Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

[Q] How to determine if a review is positive or negative?

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

[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 [0]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os
```

In [2]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [3]:

```
# using SQLite Table to read data.
con = sqlite3.connect('/content/drive/My Drive/Colab Notebooks/databas
e.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 50
0000 data points
# you can change the number to any other number based on your computing
 power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 5000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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-verse
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (5000, 10)
```

Out[3]:

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin

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

In [0]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [0]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[0]:

UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

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

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

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

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

Out[9]: (4986, 10)

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

Out[10]: 99.72

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

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

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

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

```
(4986, 10)
```

Out[13]:
```
1    4178
0     808
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.

3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in
 the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
```

```
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [16]:

```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
```

```
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 4986/4986 [00:01<00:00, 2574.38it/s]
```

In [17]:
```
preprocessed_reviews[1500]
```

Out[17]:
```
'wow far two two star reviews one obviously no idea ordering wants cris
py cookies hey sorry reviews nobody good beyond reminding us look order
ing chocolate oatmeal cookies not like combination not order type cooki
e find combo quite nice really oatmeal sort calms rich chocolate flavor
gives cookie sort coconut type consistency let also remember tastes dif
fer given opinion soft chewy cookies advertised not crispy cookies blur
b would say crispy rather chewy happen like raw cookie dough however no
t see taste like raw cookie dough soft however confusion yes stick toge
ther soft cookies tend not individually wrapped would add cost oh yeah
chocolate chip cookies tend somewhat sweet want something hard crisp su
ggest nabiso ginger snaps want cookie soft chewy tastes like combinatio
n chocolate oatmeal give try place second order'
```

[4] Featurization

In [18]:
```
#here preprocessed_review is my X and final['Score'] is my Y
print(len(preprocessed_reviews))
print(len(final['Score']))
X=preprocessed_reviews
Y=final['Score']
#if both are of same lenght then proceed....
```

```
4986
4986
```

In [0]:
```
#here i am performing splittig operation as train test and cv...
from sklearn.model_selection import train_test_split
```

```
# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

[4.1] BAG OF WORDS

In [20]:

```
#BoW
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=None)
vectorizer.fit(X_train) # fitting on train data ,we cant perform fit on
 test or cv

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
#you can also check X_train_bow is of sparse matrix type or not
#below is code for that
print(type(X_train_bow))
#displaying number of unique words in each of splitted dataset
print("the number of unique words in train: ", X_train_bow.get_shape()[
1])
print("the number of unique words in cv: ", X_cv_bow.get_shape()[1])
print("the number of unique words in test: ", X_test_bow.get_shape()[1
])
```

```
After vectorizations
(2237, 1263) (2237,)
(1103, 1263) (1103,)
```

```
(1646, 1263) (1646,)
========================================================================
==============================
<class 'scipy.sparse.csr.csr_matrix'>
the number of unique words in train:  1263
the number of unique words in cv:  1263
the number of unique words in test:  1263
```

[4.3] TF-IDF

In [21]:

```
#below code for converting to tfidf
#i refered sample solution to write this code
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(X_train)
X_test_tf_idf = tf_idf_vect.transform(X_test)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(X_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_shap
e())
print("the number of unique words including both unigrams and bigrams "
, X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['able', 'absolute',
'absolutely', 'absolutely delicious', 'absolutely love', 'acid', 'acros
s', 'actual', 'actually', 'add']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2237, 1523)
the number of unique words including both unigrams and bigrams  1523
```

[4.4] Word2Vec

```
In [22]: #in average w2v the output is of list form and here we write same code
          of all train ,test and cv
         #this code is for train data:
         # Train your own Word2Vec model using your own text corpus
         i=0
         list_of_sentance_train=[]
         for sentance in X_train:
             list_of_sentance_train.append(sentance.split())


         #training word2vect model
         from gensim.models import Word2Vec
         from gensim.models import KeyedVectors
         # this line of code trains your w2v model on the give list of sentances
         w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
         4)
         w2v_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(w2v_words))
         print("sample words ", w2v_words[0:50])

         #this is the actuall code to convert word2vect to avg w2v:
         from tqdm import tqdm
         import numpy as np
         # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors_train = []; # the avg-w2v for each sentence/review is stor
         ed in this list
         for sent in tqdm(list_of_sentance_train): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
         u might need to change this to 300 if you use google's w2v
             cnt_words =0; # num of words with a valid vector in the sentence/re
         view
             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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
  8%|█         | 186/2237 [00:00<00:01, 1843.91it/s]
```

```
number of words that occured minimum 5 times  2369
sample words  ['tomatoes', 'perfect', 'italian', 'recipes', 'full', 'fl
avor', 'not', 'always', 'best', 'choice', 'cooking', 'used', 'product',
'years', 'first', 'time', 'amazon', 'glad', 'far', 'pancake', 'waffle',
'mix', 'market', 'bar', 'none', 'drinking', 'green', 'tea', 'purchasin
g', 'going', 'every', 'summer', 'year', 'stopped', 'know', 'buy', 'purc
hased', 'pleased', 'could', 'get', 'happy', 'flaxseed', 'great', 'way',
'fiber', 'diet', 'brand', 'mild', 'taste', 'put']
```

```
100%|██████████| 2237/2237 [00:01<00:00, 1598.25it/s]
```

```
(2237, 50)
[ 0.5815557  -0.1249199  -0.22773263  0.14877247  0.19424616 -0.5831709
2
  0.15959836 -0.19497902  0.03360663 -0.07828729 -0.36949081 -0.3113993
4
  0.87857483 -0.19647899  0.52671479 -0.06469363 -0.27101788  0.1192673
1
  0.44854099 -0.03302139 -0.18821708 -0.38120448  0.3814335   0.0579020
7
 -0.02593781 -0.19189688  0.54454326 -0.26339874  0.49361477  0.2693361
5
  0.33985613  0.30827539 -0.25443215 -0.30647906  0.00842176 -0.5061643
5
 -0.999982    0.23799093  0.22519295 -0.50063275 -0.15111828  0.1475994
2
 -0.12880595  0.03188713 -0.53452821 -0.09460729 -0.06102168  0.4237058
8
  0.27149406 -0.16063957]
```

In [23]:
```
#this code is for test data:
```

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


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
#i made below two statement as comment to avoid data leakage problem
#w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=
4)
#w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is store
d in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
```

```
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
  9%|█          | 144/1646 [00:00<00:01, 1438.14it/s]
```

```
number of words that occured minimum 5 times  2369
sample words  ['tomatoes', 'perfect', 'italian', 'recipes', 'full', 'fl
avor', 'not', 'always', 'best', 'choice', 'cooking', 'used', 'product',
'years', 'first', 'time', 'amazon', 'glad', 'far', 'pancake', 'waffle',
'mix', 'market', 'bar', 'none', 'drinking', 'green', 'tea', 'purchasin
g', 'going', 'every', 'summer', 'year', 'stopped', 'know', 'buy', 'purc
hased', 'pleased', 'could', 'get', 'happy', 'flaxseed', 'great', 'way',
'fiber', 'diet', 'brand', 'mild', 'taste', 'put']
```

```
100%|██████████| 1646/1646 [00:01<00:00, 1642.63it/s]
```

```
(1646, 50)
[ 0.59260428 -0.13274156 -0.23257391  0.15118477  0.20004071 -0.5902876
5
  0.16233429 -0.20037775  0.0343786  -0.08306612 -0.37788957 -0.3165607
6
  0.89658557 -0.20245428  0.53625456 -0.06305512 -0.27775117  0.1207708
5
  0.45624327 -0.03677419 -0.19391277 -0.3896531   0.39214232  0.0603857
2
 -0.02970573 -0.19105635  0.55451854 -0.27247281  0.50782258  0.2727591
2
  0.34249781  0.31733466 -0.26386368 -0.31504892  0.01273812 -0.5145355
1
 -1.01841334  0.23925593  0.22856011 -0.51042768 -0.14972638  0.1535070
7
 -0.12715844  0.03234488 -0.54349443 -0.09246928 -0.06204891  0.4350311
7
  0.27914118 -0.16160081]
```

In [24]:

```
#this code is for cv data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_cv=[]
```

```
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
#w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
#w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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_cv.append(sent_vec)
sent_vectors_cv= np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
  7%|▋         | 78/1103 [00:00<00:01, 771.44it/s]
```

```
number of words that occured minimum 5 times  2369
```

```
sample words  ['tomatoes', 'perfect', 'italian', 'recipes', 'full', 'fl
avor', 'not', 'always', 'best', 'choice', 'cooking', 'used', 'product',

'years', 'first', 'time', 'amazon', 'glad', 'far', 'pancake', 'waffle',
'mix', 'market', 'bar', 'none', 'drinking', 'green', 'tea', 'purchasin
g', 'going', 'every', 'summer', 'year', 'stopped', 'know', 'buy', 'purc
hased', 'pleased', 'could', 'get', 'happy', 'flaxseed', 'great', 'way',
'fiber', 'diet', 'brand', 'mild', 'taste', 'put']
```

```
100%|██████████| 1103/1103 [00:00<00:00, 1435.79it/s]
```

```
(1103, 50)
[ 0.65805155 -0.14557359 -0.26066039  0.1698221   0.22372781 -0.6561089
1
  0.1794716  -0.22089531  0.03701629 -0.09098244 -0.42037371 -0.3509628
4
  0.99718934 -0.22482419  0.59538869 -0.07210336 -0.31124395  0.1361086
2
  0.5085497  -0.04234302 -0.21436395 -0.43215606  0.43433652  0.0672013
5
 -0.03062168 -0.21221082  0.61597491 -0.30563171  0.5671878   0.3019141
8
  0.38142598  0.34938096 -0.29319007 -0.35130721  0.01021815 -0.5759048
7
 -1.12801419  0.26547112  0.2569954  -0.57131486 -0.16755697  0.1722751
2
 -0.13990711  0.03749315 -0.60333667 -0.10620004 -0.0668366   0.4838974
1
  0.30992998 -0.17686341]
```

[4.4.1]TFIDF weighted vector

In [25]:

```python
#this is for train data
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
```

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

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

```
print(tfidf_sent_vectors_train.shape)
print(tfidf_sent_vectors_train[0])
```

```
100%|██████████| 2237/2237 [00:09<00:00, 244.49it/s]
```

```
(2237, 50)
[ 0.5815557  -0.1249199  -0.22773263  0.14877247  0.19424616 -0.5831709
2
  0.15959836 -0.19497902  0.03360663 -0.07828729 -0.36949081 -0.3113993
4
  0.87857483 -0.19647899  0.52671479 -0.06469363 -0.27101788  0.1192673
1
  0.44854099 -0.03302139 -0.18821708 -0.38120448  0.3814335   0.0579020
7
 -0.02593781 -0.19189688  0.54454326 -0.26339874  0.49361477  0.2693361
5
  0.33985613  0.30827539 -0.25443215 -0.30647906  0.00842176 -0.5061643
5
 -0.999982    0.23799093  0.22519295 -0.50063275 -0.15111828  0.1475994
2
 -0.12880595  0.03188713 -0.53452821 -0.09460729 -0.06102168  0.4237058
8
  0.27149406 -0.16063957]
```

