_depth=8,optimal_estimator=400)
```

```
AUC Score for train data 0.9117938822366751
AUC Score for test data 0.8518323474514458
```




```
macro f1 score for train data : 0.7615347925477747
macro f1 score for test data : 0.708712055349226
micro f1 score for train data: 0.8417503720238095
micro f1 score for test data: 0.8103068509797965
hamming loss for train data: 0.15824962797619047
hamming loss for test data: 0.18969314902020357
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.51      0.80      0.62      6998
           1       0.96      0.85      0.90     36010

    accuracy                           0.84     43008
   macro avg       0.73      0.83      0.76     43008
weighted avg       0.88      0.84      0.85     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.44      0.68      0.54      4234
           1       0.93      0.83      0.88     22098

    accuracy                           0.81     26332
   macro avg       0.69      0.76      0.71     26332
weighted avg       0.85      0.81      0.83     26332
```

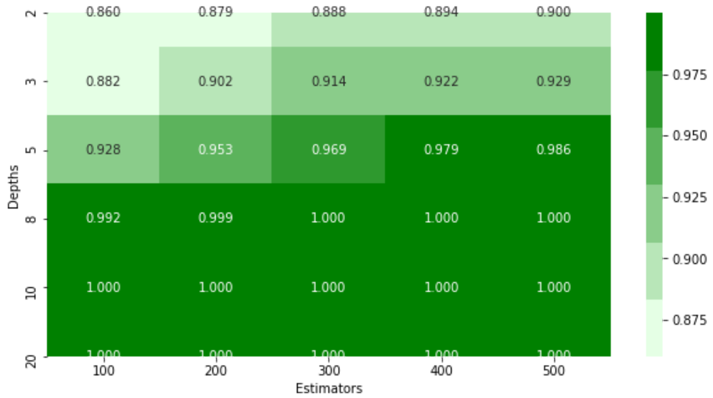




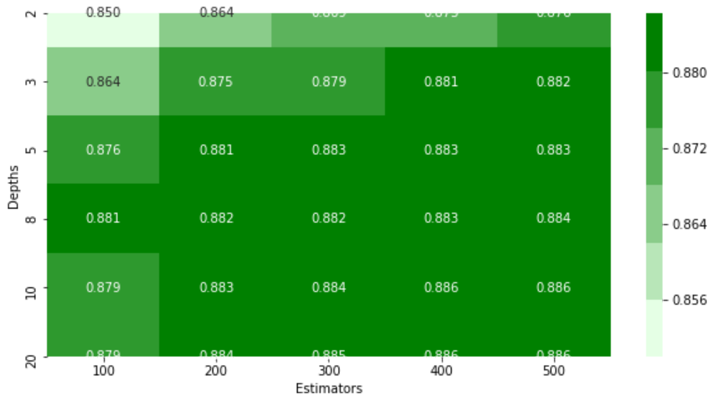
Training XGBoost classifier

In [76]:
```
xgb_Classifier(tfidfw2v_X_train_sc,tfidfw2v_X_cv_sc,y_tr,y_cv)
```

```
--------------------------------------- AUC Score for train data ---------------------------------------
```



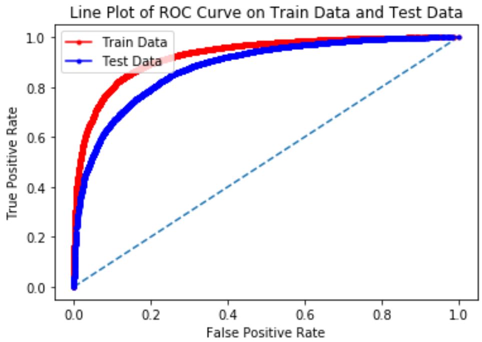

