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 [0]:

```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 [0]:
```
!cp "/content/drive/My Drive/final.sqlite" "final.sqlite"
```

In [0]:
```
import os
if os.path.isfile('final.sqlite'):
    conn = sqlite3.connect('final.sqlite')
    final = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score !=
 3 """, conn)
    conn.close()
else:
    print("Please the above cell")

print("Preprocessed Amzon fine food data columns  shape : ",final.shape
)
print("fPreprocessed Amzon fine food data columns       :",final.column
s.values)
```

```
Preprocessed Amzon fine food data columns  shape :  (364171, 12)
fPreprocessed Amzon fine food data columns       : ['index' 'Id' 'Produ
ctId' 'UserId' 'ProfileName' 'HelpfulnessNumerator'
 'HelpfulnessDenominator' 'Score' 'Time' 'Summary' 'Text' 'CleanedTex
t']
```

In [0]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 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)
```

```python
# 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-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

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

Out[0]:

	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]:

```python
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(*)
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 [0]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[0]:

UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)

	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 [0]:
```
display['COUNT(*)'].sum()
```

Out[0]:
```
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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

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=Tr
ue, inplace=False, kind='quicksort', na_position='last')
```

In [0]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
,"Text"}, keep='first', inplace=False)
final.shape
```

Out[0]: (4986, 10)

In [0]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[0]: 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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

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

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

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

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

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

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

```
Why is this $[...] when the same product is available for $[...] here?<
br />http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<
br /><br />The Victor M380 and M502 traps are unreal, of course -- tota
l fly genocide. Pretty stinky, but only right nearby.
==================================================
```

```
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.<br /><br />These are chocolate-oatmeal cookies.  If you don't li
ke that combination, don't order this type of cookie.  I find the combo
quite nice, really.  The oatmeal sort of "calms" the rich chocolate fla
vor and gives the cookie sort of a coconut-type consistency.  Now let's
also remember that tastes differ; so, I've given my opinion.<br /><br /
>Then, these are soft, chewy cookies -- as advertised.  They are not "c
rispy" cookies, or the blurb would say "crispy," rather than "chewy."
I happen to like raw cookie dough; however, I don't see where these tas
te like raw cookie dough.  Both are soft, however, so is this the confu
sion?  And, yes, they stick together.  Soft cookies tend to do that.  T
hey aren't individually wrapped, which would add to the cost.  Oh yeah,
chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you
want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you
want a cookie that's soft, chewy and tastes like a combination of choco
late and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />Thi
s k cup is great coffee.  dcaf is very good as well
==================================================
```

In [0]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
```

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

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor M380 and M502 traps are unreal, of course -- t
otal fly genocide. Pretty stinky, but only right nearby.
```

In [0]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how
-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...] here?
/>The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
```

```
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.These are chocolate-oatmeal cookies.  If you don't like that comb
ination, don't order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and give
s the cookie sort of a coconut-type consistency.  Now let's also rememb
er that tastes differ; so, I've given my opinion.Then, these are soft,
chewy cookies -- as advertised.  They are not "crispy" cookies, or the
blurb would say "crispy," rather than "chewy."  I happen to like raw co
okie dough; however, I don't see where these taste like raw cookie doug
h.  Both are soft, however, so is this the confusion?  And, yes, they s
tick together.  Soft cookies tend to do that.  They aren't individually
wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies
tend to be somewhat sweet.So, if you want something hard and crisp, I s
uggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy
and tastes like a combination of chocolate and oatmeal, give these a tr
y.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cu
p is great coffee.  dcaf is very good as well
```

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)
```

```python
    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]:

```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I am sorry; but
these reviews do nobody any good beyond reminding us to look  before or
dering.<br /><br />These are chocolate-oatmeal cookies.  If you do not
like that combination, do not order this type of cookie.  I find the co
mbo quite nice, really.  The oatmeal sort of "calms" the rich chocolate
flavor and gives the cookie sort of a coconut-type consistency.  Now le
t is also remember that tastes differ; so, I have given my opinion.<br
/><br />Then, these are soft, chewy cookies -- as advertised.  They are
not "crispy" cookies, or the blurb would say "crispy," rather than "che
wy."  I happen to like raw cookie dough; however, I do not see where th
ese taste like raw cookie dough.  Both are soft, however, so is this th
e confusion?  And, yes, they stick together.  Soft cookies tend to do t
hat.  They are not individually wrapped, which would add to the cost.
Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />S
o, if you want something hard and crisp, I suggest Nabiso is Ginger Sna
ps.  If you want a cookie that is soft, chewy and tastes like a combina
tion of chocolate and oatmeal, give these a try.  I am here to place my
second order.
==================================================
```

In [0]:

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

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor  and  traps are unreal, of course -- total fly
genocide. Pretty stinky, but only right nearby.
```

In [0]:

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

```
Wow So far two two star reviews One obviously had no idea what they wer
e ordering the other wants crispy cookies Hey I am sorry but these revi
ews do nobody any good beyond reminding us to look before ordering br b
r These are chocolate oatmeal cookies If you do not like that combinati
on do not order this type of cookie I find the combo quite nice really
The oatmeal sort of calms the rich chocolate flavor and gives the cooki
e sort of a coconut type consistency Now let is also remember that tast
es differ so I have given my opinion br br Then these are soft chewy co
okies as advertised They are not crispy cookies or the blurb would say
crispy rather than chewy I happen to like raw cookie dough however I do
not see where these taste like raw cookie dough Both are soft however s
o is this the confusion And yes they stick together Soft cookies tend t
o do that They are not individually wrapped which would add to the cost
Oh yeah chocolate chip cookies tend to be somewhat sweet br br So if yo
u want something hard and crisp I suggest Nabiso is Ginger Snaps If you
want a cookie that is soft chewy and tastes like a combination of choco
late and oatmeal give these a try I am here to place my second order
```

In [0]:

```python
# 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 [0]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 4986/4986 [00:01<00:00, 3137.37it/s]
```

In [0]:

```
preprocessed_reviews[1500]
```

Out[0]:

```
'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'
```

[3.2] Preprocessing Review Summary

In [0]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
```

```
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

[4.2] Bi-Grams and n-Grams.

In [0]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features
=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
```

```python
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape
())
print("the number of unique words including both unigrams and bigrams "
, final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [0]:

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

In [0]:

```python
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
```

```
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

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

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

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

[4.4.1.1] Avg W2v

```
In [0]: # average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sentance): # 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.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
██████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# 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_)))
```

In [0]:

```
# 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 = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sentance): # 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.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████████
██████████| 4986/4986 [00:20<00:00, 245.63it/s]
```

[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

[5.1.1] Applying Random Forests on BOW, SET 1

In [0]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split

preprocessed_reviews=final['CleanedText'][:40000]
score=final['Score'][:40000]
X_train, X_test, y_train, y_test = train_test_split(preprocessed_review
s, score, test_size=0.2, random_state=42)
```

In [0]:
```
#BoW
count_vect = CountVectorizer(max_df=0.95, min_df=2,stop_words='english'
,max_features=500) #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

X_train_bow = count_vect.transform(X_train)
print("the type of count vectorizer ",type(X_train_bow))
print("the shape of out text BOW vectorizer ",X_train_bow.get_shape())
print("the number of unique words ", X_train_bow.get_shape()[1])
```

```
X_test_bow = count_vect.transform(X_test)
print("the type of count vectorizer ",type(X_test_bow))
print("the shape of out text BOW vectorizer ",X_test_bow.get_shape())
print("the number of unique words ", X_test_bow.get_shape()[1])
```

```
some feature names  ['abl', 'absolut', 'actual', 'ad', 'add', 'addict', 'addit', 'ago', 'allergi', 'alreadi']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (32000, 500)
the number of unique words  500
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (8000, 500)
the number of unique words  500
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(with_mean=False)
scaler.fit(X_train_bow)
X_train_tfidf=scaler.transform(X_train_bow)

X_test_tfidf=scaler.transform(X_test_bow)
```

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
from  sklearn.model_selection import RandomizedSearchCV
clf=RandomForestClassifier()
param_grid={'n_estimators' :  [5, 10, 50, 100, 200, 500, 1000] , 'max_depth' :  [2, 3, 4, 5, 6, 7, 8, 9, 10]}
gcv=RandomizedSearchCV(clf,param_grid,cv=10,scoring='roc_auc')
gcv.fit(X_train_bow,y_train)
print(gcv.best_params_)
print(gcv.best_score_)

optimal_depth          =  gcv.best_params_['max_depth']
optimal_estimators     =  gcv.best_params_['n_estimators']
```

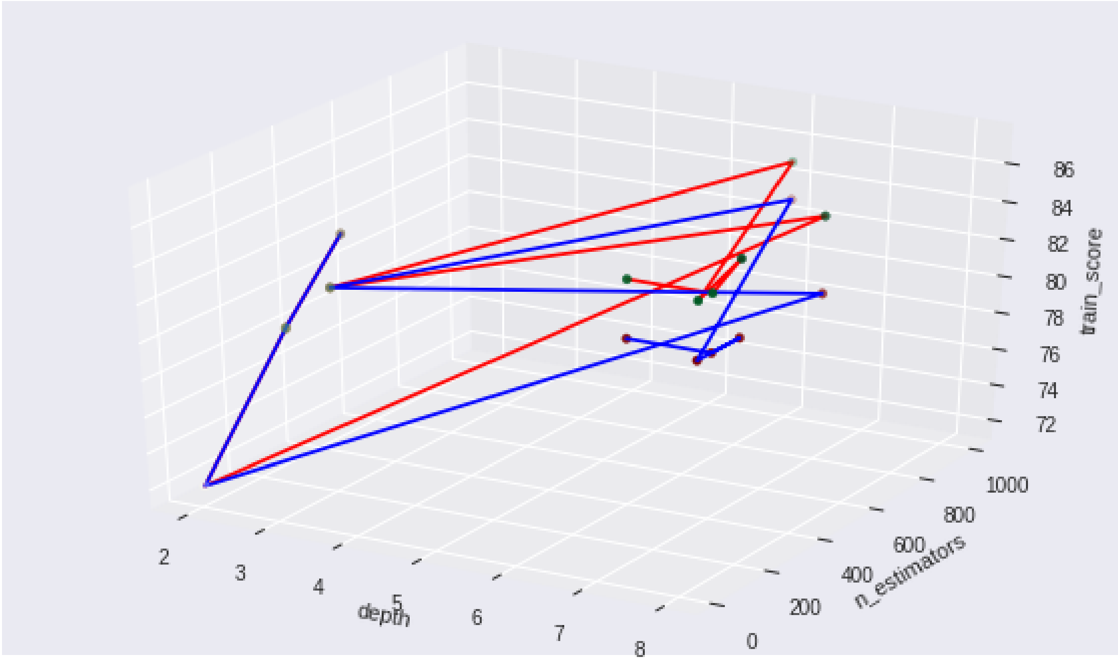
```
{'n_estimators': 500, 'max_depth': 8}
0.8349917055482534
```

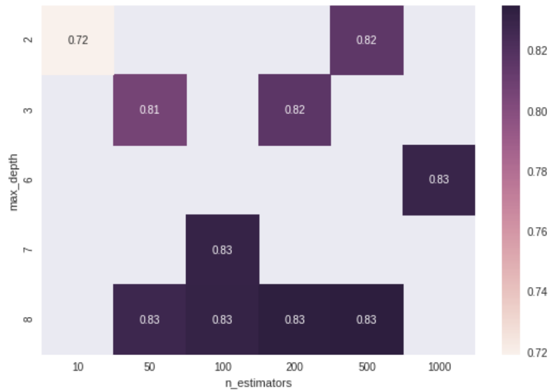
In [0]:

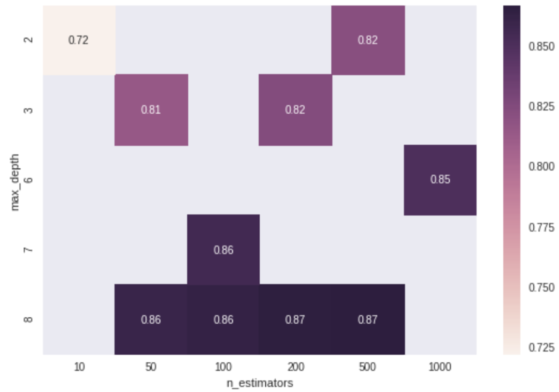
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='G
reens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score"
)
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrR
d',label="test_score")
```

```
print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_scor
e")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

```
[2, 3, 2, 8, 3, 6, 8, 8, 8, 7]
[500, 50, 10, 500, 200, 1000, 50, 200, 100, 100]
[82.0, 81.0, 72.0, 87.0, 82.0, 85.0, 86.0, 87.0, 86.0, 86.0]
[82.0, 81.0, 72.0, 83.0, 82.0, 83.0, 83.0, 83.0, 83.0, 83.0]
ploting 3d grap
ploting Heat Map
```







In [0]:

```
df_params
```

Out[0]:

	max_depth	n_estimators
0	2	500

	max_depth	n_estimators
1	3	50
2	2	10
3	8	500
4	3	200
5	6	1000
6	8	50
7	8	200
8	8	100
9	7	100

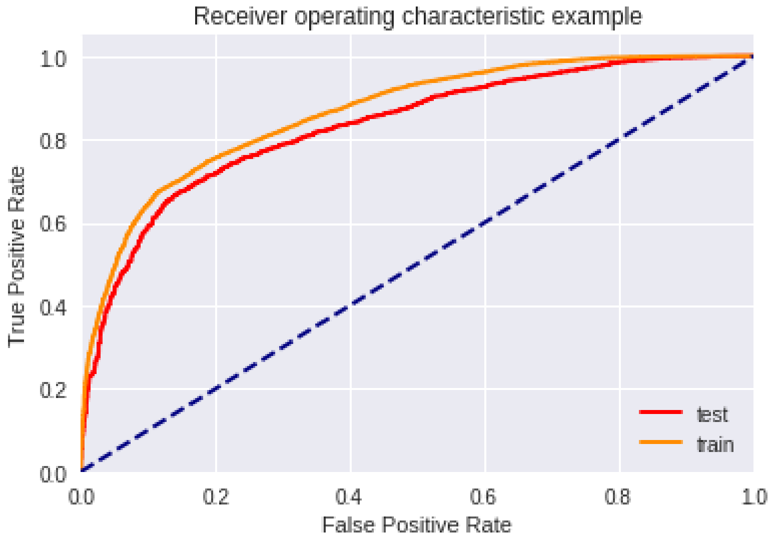
In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.ensemble import RandomForestClassifier

clf1=RandomForestClassifier(n_estimators=optimal_estimators,max_depth=o
ptimal_depth)
clf1.fit(X_train_bow,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_bow, y_train)

pred = sig_clf.predict_proba(X_test_bow)[:,1]
pred_train = sig_clf.predict_proba(X_train_bow)[:,1]


pred_train_without_CCV=clf1.predict(X_train_bow)
pred_without_CCV=clf1.predict(X_test_bow)
```

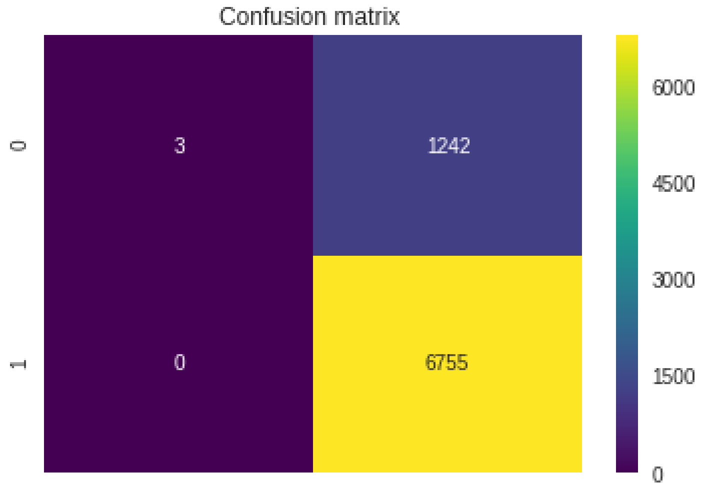
```
print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")
```

```
tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("         ")
print("         ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  84.475
Precision Score :  84.46917594097786
Recall Score :  100.0
F1 Score :  91.58080260303689

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      1245
           1       0.84      1.00      0.92      6755

   micro avg       0.84      0.84      0.84      8000
   macro avg       0.92      0.50      0.46      8000
weighted avg       0.87      0.84      0.77      8000


AUC Score for train data : 0.8634610585217078
AUC Score for test data : 0.8313486068626839
```




```
TrueNegative : 3
FalsePostive : 1242
FalseNegative : 0
TruePostive  :  6755
```



0 1

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

In [0]:
```
-np.sort(-clf1.feature_importances_)[:20]


features=count_vect.get_feature_names()
imp=clf1.feature_importances_.argsort()[::-1][:20]
top20Features=[features[i] for i in imp]
top20Features
print(top20Features)

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

wordcloud = WordCloud(width = 400, height = 400, background_color ='black',  min_font_size = 10).generate(' '.join(top20Features))

# plot the WordCloud image
plt.figure(figsize = (4, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

```
['great', 'love', 'product', 'tast', 'like', 'veri', 'best', 'use', 'just', 'good', 'tri', 'flavor', 'did', 'order', 'dog', 'food', 'onli', 'time', 'buy', 'didnt']
```

In [0]:

```
#a=clf1.feature_importances_.tolist()
#a.sort()
#print(a[::-1][:20])

#np.sort(clf1.feature_importances_,)[::-1][:20]
-np.sort(-clf1.feature_importances_)[:20]


features=count_vect.get_feature_names()
imp=clf1.feature_importances_.tolist()

print([i for i in zip(features,imp)])

df=pd.DataFrame([features,imp],index=['feature','values'])
df=df.T
df1=df.sort_values('values',ascending=False)


print(len(imp))
len(features)
df1[:20]
```

```
[('add', 0.0044907245051340255), ('alway', 0.005562703837472016), ('ama
zon', 0.011375769641162808), ('ani', 0.008867427638205094), ('bag', 0.0
11957502425203879), ('becaus', 0.011379591450007774), ('befor', 0.00781
8718892939955), ('best', 0.01733912219391957), ('better', 0.00970499625
723798), ('bit', 0.006421830449315), ('bottl', 0.005776588192260454),
('bought', 0.010886069038605832), ('box', 0.0116585777737473), ('bran
d', 0.009106228894106803), ('buy', 0.012825606305026584), ('cat', 0.008
625360953580512), ('chocol', 0.0076941147605365445), ('coffe', 0.008020
58840060532), ('come', 0.00670895188303685), ('cup', 0.0047731570609380
61), ('day', 0.010783424858340338), ('delici', 0.008656993874789555),
('did', 0.01518543524674542), ('didnt', 0.012730591258581524), ('diffe
r', 0.007830141246172519), ('doe', 0.007108266224982648), ('dog', 0.014
413099010717306), ('dont', 0.012216358128775744), ('drink', 0.006819678
635322799), ('eat', 0.012032961522191878), ('enjoy', 0.0073940288169380
18), ('everi', 0.0050955098381189135), ('favorit', 0.00622719446398589
7), ('flavor', 0.015429668071205254), ('food', 0.013223788491491957),
('fresh', 0.005307394376782559), ('good', 0.01673559891022993), ('got',
0.009281397591786476), ('great', 0.02813926611838967), ('help', 0.00491
6083456355531), ('high', 0.007164121257632216), ('hot', 0.0059226111674
22086), ('ingredi', 0.009126661675397467), ('ive', 0.00826031658065426
4), ('just', 0.01678987589472409), ('know', 0.009043697248119549), ('li
ke', 0.020377875849354084), ('littl', 0.009409523774771826), ('local',
0.005889664331044513), ('long', 0.005102971180784628), ('look', 0.01180
6048842219006), ('lot', 0.007370172032687678), ('love', 0.0271192549413
43863), ('make', 0.011909902069019958), ('mani', 0.006670584677781024),
('milk', 0.004107540365873081), ('mix', 0.006889127716397252), ('mont
h', 0.006211480187335332), ('natur', 0.005489018897415863), ('need', 0.
0072400453039554715), ('nice', 0.00679367278365118), ('oil', 0.00436832
53138147565), ('old', 0.008857224141714448), ('onli', 0.013168299674576
594), ('order', 0.014645679669403161), ('packag', 0.01030859318175575
4), ('perfect', 0.007993404633559416), ('price', 0.011636804051079853),
('product', 0.025393089412330226), ('purchas', 0.011239729603998326),
('qualiti', 0.007018252928108116), ('realli', 0.010940775479691056),
('recommend', 0.009198156988103353), ('review', 0.010203349395123669),
('sauc', 0.005149779877810109), ('say', 0.008510990215190039), ('ship',
0.010133705057587794), ('sinc', 0.0068831070981813505), ('small', 0.006
949813936013746), ('smell', 0.009195116854539043), ('start', 0.00556282
7936765836), ('store', 0.00878837245910199), ('stuff', 0.00726045348245
1339), ('sugar', 0.006260114255773394), ('sweet', 0.00551811076012660
1), ('tast', 0.022976509445782275), ('tea', 0.012210209970606824), ('th
```

```
ing', 0.008886149503862314), ('think', 0.00973521637383012), ('time',
0.012944895546747284), ('treat', 0.008679317670083923), ('tri', 0.01630
5309677121547), ('use', 0.016895330483878333), ('veri', 0.0174871659869
6611), ('want', 0.009199722960260264), ('water', 0.00785295490966159),
('way', 0.007739702745002463), ('wonder', 0.006309442634762174), ('wor
k', 0.009057776544844858), ('year', 0.009321541667259247)]
100
```