In [26]:

```
#this is for test data
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
#model = TfidfVectorizer()
tf_idf_matrix = model.transform(X_test)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```python
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

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

```
100%|██████████| 1646/1646 [00:06<00:00, 253.27it/s]
```

```
(1646, 50)
[ 0.59260428 -0.13274156 -0.23257391  0.15118477  0.20004071 -0.5902876
5
  0.16233429 -0.20037775  0.0343786  -0.08306612 -0.37788957 -0.3165607
6
  0.89658557 -0.20245428  0.53625456 -0.06305512 -0.27775117  0.1207708
5
```

```
5
  0.45624327 -0.03677419 -0.19391277 -0.3896531   0.39214232  0.0603857
2

 -0.02970573 -0.19105635  0.55451854 -0.27247281  0.50782258  0.2727591
2
  0.34249781  0.31733466 -0.26386368 -0.31504892  0.01273812 -0.5145355
1
 -1.01841334  0.23925593  0.22856011 -0.51042768 -0.14972638  0.1535070
7
 -0.12715844  0.03234488 -0.54349443 -0.09246928 -0.06204891  0.4350311
7
  0.27914118 -0.16160081]
```

In [27]:

```
#this is for cv data
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())


# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
#model = TfidfVectorizer()
tf_idf_matrix = model.transform(X_cv)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
```

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

```
100%|██████████| 1103/1103 [00:04<00:00, 231.73it/s]
```

```
(1103, 50)
[ 0.65805155 -0.14557359 -0.26066039  0.1698221   0.22372781 -0.6561089
1
  0.1794716  -0.22089531  0.03701629 -0.09098244 -0.42037371 -0.3509628
4
  0.99718934 -0.22482419  0.59538869 -0.07210336 -0.31124395  0.1361086
2
  0.5085497  -0.04234302 -0.21436395 -0.43215606  0.43433652  0.0672013
5
 -0.03062168 -0.21221082  0.61597491 -0.30563171  0.5671878   0.3019141
8
  0.38142598  0.34938096 -0.29319007 -0.35130721  0.01021815 -0.5759048
7
 -1.12801419  0.26547112  0.2569954  -0.57131486 -0.16755697  0.1722751
2
 -0.13990711  0.03749315 -0.60333667 -0.10620004 -0.0668366   0.4838974
1
  0.30992998  0.17686341]
```

```
0.30992998 -0.17686341]
```

[5] Assignment 9: Random Forests

1. **Apply Random Forests & GBDT on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)**

    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

    - Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.

4. **Feature engineering**

    - To increase the performance of your model, you can also experiment with with feature engineering like :
        - Taking length of reviews as another feature.
        - Considering some features from review summary as well.

5. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

(or)

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
  - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

6. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

4. For more details please go through this link.

[5.1] Applying RF

In [0]:
```
#this is the function for Random forest classifiere
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
def Random_Forest_Classifier(xtrain,xcv,ytrain,ycv):
    predicted_cv = []
    predicted_train = []
    d = [2,4,7,10,20]#depth
    e = [100, 200, 300, 400, 500] #estimator
    for i in d:
        for j in e:
            clf = RandomForestClassifier(n_estimators=j, max_depth=i, n
_jobs = -1, class_weight='balanced')
            clf.fit(xtrain,ytrain)
            prob_cv = clf.predict_proba(xcv)
            prob_train = clf.predict_proba(xtrain)
            prob_cv = prob_cv[:,1]
            prob_train = prob_train[:,1]
            auc_score_cv = roc_auc_score(ycv,prob_cv)
            auc_score_train = roc_auc_score(ytrain,prob_train)
            predicted_cv.append(auc_score_cv)
            predicted_train.append(auc_score_train)
    cmap=sns.light_palette("yellow")
    #Heat map:
    print("for training data:")
    predicted_train = np.array(predicted_train)
    predicted_train = predicted_train.reshape(len(d),len(e))
    plt.figure(figsize=(8,4))
    cmap=sns.light_palette("yellow")
    sns.heatmap(predicted_train,annot=True, cmap=cmap, fmt=".3f", xtick
labels=e,yticklabels=d)
    plt.xlabel('Estimators')
    plt.ylabel('Depths')
    plt.show()
```

```
    print("for cv data:")
    predicted_cv = np.array(predicted_cv)
    predicted_cv = predicted_cv.reshape(len(d),len(e))
    plt.figure(figsize=(8,4))
    sns.heatmap(predicted_cv, annot=True, cmap=cmap, fmt=".3f", xtickla
bels=e, yticklabels=d)
    plt.xlabel('Estimators')
    plt.ylabel('Depths')
    plt.show()
```

In [78]:

```
#installing scikitplot library
pip install scikit-plot
```

```
Requirement already satisfied: scikit-plot in /usr/local/lib/python3.6/
dist-packages (0.3.7)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/pyt
hon3.6/dist-packages (from scikit-plot) (0.21.3)
Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/pyth
on3.6/dist-packages (from scikit-plot) (3.0.3)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.6/d
ist-packages (from scikit-plot) (1.3.0)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.
6/dist-packages (from scikit-plot) (0.13.2)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.
6/dist-packages (from scikit-learn>=0.18->scikit-plot) (1.16.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/pyth
on3.6/dist-packages (from matplotlib>=1.4.0->scikit-plot) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.
6/dist-packages (from matplotlib>=1.4.0->scikit-plot) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/p
ython3.6/dist-packages (from matplotlib>=1.4.0->scikit-plot) (2.5.3)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.0->scik
it-plot) (2.4.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/d
ist-packages (from kiwisolver>=1.0.1->matplotlib>=1.4.0->scikit-plot)
(41.0.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-pac
kages (from cycler>=0.10->matplotlib>=1.4.0->scikit-plot) (1.12.0)
```

In [0]:

```
#testing random forest function:
import scikitplot.metrics as skplt
def Random_forest_Test(xtrain,ytrain,xtest,ytest,optimal_depth,optimal_
estimator):
    clf = RandomForestClassifier(n_estimators = optimal_estimator, max_
depth = optimal_depth,class_weight='balanced')
    clf.fit(xtrain,ytrain)
    prob_test= clf.predict_proba(xtest)
    prob_train= clf.predict_proba(xtrain)
    prob_test = prob_test[:, 1]
    prob_train = prob_train[:,1]
    print("printing auc score for train data:",roc_auc_score(ytrain,pro
b_train))
    print("printing auc score for test data:",roc_auc_score(ytest,prob_
test))
    # code to calculate roc curve:
    fpr_train, tpr_train, thresholds = roc_curve(ytrain,prob_train)
    fpr_test, tpr_test, thresholds = roc_curve(ytest,prob_test)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr_test, tpr_test, marker='.',color ='b',label='Test Dat
a')
    plt.plot(fpr_train, tpr_train, marker='.',color= 'r',label='Train D
ata')
    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()
    print("Train confusion matrix")
    ax= plt.subplot()
    arr1=confusion_matrix(ytrain, clf.predict(xtrain))
    df_1= pd.DataFrame(arr1, range(2),range(2))
    plt.figure(figsize = (5,2))
    sns.heatmap(df_1, annot=True,fmt="d",ax=ax)
    ax.set_title('Confusion Matrix');
    ax.set_xlabel('Actual Labels')
    ax.set_ylabel('Predicted Labels')
```

```
    ax.xaxis.set_ticklabels(['False', 'True']);
    ax.yaxis.set_ticklabels(['True', 'False']);
    print("Test confusion matrix")
    ax= plt.subplot()
    arr1=confusion_matrix(ytest, clf.predict(xtest))
    df_1= pd.DataFrame(arr1, range(2),range(2))
    plt.figure(figsize = (5,2))
    sns.heatmap(df_1, annot=True,fmt="d",ax=ax)
    ax.set_title('Confusion Matrix');
    ax.set_xlabel('Actual Labels')
    ax.set_ylabel('Predicted Labels')
    ax.xaxis.set_ticklabels(['False', 'True']);
    ax.yaxis.set_ticklabels(['True', 'False']);
```

In [0]:

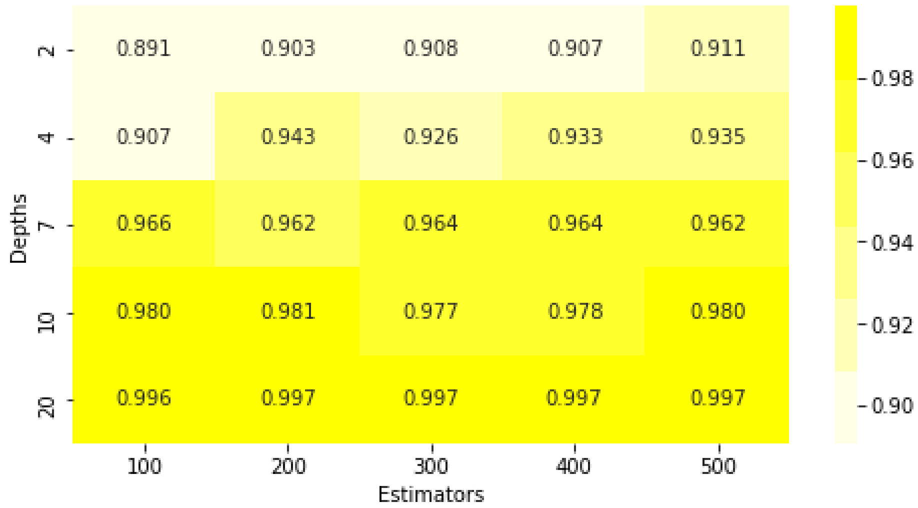
```
#below reference and code for feature extraction and cloud word represe
ntation
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get
-most-informative-features-for-scikit-learn-classifiers
#Code Reference:https://stackoverflow.com/questions/45588724/generating
-word-cloud-for-items-in-a-list-in-python
from wordcloud import WordCloud
def imp_feature(vectorizer,classifier, n =20):
    features = []
    feature_names = vectorizer.get_feature_names()
    coefficient = sorted(zip(classifier.feature_importances_, feature_n
ames))
    top = coefficient[:-(n + 1):-1]
    print('\033[1m' + "feature_importances\timportant features" + '\033
[0m')
    print("="*50)
    for (coef1, feat1) in top:
        print("%.4f\t\t\t%-15s" % (coef1, feat1))
        features.append(feat1)
    wordcloud = WordCloud(background_color='black',width=1600,height=80
0).generate(" ".join(features))    #top 20 features in word cloud
    fig = plt.figure(figsize=(15,10))
    plt.imshow(wordcloud)
    plt.axis('off')
```