```
--------------------------------------- AUC Score for CV data ---------------------------------------
```



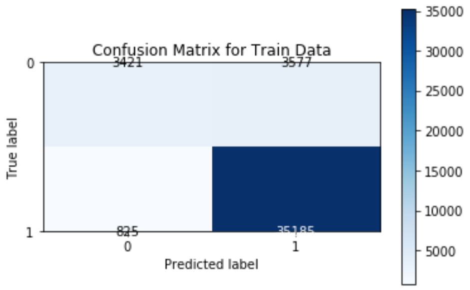
Testing XGBoost Classifier:

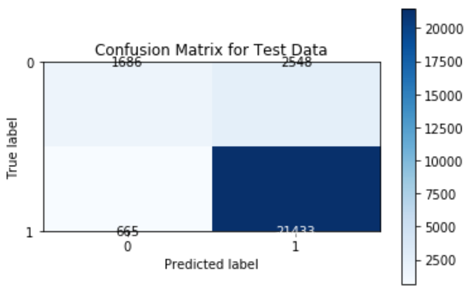
In [77]:
```
testing1(tfidfw2v_X_train_sc,y_tr,tfidfw2v_X_test_sc,y_test,optimal_depth=3,optimal_estimator=500)
```

```
AUC Score for train data 0.9293100305804038
AUC Score for test data 0.8820471765463698
```




```
macro f1 score for train data : 0.7748150036016999
macro f1 score for test data : 0.7211724086212047
micro f1 score for train data: 0.8976469494047619
micro f1 score for test data: 0.8779811636032204
hamming loss for train data: 0.1023530505952381
hamming loss for test data: 0.12201883639677959
Precision recall report for train data:
              precision    recall  f1-score   support

           0       0.81      0.49      0.61      6998
           1       0.91      0.98      0.94     36010

    accuracy                           0.90     43008
   macro avg       0.86      0.73      0.77     43008
weighted avg       0.89      0.90      0.89     43008

Precision recall report for test data:
              precision    recall  f1-score   support

           0       0.72      0.40      0.51      4234
           1       0.89      0.97      0.93     22098

    accuracy                           0.88     26332
   macro avg       0.81      0.68      0.72     26332
weighted avg       0.87      0.88      0.86     26332
```





In [78]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Model","Best Hyper Parameter(Depth)","Best Hyper parameter(n_estimator)","Test Auc Score"]
x.add_row(["BoW","Random Forest",5,400,90.4])
x.add_row(["Tf-Idf","Random Forest",5,400,81.01])
x.add_row(["Avg-W2V","Random Forest",8,300,87.92])
x.add_row(["TfIdf-W2V","Random Forest",8,400,85.18])
x.add_row(["BoW","XGBoost",3,400,91.8])
x.add_row(["Tf-Idf","XGBoost",5,300,92.86])
x.add_row(["Avg-W2V","XGBoost",3,300,90.24])
x.add_row(["TfIdf-W2V","XGBoost",3,500,88.2])
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))
printmd('****Final Conclusion:****')
print(x)
```

Final Conclusion:

```
+------------+---------------+-----------------------------+-----------------------------------+----------------+
| Vectorizer |     Model     | Best Hyper Parameter(Depth) | Best Hyper parameter(n_estimator) | Test Auc Score |
+------------+---------------+-----------------------------+-----------------------------------+----------------+
|    BoW     | Random Forest |              5              |                400                |      90.4      |
|   Tf-Idf   | Random Forest |              5              |                400                |     81.01      |
|  Avg-W2V   | Random Forest |              8              |                300                |     87.92      |
| TfIdf-W2V  | Random Forest |              8              |                400                |     85.18      |
|    BoW     |    XGBoost    |              3              |                400                |      91.8      |
|   Tf-Idf   |    XGBoost    |              5              |                300                |     92.86      |
|  Avg-W2V   |    XGBoost    |              3              |                300                |     90.24      |
| TfIdf-W2V  |    XGBoost    |              3              |                500                |      88.2      |
+------------+---------------+-----------------------------+-----------------------------------+----------------+
```