Out[0]:

	feature	values
38	great	0.0281393
52	love	0.0271193
68	product	0.0253931
85	tast	0.0229765
46	like	0.0203779
93	veri	0.0174872
7	best	0.0173391
92	use	0.0168953
44	just	0.0167899
36	good	0.0167356
91	tri	0.0163053
33	flavor	0.0154297
22	did	0.0151854
64	order	0.0146457
26	dog	0.0144131
34	food	0.0132238
63	onli	0.0131683
89	time	0.0129449

	feature	values
14	buy	0.0128256
23	didnt	0.0127306

[5.1.3] Applying Random Forests on TFIDF, SET 2

In [0]:

```
#BoW
#tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect = TfidfVectorizer(max_df=0.95, min_df=2,stop_words='englis
h',max_features=500) #in scikit-learn
tf_idf_vect.fit(X_train)
print("some feature names ", tf_idf_vect.get_feature_names()[:10])
print('='*50)

X_train_tfidf = tf_idf_vect.transform(X_train)
print("the type of count vectorizer ",type(X_train_tfidf))
print("the shape of out text BOW vectorizer ",X_train_tfidf.get_shape
())
print("the number of unique words ", X_train_tfidf.get_shape()[1])

X_test_tfidf = tf_idf_vect.transform(X_test)
print("the type of count vectorizer ",type(X_test_tfidf))
print("the shape of out text BOW vectorizer ",X_test_tfidf.get_shape())
print("the number of unique words ", X_test_tfidf.get_shape()[1])
```

```
some feature names  ['abl', 'absolut', 'actual', 'ad', 'add', 'addict',
'addit', 'ago', 'allergi', 'alreadi']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (32000, 500)
the number of unique words  500
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (8000, 500)
the number of unique words  500
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler(with_mean=False)
scaler.fit(X_train_tfidf)
X_train_tfidf=scaler.transform(X_train_tfidf)

X_test_tfidf=scaler.transform(X_test_tfidf)
```

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier()
param_grid={'n_estimators' : [5, 10, 50, 100, 200, 500, 1000], 'max_dep
th' :  [2,3,4,5,6,7,8,9,10]}
gcv=RandomizedSearchCV(clf,param_grid,cv=3,scoring='roc_auc')
gcv.fit(X_train_tfidf,y_train)
print(gcv.best_params_)
print(gcv.best_score_)

optimal_depth           =  gcv.best_params_['max_depth']
optimal_estimators      =  gcv.best_params_['n_estimators']
```

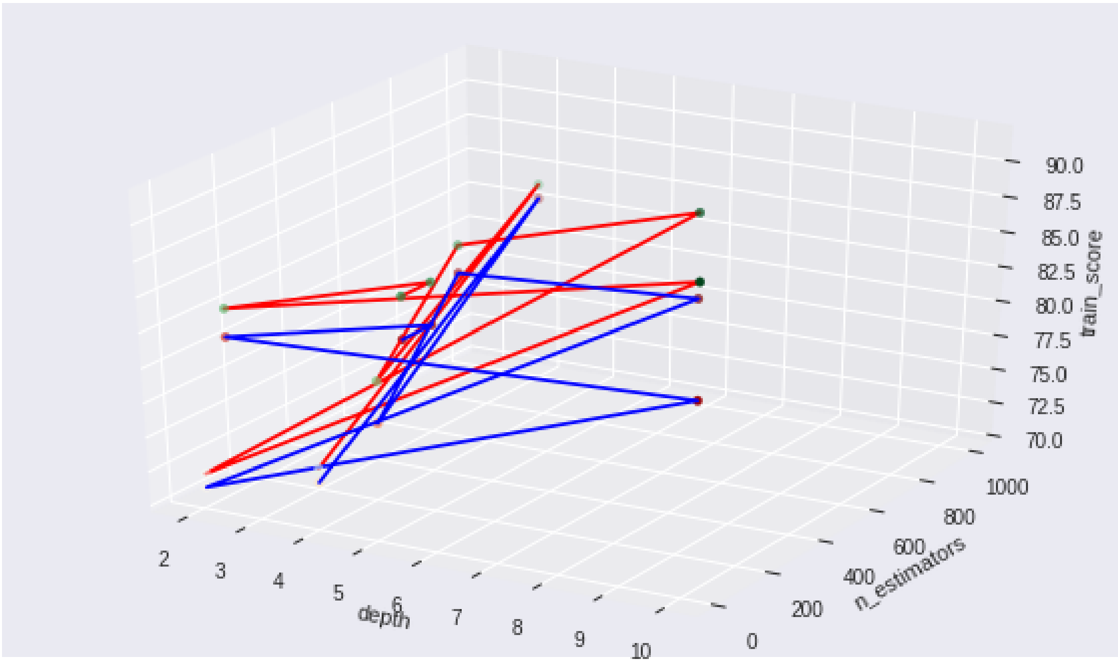
```
{'n_estimators': 500, 'max_depth': 8}
0.8380049265304902
```

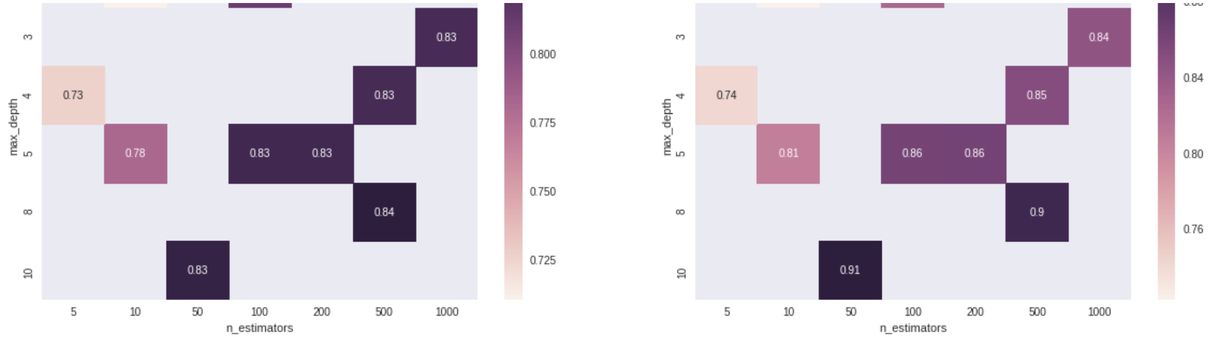
In [0]:
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")
```

```python
from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_score")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='Greens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score")
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrRd',label="test_score")



print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_score")
```

```
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

```
[4, 3, 5, 4, 8, 2, 10, 2, 5, 5]
[5, 1000, 10, 500, 500, 10, 50, 100, 200, 100]
[74.0, 84.0, 81.0, 85.0, 90.0, 72.0, 91.0, 83.0, 86.0, 86.0]
[73.0, 83.0, 78.0, 83.0, 84.0, 71.0, 83.0, 81.0, 83.0, 83.0]
ploting 3d grap
ploting Heat Map
```





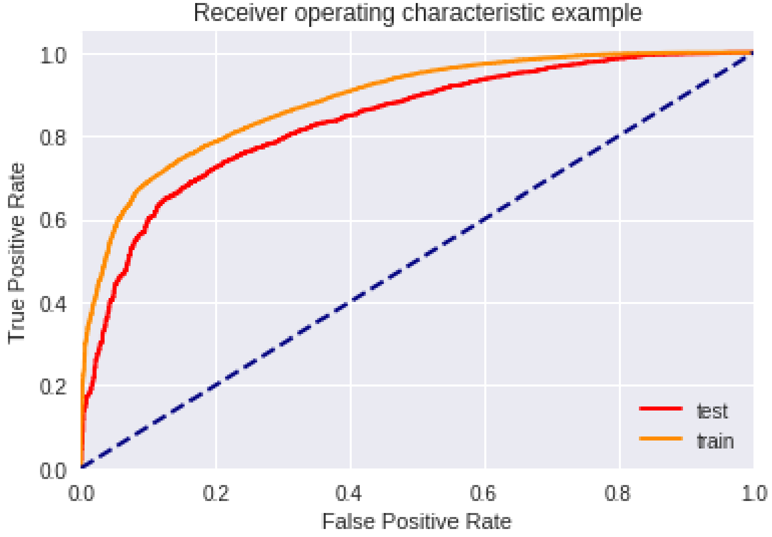
In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.ensemble import RandomForestClassifier

clf1=RandomForestClassifier(n_estimators=optimal_estimators,max_depth=o
ptimal_depth)
clf1.fit(X_train_tfidf,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_tfidf, y_train)

pred = sig_clf.predict_proba(X_test_tfidf)[:,1]
pred_train = sig_clf.predict_proba(X_train_tfidf)[:,1]


pred_train_without_CCV=clf1.predict(X_train_tfidf)
pred_without_CCV=clf1.predict(X_test_tfidf)


print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
```

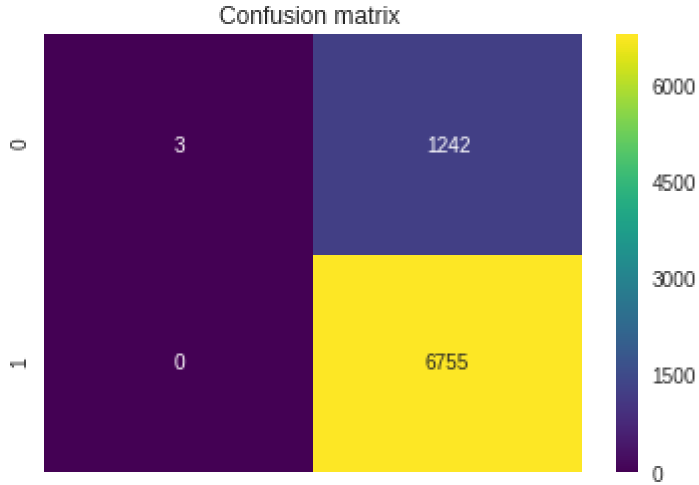
```
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("         ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("         ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("         ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("        ")

tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
```

```
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  84.475
Precision Score :  84.46917594097786
Recall Score :  100.0
F1 Score :  91.58080260303689

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      1245
           1       0.84      1.00      0.92      6755

   micro avg       0.84      0.84      0.84      8000
   macro avg       0.92      0.50      0.46      8000
weighted avg       0.87      0.84      0.77      8000


AUC Score for train data : 0.8839775262242905
AUC Score for test data : 0.8352311392126611
```




```
TrueNegative : 3
FalsePostive : 1242
FalseNegative : 0
TruePostive  :  6755
```



0 1

In [0]:

```python
#a=clf1.feature_importances_.tolist()
#a.sort()
#print(a[::-1][:20])

#np.sort(clf1.feature_importances_,)[::-1][:20]
-np.sort(-clf1.feature_importances_)[:20]


features=tf_idf_vect.get_feature_names()
imp=clf1.feature_importances_.tolist()

print([i for i in zip(features,imp)])

df=pd.DataFrame([features,imp],index=['feature','values'])
df=df.T
df1=df.sort_values('values',ascending=False)


print(len(imp))
len(features)
df1[:20]
```

```
[('add', 0.0036903995227348463), ('alway', 0.004458675068223361 5), ('am
azon', 0.010896183579977261), ('ani', 0.008210415758523602), ('bag', 0.
011480623241144334), ('becaus', 0.012798741074831 74), ('befor', 0.00812
6347047916848), ('best', 0.017997773316035234), ('better', 0.0088049551
68016194), ('bit', 0.005243049916200226), ('bottl', 0.00496495172035196
5), ('bought', 0.01193696001172303), ('box', 0.01475350401097236), ('br
and', 0.0083073330491009), ('buy', 0.017129248443937692), ('cat', 0.006
406308139799676), ('chocol', 0.0062173004384017795), ('coffe', 0.006433
342300903258), ('come', 0.005856093181462949), ('cup', 0.00377010804654
9947), ('day', 0.008837474596411 44), ('delici', 0.008717022932292246),
('did', 0.02062809882318903), ('didnt', 0.016910863563391554), ('diffe
r', 0.006914671201960943 5), ('doe', 0.007316823686131514), ('dog', 0.01
3281702886759646), ('dont', 0.016560344242231788), ('drink', 0.00507462
7530016528), ('eat', 0.011732615274007131), ('enjoy', 0.006856006982804
618), ('everi', 0.0040030856173251216), ('favorit', 0.00588916404103869
7), ('flavor', 0.01583667017165380 6), ('food', 0.01030890289880451 9),
```

```
7), ('flavor', 0.01583667017165380 6), ('food', 0.010308902898804519),
('fresh', 0.003859935250627891), ('good', 0.02130831234466467 8), ('go
t', 0.010054293007107092), ('great', 0.03195558358558468), ('help', 0.0
038678530153247056), ('high', 0.006761864591578539), ('hot', 0.00420144
0228836164), ('ingredi', 0.010824711291516305), ('ive', 0.0069602018786
74337), ('just', 0.017362347039457694), ('know', 0.008351731822018604),
('like', 0.022569287722526976), ('littl', 0.008351496289708627), ('loca
l', 0.005632462292204849), ('long', 0.004093849953874539), ('look', 0.0
13963021770144475), ('lot', 0.006507235537406034), ('love', 0.031736411
03717234), ('make', 0.012605953849243236), ('mani', 0.00602607493087795
7), ('milk', 0.003452784662287880 3), ('mix', 0.004916623108975595), ('m
onth', 0.006487339781865628), ('natur', 0.004715066853276 79), ('need',
0.006139524958993888), ('nice', 0.006013362998190329), ('oil', 0.002859
698304258652), ('old', 0.009795356511523471), ('onli', 0.01421404655297
3465), ('order', 0.01445230856587125), ('packag', 0.01056238331489929
6), ('perfect', 0.007596758502032159), ('price', 0.011080639586974807),
('product', 0.028758883233112265), ('purchas', 0.011114606460102974),
('qualiti', 0.005472177850773963), ('realli', 0.009821045297704651),
('recommend', 0.007917902587775989), ('review', 0.012690528603958633),
('sauc', 0.003703293220736522 7), ('say', 0.008980734050774257), ('shi
p', 0.009371233779228943), ('sinc', 0.005696524943159146), ('small', 0.
007390655560099957), ('smell', 0.009363723072503148), ('start', 0.00459
8649896622084), ('store', 0.007702541579583032), ('stuff', 0.0062106683
33465772), ('sugar', 0.005523172103813993), ('sweet', 0.004461847072667
189), ('tast', 0.027462712248380133), ('tea', 0.010417455492984451),
('thing', 0.008574345895371066), ('think', 0.010990880781693102), ('tim
e', 0.01199585980182042), ('treat', 0.006947504579220288), ('tri', 0.01
4939073576302033), ('use', 0.0176705799648254), ('veri', 0.018113694563
07616), ('want', 0.010283618185887022), ('water', 0.00738969760569723
9), ('way', 0.008256843738625829), ('wonder', 0.006100928668749064),
('work', 0.008002436456470047), ('year', 0.008445832169316497)]
100
```

Out[0]:

	feature	values
38	great	0.0319556

	feature	values
52	love	0.0317364

52	love	0.0317364
68	product	0.0287589
85	tast	0.0274627
46	like	0.0225693
36	good	0.0213083
22	did	0.0206281
93	veri	0.0181137
7	best	0.0179978
92	use	0.0176706
44	just	0.0173623
14	buy	0.0171292
23	didnt	0.0169109
27	dont	0.0165603
33	flavor	0.0158367
91	tri	0.0149391
12	box	0.0147535
64	order	0.0144523
63	onli	0.014214
50	look	0.013963

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

In [0]:

```
# Please write all the code with proper documentation
-np.sort(-clf1.feature_importances_)[:20]


features=tf_idf_vect.get_feature_names()
imp=clf1.feature_importances_.argsort()[::-1][:20]
```

```python
top20Features=[features[i] for i in imp]
top20Features
print(top20Features)

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

wordcloud = WordCloud(width = 400, height = 400, background_color ='black',  min_font_size = 10).generate(' '.join(top20Features))

# plot the WordCloud image
plt.figure(figsize = (4, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

```
['great', 'love', 'product', 'tast', 'like', 'good', 'did', 'veri', 'best', 'use', 'just', 'buy', 'didnt', 'dont', 'flavor', 'tri', 'box', 'order', 'onli', 'look']
```

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

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


#**********************************************************************
********************

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")

#**********************************************************************
********************

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])

#***********************************************************************
*********************



# average Word2Vec
# compute average word2vec for each review.
X_train_AvgW2V = []; # the avg-w2v for each sentence/review is stored i
n this list
for sent in tqdm(list_of_sentance): # 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
    X_train_AvgW2V.append(sent_vec)
print(len(X_train_AvgW2V))
print(len(X_train_AvgW2V[0]))
#***********************************************************************
***********************
```

```
  0%|          | 35/32000 [00:00<01:33, 343.50it/s]
```

```
[('wonder', 0.8224478363990784), ('excel', 0.8194332122802734), ('fanta
st', 0.7982593178749084), ('good', 0.769319474697113), ('awesom', 0.768
5434818267822), ('perfect', 0.7560844421386719), ('terrif', 0.715914368
6294556), ('amaz', 0.6943674087524414), ('decent', 0.6431294679641724),
('fabul', 0.6373891830444336)]
==================================================
[('best', 0.7055174112319946), ('closest', 0.6775582432746887), ('horri

bl', 0.658958911895752), ('disgust', 0.6368822455406189), ('tastiest',
```

```
0.6175901293754578), ('greatest', 0.6026514768600464), ('terribl', 0.59
06044244766235), ('aw', 0.5675726532936096), ('superior', 0.55436360836
02905), ('biggest', 0.5535221695899963)]
number of words that occured minimum 5 times  8343
sample words  ['this', 'product', 'better', 'than', 'ani', 'have', 'tr
i', 'the', 'pure', 'white', 'powder', 'and', 'doe', 'not', 'filler', 'b
est', 'valu', 'wasabi', 'pea', 'out', 'there', 'bag', 'repres', 'lot',
'but', 'leav', 'work', 'theyll', 'soon', 'disappear', 'tasti', 'first',
'had', 'cooki', 'airlin', 'kept', 'wrapper', 'been', 'long', 'time', 's
inc', 'enjoy', 'such', 'delight', 'then', 'bought', 'case', 'give', 'fr
iend', 'whenev']
```

```
100%|██████████| 32000/32000 [01:31<00:00, 349.25it/s]
```

```
32000
50
```

In [0]:

```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

#**********************************************************************
*********************



# average Word2Vec
# compute average word2vec for each review.
X_test_AvgW2V = []; # the avg-w2v for each sentence/review is stored 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
    X_test_AvgW2V.append(sent_vec)
print(len(X_test_AvgW2V))
print(len(X_test_AvgW2V[0]))
```

```
100%|██████████| 8000/8000 [00:23<00:00, 340.84it/s]
```

```
8000
50
```

In [0]:
```
len(X_train_AvgW2V),len(X_test_AvgW2V)
```

Out[0]:
```
(32000, 8000)
```

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
from  sklearn.model_selection import GridSearchCV
clf=RandomForestClassifier()
param_grid={'n_estimators' :  [5, 10, 50, 100, 200, 500, 1000], 'max_depth' :  [2,3,4,5,6,7,8,9,10]}
gcv=GridSearchCV(clf,param_grid,cv=5,scoring='roc_auc')
gcv.fit(X_train_AvgW2V,y_train)
print(gcv.best_params_)
print(gcv.best_score_)

optimal_depth          =  gcv.best_params_['max_depth']
optimal_estimators     =  gcv.best_params_['n_estimators']
```

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
from  sklearn.model_selection import GridSearchCV
clf=RandomForestClassifier()
param_grid={'n_estimators' :  [5, 10, 50, 100, 200, 500, 1000], 'max_depth' :  [2,3,4,5,6,7,8,9,10]}
gcv=GridSearchCV(clf,param_grid,cv=5,scoring='roc_auc')
gcv.fit(X_train_AvgW2V,y_train)
print(gcv.best_params_)
print(gcv.best_score_)
```

```
optimal_depth           =  gcv.best_params_['max_depth']
optimal_estimators      =  gcv.best_params_['n_estimators']
```

```
{'max_depth': 10, 'n_estimators': 100}
0.8668456827341905
```