```
    plt.tight_layout(pad=0)
    plt.show()
```

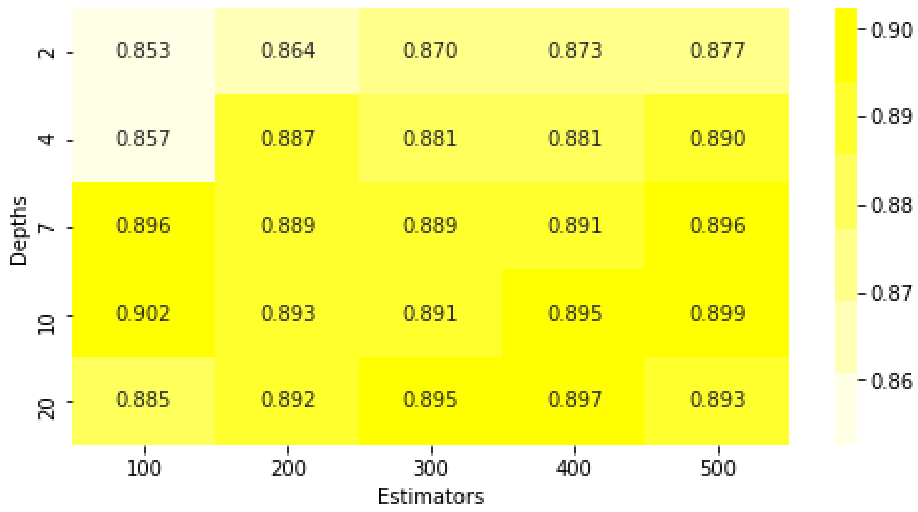
[5.1.1] Applying Random Forests on BOW, SET 1

In [38]:
```
#Hyperparameret tuning :to find optimal hyper parameters
Random_Forest_Classifier(X_train_bow,X_cv_bow,y_train,y_cv)
```

```
for training data:
```



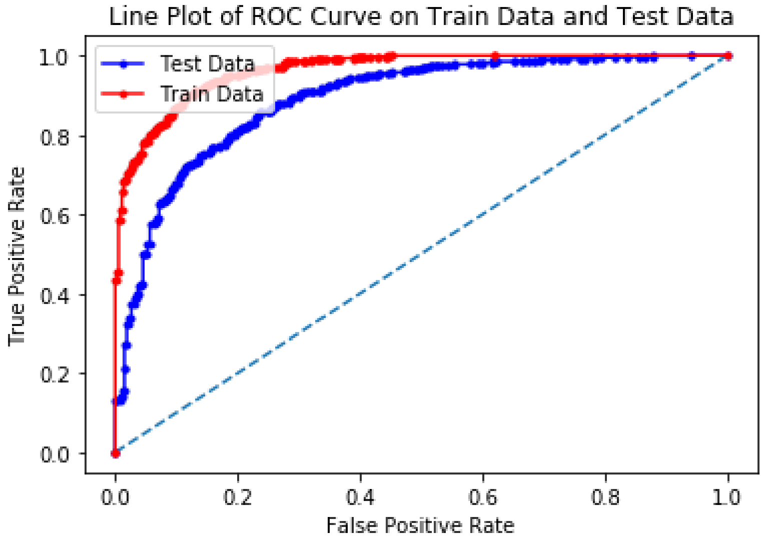

```
for cv data:
```



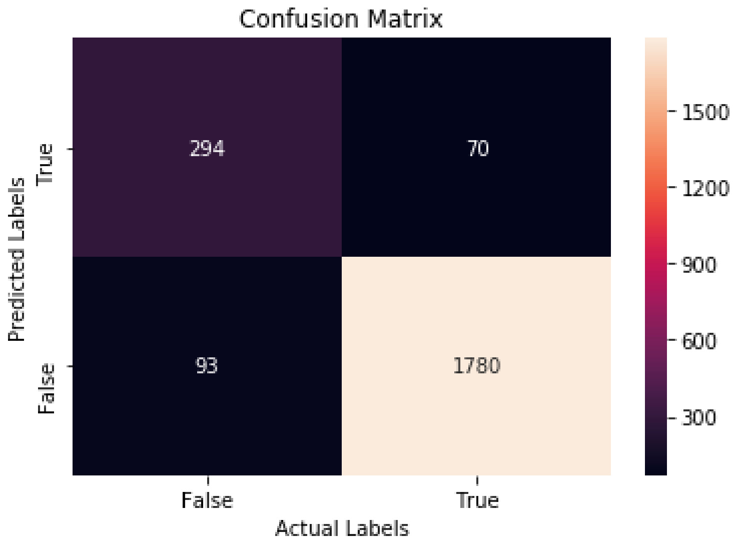
In [39]:

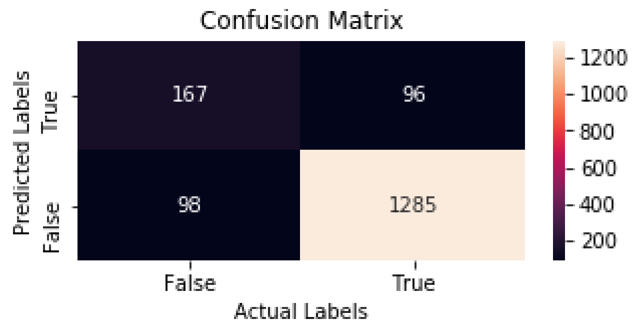
```
#Testing random classifier on bow data
Random_forest_Test(X_train_bow,y_train,X_test_bow,y_test,optimal_depth=
7,optimal_estimator=400)
```

```
printing auc score for train data: 0.9632897801611096
printing auc score for test data: 0.8902685790794795
```




```
Train confusion matrix
Test confusion matrix
```





<Figure size 360x144 with 0 Axes>

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

In [44]:
```
#print top 20 features Random forest classifier
clf = RandomForestClassifier(max_depth =10, n_estimators = 400,class_we
ight='balanced')
clf.fit(X_train_bow,y_train)
features = imp_feature(vectorizer,clf)
```

```
feature_importances      important features
==================================================
0.0450                   not
0.0407                   great
0.0203                   disappointed
0.0189                   love
0.0163                   delicious
0.0157                   perfect
0.0127                   excellent
0.0125                   awful
0.0124                   would
0.0116                   terrible
0.0115                   return
0.0109                   away
0.0108                   product
```

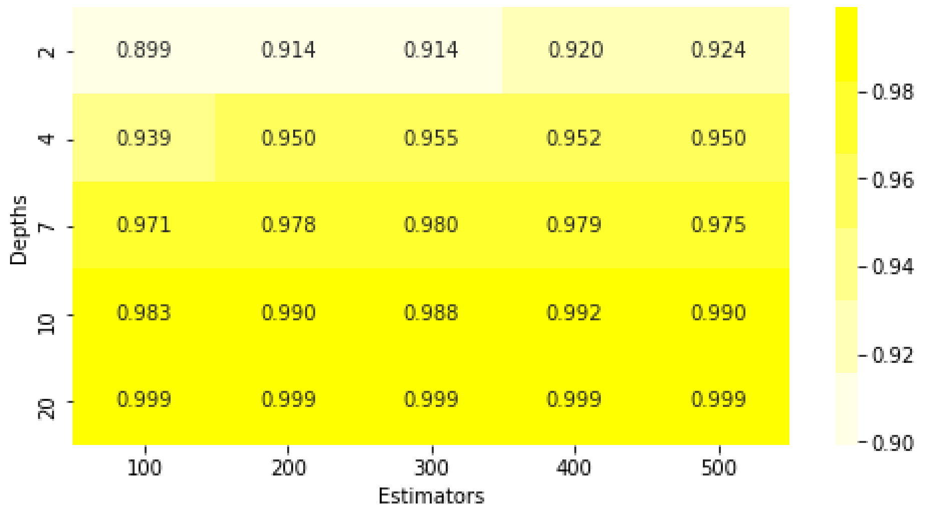
```
0.0103                  highly
0.0102                  received
0.0098                  waste
0.0090                  loves
0.0087                  money
0.0081                  nice
0.0081                  best
```

[5.1.3] Applying Random Forests on TFIDF, SET 2

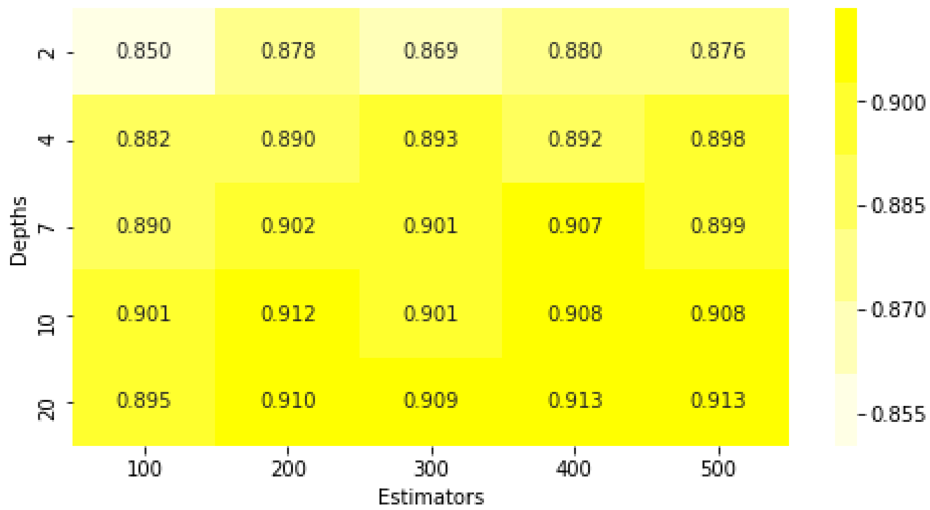
In [45]:

```
#Hyperparameret tuning :to find optimal hyper parameters
Random_Forest_Classifier(X_train_tf_idf,X_cv_tf_idf,y_train,y_cv)
```