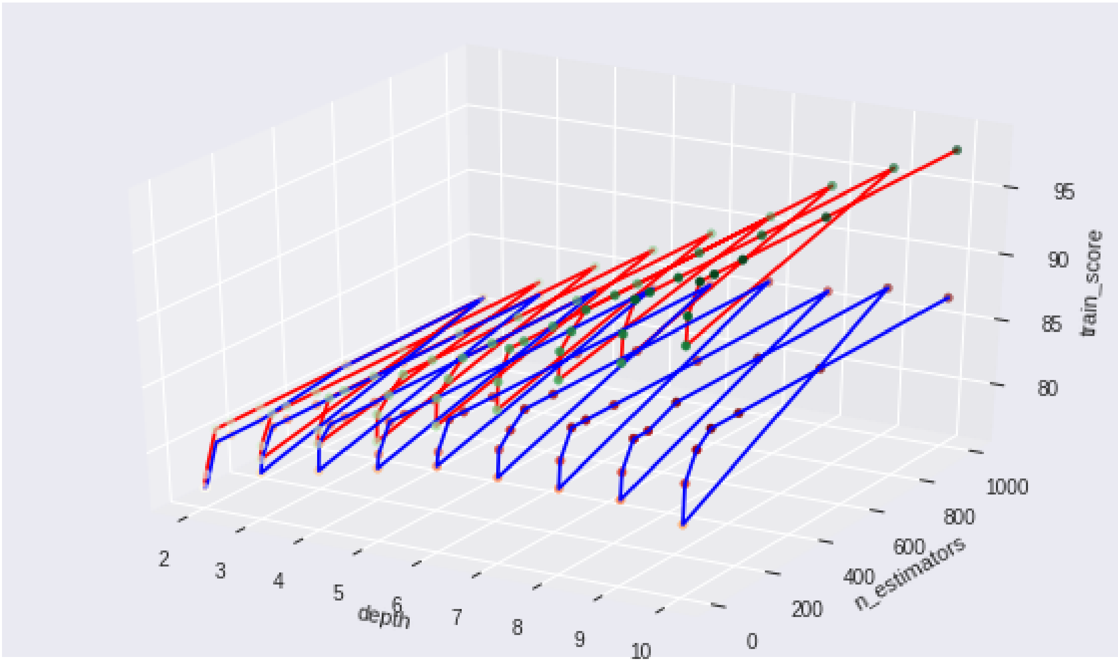
In [0]:

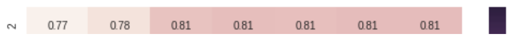
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth          = [i[0] for i in hyperparameters]
n_estimators   = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
```

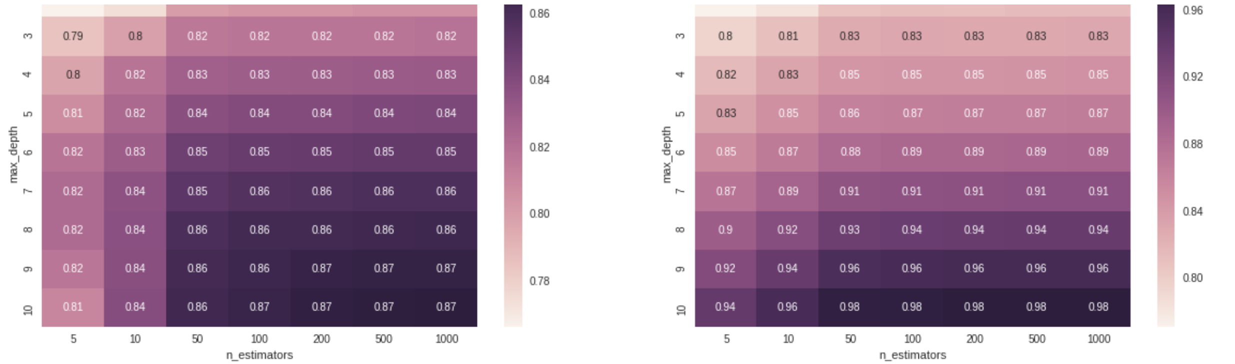
```
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='Greens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score")
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrRd',label="test_score")


print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_score")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_score")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

```
[2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5, 5,
5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8,
8, 8, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10]
[5, 10, 50, 100, 200, 500, 1000, 5, 10, 50, 100, 200, 500, 1000, 5, 10,
50, 100, 200, 500, 1000, 5, 10, 50, 100, 200, 500, 1000, 5, 10, 50, 10
```

```
50, 100, 200, 500, 1000, 5, 10, 50, 100, 200, 500, 1000, 5, 10, 50, 10
0, 200, 500, 1000, 5, 10, 50, 100, 200, 500, 1000, 5, 10, 50, 100, 200,
500, 1000, 5, 10, 50, 100, 200, 500, 1000, 5, 10, 50, 100, 200, 500, 10
00]
[77.0, 78.0, 81.0, 81.0, 81.0, 81.0, 81.0, 80.0, 81.0, 83.0, 83.0, 83.
0, 83.0, 83.0, 82.0, 83.0, 85.0, 85.0, 85.0, 85.0, 85.0, 83.0, 85.0, 8
6.0, 87.0, 87.0, 87.0, 87.0, 85.0, 87.0, 88.0, 89.0, 89.0, 89.0, 89.0,
87.0, 89.0, 91.0, 91.0, 91.0, 91.0, 91.0, 90.0, 92.0, 93.0, 94.0, 94.0,
94.0, 94.0, 92.0, 94.0, 96.0, 96.0, 96.0, 96.0, 96.0, 94.0, 96.0, 98.0,
98.0, 98.0, 98.0, 98.0]
[77.0, 77.0, 80.0, 80.0, 80.0, 81.0, 81.0, 79.0, 80.0, 82.0, 82.0, 82.
0, 82.0, 82.0, 80.0, 82.0, 83.0, 83.0, 83.0, 83.0, 83.0, 81.0, 82.0, 8
4.0, 84.0, 84.0, 84.0, 84.0, 82.0, 83.0, 85.0, 85.0, 85.0, 85.0, 85.0,
82.0, 84.0, 85.0, 86.0, 86.0, 86.0, 86.0, 82.0, 84.0, 86.0, 86.0, 86.0,
86.0, 86.0, 82.0, 84.0, 86.0, 86.0, 87.0, 87.0, 87.0, 81.0, 84.0, 86.0,
87.0, 87.0, 87.0, 87.0]
ploting 3d grap
ploting Heat Map
```







In [0]:

```
df3
```

Out[0]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth	paran
0	8.195508	0.064418	0.846148	0.872823	5	
1	13.915090	0.096795	0.867483	0.979746	10	
2	18.084755	0.133426	0.865912	1.000000	50	
3	18.160068	0.131030	0.866421	1.000000	100	

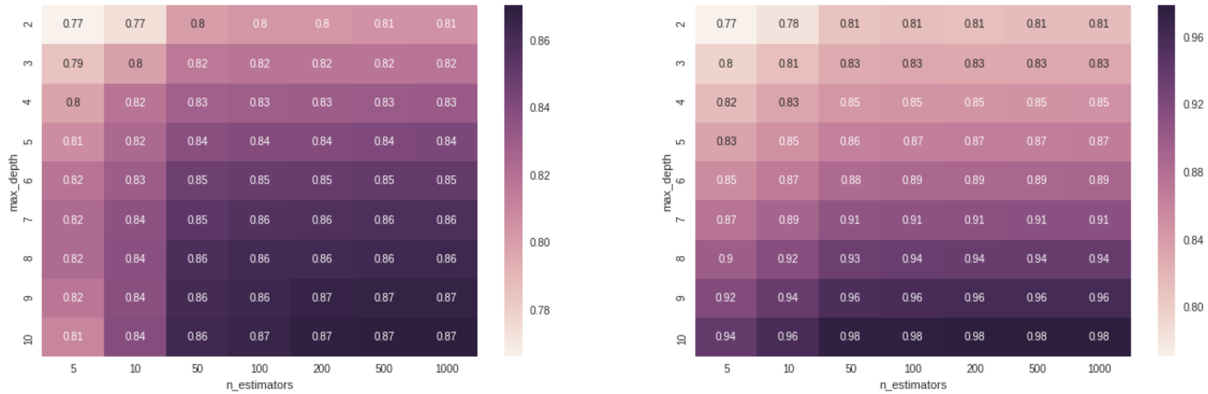
4 rows × 23 columns

In [0]:

```
print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)
```

```
final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_score")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_score")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

ploting Heat Map



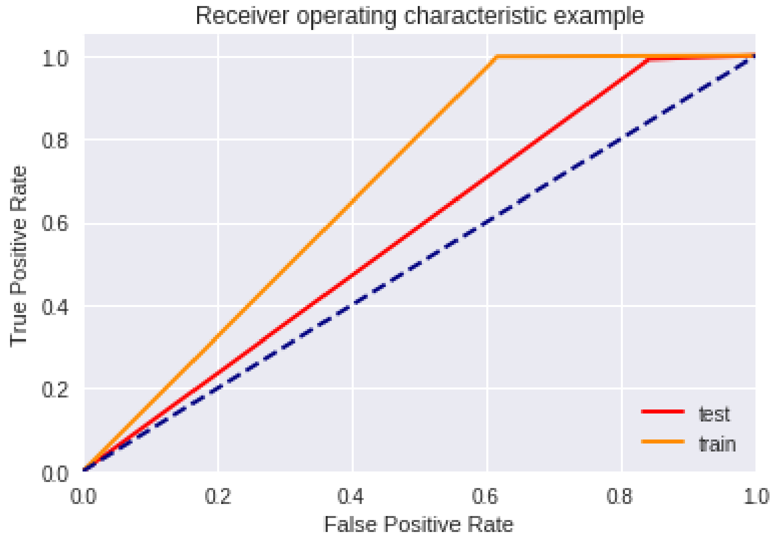
In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.ensemble import RandomForestClassifier

clf1=RandomForestClassifier(n_estimators=optimal_estimators,max_depth=o
```

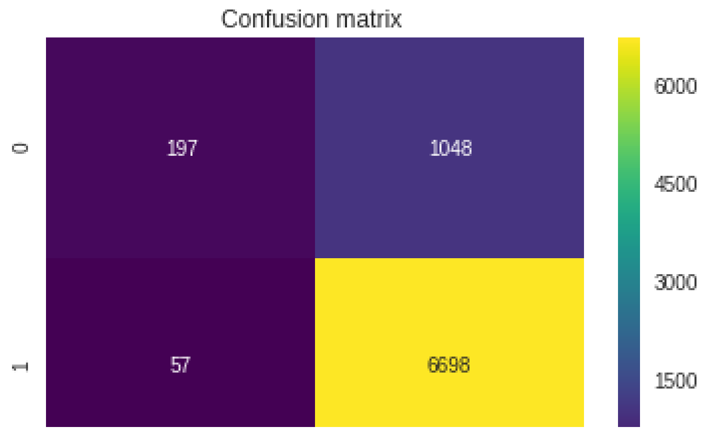
```
ptimal_depth)
clf1.fit(X_train_AvgW2V,y_train)
pred_train=clf1.predict(X_train_AvgW2V)
pred=clf1.predict(X_test_AvgW2V)

print("Accuracy Score : ",accuracy_score(y_test,pred)*100)
print("Precision Score : ",precision_score(y_test,pred)*100)
print("Recall Score : ",recall_score(y_test,pred)*100)
print("F1 Score : ",f1_score(y_test,pred)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test,pred))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
print("       ")

tn, fp, fn, tp=confusion_matrix(y_test,pred).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred),columns=[
'0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  86.1875
Precision Score :  86.47043635424735
Recall Score :  99.1561806069578
F1 Score :  92.37983587338803

Classification Report
              precision    recall  f1-score   support

           0       0.78      0.16      0.26      1245
           1       0.86      0.99      0.92      6755

   micro avg       0.86      0.86      0.86      8000
   macro avg       0.82      0.57      0.59      8000
weighted avg       0.85      0.86      0.82      8000


AUC Score for train data : 0.690836067671144
AUC Score for test data : 0.5748973688982428
```




```
TrueNegative : 197
FalsePostive : 1048
FalseNegative : 57
TruePostive  :  6698
```



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

In [0]:
```
# 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_)))

#***********************************************************************
**********************

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

# 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

X_train_Avgtfidf = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in list_of_sentance: # 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
    X_train_Avgtfidf.append(sent_vec)
    row += 1


#***********************************************************************
***********************

X_test_Avgtfidf = []; # the tfidf-w2v for each sentence/review is store
d in this list
row=0;
for sent in 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
    X_test_Avgtfidf.append(sent_vec)
    row += 1
```

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
clf=RandomForestClassifier()
param_grid={'n_estimators' : np.arange(100,500) , 'max_depth' :  [5, 10
, 50, 100]}
gcv=RandomizedSearchCV(clf,param_grid,cv=10,scoring='roc_auc')
gcv.fit(X_train_Avgtfidf,y_train)
print(gcv.best_params_)
print(gcv.best_score_)

optimal_depth          =  gcv.best_params_['max_depth']
optimal_estimators     =  gcv.best_params_['n_estimators']
```

```
{'n_estimators': 455, 'max_depth': 50}
0.8466345074231709
```

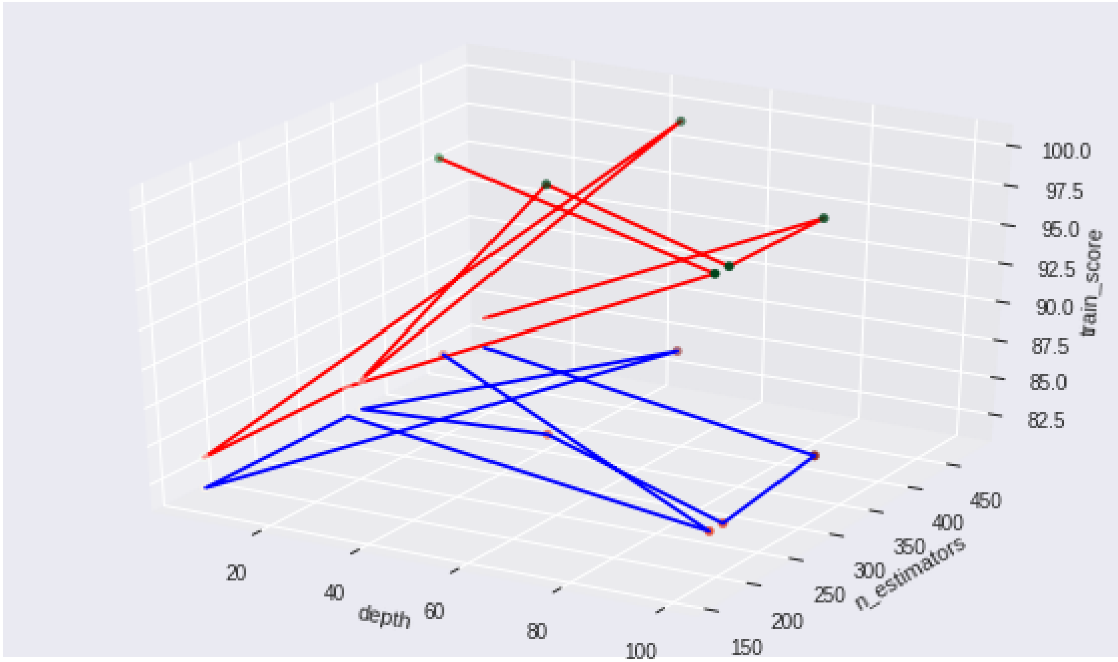
In [0]:

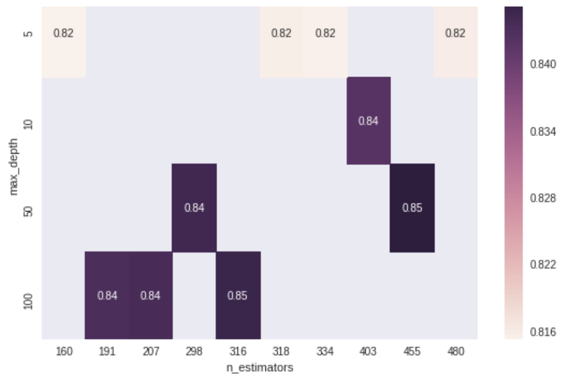
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
```

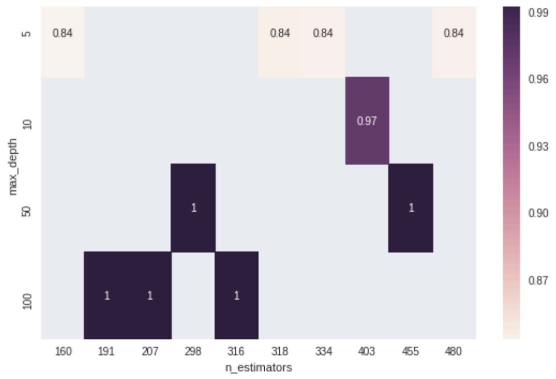
```
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='G
reens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score"
)
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrR
d',label="test_score")


print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_scor
e")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)
```

```
plt.show()
#fig.show()
```

```
[10, 100, 5, 5, 50, 5, 50, 100, 100, 5]
[403, 191, 318, 160, 455, 334, 298, 207, 316, 480]
[97.0, 100.0, 84.0, 84.0, 100.0, 84.0, 100.0, 100.0, 100.0, 84.0]
[84.0, 84.0, 82.0, 82.0, 85.0, 82.0, 84.0, 84.0, 85.0, 82.0]
ploting 3d grap
ploting Heat Map
```







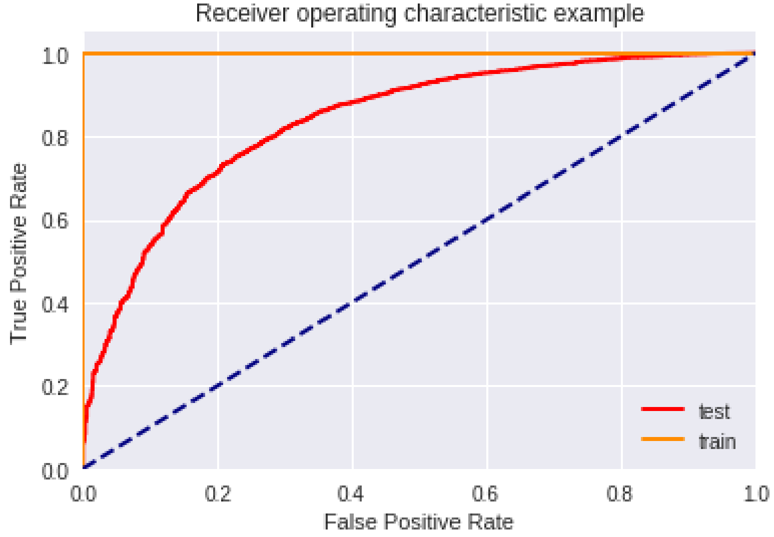
In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.ensemble import RandomForestClassifier

clf1=RandomForestClassifier(n_estimators=455,max_depth=50)
clf1.fit(X_train_Avgtfidf,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_Avgtfidf, y_train)

pred = sig_clf.predict_proba(X_test_Avgtfidf)[:,1]
pred_train = sig_clf.predict_proba(X_train_Avgtfidf)[:,1]


pred_train_without_CCV=clf1.predict(X_train_Avgtfidf)
pred_without_CCV=clf1.predict(X_test_Avgtfidf)


print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
```

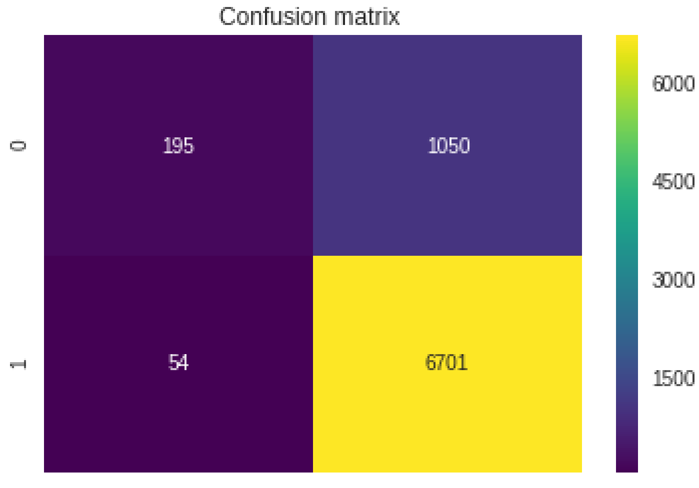
```
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")

tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
```

```
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  86.2
Precision Score :  86.45336085666365
Recall Score :  99.20059215396003
F1 Score :  92.38935612849856

Classification Report
              precision    recall  f1-score   support

           0       0.78      0.16      0.26      1245
           1       0.86      0.99      0.92      6755

   micro avg       0.86      0.86      0.86      8000
   macro avg       0.82      0.57      0.59      8000
weighted avg       0.85      0.86      0.82      8000


AUC Score for train data : 1.0
AUC Score for test data : 0.8386619460818849
```




```
TrueNegative : 195
FalsePostive : 1050
FalseNegative : 54
TruePostive  :  6701
```



0 1

[5.2] Applying GBDT using XGBOOST

In [0]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split

preprocessed_reviews_xg=final['CleanedText'][:100000]
score_xg=final['Score'][:100000]
X_train_xg, X_test_xg, y_train_xg, y_test_xg = train_test_split(preproc
essed_reviews_xg, score_xg, test_size=0.2, random_state=42)
```

In [0]:
```
#BoW
count_vect = CountVectorizer(max_df=0.95, min_df=2,stop_words='english'
,max_features=1000) #in scikit-learn
count_vect.fit(X_train_xg)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

X_train_bow_xg = count_vect.transform(X_train_xg)
print("the type of count vectorizer ",type(X_train_bow_xg))
print("the shape of out text BOW vectorizer ",X_train_bow_xg.get_shape
())
print("the number of unique words ", X_train_bow_xg.get_shape()[1])

X_test_bow_xg = count_vect.transform(X_test_xg)
print("the type of count vectorizer ",type(X_test_bow_xg))
print("the shape of out text BOW vectorizer ",X_test_bow_xg.get_shape
())
print("the number of unique words ", X_test_bow_xg.get_shape()[1])
```

```
some feature names  ['abl', 'abov', 'absolut', 'acid', 'actual', 'ad',
'add', 'addict', 'addit', 'admit']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text BOW vectorizer  (80000, 1000)
the number of unique words  1000
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (20000, 1000)
the number of unique words  1000
```

[5.2.1] Applying XGBOOST on BOW, SET 1

In [0]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split

preprocessed_reviews=final['CleanedText'][:40000]
score=final['Score'][:40000]
X_train, X_test, y_train, y_test = train_test_split(preprocessed_review
s, score, test_size=0.2, random_state=42)
```

In [0]:
```
#BoW
count_vect = CountVectorizer(max_df=0.95, min_df=2,stop_words='english'
,max_features=1000) #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

X_train_bow = count_vect.transform(X_train)
print("the type of count vectorizer ",type(X_train_bow))
print("the shape of out text BOW vectorizer ",X_train_bow.get_shape())
print("the number of unique words ", X_train_bow.get_shape()[1])

X_test_bow = count_vect.transform(X_test)
print("the type of count vectorizer ",type(X_test_bow))
print("the shape of out text BOW vectorizer ",X_test_bow.get_shape())
print("the number of unique words ", X_test_bow.get_shape()[1])
```

```
some feature names  ['abl', 'abov', 'absolut', 'acid', 'activ', 'actua
l', 'ad', 'add', 'addict', 'addit']
```

```
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (32000, 1000)
the number of unique words  1000
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (8000, 1000)
the number of unique words  1000
```

In [0]:
```
from sklearn.ensemble import  AdaBoostClassifier, GradientBoostingClass
ifier
clf=GradientBoostingClassifier()
#param_grid={ 'n_estimators' : np.arange(20,200,20),
#              'max_depth' :  [5,10,15,20,30],
#              'learning_rate' : [0.1,0.001] }
param_grid={ 'n_estimators' : [50,100,150,200],
             'max_depth' :  [5,10,20]}
gcv=RandomizedSearchCV(clf,param_grid,cv=5)
gcv.fit(X_train_bow,y_train)
print(gcv.best_params_)
print(gcv.best_score_)
```

```
{'n_estimators': 200, 'max_depth': 10}
0.88846875
```

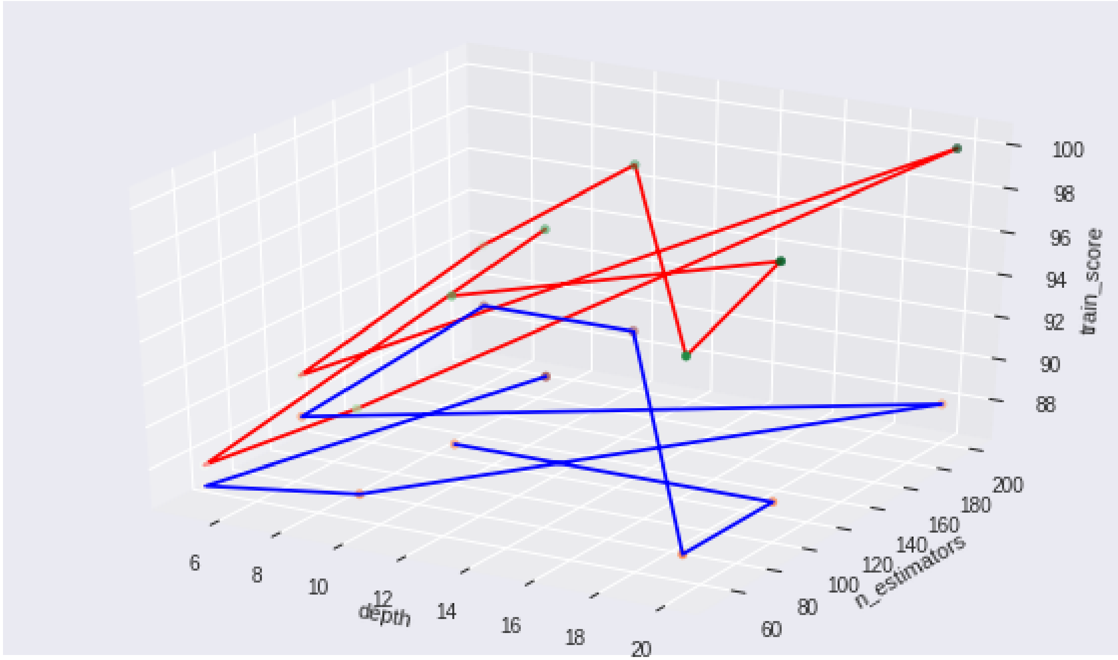
In [0]:
```
```

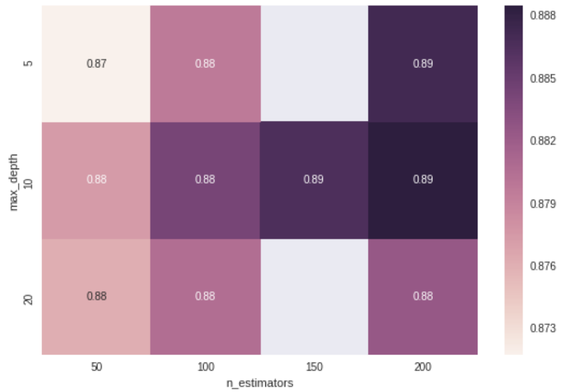
In [0]:
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))
```

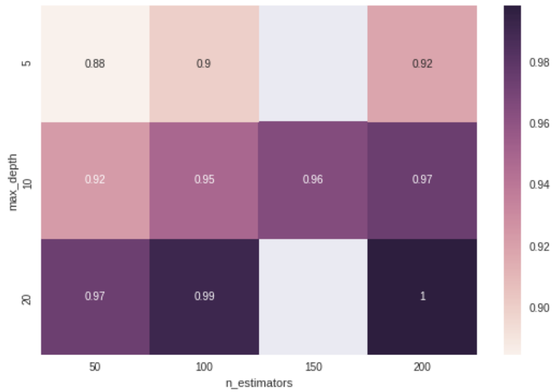
```
print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='G
reens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score"
)
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrR
d',label="test_score")


print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)
```

```
df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_scor
e")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

```
[10, 20, 20, 10, 5, 5, 20, 10, 5, 10]
[100, 100, 50, 200, 200, 100, 200, 50, 50, 150]
[95.0, 99.0, 97.0, 97.0, 92.0, 90.0, 100.0, 92.0, 88.0, 96.0]
[88.0, 88.0, 88.0, 89.0, 89.0, 88.0, 88.0, 88.0, 87.0, 89.0]
ploting 3d grap
ploting Heat Map
```







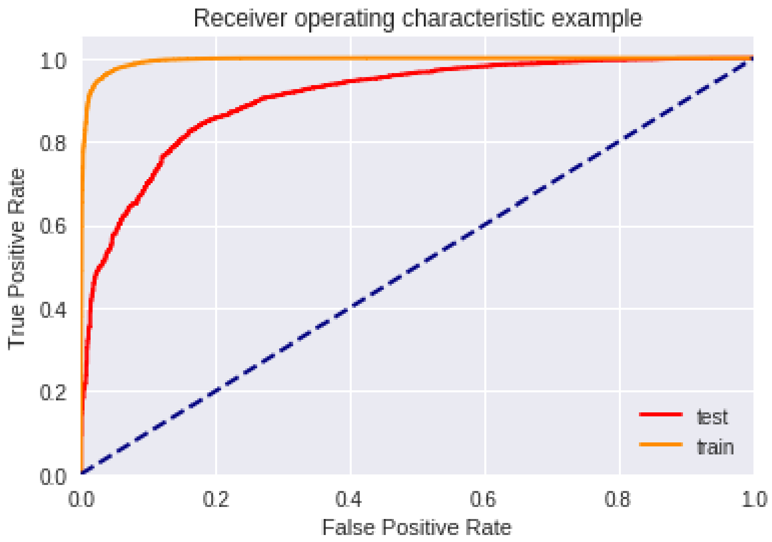
In [0]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score




clf1=GradientBoostingClassifier(n_estimators=200,max_depth=10)
clf1.fit(X_train_bow,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_bow, y_train)

pred = sig_clf.predict_proba(X_test_bow)[:,1]
pred_train = sig_clf.predict_proba(X_train_bow)[:,1]


pred_train_without_CCV=clf1.predict(X_train_bow)
pred_without_CCV=clf1.predict(X_test_bow)


print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("         ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("         ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("         ")
```

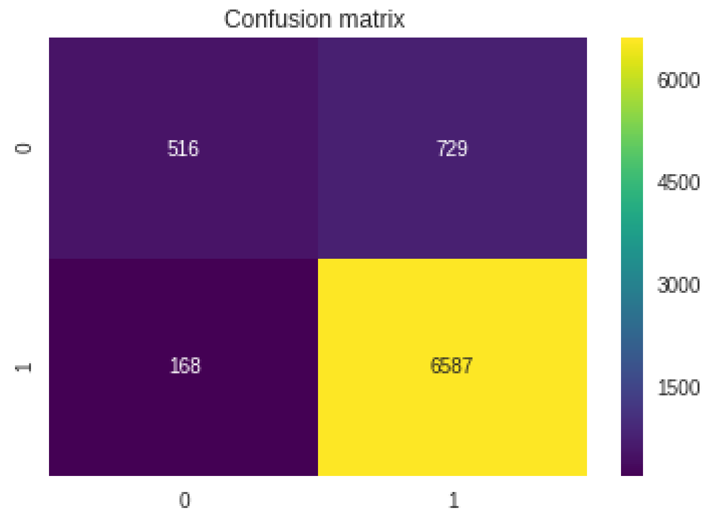
```
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")

tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  88.7875
Precision Score :  90.03553854565337
Recall Score :  97.51295336787564
F1 Score :  93.6251865539052
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.75      0.41      0.53      1245
           1       0.90      0.98      0.94      6755

   micro avg       0.89      0.89      0.89      8000
   macro avg       0.83      0.69      0.74      8000
weighted avg       0.88      0.89      0.87      8000


AUC Score for train data : 0.9942886344329558
AUC Score for test data : 0.9056362236510809
```




```
TrueNegative : 516
FalsePostive : 729
FalseNegative : 168
TruePostive  :  6587
```

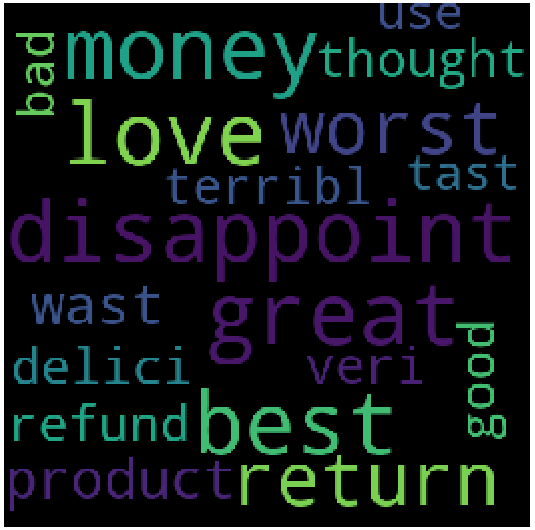


In [0]:

```
-np.sort(-clf1.feature_importances_)[:20]


features=count_vect.get_feature_names()
imp=clf1.feature_importances_.argsort()[::-1][:20]
top20Features=[features[i] for i in imp]
top20Features
print(top20Features)

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

wordcloud = WordCloud(width = 400, height = 400, background_color ='bla
ck',  min_font_size = 10).generate(' '.join(top20Features))

# plot the WordCloud image
plt.figure(figsize = (4, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
```

```
plt.show()
```

```
['disappoint', 'money', 'best', 'great', 'love', 'return', 'worst', 'pr
oduct', 'wast', 'thought', 'refund', 'delici', 'good', 'terribl', 'lik
e', 'veri', 'bad', 'tast', 'use', 'did']
```



In [0]:
```
#a=clf1.feature_importances_.tolist()
#a.sort()
#print(a[::-1][:20])

#np.sort(clf1.feature_importances_,)[::-1][:20]
-np.sort(-clf1.feature_importances_)[:50]


features=count_vect.get_feature_names()
imp=clf1.feature_importances_.tolist()

print([i for i in zip(features,imp)])

df=pd.DataFrame([features,imp],index=['feature','values'])
```

```
df=df.T
df1=df.sort_values('values',ascending=False)


print(len(imp))
len(features)
df1[:20]
```

```
[('abl', 0.0010486297363098372), ('abov', 0.00030587842799058236), ('ab
solut', 0.00044030639751752263), ('acid', 0.00035593987203346447), ('acti
v', 0.00015431996396337693), ('actual', 0.0016541530301405862), ('ad',
0.00042237883307507723), ('add', 0.0011137720286859745), ('addict', 0.00
04915089094309154), ('addit', 0.00019541243394682009), ('adult', 0.00010
192457008860951), ('advertis', 0.00045221406748092443), ('aftertast',
0.0012525070616734887), ('age', 0.00036939805953659565), ('ago', 0.0011
37197840829147), ('agre', 0.00066253169811159775), ('air', 0.00067348833
44715388), ('allerg', 0.00042050232334150016), ('allergi', 0.00029281220
191165416), ('allow', 0.00020085960555696388), ('almond', 0.00061427750
12729039), ('alon', 0.00026755008027701206), ('alot', 0.000126742801400
65433), ('alreadi', 0.0003720300664565642), ('altern', 0.00073627281301
47893), ('alway', 0.0027544712432930594), ('amaz', 0.0014378195403858882
9), ('amazon', 0.0022507981611771806), ('amazoncom', 0.000394068976219З
5635), ('american', 0.00036088093136731595), ('ani', 0.0025298497747373
206), ('anim', 0.00041778881726774246), ('anni', 0.000161183844868586З
8), ('anoth', 0.0018036970959743614), ('anymor', 0.000285838159318678l
6), ('anyon', 0.0007458130810603113), ('anyth', 0.0011740478238357593),
('anywher', 0.00027018674698184753), ('apart', 0.0008179051350318614),
('appar', 0.00047971117284335522), ('appear', 0.0007143084391957595),
('appl', 0.0001693875788148367), ('appreci', 0.00031324527603342027),
('area', 0.00024244750994995557), ('arent', 0.0003058669881035665), ('aro
ma', 0.0004902293544411684), ('arriv', 0.0012105959148820684), ('artifi
ci', 0.0007953548279805123), ('asian', 7.869873657914377e-05), ('ask',
0.0005047275238361681), ('ate', 0.0011143100074358433), ('authent', 7.07
8687149415227e-05), ('avail', 0.00047256457137877516), ('avoid', 0.0012
393632061181287), ('aw', 0.004848248123621178), ('away', 0.002274560298
722442), ('awesom', 0.00032649070222588494), ('babi', 0.00067792829362О
2768), ('bad', 0.006687196354182589), ('bag', 0.0018390120178170312),
('bake', 5.2989523223305le-05), ('balanc', 0.00019394894577430565), ('b
all', 0.0015553144041529027), ('bar', 0.00165548922542495), ('barri',
0.00020726537707758О2), ('base', 0.0006031863426910795), ('basic', 0.00
```

03649549306193599), ('basket', 0.00042561055550682244), ('batch', 0.001
415494390286974), ('bean', 0.0014978273623054818), ('beat', 0.000346122
50339446894), ('beauti', 0.0005854999941345839), ('becam', 0.0002651079
1986523075), ('becaus', 0.0020650495677777235), ('becom', 0.00035139617
526280456), ('bed', 0.0002073727025811255), ('beef', 0.0007228269906399
242), ('befor', 0.0013491470915974507), ('believ', 0.000895188365165992
3), ('benefit', 9.427048181520455e-05), ('berri', 6.424726544420497e-0
5), ('best', 0.019202014088210713), ('better', 0.0018919843064551648),
('beverag', 0.00011674832916153732), ('big', 0.0013730568621581915),
('bigger', 0.000280714429460123), ('birthday', 0.00020051675398775808),
('biscuit', 0.00015199751429429963), ('bit', 0.0008956887170082397),
('bite', 0.0005575688700786149), ('bitter', 0.001422229094710125), ('bl
ack', 0.0003764803662782954), ('bland', 0.002134952195676178), ('blen
d', 0.0004885038555879714), ('blood', 0.0006160183025060237), ('blue',
0.0007014739954642677), ('bodi', 0.00016630025348382912), ('boil', 0.00
02314535108915906), ('bone', 0.0008852690236696611), ('bonsai', 0.00048
84656306360094), ('book', 0.0006200808366899124), ('bother', 0.00086506
74040323377), ('bottl', 0.001974614538210583), ('bought', 0.00224919702
61414875), ('bowl', 0.000746568461343401), ('box', 0.002369458559737206
3), ('boy', 8.805747752444853e-05), ('brand', 0.0020963588893977148),
('bread', 0.0003525300136509046), ('break', 0.0003645463479840131), ('b
reakfast', 0.0002508905955303643), ('breast', 0.00011378641058879342),
('breath', 0.0004063110976939095), ('brew', 0.0003901299896013704З),
('bring', 0.0002753598385800421З), ('broken', 0.0009950729088322595),
('brought', 0.00020107570844867765), ('brown', 0.0004286051444262719),
('bubbl', 0.0001201637149168841б), ('bulk', 0.000426909281228404), ('bu
nch', 0.0004793546697201236З), ('burn', 0.0006881412915814816), ('bus
i', 0.0009471090516364292), ('butter', 0.0007525074224189323), ('buy',
0.00400776695708482), ('caffein', 0.0007833599021135458), ('cake', 0.00
06756073495190003), ('calm', 4.378457121343979e-05), ('calori', 0.00088
38167125765638), ('came', 0.0014671855873017302), ('candi', 0.001480151
1274979316), ('canida', 5.8485423597284514e-05), ('carb', 0.00038549391
80936333), ('care', 0.001113873681004598), ('carri', 0.0003824640696230
084), ('case', 0.0009186329965858216), ('cat', 0.002545118359611674),
('catnip', 0.0006634266540074753), ('caught', 0.0004443585691741141),
('caus', 0.0020245881983001414), ('cereal', 0.00032066375001939615),
('certain', 0.00033560709770971844), ('chai', 0.00034330980829595434),
('chanc', 0.0003003441667963016), ('chang', 0.002077175783312272), ('ch
arg', 0.0008457689853700322), ('cheap', 0.0013406610216329807), ('cheap