```
for training data:
```



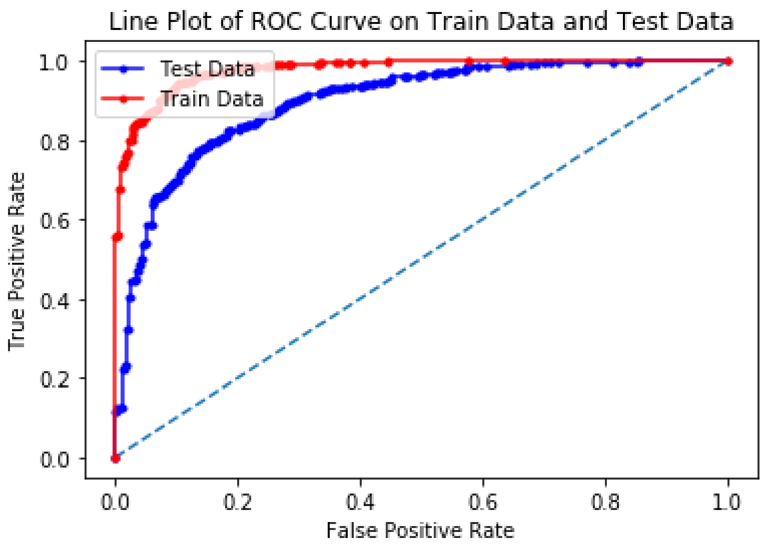
for cv data:



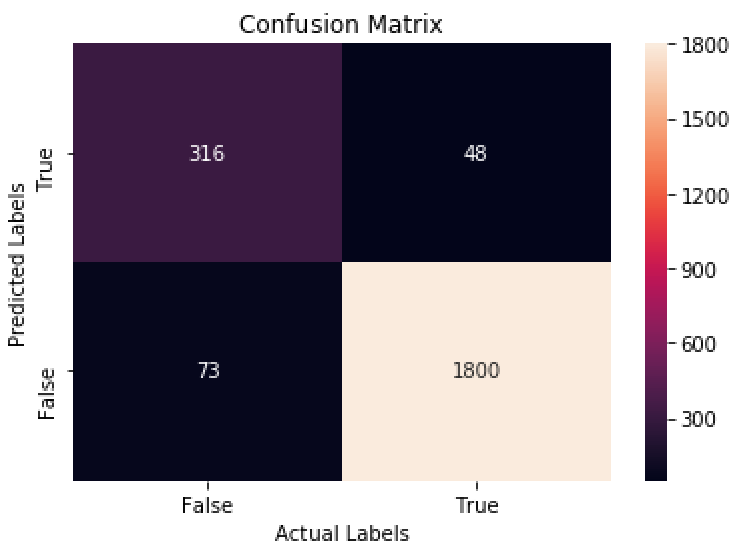
In [46]:
```
#Testing random classifier on tfidf data:
Random_forest_Test(X_train_tf_idf,y_train,X_test_tf_idf,y_test,optimal_
depth=7,optimal_estimator=500)
```

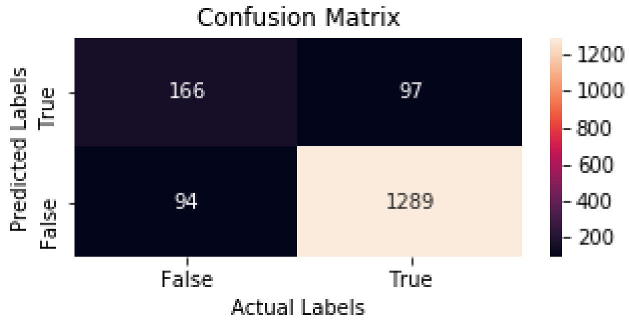
printing auc score for train data: 0.9778268981419008

```
printing auc score for test data: 0.8974016369329914
```




```
Train confusion matrix
Test confusion matrix
```






```
<Figure size 360x144 with 0 Axes>
```

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

In [47]:

```
#print top 20 features Random forest classifier
clf = RandomForestClassifier(max_depth =7, n_estimators = 500,class_wei
ght='balanced')
clf.fit(X_train_tf_idf,y_train)
features = imp_feature(tf_idf_vect,clf)
```

```
feature_importances        important features
==================================================
0.0535                     great
0.0480                     not
0.0265                     love
0.0209                     excellent
0.0189                     delicious
0.0164                     disappointed
0.0162                     perfect
0.0152                     awful
0.0146                     nice
0.0144                     best
0.0140                     always
0.0126                     good
0.0124                     would
0.0120                     not buy
0.0116                     favorite
0.0115                     terrible
```

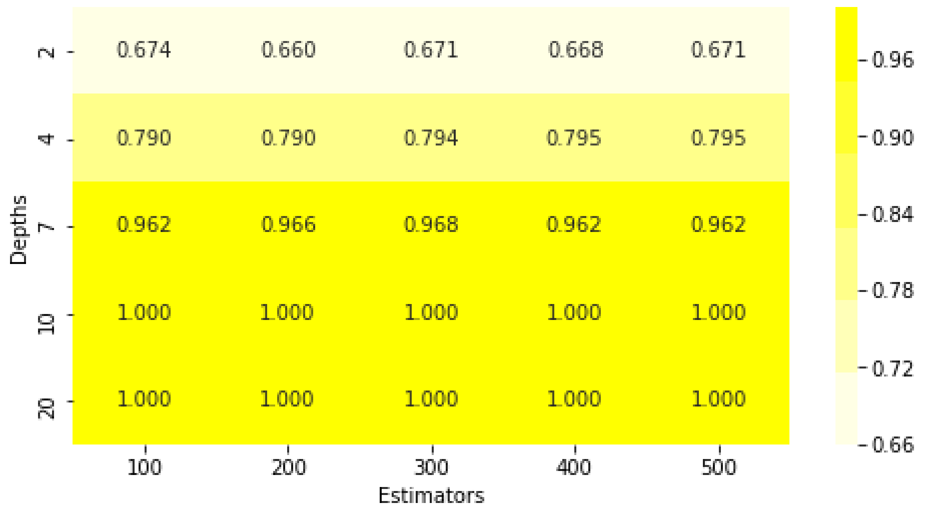
```
0.0115                  terrible
0.0115                  highly
0.0110                  received
0.0108                  away
0.0103                  return
```

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

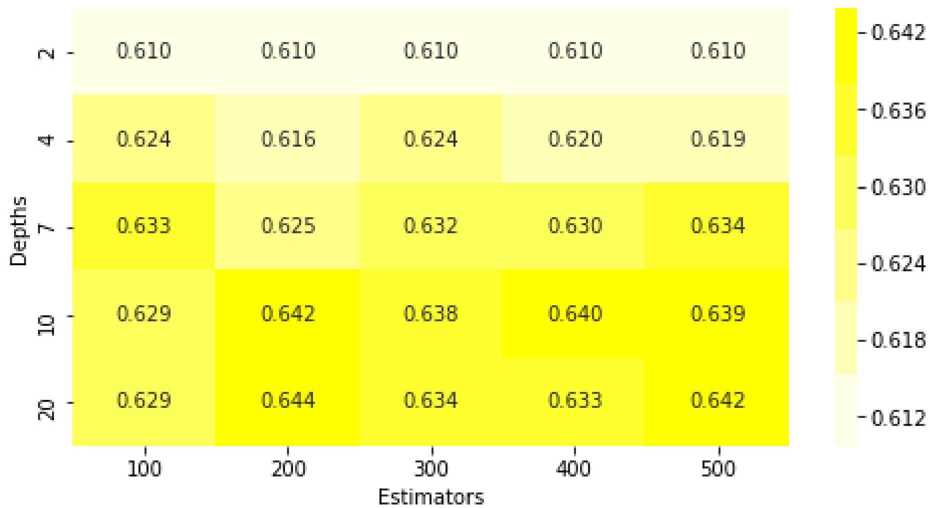
In [48]:

```
#Hyperparameret tuning :to find optimal hyper parameters
Random_Forest_Classifier(sent_vectors_train,sent_vectors_cv,y_train,y_cv)
```

```
for training data:
```

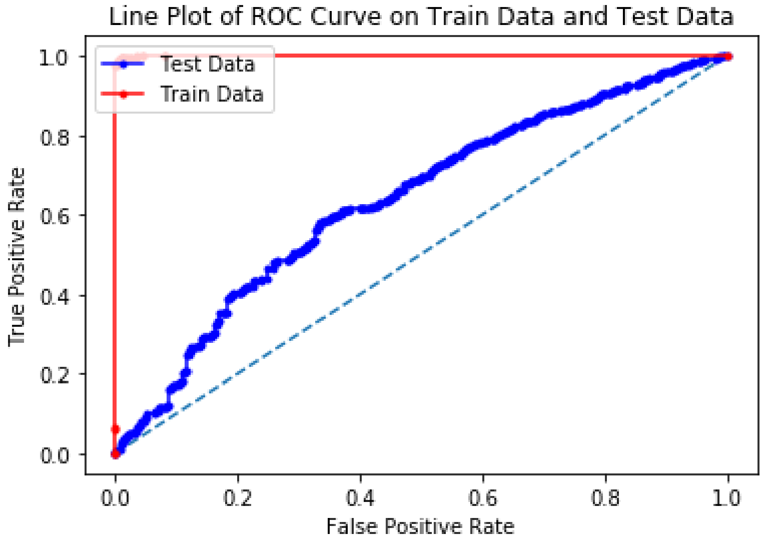


for cv data:

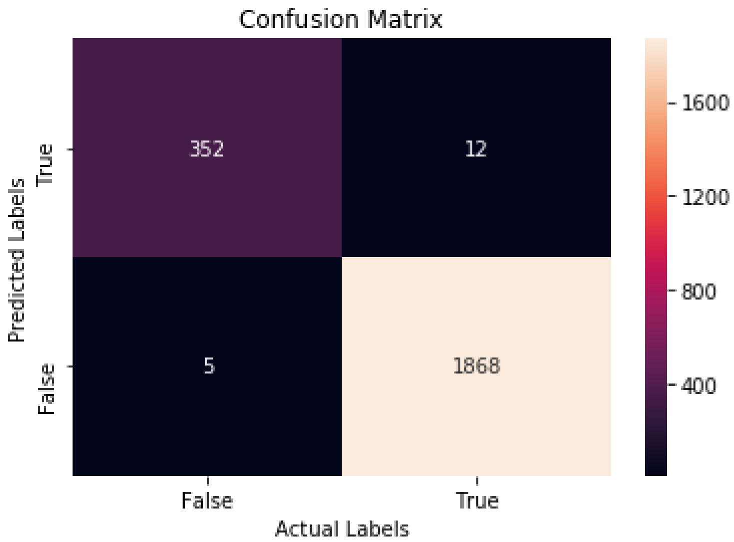


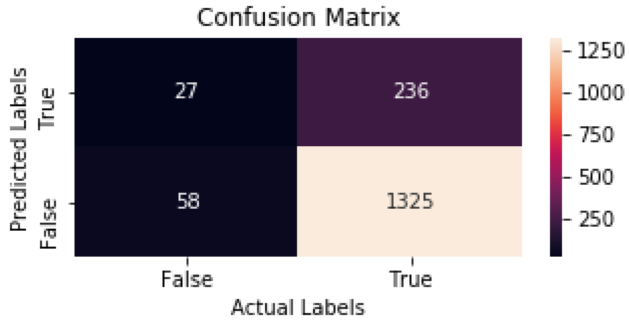
In [49]:
```
#Testing random classifier on w2v
Random_forest_Test(sent_vectors_train,y_train,sent_vectors_test,y_test,
optimal_depth=10,optimal_estimator=500)
```

```
printing auc score for train data: 0.9997535833093762
printing auc score for test data: 0.6403861116380601
```