er', 0.001149617474039386), ('check', 0.0010021052017399749), ('chedda
r', 0.00012205757031623686), ('chees', 0.0008676281137950659), ('chemi
c', 0.001565671522082633), ('cherri', 0.00014986532972792354), ('chew',
0.0013788399251452167), ('chewer', 0.00031925646827679025), ('chewi',
0.00023745474404179224), ('chicken', 0.000759579884728704), ('child',
6.949909915630794e-05), ('children', 0.0006899311143457313), ('chili',
0.00023198968171435876), ('china', 0.0023262068204402847), ('chines',
0.0007138642683886032), ('chip', 0.0006841200177244869), ('chocol', 0.0
02595216422402262), ('choic', 0.000149263343592151), ('christma', 0.001
0468466126729364), ('chunk', 0.0005087323008875637), ('cinnamon', 0.000
7435363307381163), ('claim', 0.0008848167675733499), ('clean', 0.000499
0824042318247), ('clear', 0.000259900554748042), ('close', 0.0005345626
312230352), ('clump', 0.000340161843325575), ('coat', 0.000270989849992
5759), ('cocoa', 0.0009095066067586343), ('coconut', 0.0016356315355188
298), ('coffe', 0.00257663371967161), ('cold', 0.00034299048398106035),
('color', 0.0007574650969767427), ('combin', 9.886691726742694e-05),
('come', 0.0010736571919529824), ('comment', 7.338982052031456e-05),
('compani', 0.0029929388702753567), ('compar', 0.0010695542590586685),
('complaint', 0.00011518938868401359), ('complet', 0.000523615579439779
7), ('concentr', 0.0005961827036785858), ('concern', 0.0003372282790931
8706), ('condit', 0.00014760933804229586), ('consid', 0.000189221824471
6322), ('consist', 0.0005982359397025477), ('constant', 0.0003417754795
0853), ('consum', 0.0006989362625557505), ('contact', 0.000499520373097
9255), ('contain', 0.001037128400622934), ('content', 0.000874459557627
3867), ('continu', 0.00019832615388539629), ('control', 0.0002071425740
580032), ('conveni', 0.00023115315025947086), ('cook', 0.00063057147878
62512), ('cooki', 0.0013243590900622554), ('cool', 0.000298382374197734
57), ('corn', 0.0008653535673897315), ('cost', 0.0008490422893233918),
('countri', 0.00014209357142112253), ('coupl', 0.0004202462983871198),
('cours', 0.00038090984378762443), ('cover', 0.0005260045607260978),
('cracker', 0.00036133995805683603), ('crave', 0.00017478991308737368),
('crazi', 0.00029697622886171605), ('cream', 0.00030981624175111867),
('creami', 0.0001209632060998906), ('creat', 0.0004554468702020257),
('crunch', 0.0003705035226160306), ('crunchi', 0.0004563712571005181),
('cube', 0.00018242001180503322), ('cup', 0.0009073410768615657), ('cur
rent', 0.00010287875249851584), ('custom', 0.0006869502701058645), ('cu
t', 0.00035093191413585227), ('daili', 0.00033572275150301975), ('dar
k', 0.0002423291401039231), ('date', 0.0013403765244019677), ('daughte
r', 0.00014438986113294067), ('day', 0.00399230342447302), ('dead', 0.0

```
006855824016554523), ('deal', 0.0011082763072143462), ('decaf', 0.00010
862351411288272), ('decent', 0.0009122061882821996), ('decid', 0.000935
942804182475), ('deep', 4.9458076499863213e-05), ('definit', 0.00158198
6238212116), ('delic', 5.6632809721017554e-05), ('delici', 0.0081649350
40770815), ('delight', 0.0003436075747245827 6), ('deliv', 0.00067600142
05557161), ('deliveri', 0.0002768860130094715), ('depend', 0.0001174567
9541159303), ('describ', 0.000689001347013766), ('descript', 0.00450001
7778732406), ('dessert', 0.00014142351006527065), ('develop', 0.0003726
6517707124167), ('diabet', 0.0002337989614928931 6), ('did', 0.006271006
320149338), ('didnt', 0.004051439278835143), ('die', 0.0005929390510233
814), ('diet', 0.0009323582783140579), ('differ', 0.001882874597177319
4), ('difficult', 0.0006175551422250209), ('digest', 0.0001097912880110
6397), ('dinner', 0.000180262492576777), ('dip', 0.0001508593118380638
2), ('direct', 0.0011482238750740443), ('disappoint', 0.036432969107522
396), ('discov', 0.00017150657755845604), ('dish', 0.00022038046416288 4
67), ('doctor', 0.00021111983390436603), ('doe', 0.001158031481412491
4), ('doesnt', 0.0012278704155467407), ('dog', 0.004711040346402371),
('dollar', 0.0011436038815470014), ('dont', 0.003559668505273 08), ('doo
r', 0.00019152391521168213), ('doubl', 0.0008031801248792612), ('dran
k', 0.00016447150429137853), ('dress', 0.00022447639735765807), ('dri',
0.00155170598951514), ('drink', 0.0010134778783147949), ('drinker', 0.0
0018764334366561211), ('drop', 0.00021653761951465718), ('dure', 0.0004
054289717243426), ('ear', 0.0004182386473130766), ('earl', 0.0002894798
2354534746), ('earli', 0.00018455909298004315), ('easi', 0.001118888644
6197028), ('easier', 0.0004634308875468445), ('easili', 0.0004536789106
7102384), ('eat', 0.0030600870216495727), ('eaten', 0.00058881834421789
52), ('eater', 0.0002641016344962995), ('effect', 0.000539026698988224
7), ('egg', 0.00023343341983702302), ('els', 0.0005695725351712 27), ('e
mail', 0.0016585325699732905), ('empti', 9.43608201329602e-05), ('end',
0.0013654918601170994), ('energi', 0.00016591612720788957), ('english',
0.0001535527218610621), ('enjoy', 0.0027308167878907 17), ('entir', 0.00
03481827196991612), ('especi', 0.0005781491047258696), ('espresso', 0.0
0011564555666256143), ('everi', 0.0024285860573488856), ('everyday', 3.2
443945908487674e-05), ('everyon', 0.00022805999609047222), ('everyth',
0.0005451083574417931), ('exact', 0.0005374313094278838), ('excel', 0.0
04968646573302482), ('excit', 0.0008067667179601609), ('expect', 0.0015
75374711692181), ('expens', 0.002173904608190189 6), ('experi', 0.000899
4461586118305), ('expir', 0.001885187239740904), ('extra', 0.0003181424
6896766683), ('extract', 0.00017349733989333765), ('extrem', 0.00040963
```

863324660526), ('eye', 0.00031080443108591135), ('face', 0.000128364370
5856789), ('fact', 0.0006586147778953445), ('fair', 0.00012004717122198
262), ('fall', 0.0004176038357194159), ('famili', 0.000741537425409471
8), ('fan', 0.0003010420835045984), ('fantast', 0.000759844404872398),
('far', 0.0007679214428573073), ('farm', 0.0009785596556929656), ('fas
t', 0.0014887770293381432), ('fat', 0.0007030951341165468), ('favorit',
0.0038351904314578448), ('fed', 0.0005878515438072747), ('feed', 0.0005
832859097498964), ('feel', 0.0004108771946746832), ('felin', 0.00057750
81766761359), ('felt', 0.0002904308315949152), ('fiber', 0.000208005884
01492212), ('figur', 0.0004255787632440958), ('filler', 0.0002177732211
7764276), ('final', 0.00020450609000142894), ('fine', 0.001209534700273
5245), ('finish', 0.0005174113164122902), ('fish', 0.000729002765587523
9), ('fit', 0.00024296710378872237), ('flavor', 0.004683879728524735),
('flea', 0.0002544222112315868), ('fli', 0.0020007335791929507), ('floo
r', 0.00028259767261830924), ('flour', 0.00012778213763969492), ('flowe
r', 0.0002786186755192601), ('follow', 0.0003717957955773769), ('food',
0.004196785064213095), ('form', 0.00020172154936053184), ('formula', 0.
0006953943191907063), ('forward', 0.00026611954583526034), ('frank', 0.
00012256629050748865), ('free', 0.0010828284811591038), ('freez', 0.000
28895361631984547), ('french', 0.0005615490380933536), ('fresh', 0.0009
737633270820646), ('fri', 0.0003438830535992454), ('fridg', 0.000244354
0620838378), ('friend', 0.0009542835233433483), ('frozen', 0.0002006286
5707844578), ('fruit', 0.000946047571863482), ('fun', 0.000469274518929
1318), ('futur', 0.000772978932843096), ('garlic', 0.000343564825951873
6), ('gas', 0.000261733603060349), ('gave', 0.002191434054830249), ('ge
neral', 8.222593355058107e-05), ('gift', 0.0014745418239537206), ('ging
er', 0.0007625401284807164), ('girl', 6.972183557984848e-05), ('given',
0.0002550909945833636), ('glad', 0.001099339411710957), ('glass', 0.000
27768286785522227), ('gluten', 0.0006736637041441517), ('goe', 0.000227
49026707052615), ('gone', 0.00027639432113780527), ('good', 0.007541904
49953401), ('got', 0.0014445206912047945), ('gotten', 0.000458519553601
4679), ('gourmet', 0.00013901317850127903), ('grain', 0.000396720353798
4537), ('gram', 0.0012532050495476527), ('granola', 0.00048504906646323
225), ('great', 0.018250422014061584), ('green', 0.000276992646805459
5), ('grew', 0.0002727843413720313), ('grey', 0.00015189813987910705),
('grill', 0.00014519105045741141), ('groceri', 0.001217826449147862),
('ground', 0.000999532695224283), ('grow', 0.0006635597845657947), ('gu
ess', 0.0012670759871337763), ('guest', 6.883176925338352e-05), ('gum',
0.001080218184643445), ('guy', 0.0007549974973315175), ('hair', 0.00028

454307633041335), ('half', 0.0009205938180240974), ('hand', 0.000229251
7004487058), ('handl', 0.0003958898663623308), ('happen', 0.00065718068
26399937), ('happi', 0.0019218547276910114), ('hard', 0.001280280609561
5362), ('hate', 0.0012097517041819993), ('havent', 0.000572933328092005
8), ('head', 0.0006216552164505112), ('health', 0.0002888090043824987
6), ('healthi', 0.0005709353407925776), ('healthier', 6.694330022078667
e-05), ('heard', 0.0004069870736567996 7), ('heart', 0.00028936902007765
12), ('heat', 0.0003671060930022337 5), ('heaven', 0.00022205757271884 50
6), ('heavi', 0.0001388646648454368), ('help', 0.0015645222749325913),
('herb', 7.101343778918292e-05), ('herbal', 0.00010702352777051985),
('hes', 6.864848766615729e-05), ('high', 0.0038276618577503902), ('high
er', 0.0001617220318878050 8), ('hint', 0.0005944946849973026), ('hit',
0.0005112704333294238), ('hold', 0.0005833223372180401), ('holiday', 0.
0001650597795092875 8), ('home', 0.0007472261443650966), ('homemad', 3.8
03538557853822e-05), ('honest', 0.0005266842998863198), ('honey', 0.000
7045636508860292), ('hook', 0.0002058554115036515 3), ('hope', 0.0018860
722364360876), ('horribl', 0.0060601132420273 01), ('hot', 0.00253079991
2398319), ('hour', 0.0008896964036683582), ('hous', 0.00091454270855978
52), ('howev', 0.002220636630442524), ('huge', 0.0006046651035633441),
('human', 0.0004995886085633153), ('husband', 0.0003916412636281439),
('ice', 0.00047184454071502 1), ('idea', 0.002069353461408630 4), ('ill',
0.0016564238179650808), ('imagin', 0.00014747040789123112), ('immedi',
0.0004468871260537903), ('import', 0.0003778228928923485 3), ('impress',
0.0004254182141053834 3), ('improv', 0.00016440699674283822), ('includ',
0.000599417274538930 9), ('increas', 0.0002749050194372383 7), ('incred',
0.0001861186091958387 2), ('individu', 0.0002226683250462346 4), ('infor
m', 0.0007394253630007108), ('ingredi', 0.001996520315904139), ('insi
d', 0.0009100867526848852), ('instant', 0.00021548249140788473), ('inst
ead', 0.0007946394747583631), ('instruct', 0.0009505524262058 83), ('int
ens', 7.366246364334262e-05), ('internet', 0.0002338540316446680 6), ('i
ntroduc', 1.280597282841201 5e-05), ('isnt', 0.0007424662176673068), ('i
ssu', 0.0004705651256605556), ('italian', 0.000579236370729851 8), ('ite
m', 0.0022060191680954 22), ('ive', 0.0014285676839775711), ('jam', 0.00
017315976437828472), ('japanes', 9.079808495958988e-05), ('jar', 0.0008
494629351364187), ('jelli', 0.0004489797905496489), ('job', 0.000407203
33918424154), ('juic', 0.0005801401752184418), ('just', 0.0040123140170
469996), ('kept', 0.0004654794242692), ('kibbl', 0.000388242357753860 4
6), ('kick', 8.975452919093117e-05), ('kid', 0.0005280886182051268),
('kill', 0.0009181747576828765), ('kind', 0.0007244978519033154), ('kit

chen', 0.00025957151508086836), ('kitten', 0.00043974052919170596), ('k
itti', 0.0006181605126887699), ('knew', 0.0001725923304780184), ('kno
w', 0.001452710681439848), ('known', 0.0009890148483993376), ('kona',
0.0005241341526023303), ('lab', 5.6986584603525615e-05), ('label', 0.00
16821149367831733), ('lack', 0.001564528228987855), ('larg', 0.00073618
37993427503), ('larger', 0.00040574111326071585), ('late', 0.0001749358
6574188704), ('later', 0.0006859119186690624), ('lbs', 0.00018986700873
546293), ('leaf', 0.00021898238337936376), ('learn', 0.0002373315234659
4498), ('leav', 0.0005767341170595796), ('left', 0.001023041604213704
8), ('lemon', 0.0005399689521899305), ('let', 0.0008582564502065929),
('level', 0.00035367708926820415), ('licoric', 0.0005604384822265469),
('lid', 0.0005719056815440972), ('life', 0.000281397248665582), ('ligh
t', 0.0005954023699153987), ('like', 0.0070148461456673), ('lime', 0.00
016007362620684793), ('limit', 9.245068495305516e-05), ('line', 0.00080
00540043530182), ('liquid', 0.00011141031199561788), ('list', 0.0029118
679003508575), ('liter', 0.00035474049582091805), ('litter', 0.00040897
564630111474), ('littl', 0.002875041843492895), ('live', 0.000467595472
00530817), ('liver', 0.0003967729157776129), ('local', 0.00082819282646
43544), ('long', 0.0018746547404265158), ('longer', 0.00057548738243634
05), ('look', 0.002681370373340314), ('loos', 0.0006296799538225659),
('lose', 0.0006883325048538893), ('lost', 0.0008252337600614072), ('lo
t', 0.0022332799187097666), ('love', 0.017547496210146653), ('lover',
0.0002199277863898025), ('low', 0.0004514368078226358), ('lower', 0.000
25407119894717305), ('lunch', 0.00016178670719641677), ('mac', 0.000223
49547637310732), ('machin', 0.0002203031434205117), ('mail', 0.00060828
39824659839), ('main', 0.0004476492970223544), ('major', 0.000213929324
19012312), ('make', 0.005170430591731732), ('maker', 0.0004427879230894
447), ('mani', 0.0019131350070661455), ('manufactur', 0.000749263698367
3624), ('mapl', 0.00032631260861112725), ('market', 0.00076752421999852
28), ('matter', 0.0007940840574441805), ('mayb', 0.002772576611211649),
('meal', 0.0007162242895135156), ('mean', 0.0007602093924714939), ('mea
t', 0.000384245672612925), ('medicin', 0.0001358453512117926), ('mediu
m', 0.00015595222346298878), ('melt', 0.0013932968560077324), ('mentio
n', 0.0003181829458947678), ('mess', 0.0016877459165017166), ('mexica
n', 0.00023293905453598886), ('microwav', 0.0006382329883301335), ('mil
d', 7.565120099593292e-05), ('milk', 0.0004680215469234377), ('mind',
0.0003068089332537986), ('mint', 0.000503350551822317), ('minut', 0.001
1610773290208099), ('miss', 0.00018594756234559094), ('mix', 0.00099436
15926059537), ('moist', 9.404532125758772e-05), ('mole', 0.000320534824

64107594), ('mom', 0.0003907885678515697), ('money', 0.020053579147643
1), ('month', 0.0012519267436470307), ('morn', 0.00033803677750772747),
('mother', 0.0004678295591200095), ('mouth', 0.0014429488768052917), ('m
ovi', 0.0003590461213246027), ('mushroom', 0.0006186134625269222), ('mu
stard', 0.0003523183376404597), ('natur', 0.0005851760369050122), ('nea
r', 0.0005752920599274644), ('need', 0.003188854162379966), ('new', 0.0
010249248311338332), ('nice', 0.0033746206835011763), ('night', 0.000335
1236240606267), ('noodl', 0.0005460230930004334), ('normal', 0.00033737
91333178432), ('nose', 0.0007977560539332175), ('note', 0.0002525695130
805864), ('noth', 0.0013507840103236367), ('notic', 0.000982589947935826
3), ('number', 0.0003455445646308932), ('nurs', 0.00011297156814090l3
6), ('nut', 0.0005100948965415969), ('nutrit', 0.0004023407865351618),
('oatmeal', 0.0003418971297591486 5), ('obvious', 0.0001848921537004098
8), ('occasion', 0.0002150496728121629 2), ('odor', 0.000336773866076836
23), ('offer', 0.0007044487395306558), ('offic', 0.0002096526751619123
3), ('oil', 0.0006253024426464434), ('okay', 0.0007779922506506919), ('o
ld', 0.0020312544725125045), ('older', 0.0004499281691123926), ('oliv',
0.0003145845389593696 7), ('onc', 0.0008271045119199205), ('onion', 0.00
037335227966146903), ('onli', 0.0029102339831741974), ('onlin', 0.00043
874422861628294), ('open', 0.0021090231653787166), ('opinion', 0.000250
60658563005916), ('option', 0.0002883926224262176), ('orang', 0.0005488
280926207511), ('order', 0.0035489823633536566), ('organ', 0.0002433881
4662210018), ('origin', 0.0015949019605396937), ('otherwis', 0.00041759
24546254312), ('ounc', 0.0003994657960684539), ('outsid', 0.00020529651
462721814), ('overal', 0.0001699855573188964), ('overpow', 0.0001528078
3640968788), ('owner', 0.0001643266594397561), ('pack', 0.0009375965270
622534), ('packag', 0.0013784915112649058), ('packet', 0.00083766688630
03576), ('paid', 0.0016077814355616104), ('pain', 0.00053210914226356
6), ('pamela', 8.271169957969158e-05), ('pan', 0.0002110828857904941 2),
('pancak', 0.0001443121850428825 6), ('paper', 0.0003187044651789503),
('parti', 0.0002207734811531077 6), ('particular', 0.0003366303008732670
4), ('pass', 0.0004579442801163740 6), ('past', 0.0007320939028308261),
('pasta', 0.0004469204527300612), ('pay', 0.0017622867597107364), ('pea
nut', 0.0005395383452653837), ('peopl', 0.0007674190510509712), ('peppe
r', 0.0005544331648751187), ('peppermint', 0.0002117169576128181 2), ('p
erfect', 0.005588156295521851), ('perhap', 0.0006910425578886026), ('pe
riod', 7.396937371047153e-05), ('person', 0.0005065180732531817), ('pe
t', 0.0013630662594967802), ('pick', 0.0006671307171068789), ('picki',
0.0001975086972998844 4), ('pictur', 0.0029682090055398 08), ('pie', 0.00

03449076091032661), ('piec', 0.0011901491329659 46), ('pill', 0.00054302
87434567448), ('pine', 0.00036403654059236 83), ('place', 0.000498298607
4867281), ('plain', 0.00031405098750610 4), ('plan', 0.00014851292316262
15), ('plant', 0.0015593601895903429), ('plastic', 0.001440678822656338
2), ('play', 0.0006715747271616764), ('pleas', 0.0012914066015581544),
('pleasant', 0.00045865411735477316), ('plenti', 0.0002543670217197604
6), ('plus', 0.00028594679665046264), ('pocket', 1.8379953759069638e-0
5), ('pod', 0.0007875477311855999), ('point', 0.0004949109626384764),
('poor', 0.003848341975021527), ('pop', 0.0012251061860889493), ('popco
rn', 0.0010505547153516885), ('pork', 0.00022044542136745557), ('possib
l', 0.0001646798261484581), ('post', 0.000622885022442234), ('pot', 0.0
007154538814354806), ('potato', 0.00014593240307437383), ('pound', 0.00
11681783763210977), ('pour', 0.00023610333614183028), ('powder', 0.0007
623041758331761), ('power', 0.00041395065239107 85), ('prefer', 0.001541
2876475396915), ('premium', 0.0005122135250070803), ('prepar', 0.000163
836767395308), ('present', 0.0001304733453823786), ('preserv', 0.000386
6260618291436), ('pretti', 0.0008500305035969537), ('previous', 0.00041
254774416633135), ('price', 0.004057520120263195), ('pricey', 0.0002985
612093681879), ('probabl', 0.0011617135383157132), ('problem', 0.000988
982668181571), ('process', 0.0011062030414912157), ('produc', 0.0004770
0181422673845), ('product', 0.010456796693501462), ('prompt', 0.0004954
185731089409), ('proper', 0.0003109367891582078 5), ('protein', 0.000992
982329813463), ('provid', 0.0003284761833495224), ('pull', 0.0001044597
8685643748), ('pump', 0.0014583402628492947), ('pup', 0.000270196959901
86714), ('puppi', 0.0008231233216687374), ('purchas', 0.002362820785263
0914), ('pure', 0.00030234795077077825), ('qualiti', 0.0013957658320896
608), ('quantiti', 0.00036387380472091167), ('quick', 0.001864386751244
988), ('quit', 0.00084859170922097), ('rabbit', 0.00046924298360013 75),
('ran', 0.00042685114987498185), ('raspberri', 0.0003316433112917076),
('rate', 0.00038674192071057653), ('raw', 0.00036109616294942667), ('ra
whid', 0.00037617181837419964), ('read', 0.001012705580168606), ('read
i', 0.00020092879213271967), ('real', 0.0010812103506821465), ('reali
z', 0.001102128240160636), ('realli', 0.0026404075167671217), ('reaso
n', 0.0008228682835138076), ('receiv', 0.004454014609019156), ('recen
t', 0.00106174719685805), ('recip', 0.0004936765448017592), ('recommen
d', 0.0027650105437321895), ('red', 0.000848978513282446), ('reduc', 0.
0001737703110495215), ('refresh', 0.00010887109117491561), ('refriger',
0.00029366523075679 22), ('refund', 0.008269943102860097), ('regular',
0.0022258929710356023), ('relat', 0.0003840888568181556), ('relax', 3.6

157658149153564e-05), ('remain', 0.00017670307422568325), ('rememb', 0.
00063672436330006502), ('remind', 0.0005273161497819395), ('remov', 0.00
05282295287972714), ('replac', 0.0008805461475474371), ('requir', 0.000
21703844633735465), ('research', 0.0006796407141891904), ('rest', 0.000
772011617131399), ('restaur', 0.0005376745558298185), ('result', 0.0005
085456592402871), ('return', 0.011814000479328029), ('review', 0.003520
7317114841056), ('rice', 0.0009249640714650327), ('rich', 0.00030208868
528719445), ('right', 0.0005193344436975353), ('roast', 0.0004549655955
1801917), ('roll', 0.0007937736519835288), ('room', 0.00017296491717324
589), ('root', 0.00014257557942074593), ('round', 0.00030574623454456007), ('run', 0.0012605870130497922), ('safe', 0.0002873000810837817),
('said', 0.0006867660964866355), ('salad', 0.0003837107128778873б), ('s
ale', 0.0004948869169282627), ('salmon', 0.0007243124382045592), ('sals
a', 0.0004436787042460946), ('salt', 0.0015385687802135296), ('salti',
0.0009714698853968035), ('sampl', 0.00015131983354634696), ('sandwich',
0.00027154645604868856), ('satisfi', 0.00011897586932259644), ('sauc',
0.0012760480196875566), ('save', 0.0004903267952885839), ('saw', 0.0010
483967817069653), ('say', 0.002414387024008529), ('scent', 0.0003337174
128098792), ('scoop', 9.467827219108412e-05), ('seal', 0.00052834026987
89659), ('search', 0.0006365310962653741), ('season', 0.000267958816943
49587), ('second', 0.00116048950207008), ('seed', 0.000761058881120077
2), ('seen', 0.00028955119699802275), ('select', 3.9691425855050574e-0
5), ('sell', 0.0013866975330820781), ('seller', 0.0013437382644504992),
('send', 0.0005812130179707507), ('senseo', 0.00045063564722316245),
('sensit', 0.0005117003585941341), ('sent', 0.0010646406806900644), ('s
erv', 0.0005032521565088843), ('servic', 0.000484295867982652), ('set',
0.0006875347244284072), ('sever', 0.0005670906981103251), ('shake', 0.0
0020008741956001452), ('shape', 0.00026185214493839915), ('share', 0.00
01285415814369528), ('shelf', 0.00038063492291525217), ('shell', 0.0002
175870203743189), ('shes', 0.0007312164340079098), ('ship', 0.002593986
759276129), ('shipment', 0.00050583032354723), ('shop', 0.0008412687661
179353), ('short', 0.0007345317790236337), ('shot', 0.00021344036962407
022), ('sick', 0.000992577713604639), ('similar', 0.000600690820685542
2), ('simpl', 9.094600061583416e-05), ('simpli', 0.000230427840491867
2), ('sinc', 0.0014553947115280908), ('singl', 0.0009089192014299656),
('sit', 0.0004918077066623028), ('site', 0.0010664200198843505), ('siz
e', 0.002056130880874254), ('skin', 0.0006448128114881845), ('sleep',
4.364268502405148e-06), ('slice', 0.00029456284668781876), ('slight',
0.0007571758519201164), ('small', 0.0016199390636181179), ('smaller',

0.0003610478185391372), ('smell', 0.002278512525787545), ('smoke', 0.00
051270947394756), ('smooth', 0.00025292415735037976), ('snack', 0.00022
251360480688627), ('soda', 0.0004985780352161434), ('sodium', 0.0005501
964111889886), ('soft', 0.0007936971649429519), ('sold', 0.000161722368
75601316), ('solid', 0.0002699205470208876), ('someon', 0.0008806913549
831266), ('someth', 0.0019602797160005398), ('sometim', 0.0002595561082
0803106), ('somewhat', 0.00032763460040901167), ('son', 0.0001740285398
6808435), ('soon', 0.0003637267898742968), ('sooth', 3.3452691731448272
e-06), ('sort', 0.00025439173396372646), ('sound', 0.000690631899297864
3), ('soup', 0.0005240164772899926), ('sour', 0.00033592675186662), ('s
ourc', 0.0005542588201924277), ('soy', 0.000321745343045942), ('specia
l', 0.00047967072253433485), ('spend', 0.0009824613797691274), ('spen
t', 0.00018587271427788178), ('spice', 0.0003855015681739707), ('spic
i', 0.0007096434644711403), ('spoon', 0.00013835910592214945), ('spot',
0.00017525598181623365), ('spray', 0.0004706809762375254), ('spread',
8.999084770620642e-05), ('sprinkl', 6.30459766501272e-05), ('stale', 0.
002390648267028559), ('stand', 0.00025626151259719156), ('standard', 0.
0007123482234695304), ('star', 0.0007962920866012817), ('starbuck', 0.0
002422390497820765), ('start', 0.0010851270487258861), ('stash', 0.0001
2066374389437612), ('state', 0.0005262705899259672), ('stay', 0.0006951
827323545876), ('steak', 0.00010720724900341479), ('steep', 3.843825826
902538e-05), ('stevia', 0.0005053110734435343), ('stick', 0.00267052915
8327826), ('stir', 0.00019032935251537674), ('stock', 0.000587750769224
2335), ('stomach', 9.696033812757458e-05), ('stool', 0.0003503079868321
1766), ('stop', 0.0011105084774024954), ('store', 0.002079541141958436
6), ('straight', 0.0001926039788716656), ('strawberri', 0.0001549094633
5473805), ('strong', 0.0012978470351855196), ('stronger', 0.00016083811
932698125), ('stuck', 0.0007670934862924342), ('stuff', 0.0017924842377
680375), ('style', 0.0008346903508038348), ('subscrib', 6.7387791226479
32e-05), ('substitut', 0.0003096191868109185), ('subtl', 0.000214733653
04894575), ('success', 0.00023323404265716453), ('suffer', 0.0003905128
5418441377), ('sugar', 0.0009707849890250893), ('suggest', 0.0005688188
154306643), ('summer', 0.00011593930950570613), ('super', 0.00029162048
25406498), ('supermarket', 0.0004049957925953081), ('supplement', 0.000
10244862900092195), ('suppli', 0.0004764342690695335), ('suppos', 0.001
1101902100997998), ('sure', 0.0010371549384381436), ('surpris', 0.00058
59135077144747), ('sweet', 0.0019721973158415666), ('sweeten', 0.000319
02932620560503), ('switch', 0.0011500265942069076), ('syrup', 0.0010711
202192702763), ('tabasco', 2.734379430645063e-05), ('tabl', 4.335882376

6701005e-05), ('tablespoon', 0.00024912515148378l4), ('taken', 0.000623
3865410622113), ('talk', 0.0003066939462008419З), ('tast', 0.0066495642
22792532), ('tasti', 0.0015302212265401421), ('tea', 0.0042995906911663
054), ('teaspoon', 1.9723669772492195e-05), ('teeth', 0.000455168369299
17776), ('tell', 0.00036910438391846715), ('tend', 0.000117231238478688
27), ('terribl', 0.007507945943430323), ('test', 0.0002338136971891959
5), ('textur', 0.0009008957442349498), ('thank', 0.002154862607331872
3), ('theyr', 0.0005177213363794045), ('thing', 0.0016421469513846828),
('think', 0.0026878260505595187), ('thought', 0.008909132132170863),
('thrill', 4.456270631494112e-05), ('throat', 0.0001384990544450025),
('throw', 0.0017764362759249016), ('time', 0.0029006080137075627), ('ti
n', 0.0006946405682433041), ('tini', 0.000519022233813691), ('toast',
0.0002899580374847741), ('today', 0.00031761010780279075), ('togeth',
0.0006475555775510853), ('told', 0.0001886815740684323), ('tomato', 0.0
005051010172751121), ('ton', 0.00023808276822834213), ('took', 0.000959
6960391406041), ('total', 0.0006363721456574571), ('touch', 0.000766062
5278993385), ('toy', 0.001183660190948835), ('tradit', 0.00011280170190
435147), ('train', 0.00040487022840884375), ('trap', 0.0012590941727691
476), ('travel', 0.00012598613785630978), ('treat', 0.00266223041082707
5), ('tree', 0.00040740229723997516), ('tri', 0.0040422925034047895),
('trick', 6.613087156184457e-05), ('trip', 0.0003447538241873256), ('tr
oubl', 0.00011144266679095696), ('true', 0.0006120171570602186), ('truf
fl', 0.0005194076080286235), ('truli', 0.00015594720998650903), ('tun
a', 0.0001299261302852102), ('turkey', 0.00030758955673167), ('turn',
0.0007527020870376217), ('twice', 0.00034875726533947), ('type', 0.0005
264142838028164), ('typic', 0.00019841537540476307), ('understand', 0.0
004552260004969337), ('unfortun', 0.00347561134051238), ('uniqu', 0.000
1433442872375677), ('unless', 0.0008860067025811965), ('unlik', 0.00011
171393184552863), ('upset', 0.00015691102818666695), ('use', 0.00628823
5787391456), ('usual', 0.0010208644977317172), ('valu', 0.0006183821843
245377), ('vanilla', 0.001004861535824632), ('varieti', 0.0003944312664
6199563), ('various', 0.00012099275666496943), ('vegan', 0.000361867965
472441), ('veget', 0.0004169897028635498), ('vegetarian', 3.38854252379
42364e-05), ('veggi', 0.0003041804420896976), ('vendor', 0.000769895197
514932), ('veri', 0.006747739446726154), ('version', 0.0007610705993928
97), ('vet', 0.0002716945014553223), ('vinegar', 0.0002157914635532347
3), ('visit', 7.264559079792039e-05), ('vitamin', 0.000204630640354391
4), ('vomit', 0.001667393007097814), ('wait', 0.0003950503055152481),
('walk', 0.000297524304161967), ('want', 0.002394371045784039), ('war

```
m', 0.000303747298797898), ('warn', 0.0005336956279950371), ('wasnt',
0.0014960918279314957), ('wast', 0.009357676957439234), ('watch', 0.000
16198721119217117), ('water', 0.0019476957362144446), ('way', 0.0023236
867678898933), ('weak', 0.0020204456392886577), ('websit', 0.0004065708
917236736), ('week', 0.0013505724639698033), ('weight', 0.0007346673463
713536), ('went', 0.0007155953603974711), ('wet', 0.000503512013005922
5), ('weve', 0.0002054832406179902), ('whatev', 0.0001435255225159194),
('wheat', 0.00029020572780460134), ('whenev', 9.052991406551119e-05),
('whi', 0.002380708995536595), ('white', 0.0020763237431977084), ('wif
e', 0.0008213083117898238), ('wild', 0.00031332162169208283), ('wine',
0.0001502312172380131), ('winter', 0.00012829757640100606), ('wish', 0.
0014565819885511543), ('wonder', 0.0038399227735184563), ('wont', 0.003
6806404224403676), ('word', 0.0004239166767572577), ('work', 0.00278317
6890995696), ('world', 0.0001050207137275353), ('worri', 9.447822684816
2e-05), ('worst', 0.011677301751938245), ('worth', 0.002882848178501895
5), ('wouldnt', 0.0015407044731070962), ('wow', 0.000283854531894585),
('wrap', 0.00021537510652986521), ('write', 0.0003762047515586792), ('w
rong', 0.0013871363183462714), ('year', 0.004570175184288074), ('yeas
t', 0.00016943018947013816), ('yellow', 0.0004011151963452404), ('yes',
0.0003437398859796687), ('yogi', 0.0003121624380804915), ('youll', 0.00
045518318605263517), ('youv', 0.00032099703388040765), ('yum', 9.123847
504457213e-05), ('yummi', 0.00034989470074334996), ('zico', 0.001136391
1493618177), ('zuke', 5.916476017718606e-05)]
1000
```

Out[0]:

	feature	values
254	disappoint	0.036433
552	money	0.0200536
81	best	0.019202
376	great	0.0182504
514	love	0.0175475
722	return	0.011814
982	worst	0.0116773
673	product	0.0104568
955	wast	0.00935768

889	thought	0.00890913
708	refund	0.00826994
234	delici	0.00816494
369	good	0.0075419
882	terribl	0.00750795
495	like	0.00701485
942	veri	0.00674774
58	bad	0.0066872
875	tast	0.00664956
931	use	0.00628824
244	did	0.00627101

In [0]:

```

```

In [0]:

```

```

In [0]:

```
import xgboost as xgb
```

In [0]:

```
xgb_model = xgb.XGBClassifier()

#brute force scan for all parameters, here are the tricks
#usually max_depth is 6,7,8
#learning rate is around 0.05, but small changes may make big diff
#tuning min_child_weight subsample colsample_bytree can have
#much fun of fighting against overfit
#n_estimators is how many round of boosting
#finally, ensemble xgboost with multiple seeds may reduce variance
parameters = {'nthread':[4], #when use hyperthread, xgboost may become
 slower
              'objective':['binary:logistic'],
              'learning_rate': [0.05], #so called `eta` value
```

```
              'max_depth': [6],
              'min_child_weight': [11],
              'silent': [1],
              'subsample': [0.8],
              'colsample_bytree': [0.7],
              'n_estimators': [5], #number of trees, change it to 1000
 for better results
              'missing':[-999],
              'seed': [1337]}


clf = GridSearchCV(xgb_model, parameters, n_jobs=5,
                   cv=5,
                   scoring='roc_auc',
                   verbose=2, refit=True)

#clf.fit(dtrain, dtest)
clf.fit(X_train_bow_xg,y_train_xg)


#sample = pd.read_csv('../input/sample_submission.csv')
#s#ample.QuoteConversion_Flag = test_probs
#sample.to_csv("xgboost_best_parameter_submission.csv", index=False)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent worke
rs.
[Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:    5.7s remaining:
    8.5s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    6.0s remaining:
    0.0s
[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    6.0s finished
```

Out[0]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsam
ple_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=
0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_stat
```

```
        n_jobs=1, nthread=None, objective='binary:logistic', random_stat
e=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
       fit_params=None, iid='warn', n_jobs=5,
       param_grid={'nthread': [4], 'objective': ['binary:logistic'], 'l
earning_rate': [0.05], 'max_depth': [6], 'min_child_weight': [11], 'sil
ent': [1], 'subsample': [0.8], 'colsample_bytree': [0.7], 'n_estimator
s': [5], 'missing': [-999], 'seed': [1337]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=2)
```

In [0]:

```
import xgboost as xgb
dtrain = xgb.DMatrix(X_train_bow_xg, label=y_train_xg.values)
dtest = xgb.DMatrix(X_test_bow_xg, label=y_test_xg.values)

#*********for Lsees consumption use SVMLIGHt ********************
#import xgboost as xgb
#fr#om sklearn.datasets import dump_svmlight_file

#dump_svmlight_file(X_train, y_train, 'dtrain.svm', zero_based=True)
#dump_svmlight_file(X_test, y_test, 'dtest.svm', zero_based=True)
#dtrain_svm = xgb.DMatrix('dtrain.svm')
#dtest_svm = xgb.DMatrix('dtest.svm')


param = {
    'max_depth': 3,  # the maximum depth of each tree
    'eta': 0.3,  # the training step for each iteration
    'silent': 1,  # logging mode - quiet
    'objective': 'multi:softprob',  # error evaluation for multiclass t
raining
    'num_class': 3}  # the number of classes that exist in this datset
num_round = 20  # the number of training iterations


xgboost = xgb.train(param, dtrain, num_round)
preds = xgboost.predict(dtest)
```

```python
#Here each column represents class number 0, 1, or 2. For each line you
 need to select that column where the probability is the highest:
import numpy as np
pred = np.asarray([np.argmax(line) for line in preds])

from sklearn.metrics import precision_score
print(precision_score(y_test, pred, average='macro'))
```

In [0]:
```python
X_train_xg.shape,y_train_xg.shape
```

Out[0]: ((800,), (800,))

In [0]:
```python
import xgboost as xgb

#dtrain = xgb.DMatrix(X_train_xg.values, label=y_train_xg.values)
#dtest = xgb.DMatrix(X_test_xg.values, label=y_test_xg.values)

param_test1 = {
 'max_depth':range(3,10,2),
 'min_child_weight':range(1,6,2),
 'reg_alpha':[0, 0.001, 0.005, 0.01, 0.05]
}
param_test3 = {

}

gsearch1 = GridSearchCV(estimator = xgb.XGBClassifier( learning_rate =
0.1, n_estimators=140, max_depth=5,
 min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=27),

 param_grid = param_test1, scoring='roc_auc',n_jobs=4,iid=False, cv=5)


gsearch1.fit(X_train_bow_xg,y_train_xg)
#gsearch1.cv_results_
gsearch1.best_params_, gsearch1.best_score_
```

In [0]:

```
hyperparameters=[(i['max_depth'],i['min_child_weight'],i['reg_alpha'])
for i in gsearch1.cv_results_['params']]

depth         = [i[0] for i in hyperparameters]
min_child_weight  = [i[1] for i in hyperparameters]

train_score = gsearch1.cv_results_['mean_train_score'].tolist()
test_score  = gsearch1.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(min_child_weight)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, min_child_weight , train_score , 'red',label="train_s
core")
ax1.set_xlabel('depth')
ax1.set_ylabel('min_child_weight')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, min_child_weight, train_score , c=train_score, cma
p='Greens',label="train_score")

ax1.plot3D(depth, min_child_weight , test_score , 'blue',label="test_sc
ore")
ax1.scatter3D(depth, min_child_weight, test_score , c=test_score, cmap=
'OrRd',label="test_score")
```

```
print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gsearch1.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "min_child_weight", "mean_test_
score")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "min_child_weight", "mean_train
_score")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.calibration import CalibratedClassifierCV

#({'max_depth': 5, 'min_child_weight': 1, 'reg_alpha': 0.05},
```

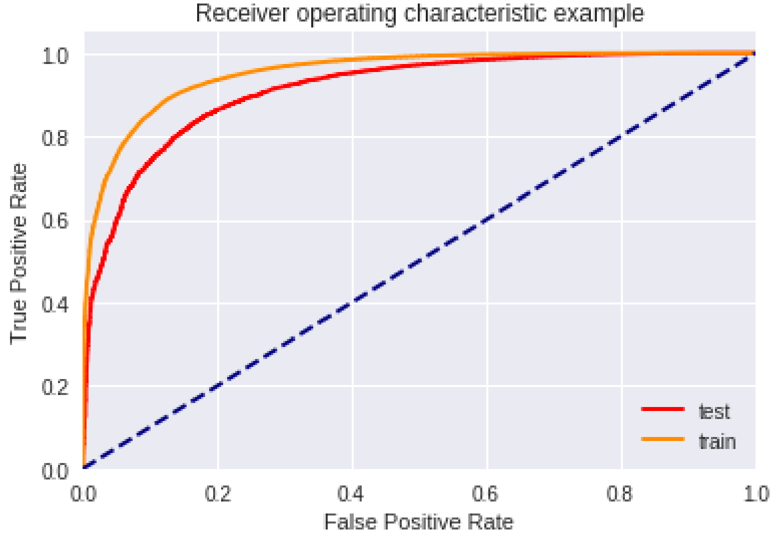
```
clf1=xgb.XGBClassifier(min_child_weight=1,max_depth=9,reg_alpha=0)
clf1.fit(X_train_bow_xg,y_train_xg)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_bow_xg, y_train_xg)

pred = sig_clf.predict_proba(X_test_bow_xg)[:,1]
pred_train = sig_clf.predict_proba(X_train_bow_xg)[:,1]


pred_train_without_CCV=clf1.predict(X_train_bow_xg)
pred_without_CCV=clf1.predict(X_test_bow_xg)


print("Accuracy Score : ",accuracy_score(y_test_xg,pred_without_CCV)*10
0)
print("Precision Score : ",precision_score(y_test_xg,pred_without_CCV)*
100)
print("Recall Score : ",recall_score(y_test_xg,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test_xg,pred_without_CCV)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test_xg,pred_without_CCV))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train_xg,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test_xg,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
```