```
Train confusion matrix
Test confusion matrix
```



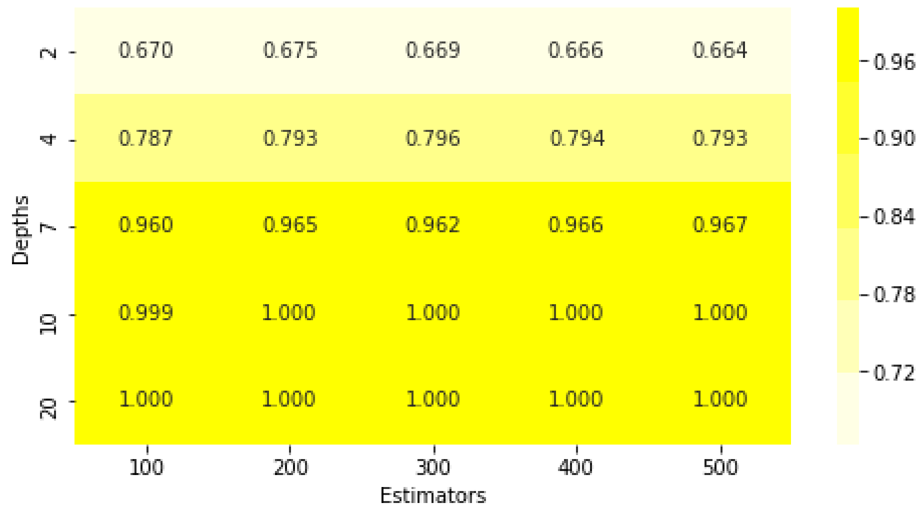


<Figure size 360x144 with 0 Axes>

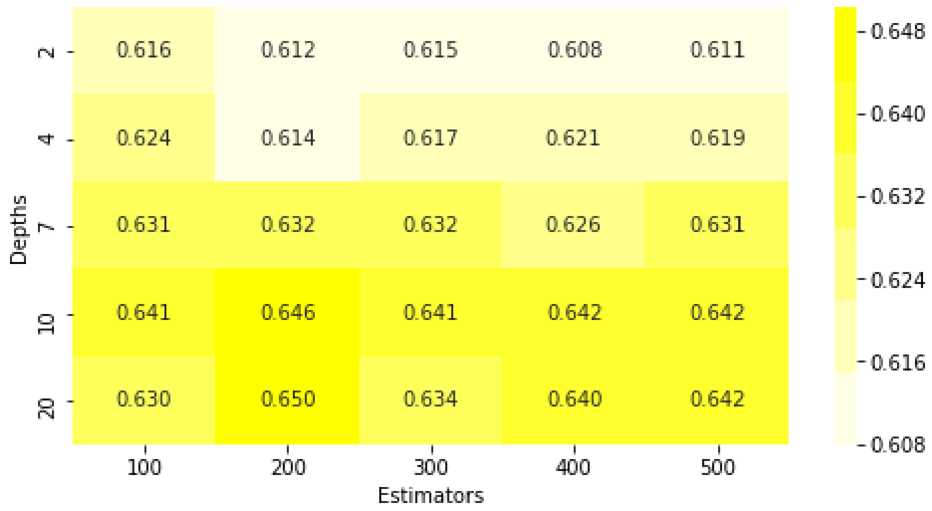
[5.1.6] Applying Random Forests on TFIDF W2V, SET 4

In [50]:
```
#Hyperparameret tuning :to find optimal hyper parameters
Random_Forest_Classifier(tfidf_sent_vectors_train,tfidf_sent_vectors_cv
,y_train,y_cv)
```

for training data:

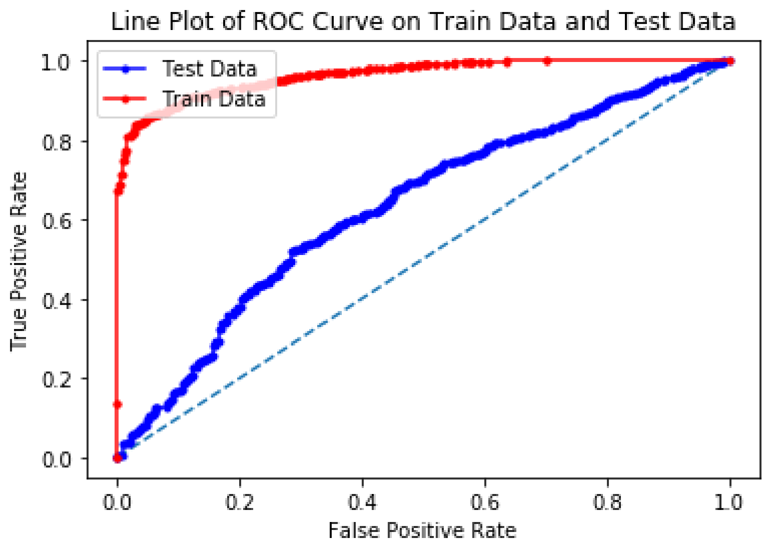


for cv data:

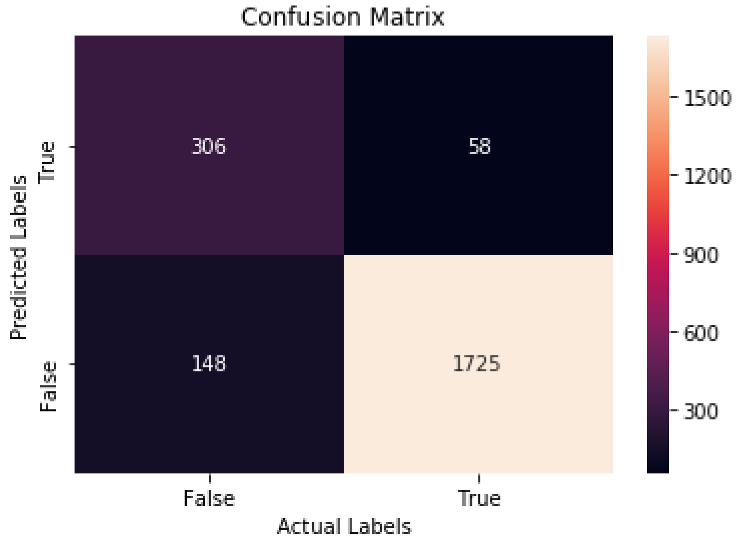


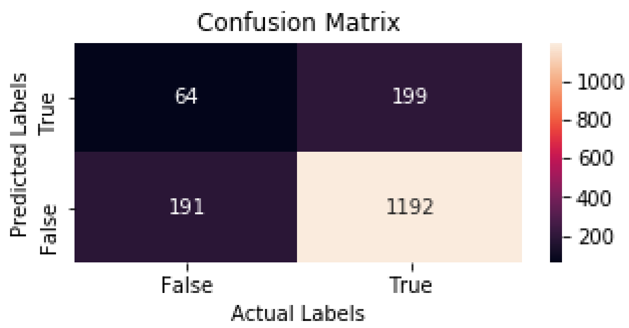
In [51]:
```
#Testing random classifier on tfidf_w2v
Random_forest_Test(tfidf_sent_vectors_train,y_train,tfidf_sent_vectors_
test,y_test,optimal_depth=7,optimal_estimator=400)
```

```
printing auc score for train data: 0.9641243700239963
printing auc score for test data: 0.6347005600323317
```




```
Train confusion matrix
Test confusion matrix
```





<Figure size 360x144 with 0 Axes>

[5.2] Applying GBDT using XGBOOST

In [0]:
```
#this is the function for xgboost
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
def XGBOOST_Classifier(xtrain,xcv,ytrain,ycv):
    predicted_cv = []
    predicted_train = []
    d = [2,4,7,10,20]#depth
    e = [100, 200, 300, 400, 500] #estimator
    for i in d:
        for j in e:
            clf = XGBClassifier(n_estimators=j, max_depth=i, scale_pos_
weight=1, objective='binary:logistic')
            clf.fit(xtrain,ytrain)
            prob_cv = clf.predict_proba(xcv)
            prob_train = clf.predict_proba(xtrain)
            prob_cv = prob_cv[:,1]
            prob_train = prob_train[:,1]
            auc_score_cv = roc_auc_score(ycv,prob_cv)
```

```
                auc_score_train = roc_auc_score(ytrain,prob_train)
                predicted_cv.append(auc_score_cv)
                predicted_train.append(auc_score_train)
    cmap=sns.light_palette("yellow")
    #Heat map:
    print("for training data:")
    predicted_train = np.array(predicted_train)
    predicted_train = predicted_train.reshape(len(d),len(e))
    plt.figure(figsize=(8,4))
    cmap=sns.light_palette("yellow")
    sns.heatmap(predicted_train,annot=True, cmap=cmap, fmt=".3f", xtick
labels=e,yticklabels=d)
    plt.xlabel('Estimators')
    plt.ylabel('Depths')
    plt.show()
    print("for cv data:")
    predicted_cv = np.array(predicted_cv)
    predicted_cv = predicted_cv.reshape(len(d),len(e))
    plt.figure(figsize=(8,4))
    sns.heatmap(predicted_cv, annot=True, cmap=cmap, fmt=".3f", xtickla
bels=e, yticklabels=d)
    plt.xlabel('Estimators')
    plt.ylabel('Depths')
    plt.show()
```

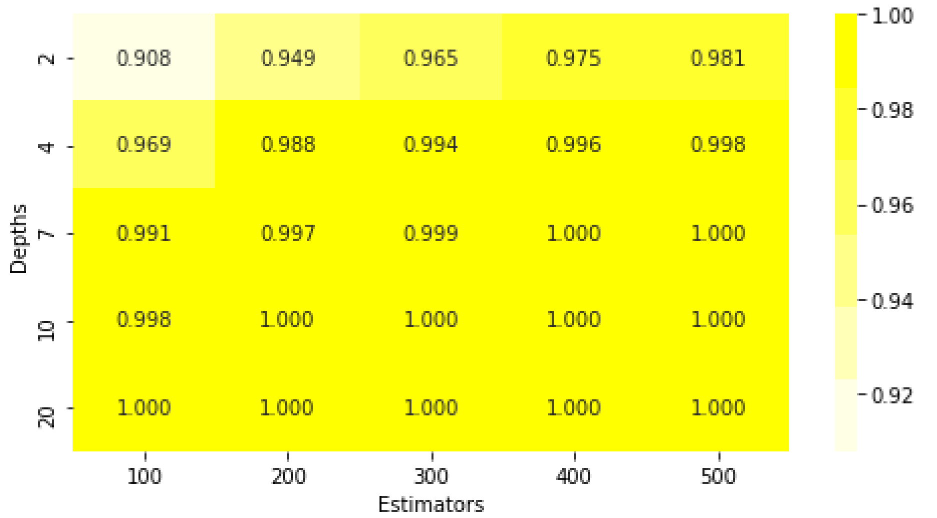
In [0]:
```
#function for testing of xgboost
import scikitplot.metrics as skplt
def XGBOOST_Test(xtrain,ytrain,xtest,ytest,optimal_depth,optimal_estima
tor):
    clf = XGBClassifier(n_estimators = optimal_estimator, max_depth = o
ptimal_depth)
    clf.fit(xtrain,ytrain)
    prob_test= clf.predict_proba(xtest)
    prob_train= clf.predict_proba(xtrain)
    prob_test = prob_test[:, 1]
    prob_train = prob_train[:,1]
    print("printing auc score for train data:",roc_auc_score(ytrain,pro
b_train))
    print("printing auc score for test data:",roc_auc_score(ytest,prob_
```