```
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")

tn, fp, fn, tp=confusion_matrix(y_test_xg,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test_xg,pred_without_CCV),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

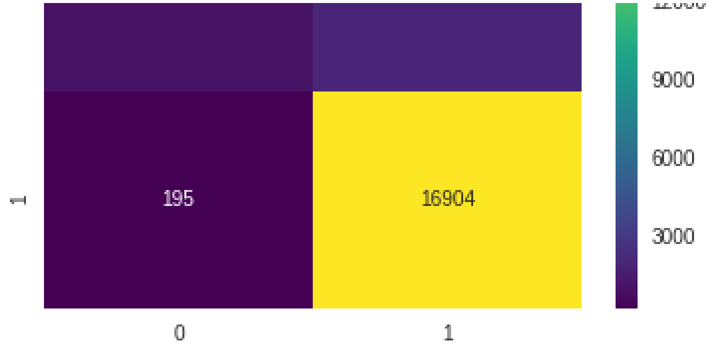
```
Accuracy Score :  89.495
Precision Score :  89.86709197235513
Recall Score :  98.85958243172115
F1 Score :  94.14909911164332

Classification Report
              precision    recall  f1-score   support

           0       0.84      0.34      0.49      2901
           1       0.90      0.99      0.94     17099
```

```
   micro avg       0.89      0.89      0.89     20000
   macro avg       0.87      0.67      0.71     20000
weighted avg       0.89      0.89      0.88     20000
```

```
AUC Score for train data : 0.9526772700620123
AUC Score for test data : 0.9128114335643238
```




```
TrueNegative : 995

FalsePostive : 1906
FalseNegative : 195
TruePostive  :  16904
```



[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [0]:
```
#BoW
#tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect = TfidfVectorizer(max_df=0.95, min_df=2,stop_words='englis
h',max_features=100) #in scikit-learn
tf_idf_vect.fit(X_train)
print("some feature names ", tf_idf_vect.get_feature_names()[:10])
print('='*50)

X_train_tfidf = tf_idf_vect.transform(X_train)
print("the type of count vectorizer ",type(X_train_tfidf))
print("the shape of out text BOW vectorizer ",X_train_tfidf.get_shape
())
print("the number of unique words ", X_train_tfidf.get_shape()[1])

X_test_tfidf = tf_idf_vect.transform(X_test)
print("the type of count vectorizer ",type(X_test_tfidf))
print("the shape of out text BOW vectorizer ",X_test_tfidf.get_shape())
print("the number of unique words ", X_test_tfidf.get_shape()[1])
```

```
some feature names  ['add', 'alway', 'amazon', 'ani', 'bag', 'becaus',
'befor', 'best', 'better', 'bit']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (32000, 100)
```

```
the number of unique words  100
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (8000, 100)
the number of unique words  100
```

In [0]:

```
from sklearn.ensemble import  AdaBoostClassifier, GradientBoostingClass
ifier
clf=GradientBoostingClassifier()
#param_grid={ 'n_estimators' : np.arange(20,200,20),
#              'max_depth' :  [5,10,15,20,30],
#              'learning_rate' : [0.1,0.001] }
param_grid={ 'n_estimators' : [50,100,150,200],
             'max_depth' :  [5,10,20]}
gcv=RandomizedSearchCV(clf,param_grid,cv=5)
gcv.fit(X_train_tfidf,y_train)
print(gcv.best_params_)
print(gcv.best_score_)
```

```
{'n_estimators': 200, 'max_depth': 5}
0.856125
```

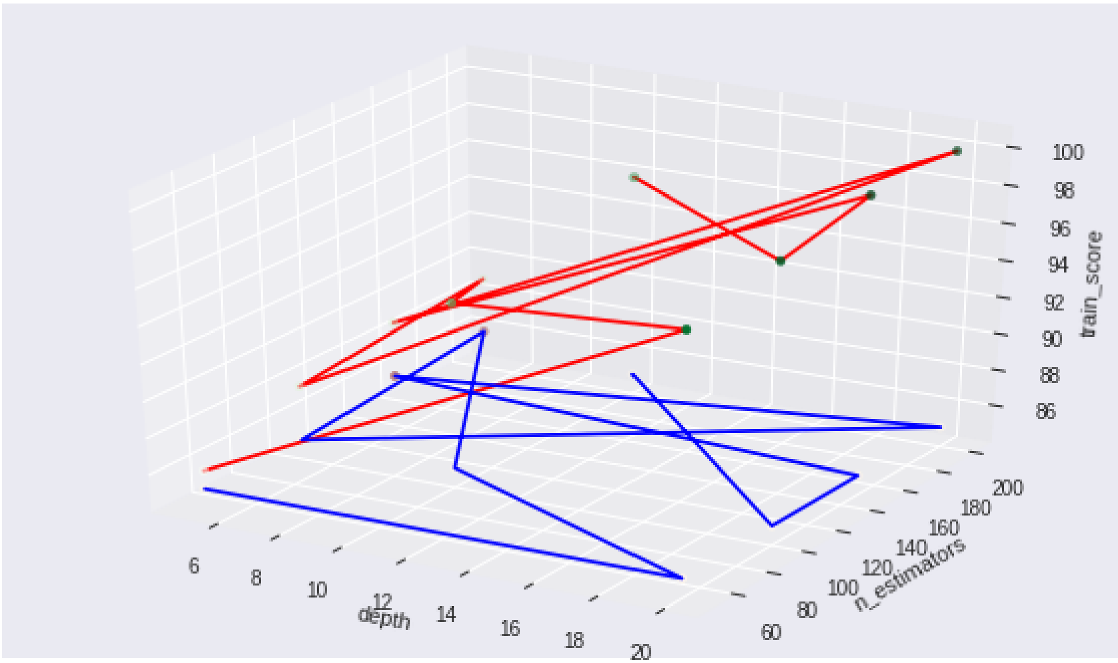
In [0]:

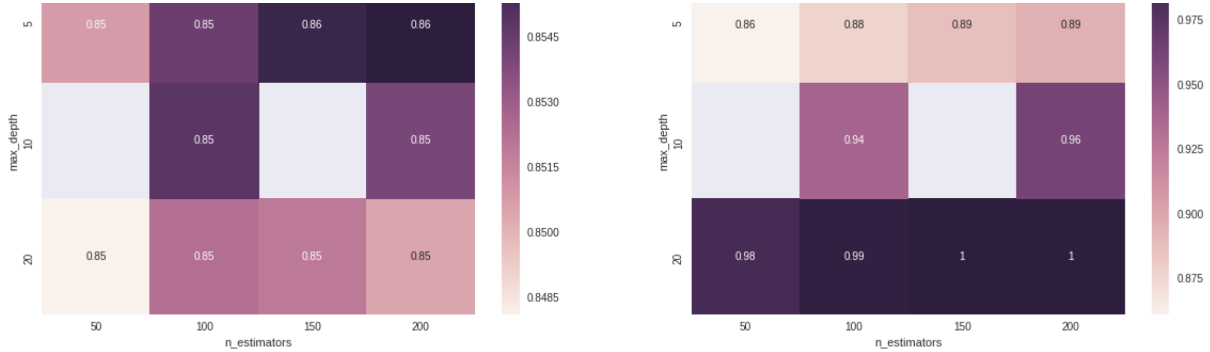
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")
```

```
from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='G
reens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score"
)
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrR
d',label="test_score")


print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_scor
e")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )
```

```
final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

```
[5, 20, 10, 5, 5, 20, 5, 20, 20, 10]
[50, 50, 100, 200, 100, 200, 150, 150, 100, 200]
[86.0, 98.0, 94.0, 89.0, 88.0, 100.0, 89.0, 100.0, 99.0, 96.0]
[85.0, 85.0, 85.0, 86.0, 85.0, 85.0, 86.0, 85.0, 85.0, 85.0]
ploting 3d grap
ploting Heat Map
```





In [0]:

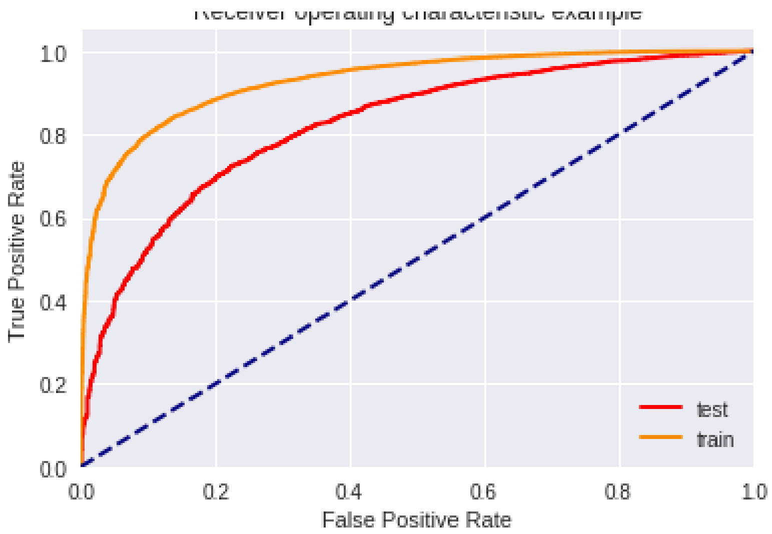
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score




clf1=GradientBoostingClassifier(n_estimators=200,max_depth=5)
clf1.fit(X_train_tfidf,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_tfidf, y_train)

pred = sig_clf.predict_proba(X_test_tfidf)[:,1]
pred_train = sig_clf.predict_proba(X_train_tfidf)[:,1]


pred_train_without_CCV=clf1.predict(X_train_tfidf)
pred_without_CCV=clf1.predict(X_test_tfidf)


print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
```

```
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")

tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
```

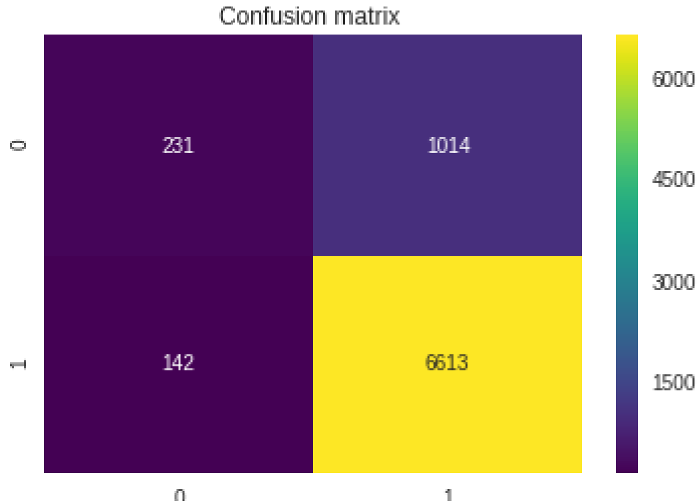
```
FalseNegative : {}
TruePostive   :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  85.55
Precision Score :  86.70512652419038
Recall Score :  97.8978534418949
F1 Score :  91.96217494089834

Classification Report
              precision    recall  f1-score   support

           0       0.62      0.19      0.29      1245
           1       0.87      0.98      0.92      6755

   micro avg       0.86      0.86      0.86      8000
   macro avg       0.74      0.58      0.60      8000
weighted avg       0.83      0.86      0.82      8000


AUC Score for train data : 0.9287001619184111
AUC Score for test data : 0.8221434070850389
```

Receiver operating characteristic example




```
TrueNegative : 231
FalsePostive : 1014
FalseNegative : 142
TruePostive  :  6613
```



In [0]:
```python
-np.sort(-clf1.feature_importances_)[:20]


features=tf_idf_vect.get_feature_names()
imp=clf1.feature_importances_.argsort()[::-1][:20]
top20Features=[features[i] for i in imp]
top20Features
print(top20Features)

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

wordcloud = WordCloud(width = 400, height = 400, background_color ='black',  min_font_size = 10).generate(' '.join(top20Features))

# plot the WordCloud image
plt.figure(figsize = (4, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

```
['great', 'love', 'best', 'did', 'product', 'didnt', 'tast', 'good', 'delici', 'box', 'perfect', 'review', 'ingredi', 'buy', 'dont', 'use', 'look', 'like', 'old', 'onli']
```

In [0]:
```
#a=clf1.feature_importances_.tolist()
#a.sort()
#print(a[::-1][:20])

#np.sort(clf1.feature_importances_,)[::-1][:20]
-np.sort(-clf1.feature_importances_)[:50]


features=tf_idf_vect.get_feature_names()
imp=clf1.feature_importances_.tolist()

print([i for i in zip(features,imp)])

df=pd.DataFrame([features,imp],index=['feature','values'])
df=df.T
df1=df.sort_values('values',ascending=False)


print(len(imp))
len(features)
df1[:20]
```

```
[('add', 0.004790526336066039), ('alway', 0.006755409983854369), ('amaz
on', 0.007637441244196877), ('ani', 0.001240021027745022), ('bag', 0.01
097278777813433), ('becaus', 0.008490865782628238), ('befor', 0.0071210
10407324348), ('best', 0.0518310234638379), ('better', 0.00268363210417
06713), ('bit', 0.0021805949282548754), ('bottl', 0.00391638765334067
3), ('bought', 0.009934506296238531), ('box', 0.019888991782301937),
('brand', 0.009004188960466737), ('buy', 0.014510953823906334), ('cat',
0.004328381514334041), ('chocol', 0.0018969214906877742), ('coffe', 0.0
04425927190579354), ('come', 0.0040088675736611725), ('cup', 0.00117707
9278719279), ('day', 0.0039654919899103774), ('delici', 0.0231505619829
27118), ('did', 0.03894529627567970б), ('didnt', 0.031504545860098l),
('differ', 0.002880902708370829), ('doe', 0.004914816458537433), ('do
g', 0.010642100929520958), ('dont', 0.014388333666259957), ('drink', 0.
0019852194333064405), ('eat', 0.007036617687499177), ('enjoy', 0.006820
384586415169), ('everi', 0.003482349357249429), ('favorit', 0.010521900
504324808), ('flavor', 0.009579705069951646), ('food', 0.00628167589415
278), ('fresh', 0.004143624102663389), ('good', 0.026923591290977517),
('got', 0.009894846133874619), ('great', 0.0723193311057017), ('help',
0.006205190808183914), ('high', 0.007853667592313644), ('hot', 0.005516
774081641801), ('ingredi', 0.015637739188538405), ('ive', 0.00166465653
6559892), ('just', 0.010996756343449425), ('know', 0.00526246492766701
5), ('like', 0.013891911478132098), ('littl', 0.004014918631639108),
('local', 0.0017144960047064926), ('long', 0.0023235280581470805), ('lo
ok', 0.014096383302413225), ('lot', 0.003380572427804842), ('love', 0.07
079388890712485), ('make', 0.00742709411776086), ('mani', 0.00241491908
9820485), ('milk', 0.0024544773947233565), ('mix', 0.00261490578548501 9
7), ('month', 0.004499817011203297), ('natur', 0.002923766820833712),
('need', 0.006757595825489652), ('nice', 0.011027187748589235), ('oil',
0.0009701376278339593), ('old', 0.012693524164771394), ('onli', 0.01155
6387920209556), ('order', 0.009018266417615559), ('packag', 0.010273946
354571361), ('perfect', 0.018167340147952224), ('price', 0.009361987516
052266), ('product', 0.037544392103856954), ('purchas', 0.0104264166207
3871), ('qualiti', 0.0019808383393637614), ('realli', 0.004086452650972
6985), ('recommend', 0.007501258748931716), ('review', 0.01786595902895
507), ('sauc', 0.0011992532585459053), ('say', 0.006621422791562313),
('ship', 0.007898965410368754), ('sinc', 0.003440408757468705), ('smal
l', 0.0033644439731685865), ('smell', 0.010597946995670994), ('start',
0.002232472149939984), ('store', 0.005207952727664496), ('stuff', 0.003
7496124083063877), ('sugar', 0.00563214586346022), ('sweet', 0.00134662
0695348708), ('tast', 0.02937251151697485), ('tea', 0.0065037375270369 2
```

```
6), ('thing', 0.0058052734985938464), ('think', 0.008250850093667566),
('time', 0.0051298855774051955), ('treat', 0.002514443379896587), ('tr
i', 0.0044139988568152955), ('use', 0.01418710772536239), ('veri', 0.00
657234399238108), ('want', 0.0067801431226612785), ('water', 0.00547362
2310692154), ('way', 0.005020260385008967), ('wonder', 0.01057574999132
1479), ('work', 0.006644031438967403), ('year', 0.010368362169787712)]
100
```

Out[0]:

	feature	values
38	great	0.0723193
52	love	0.0707939
7	best	0.051831
22	did	0.0389453
68	product	0.0375444
23	didnt	0.0315045
85	tast	0.0293725
36	good	0.0269236
21	delici	0.0231506
12	box	0.019889
66	perfect	0.0181673
73	review	0.017866
42	ingredi	0.0156377
14	buy	0.014511
27	dont	0.0143883
92	use	0.0141871
50	look	0.0140964
46	like	0.0138919
62	old	0.0126935

	feature	values
63	onli	0.0115564

[5.2.3] Applying XGBOOST on AVG W2V, SET 3

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


#************************************************************************
*********************

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")

#************************************************************************
```

```
********************

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])

#***********************************************************************
********************



# average Word2Vec
# compute average word2vec for each review.
X_train_AvgW2V = []; # the avg-w2v for each sentence/review is stored i
n this list
for sent in list_of_sentance: # 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
    X_train_AvgW2V.append(sent_vec)
print(len(X_train_AvgW2V))
print(len(X_train_AvgW2V[0]))
#***********************************************************************
**********************

i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

#***********************************************************************
********************
```

```
# average Word2Vec
# compute average word2vec for each review.
X_test_AvgW2V = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in 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
    X_test_AvgW2V.append(sent_vec)
print(len(X_test_AvgW2V))
print(len(X_test_AvgW2V[0]))
```

```
[('wonder', 0.826174259185791), ('excel', 0.8060868978500366), ('fantas
t', 0.8010183572769165), ('perfect', 0.7783036231994629), ('good', 0.77
74456739425659), ('awesom', 0.7764139771461487), ('terrif', 0.763584911
8232727), ('amaz', 0.7019591331481934), ('nice', 0.6454187631607056),
('fabul', 0.6228042840957642)]
==================================================
[('best', 0.7324355840682983), ('disgust', 0.6967191696166992), ('horri
bl', 0.6771298050880432), ('closest', 0.6473489999771118), ('nicest',
0.6233989596366882), ('terribl', 0.6099085211753845), ('greatest', 0.60
59007048606873), ('tastiest', 0.5827214121818542), ('superior', 0.57219
40994262695), ('finest', 0.5604434013366699)]
number of words that occured minimum 5 times  8343
sample words  ['this', 'product', 'better', 'than', 'ani', 'have', 'tr
i', 'the', 'pure', 'white', 'powder', 'and', 'doe', 'not', 'filler', 'b
est', 'valu', 'wasabi', 'pea', 'out', 'there', 'bag', 'repres', 'lot',
'but', 'leav', 'work', 'theyll', 'soon', 'disappear', 'tasti', 'first',
'had', 'cooki', 'airlin', 'kept', 'wrapper', 'been', 'long', 'time', 's
inc', 'enjoy', 'such', 'delight', 'then', 'bought', 'case', 'give', 'fr
```

```
iend', 'whenev']
32000
50
8000
50
```

In [0]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(with_mean=False)
scaler.fit(X_train_AvgW2V)
X_train_AvgW2V=scaler.transform(X_train_AvgW2V)

X_test_AvgW2V=scaler.transform(X_test_AvgW2V)
```