```
test))
    # code to calculate roc curve:
    fpr_train, tpr_train, thresholds = roc_curve(ytrain,prob_train)
    fpr_test, tpr_test, thresholds = roc_curve(ytest,prob_test)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr_test, tpr_test, marker='.',color ='b',label='Test Dat
a')
    plt.plot(fpr_train, tpr_train, marker='.',color= 'r',label='Train D
ata')
    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()
    print("Train confusion matrix")
    ax= plt.subplot()
    arr1=confusion_matrix(ytrain, clf.predict(xtrain))
    df_1= pd.DataFrame(arr1, range(2),range(2))
    plt.figure(figsize = (5,2))
    sns.heatmap(df_1, annot=True,fmt="d",ax=ax)
    ax.set_title('Confusion Matrix');
    ax.set_xlabel('Actual Labels')
    ax.set_ylabel('Predicted Labels')
    ax.xaxis.set_ticklabels(['False', 'True']);
    ax.yaxis.set_ticklabels(['True', 'False']);
    print("Test confusion matrix")
    ax= plt.subplot()
    arr1=confusion_matrix(ytest, clf.predict(xtest))
    df_1= pd.DataFrame(arr1, range(2),range(2))
    plt.figure(figsize = (5,2))
    sns.heatmap(df_1, annot=True,fmt="d",ax=ax)
    ax.set_title('Confusion Matrix');
    ax.set_xlabel('Actual Labels')
    ax.set_ylabel('Predicted Labels')
    ax.xaxis.set_ticklabels(['False', 'True']);
    ax.yaxis.set_ticklabels(['True', 'False']);
```

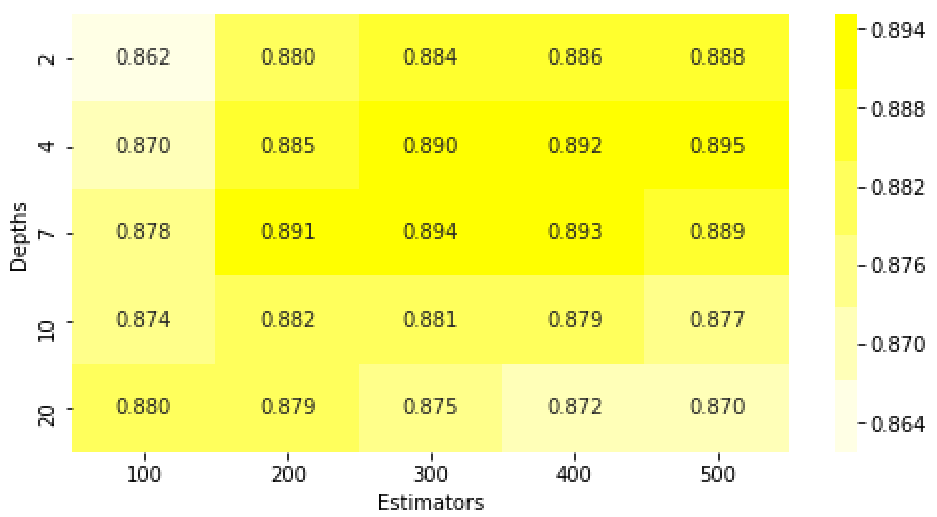
[5.2.1] Applying XGBOOST on BOW, SET 1

In [63]:
```
XGBOOST_Classifier(X_train_bow,X_cv_bow,y_train,y_cv)
```

```
for training data:
```

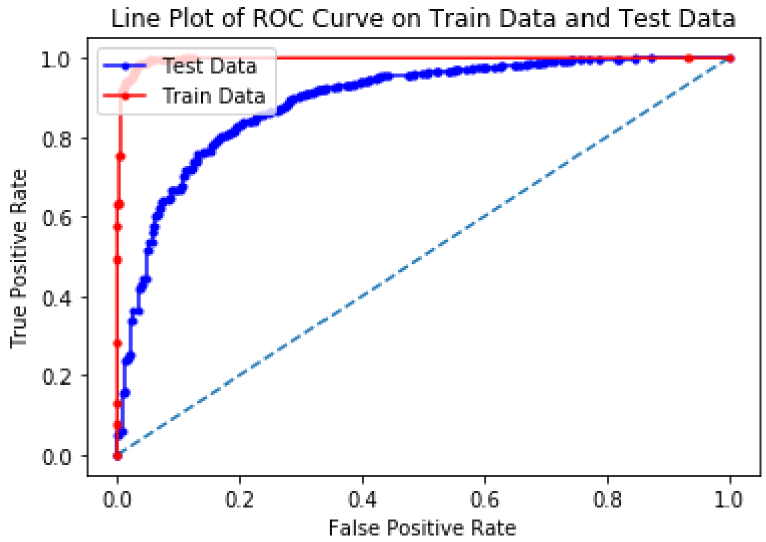



```
for cv data:
```

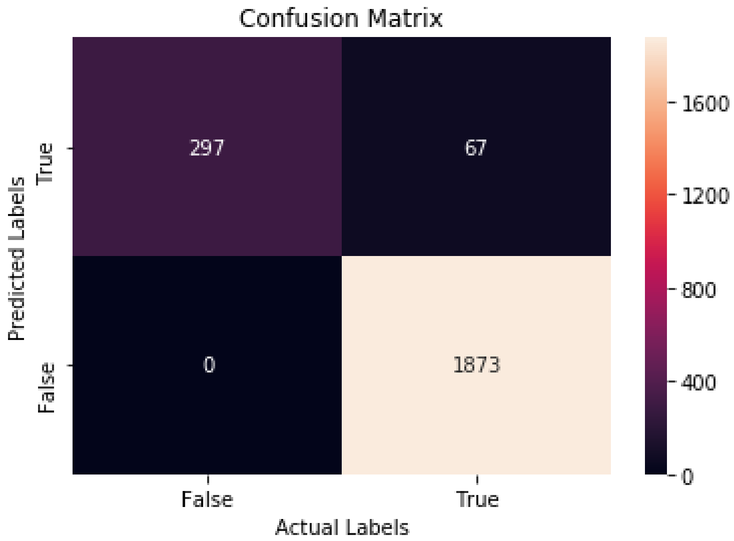


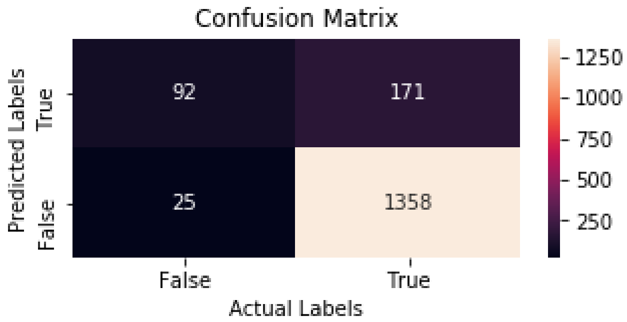
In [66]:
```
#Testing xgboost classifier on bow data
XGBOOST_Test(X_train_bow,y_train,X_test_bow,y_test,optimal_depth=4,optimal_estimator=400)
```

```
printing auc score for train data: 0.9963822216224779
printing auc score for test data: 0.8910095153259707
```




```
Train confusion matrix
Test confusion matrix
```





<Figure size 360x144 with 0 Axes>

In [67]:
```
#print top 20 features xgboost classifier
clf = XGBClassifier(n_estimators=400, max_depth=3, scale_pos_weight=1,
objective='binary:logistic')
clf.fit(X_train_bow,y_train)
features = imp_feature(vectorizer,clf)
```

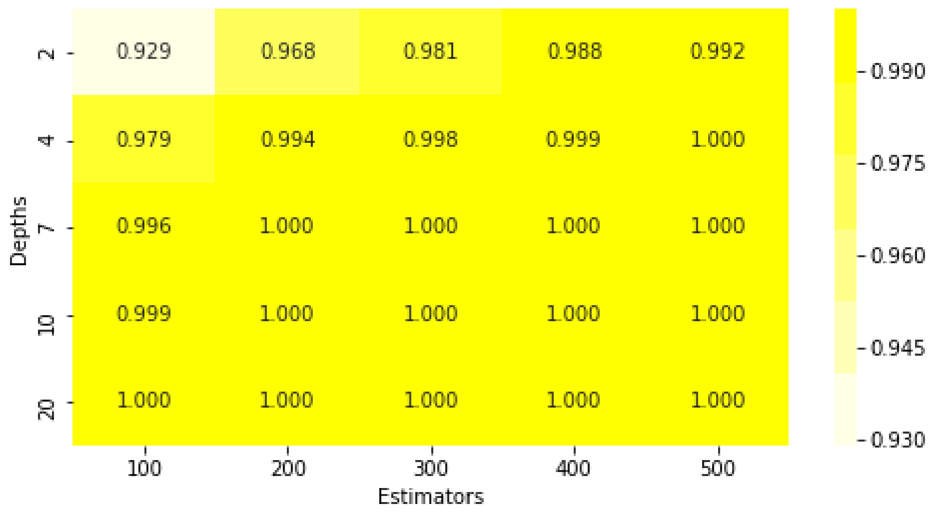
```
feature_importances     important features
==================================================
0.0134                  received
```

```
0.0120                  picture
0.0120                  money
0.0118                  terrible
0.0115                  disappointed
0.0113                  awful
0.0110                  waste
0.0105                  great
0.0104                  return
0.0094                  ingredient
0.0091                  extra
0.0091                  disappointing
0.0090                  cookies
0.0085                  love
0.0083                  beef
0.0082                  stick
0.0081                  weak
0.0079                  item
0.0078                  wont
0.0074                  worst
```

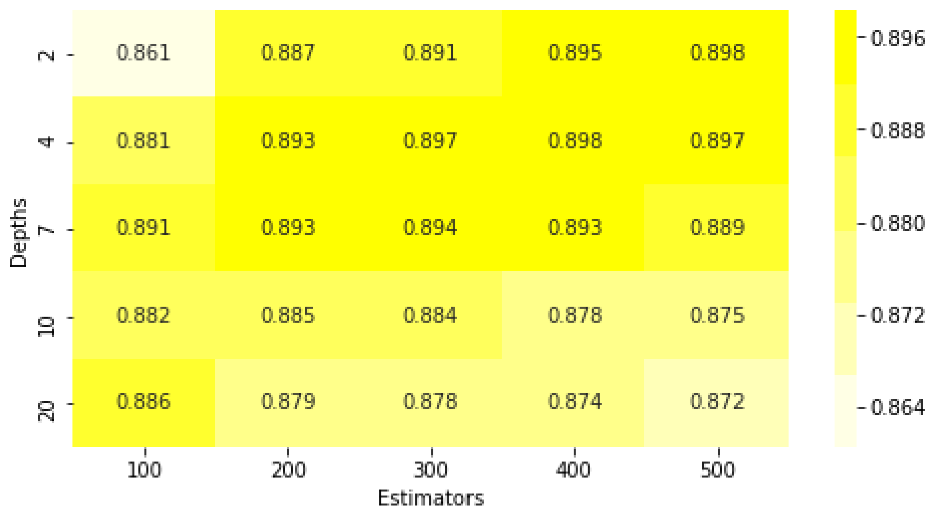
[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [82]:
```
XGBOOST_Classifier(X_train_tf_idf,X_cv_tf_idf,y_train,y_cv)
```

```
for training data:
```

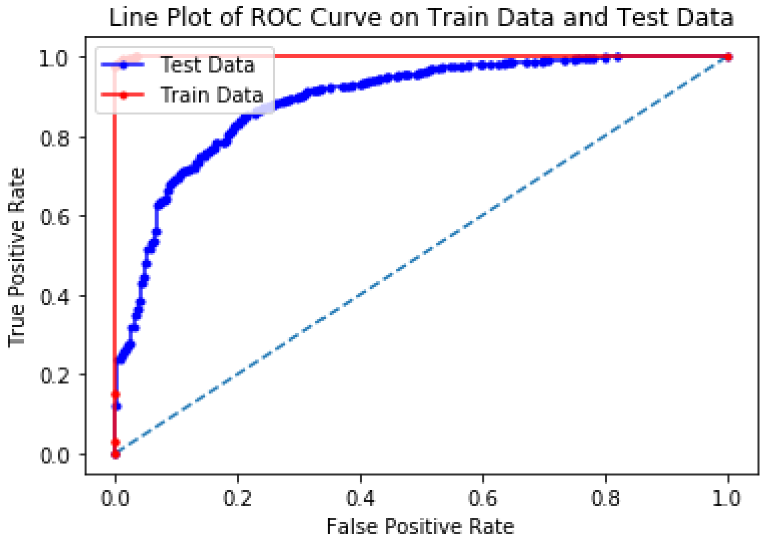



```
for cv data:
```

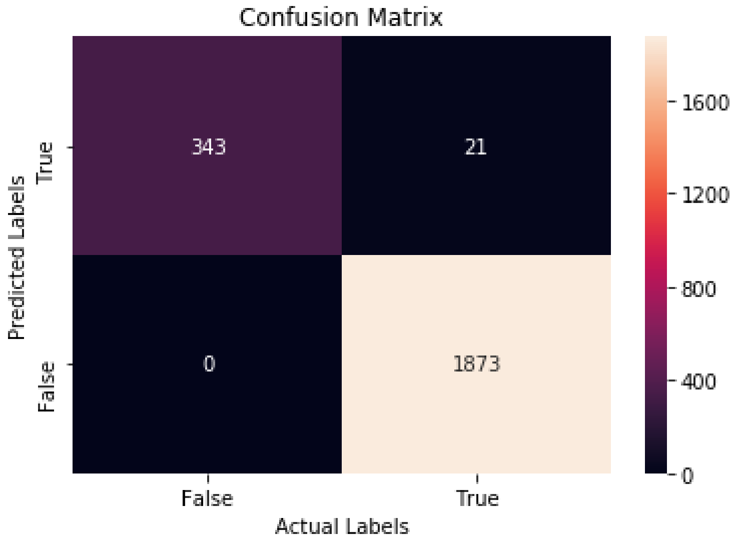


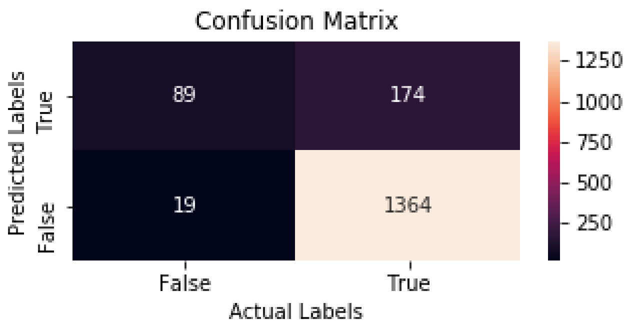
In [83]:
```
#Testing xgboost classifier on tfidf data
XGBOOST_Test(X_train_tf_idf,y_train,X_test_tf_idf,y_test,optimal_depth=
4,optimal_estimator=500)
```

```
printing auc score for train data: 0.999769717735548
printing auc score for test data: 0.8909050419405657
```




```
Train confusion matrix
Test confusion matrix
```






```
<Figure size 360x144 with 0 Axes>
```

In [81]:
```
#print top 20 features xgb classifier
clf = XGBClassifier(max_depth =4, n_estimators = 500,class_weight='bala
nced')
clf.fit(X_train_tf_idf,y_train)
features = imp_feature(tf_idf_vect,clf)
```

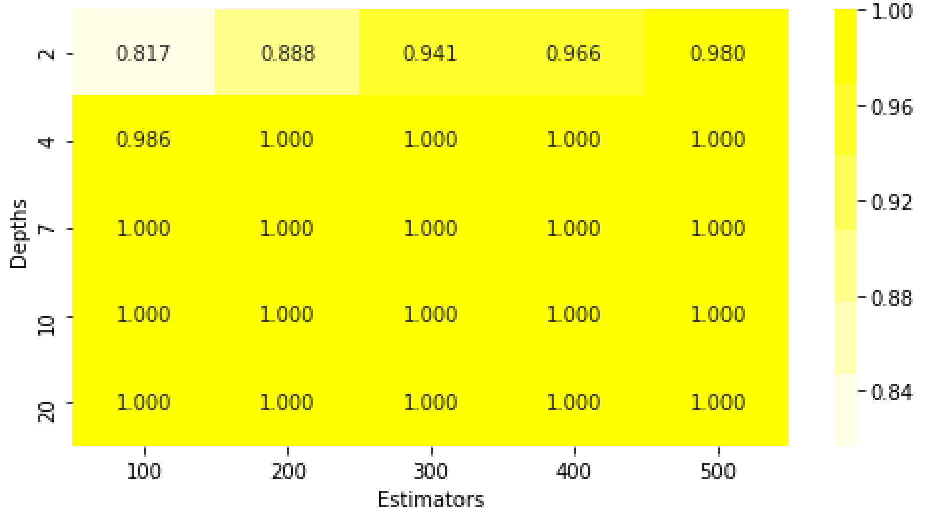
```
feature_importances     important features
==================================================
0.0188                  label
0.0177                  would not
```

0.0133	worst
0.0108	awful
0.0102	great
0.0097	not disappointed
0.0097	disappointing
0.0094	received
0.0092	return
0.0091	bottle
0.0089	weak
0.0086	not good
0.0086	disappointed
0.0084	terrible
0.0076	company
0.0075	store
0.0074	told
0.0071	not buy
0.0071	never
0.0070	not best

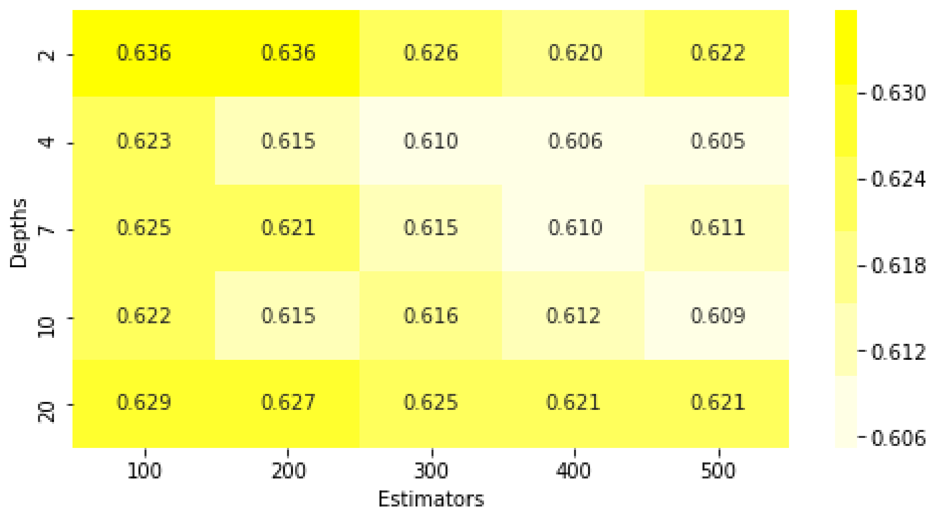
[5.2.3] Applying XGBOOST on AVG W2V, SET 3

In [72]:
```
XGBOOST_Classifier(sent_vectors_train,sent_vectors_cv,y_train,y_cv)
```

```
for training data:
```

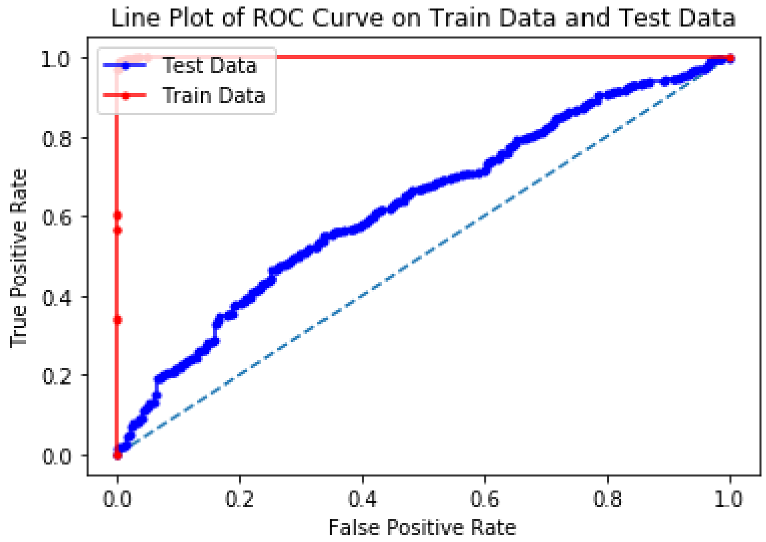



```
for cv data:
```

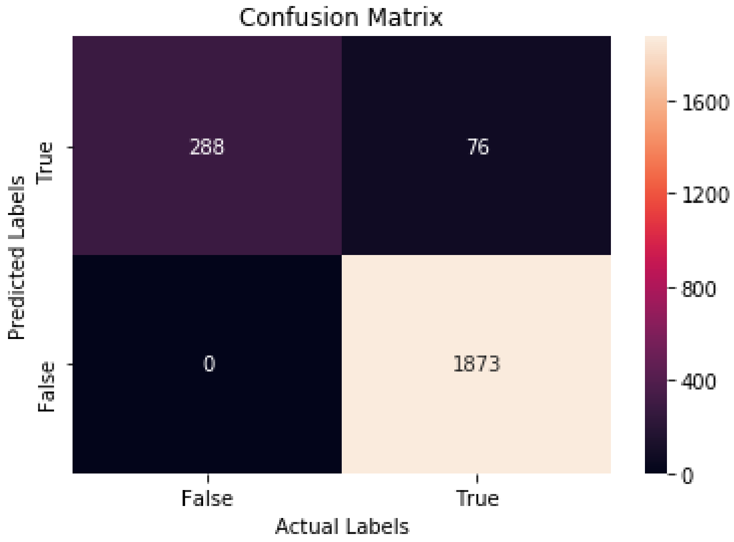


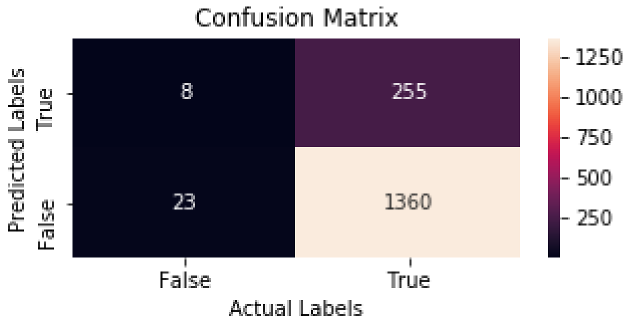
In [73]:
```
#Testing xgboost classifier on w2v
XGBOOST_Test(sent_vectors_train,y_train,sent_vectors_test,y_test,optima
l_depth=4,optimal_estimator=200)
```

```
printing auc score for train data: 0.9997623839054698
printing auc score for test data: 0.6273791751551292
```




```
Train confusion matrix
Test confusion matrix
```



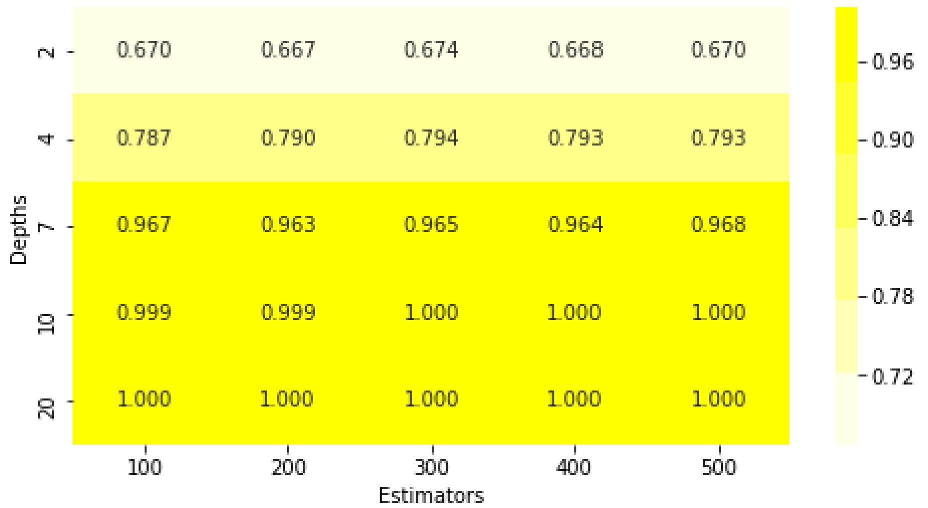



```
<Figure size 360x144 with 0 Axes>
```

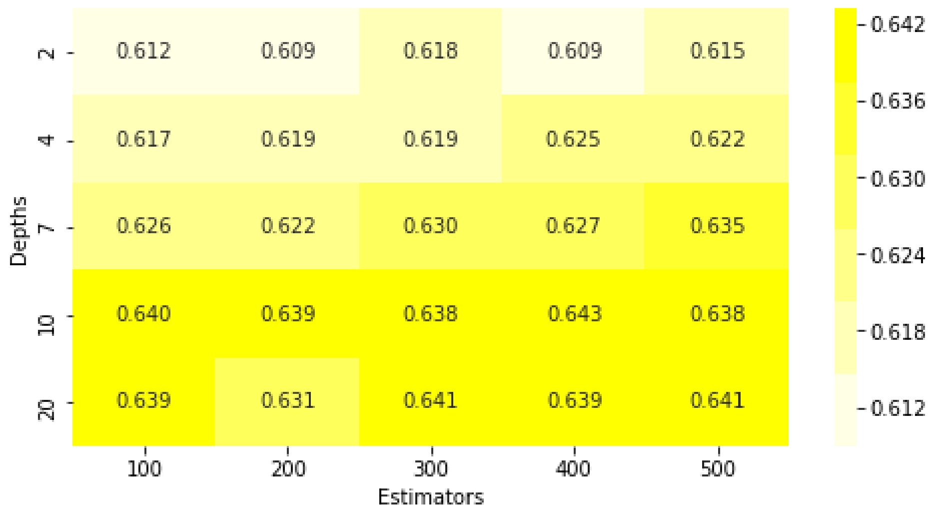
[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [74]:
```
Random_Forest_Classifier(tfidf_sent_vectors_train,tfidf_sent_vectors_cv
,y_train,y_cv)
```

```
for training data:
```

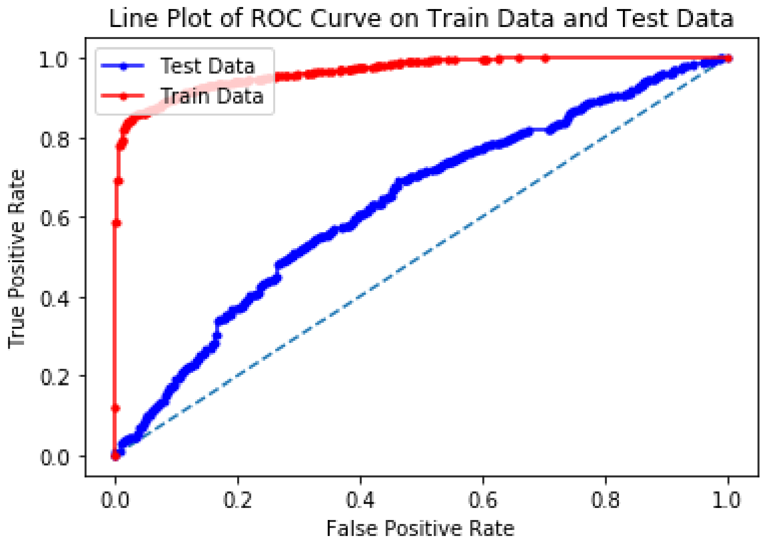


for cv data:

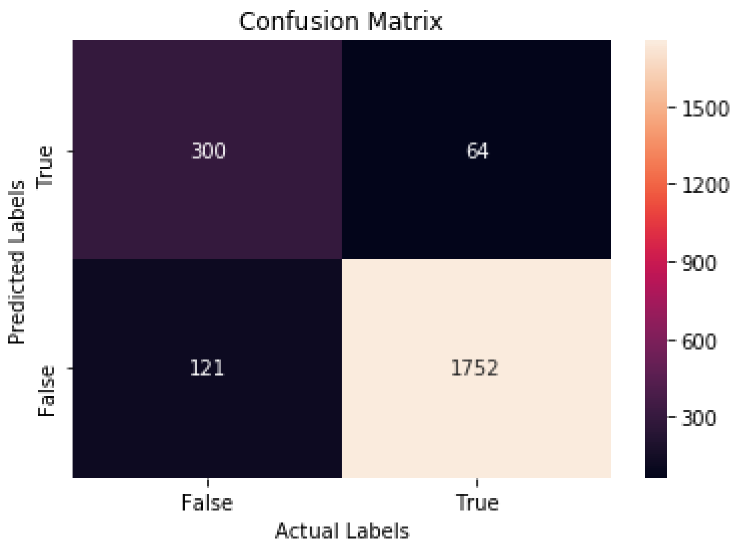


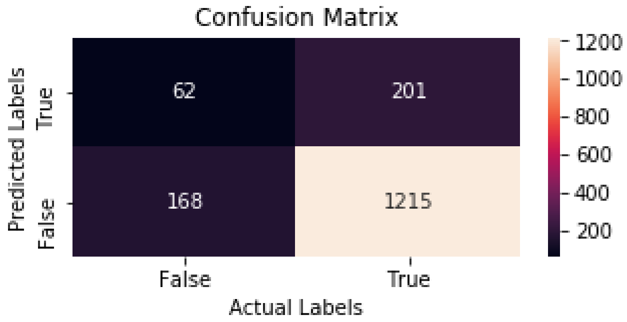
In [75]:
```
#Testing xgboost classifier on tfidf-w2v
Random_forest_Test(tfidf_sent_vectors_train,y_train,tfidf_sent_vectors_
test,y_test,optimal_depth=7,optimal_estimator=400)
```

```
printing auc score for train data: 0.9668085518325774
printing auc score for test data: 0.6337548009644542
```




```
Train confusion matrix
Test confusion matrix
```






```
<Figure size 360x144 with 0 Axes>
```

[6] Conclusions

In [88]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Model","Hyper Parameter:Depth","Hyper Parameter:n_estimator","AUC SCORE On test data"]
x.add_row(["BoW","Random Forest",7,400,89.02])
x.add_row(["Tf-Idf","Random Forest",7,500,89.74])
x.add_row(["Avg-W2V","Random Forest",10,500,64])
x.add_row(["TfIdf-W2V","Random Forest",7,400,63])
x.add_row(["BoW","XGBoost",4,400,89.10])
x.add_row(["Tf-Idf","XGBoost",4,500,89.9])
x.add_row(["Avg-W2V","XGBoost",4,200,62.73])
x.add_row(["TfIdf-W2V","XGBoost",7,400,63.37])
print(x)
```

```
+------------+---------------+-----------------------+----------------
-----------+------------------------+
| Vectorizer |     Model     | Hyper Parameter:Depth | Hyper Parameter:
n_estimator | AUC SCORE On test data |
+------------+---------------+-----------------------+----------------
-----------+------------------------+
|    BoW     | Random Forest |           7           |             400
           |         89.02          |
|   Tf-Idf   | Random Forest |           7           |             500
```

```
             |          89.74          |
|  Avg-W2V   | Random Forest |          10           |              500
             |            64           |
| TfIdf-W2V  | Random Forest |          7            |              400
             |            63           |
|    BoW     |    XGBoost    |          4            |              400
             |           89.1          |
|   Tf-Idf   |    XGBoost    |          4            |              500
             |           89.9          |
|  Avg-W2V   |    XGBoost    |          4            |              200
             |          62.73          |
| TfIdf-W2V  |    XGBoost    |          7            |              400
             |          63.37          |
+------------+---------------+-----------------------+-----------------
------------+-----------------------+
```