In [0]:
```
from sklearn.ensemble import  AdaBoostClassifier, GradientBoostingClass
ifier
clf=GradientBoostingClassifier()
#param_grid={ 'n_estimators' : np.arange(20,200,20),
#              'max_depth' :  [5,10,15,20,30],
#              'learning_rate' : [0.1,0.001] }
param_grid={ 'n_estimators' : [10,100],
             'max_depth' :  [5,10,20]}
gcv=RandomizedSearchCV(clf,param_grid,cv=5)
gcv.fit(X_train_AvgW2V,y_train)
print(gcv.best_params_)
print(gcv.best_score_)
```

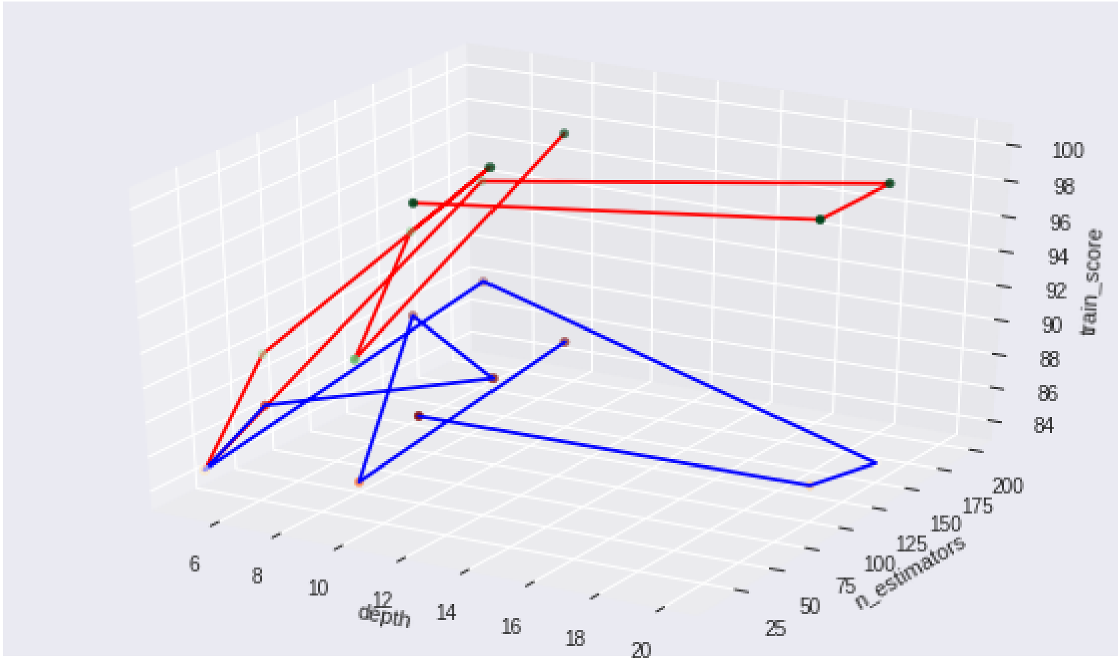
```
{'n_estimators': 100, 'max_depth': 5}
0.87915625
```

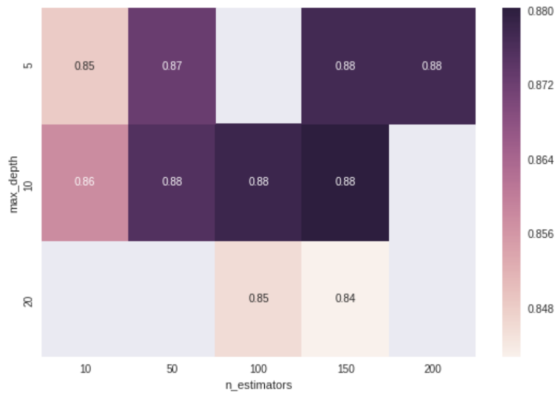
In [0]:
```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()
```

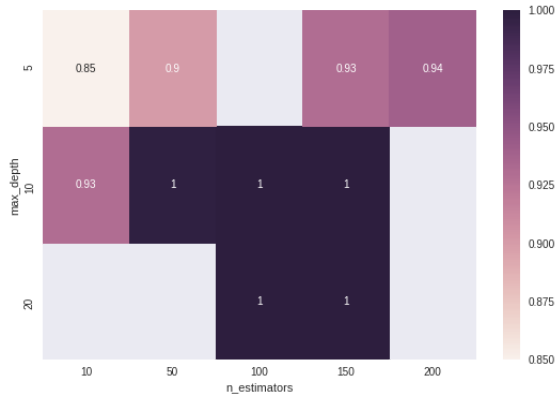
```
train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='G
reens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score"
)
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrR
d',label="test_score")


print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)
```

```
df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_scor
e")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

```
[10, 10, 5, 10, 5, 5, 5, 20, 20, 10]
[150, 10, 150, 100, 50, 10, 200, 150, 100, 50]
[100.0, 93.0, 93.0, 100.0, 90.0, 85.0, 94.0, 100.0, 100.0, 100.0]
[88.0, 86.0, 88.0, 88.0, 87.0, 85.0, 88.0, 84.0, 85.0, 88.0]
ploting 3d grap
ploting Heat Map
```







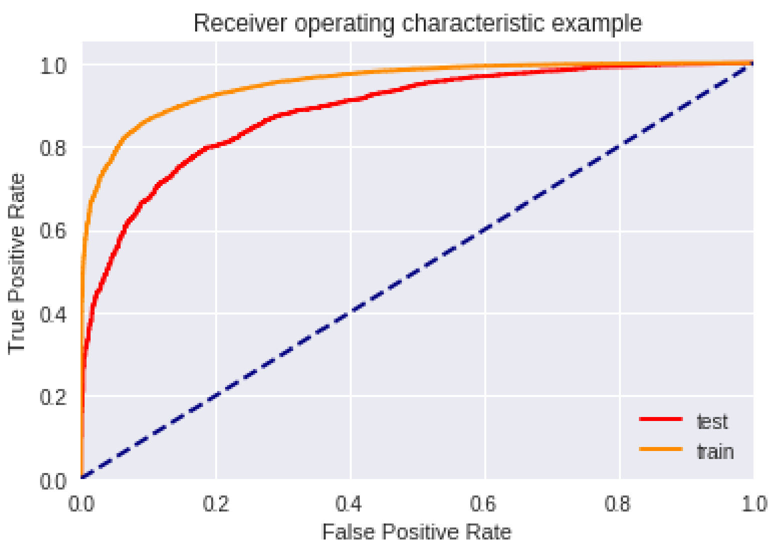
In [0]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.ensemble import GradientBoostingClassifier

clf1=GradientBoostingClassifier(n_estimators=100,max_depth=5)
clf1.fit(X_train_AvgW2V,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_AvgW2V, y_train)

pred = sig_clf.predict_proba(X_test_AvgW2V)[:,1]
pred_train = sig_clf.predict_proba(X_train_AvgW2V)[:,1]


pred_train_without_CCV=clf1.predict(X_train_AvgW2V)
pred_without_CCV=clf1.predict(X_test_AvgW2V)


print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")
```

```
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")

tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
print("""
TrueNegative : {}
FalsePostive : {}
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

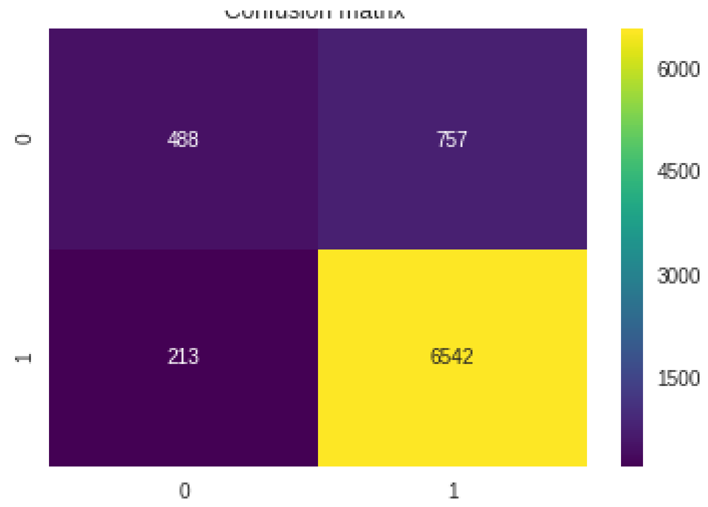
```
Accuracy Score :  87.875
Precision Score :  89.62871626250171
Recall Score :  96.84678016284234
F1 Score :  93.09805037711682

Classification Report

              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.70      0.39      0.50      1245
           1       0.90      0.97      0.93      6755

   micro avg       0.88      0.88      0.88      8000
   macro avg       0.80      0.68      0.72      8000
weighted avg       0.87      0.88      0.86      8000


AUC Score for train data : 0.9519917924771961
AUC Score for test data : 0.8847887181590908
```




```
TrueNegative : 488
FalsePostive : 757
FalseNegative : 213
TruePostive  :  6542
```

Confusion matrix



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [0]:
```
# 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_)))

#***********************************************************************
***********************

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

# 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
```

```
X_train_Avgtfidf = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in list_of_sentance: # 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
    X_train_Avgtfidf.append(sent_vec)
    row += 1


#***********************************************************************
**********************


X_test_Avgtfidf = []; # the tfidf-w2v for each sentence/review is store
d in this list
row=0;
for sent in 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
    X_test_Avgtfidf.append(sent_vec)
    row += 1
```

In [0]:

```
from sklearn.ensemble import  AdaBoostClassifier, GradientBoostingClass
ifier
clf=GradientBoostingClassifier()
#param_grid={ 'n_estimators' : np.arange(20,200,20),
#              'max_depth' :  [5,10,15,20,30],
#              'learning_rate' : [0.1,0.001] }
param_grid={ 'n_estimators' : [10,50,100,200],
             'max_depth' :  [5,10,20]}
gcv=RandomizedSearchCV(clf,param_grid,cv=5)
gcv.fit(X_train_Avgtfidf,y_train)
print(gcv.best_params_)
print(gcv.best_score_)
```

```
{'n_estimators': 200, 'max_depth': 5}
0.8721875
```

In [0]:

```
hyperparameters=[(i['max_depth'],i['n_estimators']) for i in gcv.cv_res
ults_['params']]

depth         = [i[0] for i in hyperparameters]
n_estimators  = [i[1] for i in hyperparameters]

train_score = gcv.cv_results_['mean_train_score'].tolist()
test_score  = gcv.cv_results_['mean_test_score'].tolist()

train_score=  list(map(lambda x : round(x,2)*100,train_score))
test_score=  list(map(lambda x : round(x,2)*100,test_score))

print(depth)
```

```
print(n_estimators)
print(train_score)
print(test_score)

print("ploting 3d grap")

from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10, 6))

ax1 = plt.axes(projection='3d')

ax1.plot3D(depth, n_estimators , train_score , 'red',label="train_scor
e")
ax1.set_xlabel('depth')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('train_score')
#ax1.label_outer()

#ax1.legend()
ax1.scatter3D(depth, n_estimators, train_score , c=train_score, cmap='G
reens',label="train_score")

ax1.plot3D(depth, n_estimators , test_score , 'blue',label="test_score"
)
ax1.scatter3D(depth, n_estimators, test_score , c=test_score, cmap='OrR
d',label="test_score")


print("ploting Heat Map")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

results_df=pd.DataFrame(gcv.cv_results_)

#df2=pd.DataFrame(train_score,test_score)

df_params = results_df['params'].apply(pd.Series)

df3=pd.concat([results_df,df_params],axis=1).drop('params',axis=1)
```

```
#df3=pd.DataFrame(depth,n_estimators,test_score,train_score)

final_df1_test = df3.pivot("max_depth", "n_estimators", "mean_test_scor
e")
sns.heatmap(final_df1_test, annot=True ,ax=ax1 )



final_df_train = df3.pivot("max_depth", "n_estimators", "mean_train_sco
re")
sns.heatmap(final_df_train, annot=True ,ax=ax2)

plt.show()
#fig.show()
```

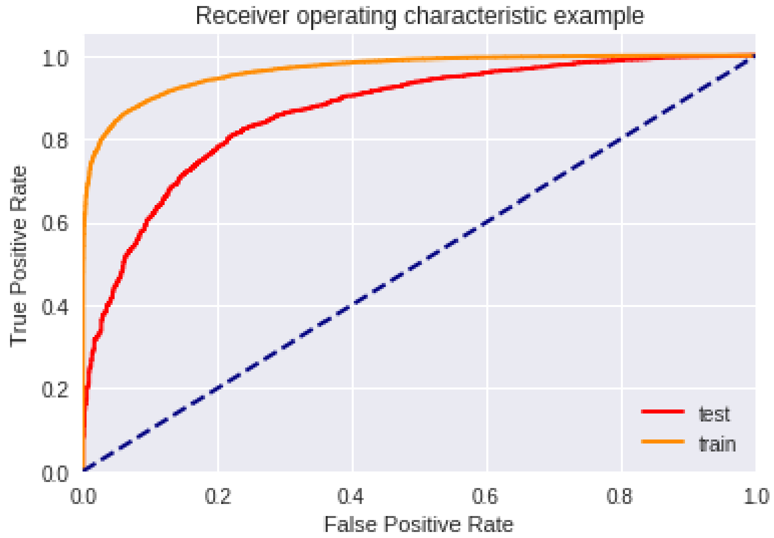
In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.ensemble import GradientBoostingClassifier

clf1=GradientBoostingClassifier(n_estimators=200,max_depth=5)
clf1.fit(X_train_Avgtfidf,y_train)
sig_clf = CalibratedClassifierCV(clf1, method="sigmoid" ,cv= 5)
sig_clf.fit(X_train_Avgtfidf, y_train)

pred = sig_clf.predict_proba(X_test_Avgtfidf)[:,1]
pred_train = sig_clf.predict_proba(X_train_Avgtfidf)[:,1]


pred_train_without_CCV=clf1.predict(X_train_Avgtfidf)
pred_without_CCV=clf1.predict(X_test_Avgtfidf)
```

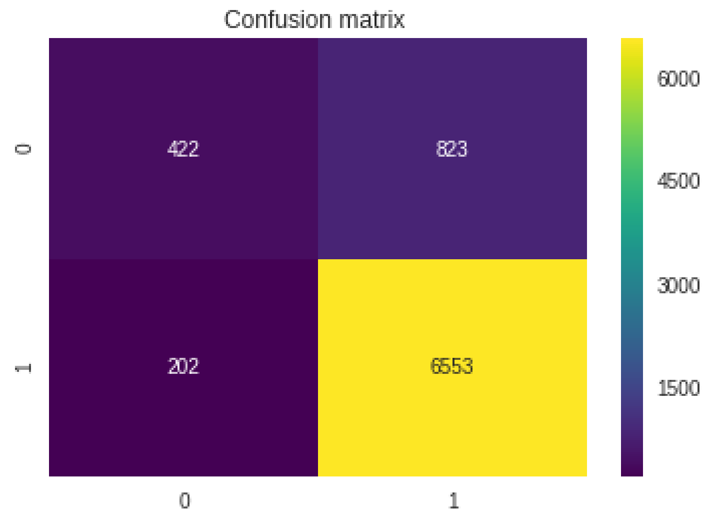
```
print("Accuracy Score : ",accuracy_score(y_test,pred_without_CCV)*100)
print("Precision Score : ",precision_score(y_test,pred_without_CCV)*100
)
print("Recall Score : ",recall_score(y_test,pred_without_CCV)*100)
print("F1 Score : ",f1_score(y_test,pred_without_CCV)*100)

print("        ")
print("Classification Report")
print(classification_report(y_test,pred_without_CCV))
print("        ")

fpr_train,tpr_train,thresholds_train=roc_curve(y_train,pred_train)
print("AUC Score for train data :",metrics.auc(fpr_train,tpr_train))

fpr,tpr,thresholds=roc_curve(y_test,pred)
print("AUC Score for test data :",metrics.auc(fpr,tpr))

print("        ")


plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw,label='test')
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw,label='train')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


print("       ")

tn, fp, fn, tp=confusion_matrix(y_test,pred_without_CCV).ravel()
```

```
print("""
TrueNegative : {}
FalsePostive : {}
FalseNegative : {}
TruePostive  :  {}""".format(tn, fp, fn, tp))
print("        ")
print("        ")

confusionmatrix_DF=pd.DataFrame(confusion_matrix(y_test,pred_without_CC
V),columns=['0','1'],index=['0','1'])
sns.heatmap(confusionmatrix_DF,annot=True,fmt='g',cmap='viridis')
plt.title("Confusion matrix ")
plt.show()
```

```
Accuracy Score :  87.1875
Precision Score :  88.84219088937093
Recall Score :  97.00962250185047
F1 Score :  92.7464439883943

Classification Report
              precision    recall  f1-score   support

           0       0.68      0.34      0.45      1245
           1       0.89      0.97      0.93      6755

   micro avg       0.87      0.87      0.87      8000
   macro avg       0.78      0.65      0.69      8000
weighted avg       0.86      0.87      0.85      8000


AUC Score for train data : 0.9655638353235834
AUC Score for test data : 0.8657696366517142
```




```
TrueNegative : 422
FalsePostive : 823
FalseNegative : 202
TruePostive  :  6553
```



[6] Conclusions

In [2]:

```
from prettytable import PrettyTable

x = PrettyTable()

print("RandomForest Table")
x.field_names = ["RandomForest with Different Vectorization" , "n_estim
ators" , "max_depth" , 'Test_Accuracy','F1-Score','AUC_Score']


x.add_row([ "RF with BOW" , 100 ,500  , 85.55 ,  92.01 , 83.13 ])
x.add_row([ "RF with TFIDF" , 100 ,50 , 84.975 , 91.708 ,83.52])
x.add_row([ "Rf with AVG_W2V" , 10 , 100, 85.9,  92.24 ,83.62 ])
x.add_row([ "Rf with AVG_W2VTFIDF" , 455 , 50 , 85.97,  92.27 , 83.86
])


print(x)
```

```
RandomForest Table
```

```
+-----------------------------------------+--------------+-----------
+--------------+---------+----------+
| RandomForest with Different Vectorization | n_estimators | max_depth
| Test_Accuracy | F1-Score | AUC_Score |
+-----------------------------------------+--------------+-----------
+--------------+---------+----------+
|                RF with BOW                |     100      |    500
|     85.55     |  92.01   |   83.13   |
|               RF with TFIDF               |     100      |     50
|     84.975    |  91.708  |   83.52   |
|              Rf with AVG_W2V              |      10      |    100
|      85.9     |  92.24   |   83.62   |
|            Rf with AVG_W2VTFIDF           |     455      |     50
|     85.97     |  92.27   |   83.86   |
+-----------------------------------------+--------------+-----------
+--------------+---------+----------+
```

In [3]:

```
x1 = PrettyTable()

x1.field_names = ["GBDT with Different Vectorization" , "n_estimators"
, "max_depth" , 'Test_Accuracy','F1-Score','AUC_Score']

x1.title = "GradientBoostingDescisionTree Table"
x1.add_row([ "XGBOOST with BOW with 100000 rows" , 200 ,10  , 86.8625 ,
  92.65 , 91.28])
x1.add_row([ "GDBt with BOW" , 200 ,10  , 86.8625 ,  92.65 , 90.56])
x1.add_row([ "GDBt with TFIDF" , 200 ,5 , 85.475, 91.92,82.214])
x1.add_row([ "GDBt with AVG_W2V" , 100 , 5, 87,  92.79 ,88.47 ])
x1.add_row([ "GDBt with AVG_W2VTFIDF" , 200 , 5 , 85.97,  92.27 , 86.57
 ])

print(x1)
```

```
+-----------------------------------+--------------+-----------+-------
--------+---------+----------+
| GBDT with Different Vectorization | n_estimators | max_depth | Test_A
ccuracy | F1-Score | AUC_Score |
+-----------------------------------+--------------+-----------+-------
--------+---------+----------+
```

```
| XGBOOST with BOW with 100000 rows |     200      |     10     |    86.
8625    |  92.65    |   91.28    |
|           GDBt with BOW            |     200      |     10     |    86.
8625    |  92.65   |   90.56    |
|          GDBt with TFIDF           |     200      |     5      |     8
5.475    |  91.92    |   82.214   |
|         GDBt with AVG_W2V          |     100      |     5      |
87      |  92.79    |   88.47    |
|      GDBt with AVG_W2VTFIDF       |     200      |     5      |     8
5.97     |  92.27    |   86.57    |
+------------------------------------+--------------+------------+-------
--------+----------+-----------+
```

Summary

- XGBoost Performs better then than the Random forest
- there is overfit in data at RandomForest w2v tfidf model
- there is less over fit on XG Boost Model

In [0]: