In [2]:

```
# Assignment 9 Random Forest
```

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 [3]:

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


import sqlite3
```

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

from sklearn.ensemble import RandomForestClassifier
from wordcloud import WordCloud, STOPWORDS
import xgboost as xgb

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
```

```
C:\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunk
ize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [4]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/sesha/OneDrive/Desktop/IMP/before/MINIPJ/Personal/AMAZON food revi
ew dataset/database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data 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 Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

Out[4]:

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [5]:

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

In [6]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [7]:

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

Out[7]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [8]:

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

Out[8]:

```
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 [9]:

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

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

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 [10]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [11]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

Out[11]:

```
(87775, 10)
```

In [12]:

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

Out[12]:

```
87.775
```

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 [13]:

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

display.head()
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [14]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [15]:

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

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

```
(87773, 10)
```

Out[15]:

```
1    73592
0    14181
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 [16]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [17]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
```

```
on with the china imports.
```

In [18]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
```

In [19]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [20]:

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

```
was way to hot for my blood, took a bite and did a jig  lol
```

==========

In [21]:

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [22]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [23]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', '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', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', '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', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn'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 [24]:

```
# 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%|██████████| 87773/87773 [00:31<00:00, 2752.46it/s]
```

In [25]:

```
preprocessed_reviews[1500]
```

Out[25]:

```
'way hot blood took bite jig lol'
```

In [26]:

```
print(len(preprocessed_reviews))
final.shape
```

```
87773
```

Out[26]:

```
(87773, 10)
```

[3.2] Preprocessing Review Summary

In [27]:

```
preprocessed_summary = []
# tqdm is for printing the status bar
for summary in tqdm(final['Summary'].values):
    summary = re.sub(r"http\S+", "", summary)
    # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
    summary = BeautifulSoup(summary, 'lxml').get_text()
    # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
    summary = decontracted(summary)
    summary = re.sub("\S*\d\S*", "", summary).strip() #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
    summary = re.sub('[^A-Za-z]+', ' ', summary) #remove spacial character: https://stackoverflow.com/a/5843547/4084039
    # https://gist.github.com/sebleier/554280
    summary = ' '.join(e.lower() for e in summary.split() if e.lower() not in stopwords)
    preprocessed_summary.append(summary.strip())
```

```
 37%|███▊      | 32596/87773 [00:06<00:10, 5299.49it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:273: UserWarning: "b'...'" looks like a filename, not markup. You should probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 70%|███████   | 61447/87773 [00:11<00:04, 5322.70it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:273: UserWarning: "b'...'" looks like a filename, not markup. You should probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 74%|███████▍  | 65191/87773 [00:12<00:04, 5312.34it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:273: UserWarning: "b'...'" looks like a filename, not markup. You should probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 95%|█████████▌| 83801/87773 [00:15<00:00, 5279.05it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:273: UserWarning: "b'...'" looks like a filename, not markup. You should probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████| 87773/87773 [00:16<00:00, 5269.61it/s]
```

In [28]:

```
preprocessed_reviews = [i + ' ' + j for i, j in zip(preprocessed_reviews,preprocessed_summary)]
print(preprocessed_reviews[1500])
```

```
way hot blood took bite jig lol hot stuff
```

[4] Featurization

[4.1] BAG OF WORDS WITH RANDOM SPLITTING

In [29]:

```
Y = final['Score'].values
X = np.array(preprocessed_reviews)
```

In [30]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

#X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33,
shuffle=False,random_state=0)
#X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train,
test_size=0.33,shuffle=False,random_state=0)# this is for time series split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33) # this is random splittin
g
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, test_size=0.33) # this is random
splitting


print(X_train.shape, Y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

print("="*100)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, Y_train.shape)
print(X_cv_bow.shape, Y_cv.shape)
print(X_test_bow.shape, Y_test.shape)
print("="*100)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(39400,) (39400,)
(19407,) (19407,)
(28966,) (28966,)
====================================================================================================

After vectorizations
(39400, 38617) (39400,)
(19407, 38617) (19407,)
(28966, 38617) (28966,)
====================================================================================================

NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

[4.2] Bi-Grams and n-Grams.

In [31]:

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

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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
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_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_s
hape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [32]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

train_tf_idf = tf_idf_vect.transform(X_train)
cv_tf_idf = tf_idf_vect.transform(X_cv)
test_tf_idf = tf_idf_vect.transform(X_test)
print("the type of count vectorizer ",type(train_tf_idf))
print("the shape of out text TRAIN TFIDF vectorizer ",train_tf_idf.get_shape())
print("the shape of out text CV TFIDF vectorizer ",cv_tf_idf.get_shape())
print("the shape of out text TEST TFIDF vectorizer ",test_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams in train ", train_tf_idf.get
_shape()[1])
```

```
some sample features(unique words in the corpus) ['aa', 'ability', 'able', 'able buy', 'able
drink', 'able eat', 'able enjoy', 'able find', 'able finish', 'able get']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TRAIN TFIDF vectorizer  (39400, 24854)
the shape of out text CV TFIDF vectorizer  (19407, 24854)
the shape of out text TEST TFIDF vectorizer  (28966, 24854)
the number of unique words including both unigrams and bigrams in train  24854
```

[4.4] Word2Vec

In [33]:

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

In [34]:

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

In [35]:

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

In [36]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(sent_of_train,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=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
[('awesome', 0.7874200940132141), ('excellent', 0.7831012606620789), ('good', 0.7759498953819275),
('fantastic', 0.7616086602210999), ('wonderful', 0.7502356767654419), ('amazing',
0.7498698234558105), ('terrific', 0.7245891094207764), ('fabulous', 0.7076520323753357),
('perfect', 0.7039331197738647), ('decent', 0.6629480123519897)]
==================================================
[('nastiest', 0.8207853436470032), ('tastiest', 0.7029910087585449), ('greatest',
0.6985799074172974), ('best', 0.679274320602417), ('disgusting', 0.6786993741989136),
('experienced', 0.6721776723861694), ('coolest', 0.6439719200134277), ('terrible',
0.6356139779090881), ('horrible', 0.602164626121521), ('tasted', 0.5933910012245178)]
```

In [37]:

```
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  12443
sample words  ['happy', 'product', 'great', 'value', 'compared', 'price', 'pet', 'stores', 'love',
'healthy', 'lasts', 'long', 'not', 'harm', 'animals', 'person', 'eats', 'clean', 'food',
'craving', 'salty', 'snack', 'find', 'mom', 'organic', 'market', 'fits', 'bill', 'husband',
'loves', 'crunchy', 'tasty', 'single', 'serving', 'bags', 'case', 'always', 'top', 'fridge',
'grab', 'run', 'said', 'chips', 'fattening', 'taste', 'received', 'free', 'sample', 'planters', 'n
ut']
```

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

[4.4.1.1] Avg W2v

In [38]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(sent_of_cv): # for each review/sentence
    sent_vec_cv = np.zeros(50) # as word vectors are of zero length 50, you 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec_cv += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec_cv /= cnt_words
    sent_vectors_cv.append(sent_vec_cv)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 19407/19407 [00:
22<00:00, 852.70it/s]
```

```
19407
50
```

In [39]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train= []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(sent_of_train): # for each review/sentence
    sent_vec_train = np.zeros(50) # as word vectors are of zero length 50, you 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec_train += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec_train /= cnt_words
    sent_vectors_train.append(sent_vec_train)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 39400/39400 [00:
44<00:00, 880.25it/s]
```

```
39400
50
```

In [40]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test= []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(sent_of_test): # for each review/sentence
    sent_vec_test = np.zeros(50) # as word vectors are of zero length 50, you 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec_test += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec_test /= cnt_words
```

```
    sent_vectors_test.append(sent_vec_test)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 28966/28966 [00:
34<00:00, 842.92it/s]
```

```
28966
50
```

[4.4.1.2] TFIDF weighted W2v

In [41]:

```
# 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 value
dictionary= dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [42]:

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

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

```
100%|██████████████████████████████████████████████████████████████████████████| 19407/19407 [03
:48<00:00, 79.56it/s]
```

In [43]:

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

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
```

```
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████████████████| 28966/28966 [05
:36<00:00, 85.96it/s]
```

In [44]:

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

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

```
100%|██████████████████████████████████████████████████████████████████████████| 39400/39400 [07
:37<00:00, 86.10it/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

[5.1] Applying RF

[5.1.1] Applying Random Forests on BOW, SET 1

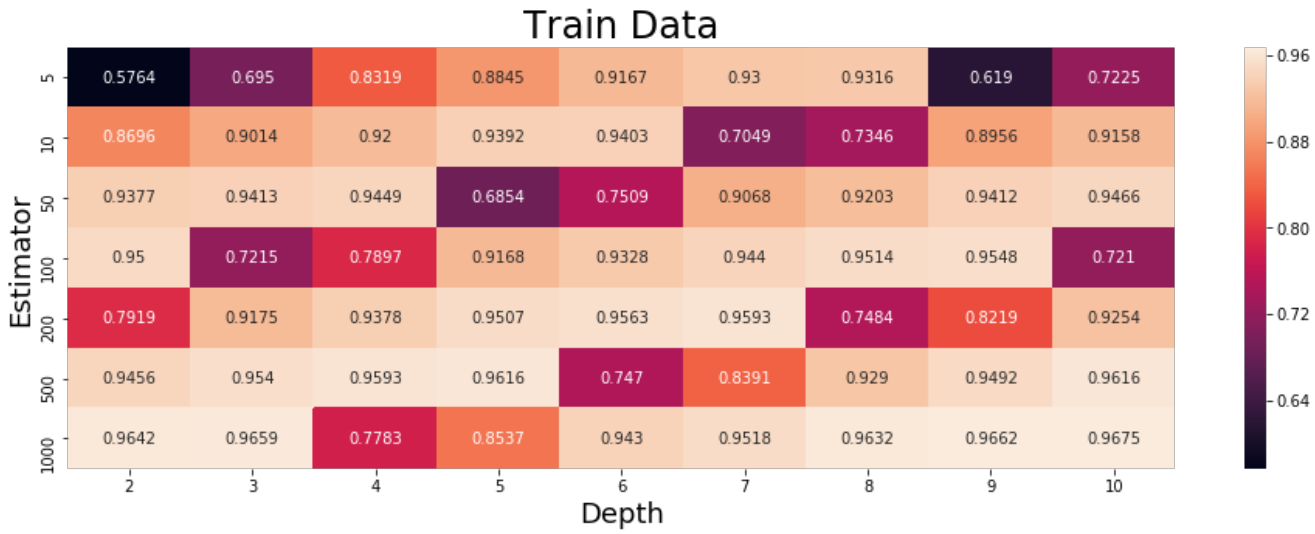
In [45]:

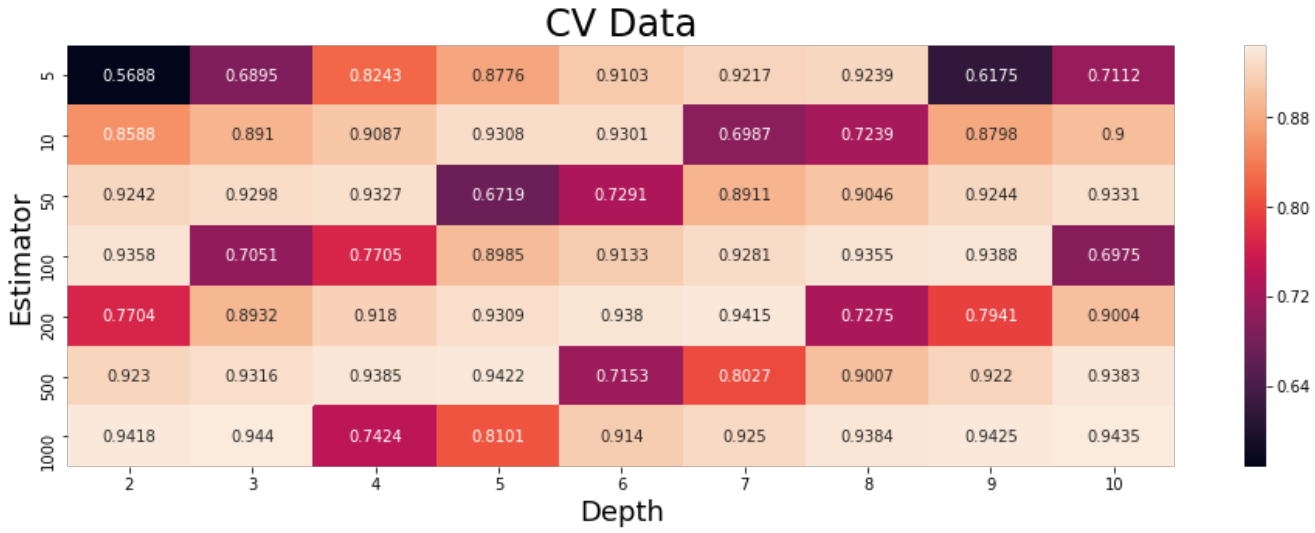
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import GridSearchCV

#Declaring the values for the chosen hyper pamameters
n_estimators = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
hyper_param = {'n_estimators':n_estimators, 'max_depth':depth}

#Using Gridsearch for the Hyperparameter tuning and Random Forest Classifier on Bow
clf = GridSearchCV(RandomForestClassifier(class_weight = 'balanced'),hyper_param,
verbose=1,scoring='roc_auc',n_jobs=-1,pre_dispatch=2)
clf.fit(X_train_bow,Y_train)
opt_estimator_bow, opt_depth_bow = clf.best_params_.get('n_estimators'), clf.best_params_.get('max_
depth')

#Computing the train_auc and cv_auc
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

#Heatmap for train_auc
df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Heatmap for cv_auc
df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Printing the Max Depth and Optimum value of number of estimators
print("Max depth is = ", opt_depth_bow , " Optimal value of n_estimator :", opt_estimator_bow)

#Cv auc scores
print("--------------------------")
```

```
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf =RandomForestClassifier(max_depth=opt_depth_bow, n_estimators=opt_estimator_bow,class_weight =
'balanced')
clf.fit(X_train_bow,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(X_test_bow)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

```
Fitting 3 folds for each of 63 candidates, totalling 189 fits
```

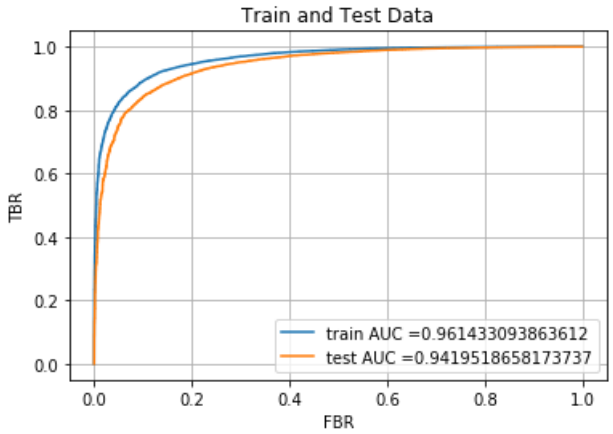
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 189 out of 189 | elapsed: 11.6min finished
```






```
Max depth is =  9  Optimal value of n_estimator : 1000
--------------------------
Cv auc scores
```

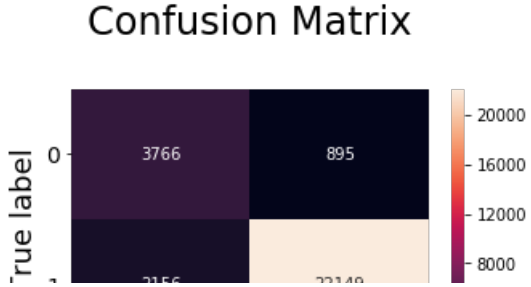
```
[0.5687926  0.68945945 0.82431893 0.87764611 0.91028735 0.9216565
 0.9239212  0.61752328 0.71115327 0.85883197 0.89101442 0.90865527
 0.93075669 0.93014649 0.69868794 0.72390699 0.87978531 0.90000244
 0.92418007 0.92977165 0.93273237 0.67194882 0.72911707 0.89105417
 0.90462582 0.9243887  0.9331144  0.93583268 0.70514164 0.77053006
 0.89854327 0.91332117 0.9280778  0.93547714 0.93882991 0.69754848
 0.77044555 0.8932284  0.91797189 0.93092409 0.93801267 0.94152455
 0.72750613 0.79408998 0.90042276 0.92300047 0.93156927 0.93851228
 0.94217457 0.71529619 0.80270863 0.9007102  0.92198806 0.93830928
 0.94184363 0.94399024 0.74243722 0.81008783 0.91404436 0.9250263
 0.93836949 0.94247983 0.9435161 ]
Maximun Auc value : 0.9439902361421594
```



In [46]:

```
#Confusion Matrix

print("Train confusion matrix")
print(confusion_matrix(Y_train, clf.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, clf.predict(X_test_bow)))

cm = confusion_matrix(Y_train, clf.predict(X_train_bow))
cm = confusion_matrix(Y_test, clf.predict(X_test_bow))
tn, fp, fn, tp = cm.ravel()
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# Code for drawing seaborn heatmaps
class_names = ['0','1']
df_heatmap = pd.DataFrame(cm, index=class_names, columns=class_names )
fig = plt.figure(figsize=(5,3))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predict label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Train confusion matrix
[[ 5410   902]
 [ 2514 30574]]
Test confusion matrix
[[ 3766   895]
 [ 2156 22149]]
```



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

In [47]:

```
#pip3 install wordcloud
#pip3 install xgboost

#from sklearn.ensemble import RandomForestClassifier
#from wordcloud import WordCloud, STOPWORDS
#import xgboost as xgb


# Please write all the code with proper documentation
clf = RandomForestClassifier(max_depth= 10, n_estimators=500,class_weight='balanced')
clf.fit(X_train_bow,Y_train)

feat = clf.feature_importances_
index=np.argsort(feat)
index_rev=index[::-1]
names=vectorizer.get_feature_names()
index_rev=index_rev[:30]

text=" "
for i in range(30):
    text = text + " " + names[index_rev[i]]
wordcloud = WordCloud(width=500, height=200, max_words=20).generate(text)

plt.figure(figsize=(12,12),facecolor='k' )
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```

[5.1.3] Applying Random Forests on TFIDF, SET 2

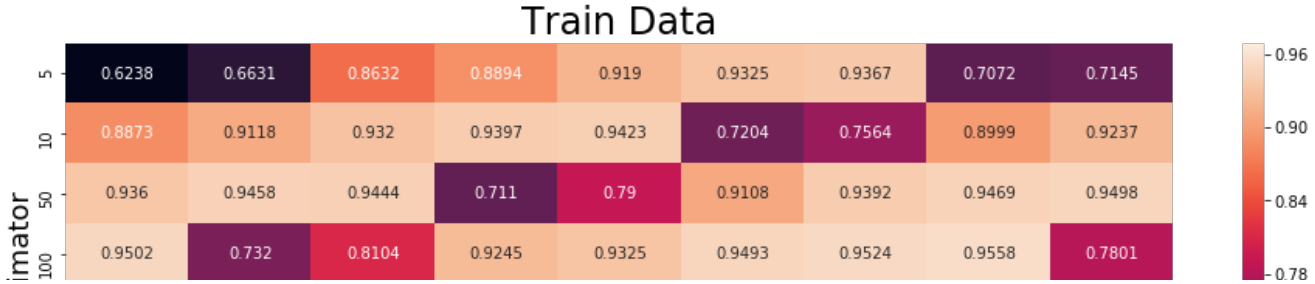
In [48]:

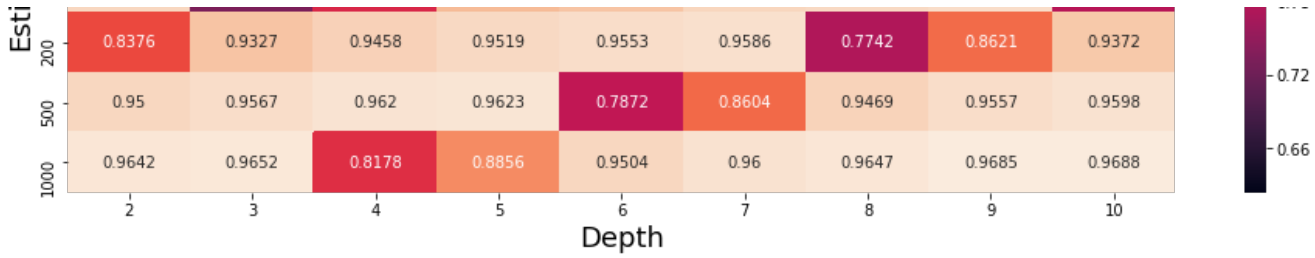
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import GridSearchCV

#Declaring the values for the chosen hyper pamameters
n_estimators = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
hyper_param = {'n_estimators':n_estimators, 'max_depth':depth}

#Using Gridsearch for the Hyperparameter tuning and Random Forest Classifier on Bow
clf = GridSearchCV(RandomForestClassifier(class_weight = 'balanced'),hyper_param,
```

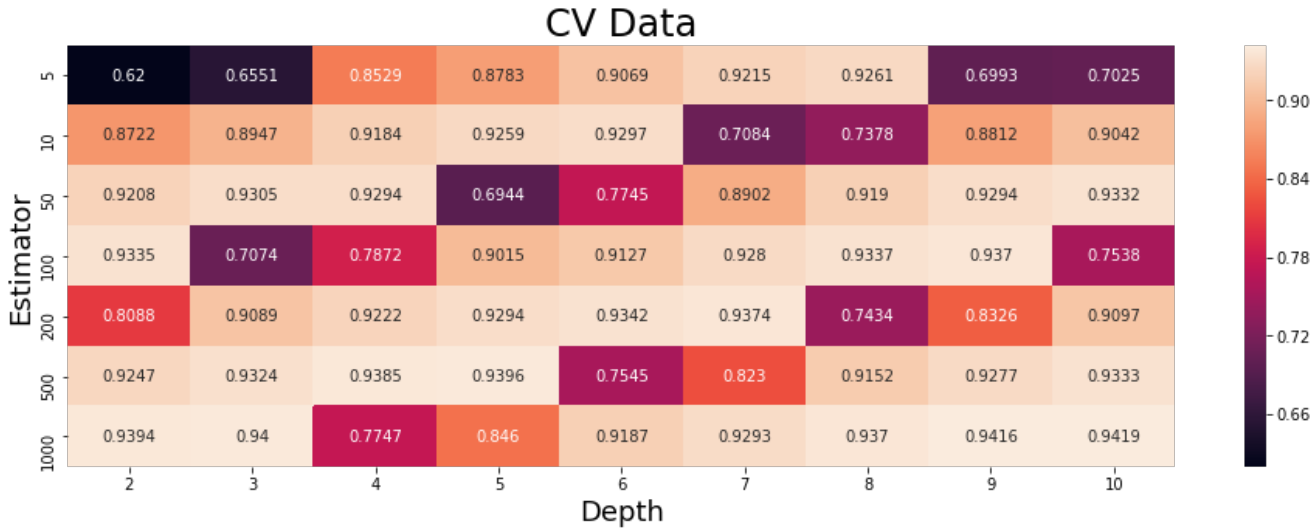
```
verbose=1,scoring='roc_auc',n_jobs=-1,pre_dispatch=2)
clf.fit(train_tf_idf,Y_train)
opt_estimator_tfidf, opt_depth_tfidf = clf.best_params_.get('n_estimators'), clf.best_params_.get('
max_depth')

#Computing the train_auc and cv_auc
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

#Heatmap for train_auc
df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Heatmap for cv_auc
df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Printing the Max Depth and Optimum value of number of estimators
print("Max depth is = ", opt_depth_tfidf , " Optimal value of n_estimator :", opt_estimator_tfidf)

#Cv auc scores
print("---------------------------")
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf =RandomForestClassifier(max_depth=opt_depth_bow, n_estimators=opt_estimator_bow,class_weight =
'balanced')
clf.fit(train_tf_idf,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(train_tf_idf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(test_tf_idf)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

```
Fitting 3 folds for each of 63 candidates, totalling 189 fits
```

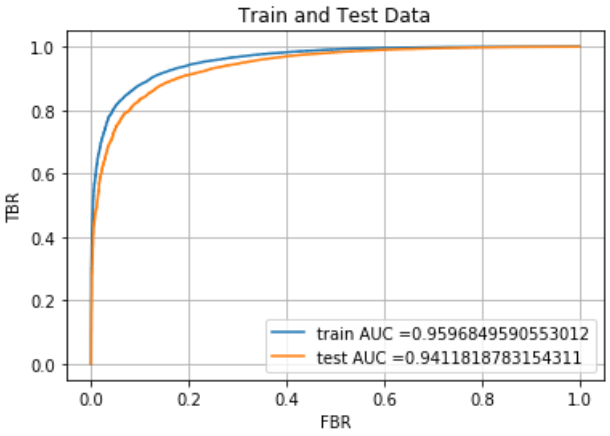
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   57.5s
[Parallel(n_jobs=-1)]: Done 189 out of 189 | elapsed:  5.1min finished
```








```
Max depth is =  10  Optimal value of n_estimator : 1000
--------------------------
Cv auc scores
[0.62001482 0.65507669 0.85290951 0.87831051 0.90693253 0.9214708
 0.92610251 0.69933478 0.70250577 0.87215426 0.89469184 0.91842648
 0.92593531 0.92967854 0.70839224 0.73781017 0.8812135  0.9042473
 0.92082239 0.93045982 0.92938741 0.69441759 0.77454937 0.89019636
 0.91896977 0.92944268 0.93316114 0.93347044 0.70738421 0.78719714
 0.90147545 0.91269569 0.92798044 0.93374879 0.93699045 0.75381466
 0.80884785 0.90889302 0.92220582 0.92943118 0.93421421 0.93741411
 0.7433877  0.83257441 0.90974709 0.92473287 0.93243828 0.93854988
 0.93962677 0.75445932 0.82299864 0.91521452 0.9276957  0.93334896
 0.93936141 0.94002525 0.77474041 0.84598703 0.91871666 0.92927632
 0.93696437 0.94163018 0.94190852]
Maximun Auc value : 0.9419085226365376
```

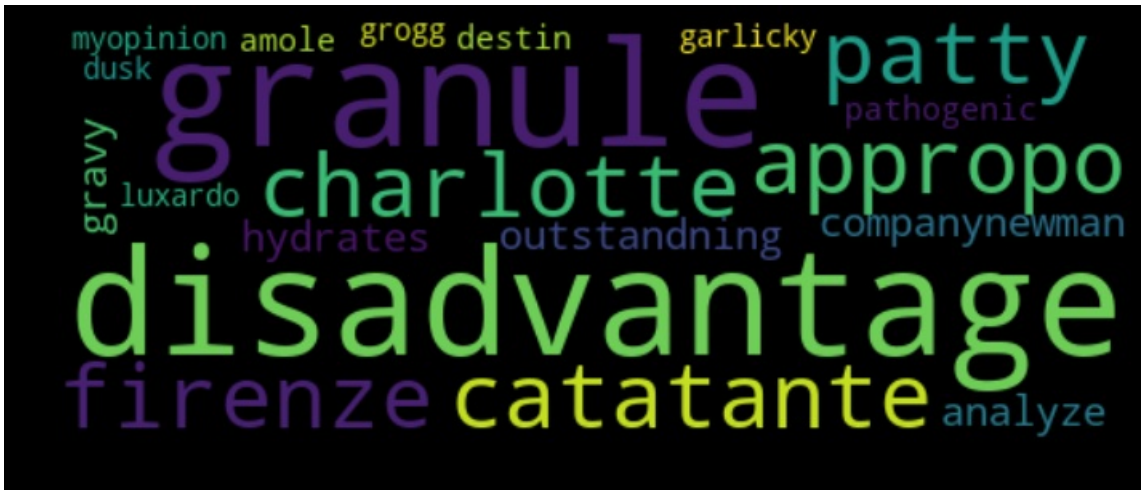


[5.1.4] Wordcloud of top 20 important features from SET 2

In [49]:

```
#pip3 install wordcloud
#pip3 install xgboost

#from sklearn.ensemble import RandomForestClassifier
#from wordcloud import WordCloud, STOPWORDS
#import xgboost as xgb
```

```
# Please write all the code with proper documentation
clf = RandomForestClassifier(max_depth= 10, n_estimators=500,class_weight='balanced')
clf.fit(train_tf_idf,Y_train)

feat = clf.feature_importances_
index=np.argsort(feat)
index_rev=index[::-1]
names=vectorizer.get_feature_names()
index_rev=index_rev[:30]

text=" "
for i in range(30):
    text = text + " " + names[index_rev[i]]
wordcloud = WordCloud(width=500, height=200, max_words=20).generate(text)

plt.figure(figsize=(12,12),facecolor='k' )
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```



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

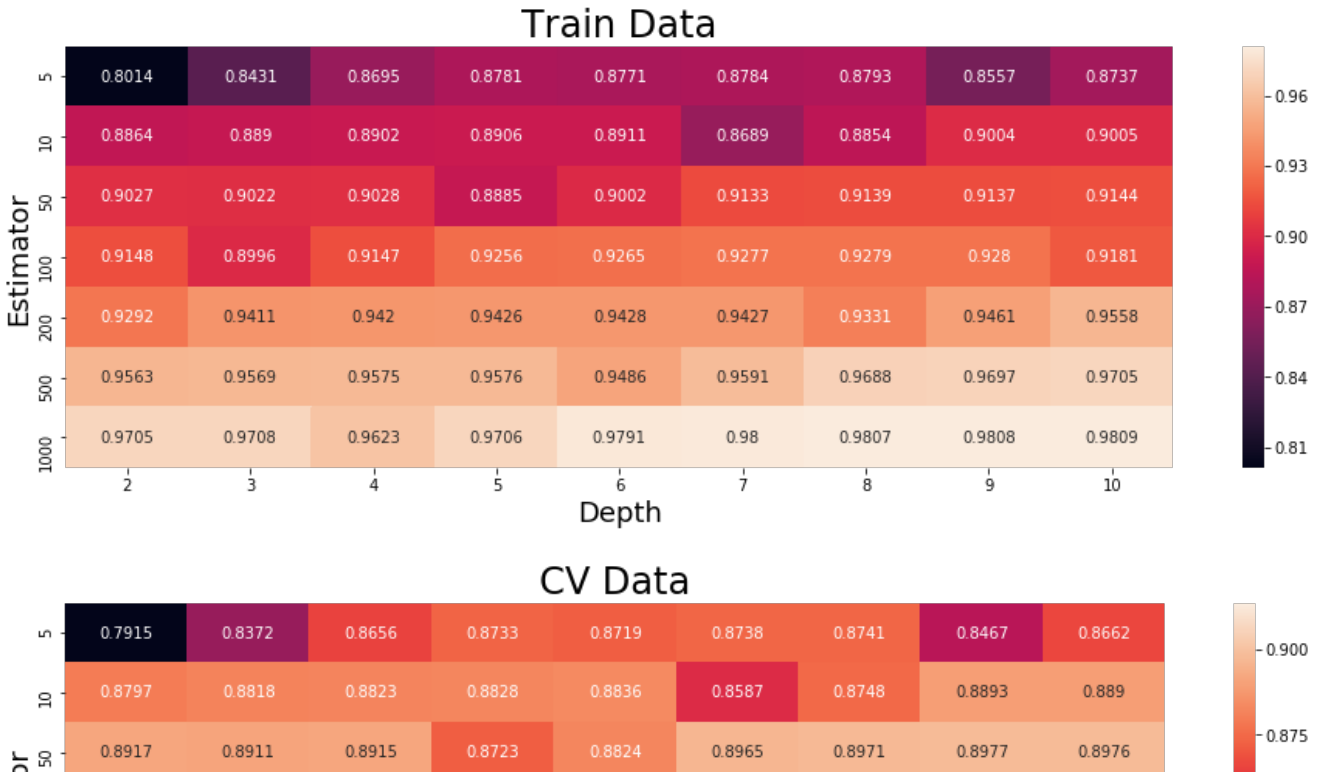
In [50]:

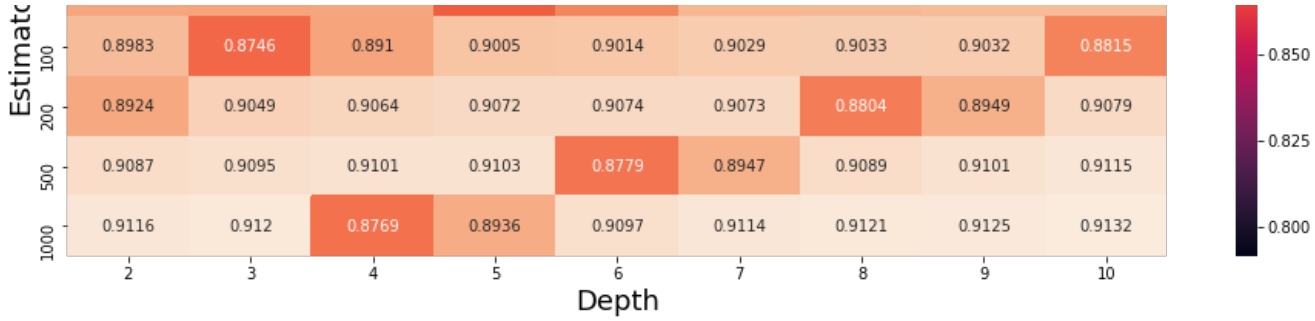
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import GridSearchCV

#Declaring the values for the chosen hyper pamameters
n_estimators = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
hyper_param = {'n_estimators':n_estimators, 'max_depth':depth}

#Using Gridsearch for the Hyperparameter tuning and Random Forest Classifier on Bow
clf = GridSearchCV(RandomForestClassifier(class_weight = 'balanced'),hyper_param,
verbose=1,scoring='roc_auc',n_jobs=-1,pre_dispatch=2)
clf.fit(sent_vectors_train,Y_train)
opt_estimator_avgw2v, opt_depth_avgw2v = clf.best_params_.get('n_estimators'), clf.best_params_.get
('max_depth')

#Computing the train_auc and cv_auc
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

#Heatmap for train_auc
df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Heatmap for cv_auc
```

```
#Heatmap for cv_auc
df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Printing the Max Depth and Optimum value of number of estimators
print("Max depth is = ", opt_depth_avgw2v , " Optimal value of n_estimator :",
opt_estimator_avgw2v)

#Cv auc scores
print("--------------------------")
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf =RandomForestClassifier(max_depth=opt_depth_avgw2v,
n_estimators=opt_estimator_avgw2v,class_weight = 'balanced')
clf.fit(sent_vectors_train,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(sent_vectors_test)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

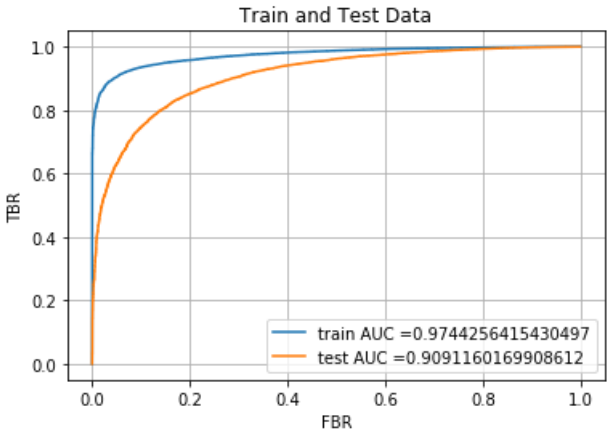
```
Fitting 3 folds for each of 63 candidates, totalling 189 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 189 out of 189 | elapsed: 30.1min finished
```






```
Max depth is =  10  Optimal value of n_estimator : 1000
--------------------------
Cv auc scores
[0.79151522 0.8371831  0.86555188 0.87334625 0.87192116 0.87384048
 0.87410215 0.84674081 0.86618043 0.87965374 0.88178715 0.88232848
 0.88276392 0.88361736 0.85874458 0.87482242 0.88932774 0.88900735
 0.89170146 0.8910819  0.89153404 0.87231564 0.88236527 0.89654881
 0.89713665 0.89765514 0.89764568 0.89827459 0.87456363 0.89103823
 0.90050253 0.90141198 0.90286689 0.90331043 0.90322267 0.88150575
 0.89239162 0.90492039 0.90639068 0.90719271 0.90743136 0.90731921
 0.8804053  0.89486794 0.90786798 0.90867181 0.90948399 0.91012563
 0.91033354 0.8779266  0.89471706 0.90894772 0.9100727  0.91145582
 0.91156134 0.9120345  0.87693369 0.89358346 0.90972524 0.91135433
 0.91207853 0.91253471 0.91324662]
Maximun Auc value : 0.9132466161550261
```



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

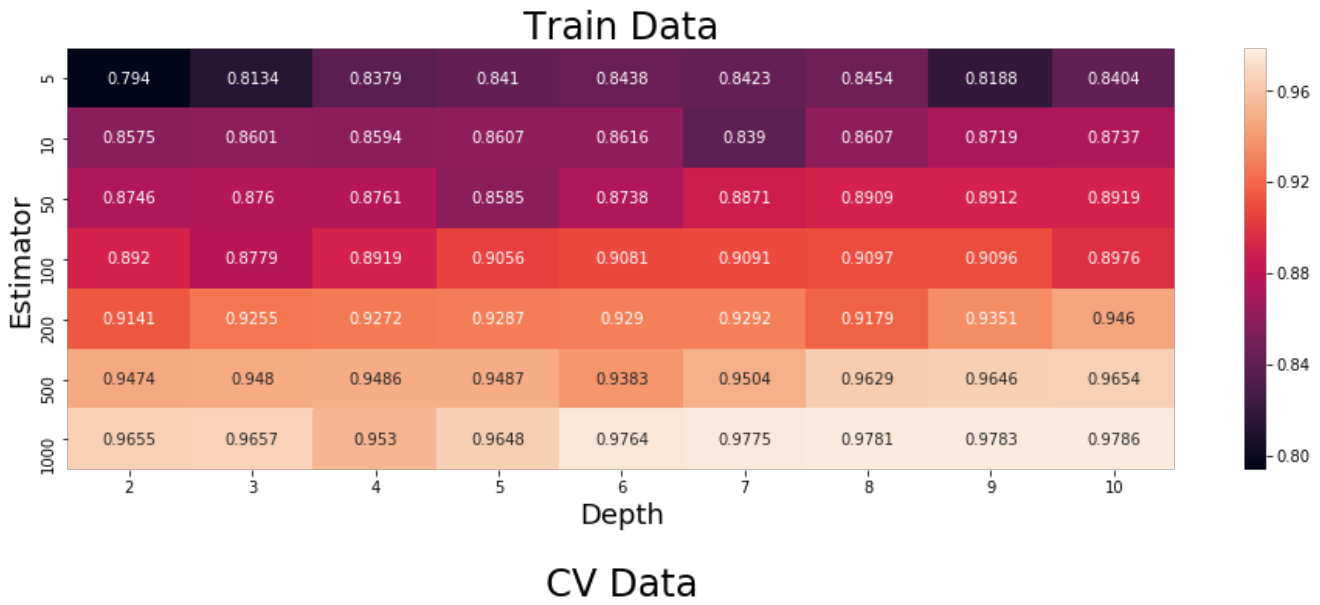
In [51]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import GridSearchCV

#Declaring the values for the chosen hyper pamameters
n_estimators = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
hyper_param = {'n_estimators':n_estimators, 'max_depth':depth}

#Using Gridsearch for the Hyperparameter tuning and Random Forest Classifier on Bow
clf = GridSearchCV(RandomForestClassifier(class_weight = 'balanced'),hyper_param,
verbose=1,scoring='roc_auc',n_jobs=-1,pre_dispatch=2)
clf.fit(tfidf_sent_vectors_train,Y_train)
opt_estimator_tfidfw2v, opt_depth_tfidfw2v = clf.best_params_.get('n_estimators'),
clf.best_params_.get('max_depth')

#Computing the train_auc and cv_auc
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

#Heatmap for train_auc
df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
```

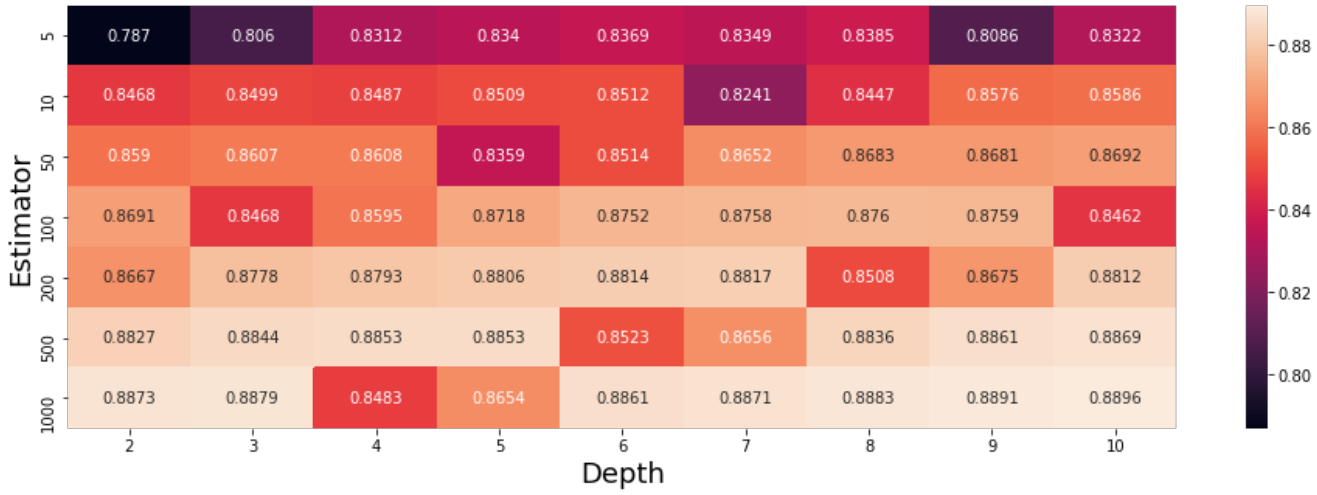
```
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Heatmap for cv_auc
df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

#Printing the Max Depth and Optimum value of number of estimators
print("Max depth is = ", opt_depth_tfidfw2v , " Optimal value of n_estimator :",
opt_estimator_tfidfw2v)

#Cv auc scores
print("--------------------------")
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf =RandomForestClassifier(max_depth=opt_depth_avgw2v,
n_estimators=opt_estimator_avgw2v,class_weight = 'balanced')
clf.fit(tfidf_sent_vectors_train,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(tfidf_sent_vectors_train)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(tfidf_sent_vectors_test)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

```
Fitting 3 folds for each of 63 candidates, totalling 189 fits
```

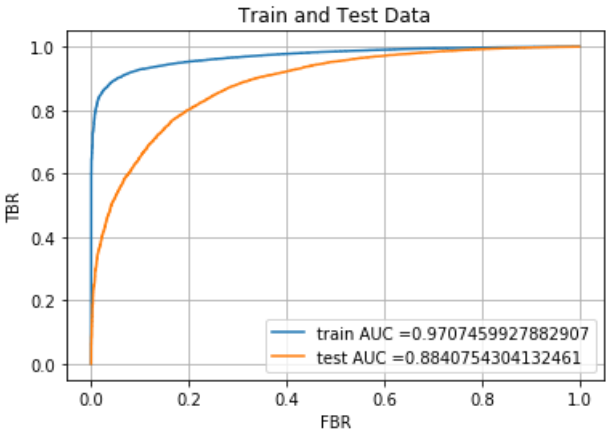
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 189 out of 189 | elapsed: 30.2min finished
```



CV Data




```
Max depth is =  10  Optimal value of n_estimator : 1000
---------------------------
Cv auc scores
[0.78699102 0.80603418 0.83122171 0.8340224  0.83691697 0.83489444
 0.83847081 0.80864663 0.83218539 0.84679296 0.84985048 0.84871701
 0.85093589 0.8512345  0.82408164 0.84474836 0.85759977 0.85863972
 0.85898381 0.860731   0.86084887 0.83593914 0.85137113 0.86523927
 0.86831358 0.86809224 0.86918637 0.86913316 0.84678997 0.85952323
 0.87183298 0.87517255 0.87579276 0.87601575 0.87593448 0.84624508
 0.86670191 0.87783201 0.87927639 0.88055576 0.88142723 0.88165647
 0.85076032 0.86751383 0.88123112 0.88269518 0.88444584 0.88528096
 0.88528995 0.85227926 0.86561193 0.88355033 0.88605621 0.88689625
 0.88732918 0.88788931 0.84827298 0.86537261 0.88613071 0.88713131
 0.88832059 0.88910155 0.88955783]
Maximun Auc value : 0.8895578342063878
```



[5.2] Applying GBDT using XGBOOST

[5.2.1] Applying XGBOOST on BOW, SET 1

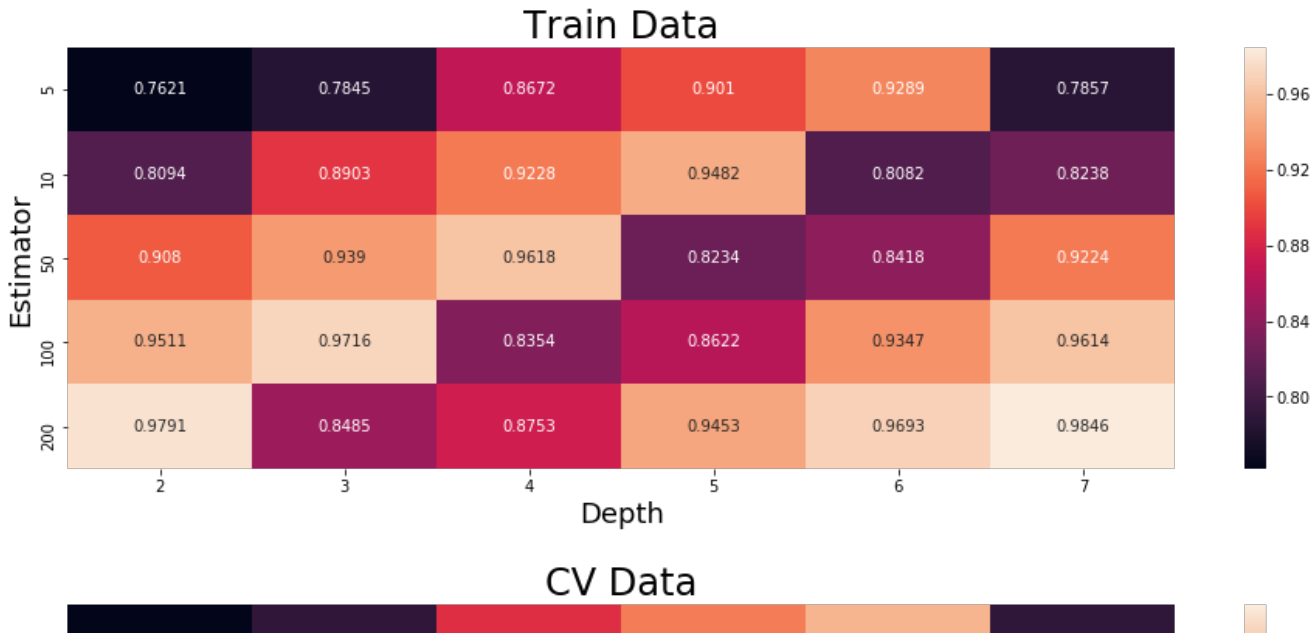
In [55]:

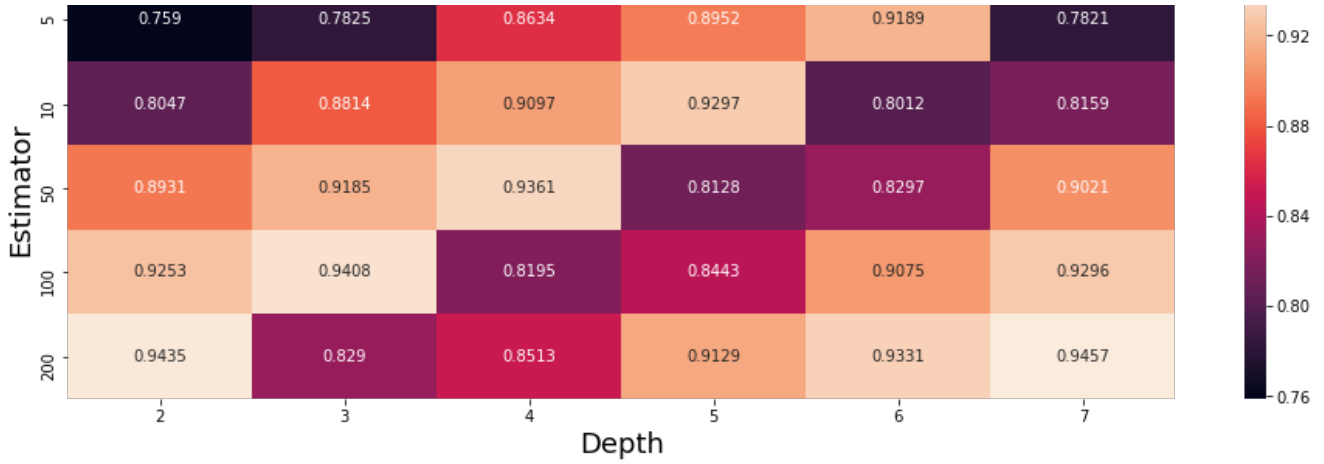
```
n_estimators = [5, 10, 50, 100, 200]
depth = [2, 3, 4, 5, 6, 7]
param = {'n_estimators':n_estimators, 'max_depth':depth}

clf = GridSearchCV(xgb.XGBClassifier(booster='gbtree',class_weight = 'balanced'),param,verbose=1,sc
oring='roc_auc',n_jobs=-1,pre_dispatch=2,cv=3)
clf.fit(X_train_bow,Y_train)
opt_estimator_bow_xg, opt_depth_bow_xg = clf.best_params_.get('n_estimators'), clf.best_params_.get
('max_depth')

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
```

```
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

print("Max depth is = ", opt_depth_bow_xg , " Optimal value of n_estimator :",
opt_estimator_bow_xg)

#Cv auc scores
print("--------------------------")
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf = xgb.XGBClassifier(max_depth=opt_depth_bow_xg, n_estimators=opt_estimator_bow_xg,random_state
=0 )
clf.fit(X_train_bow,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(X_test_bow)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

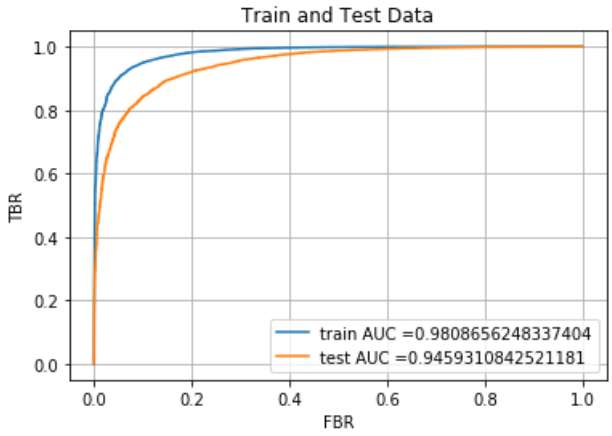
```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  9.7min
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed: 35.0min finished
```






```
Max depth is =  7  Optimal value of n_estimator : 200
--------------------------
Cv auc scores
[0.7589995  0.78247113 0.86341583 0.89515614 0.91892152 0.78210154
 0.80468028 0.88138155 0.90972488 0.92969707 0.80121011 0.81593237
 0.89306621 0.91852481 0.93612433 0.8127568  0.82972171 0.90214981
 0.92530757 0.94084057 0.81953777 0.84433495 0.90746569 0.92955882
 0.94352956 0.82903888 0.8513326  0.91292965 0.93311368 0.94568483]
Maximun Auc value : 0.9456848348636719
```



[5.2.2] Applying XGBOOST on TFIDF, SET 2

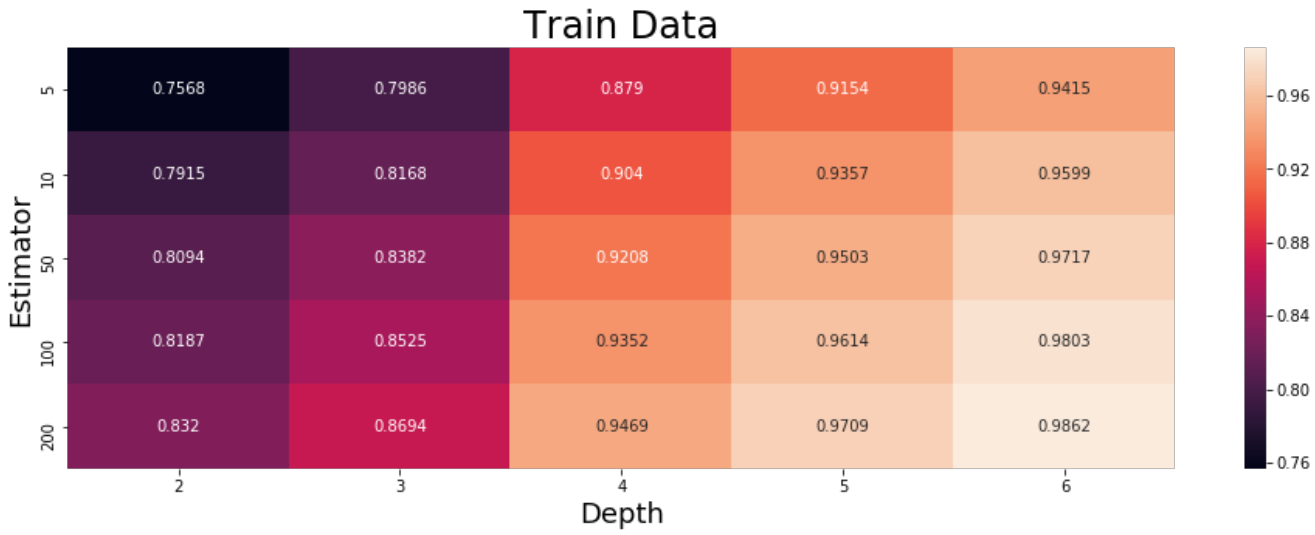
In [56]:

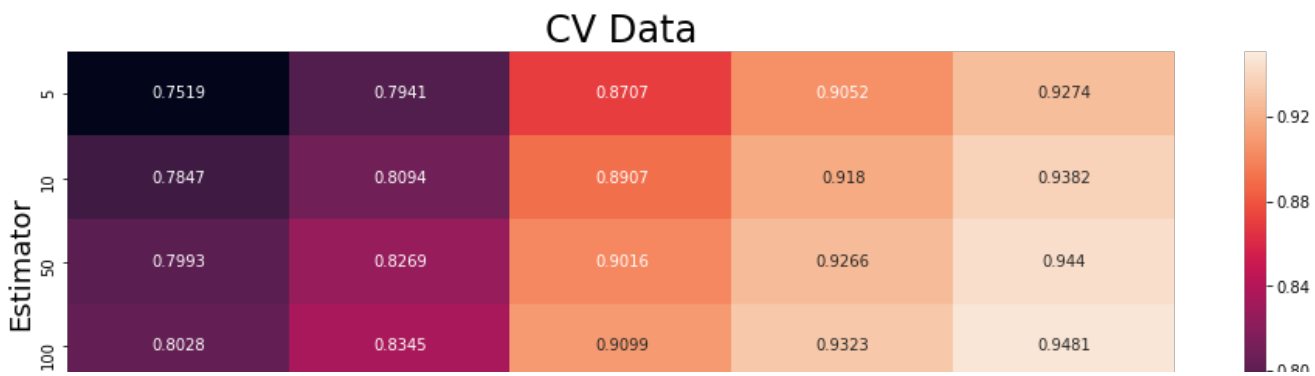
```
n_estimators = [5, 10, 50, 100, 200]
depth = [2, 3, 4, 5, 6]
param = {'n_estimators':n_estimators, 'max_depth':depth}

clf = GridSearchCV(xgb.XGBClassifier(booster='gbtree',class_weight = 'balanced'),param,verbose=1,sc
oring='roc_auc',n_jobs=-1,pre_dispatch=2,cv=3)
clf.fit(train_tf_idf,Y_train)
opt_estimator_tfidf_xg, opt_depth_tfidf_xg = clf.best_params_.get('n_estimators'),
clf.best_params_.get('max_depth')

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
```

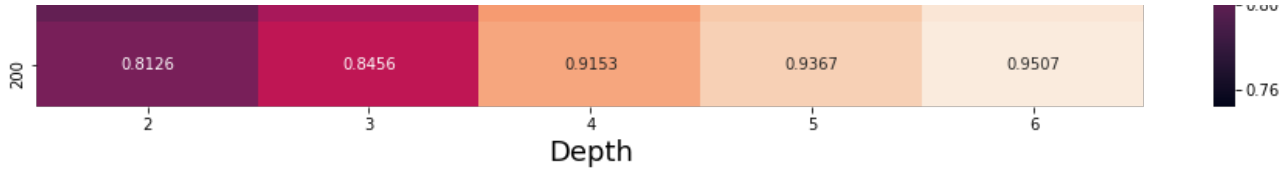
```
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

print("Max depth is = ", opt_depth_tfidf_xg , " Optimal value of n_estimator :",
opt_estimator_tfidf_xg)

#Cv auc scores
print("---------------------------")
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf = xgb.XGBClassifier(max_depth=opt_depth_tfidf_xg,
n_estimators=opt_estimator_tfidf_xg,random_state=0 )
clf.fit(train_tf_idf,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(train_tf_idf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(test_tf_idf)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
```

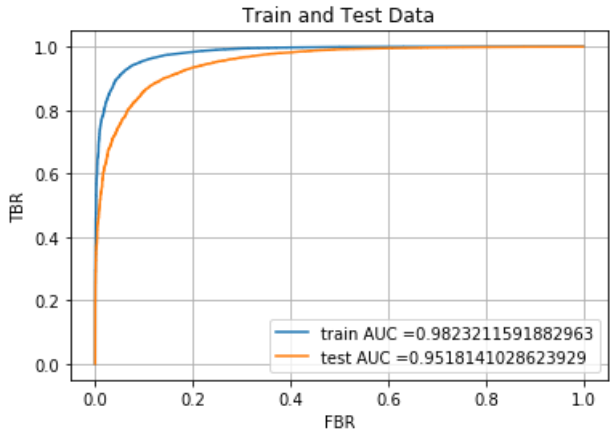
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done  75 out of  75 | elapsed: 11.2min finished
```








```
Max depth is =  6  Optimal value of n_estimator : 200
---------------------------
Cv auc scores
[0.75188902 0.79412142 0.87072952 0.90520453 0.92741483 0.78473037
 0.80939652 0.89074538 0.91798663 0.93822567 0.79926774 0.82690008
 0.90156896 0.92656417 0.94395159 0.8028296  0.83449238 0.90988056
 0.93232657 0.94805907 0.81255855 0.84557476 0.91530859 0.93667971
 0.95074341]
Maximun Auc value : 0.9507434131154211
```



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

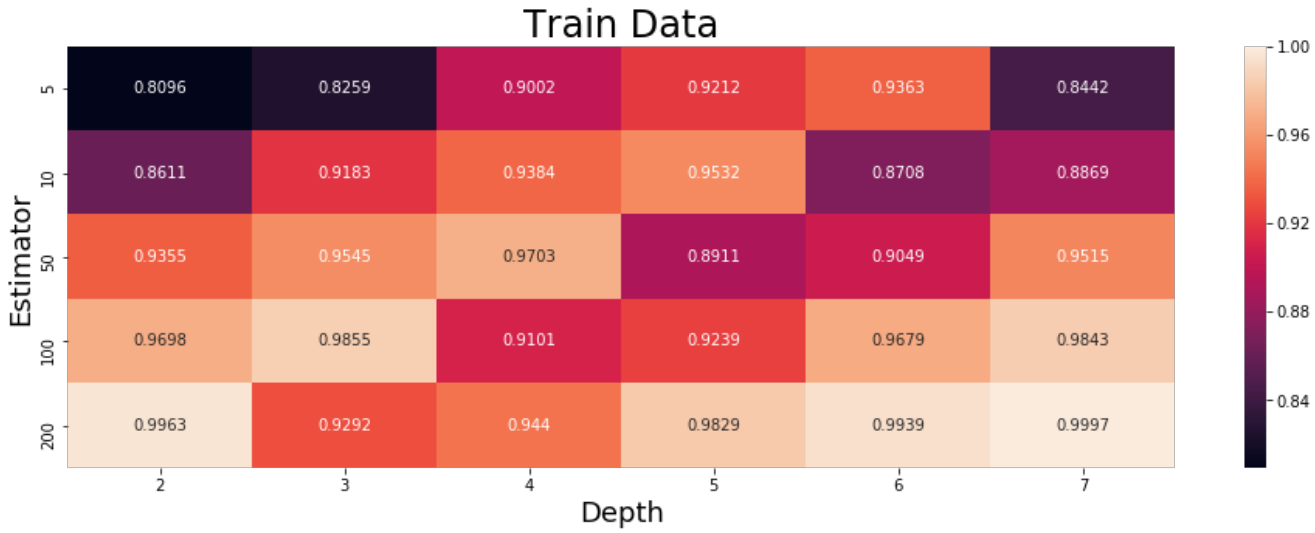
In [57]:

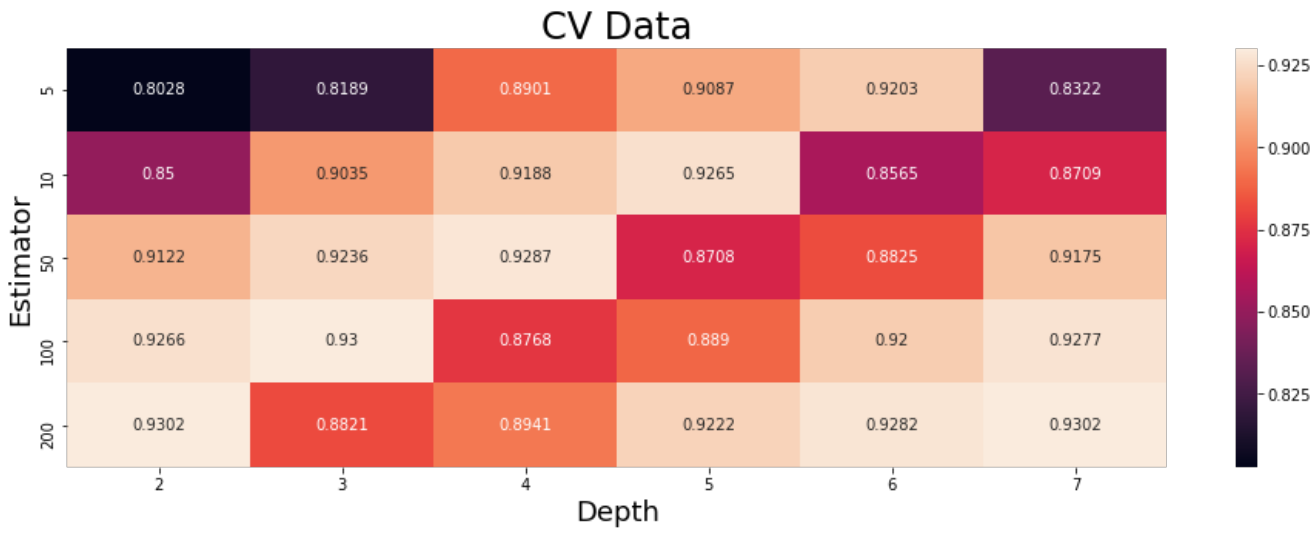
```
n_estimators = [5, 10, 50, 100, 200]
depth = [2, 3, 4, 5, 6, 7]
param = {'n_estimators':n_estimators, 'max_depth':depth}

clf = GridSearchCV(xgb.XGBClassifier(booster='gbtree',class_weight = 'balanced'),param,verbose=1,sc
oring='roc_auc',n_jobs=-1,pre_dispatch=2,cv=3)
train_avgw2v = np.array(sent_vectors_train)
clf.fit(train_avgw2v,Y_train)
opt_estimator_avgw2v_xg, opt_depth_avgw2v_xg = clf.best_params_.get('n_estimators'), clf.best_param
s_.get('max_depth')

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

print("Max depth is = ", opt_depth_avgw2v_xg , " Optimal value of n_estimator :",
opt_estimator_avgw2v_xg)

#Cv auc scores
print("---------------------------")
```

```
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf = xgb.XGBClassifier(max_depth=opt_depth_avgw2v_xg,
n_estimators=opt_estimator_avgw2v_xg,random_state=0 )
clf.fit(train_avgw2v,Y_train)

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(sent_vectors_test)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

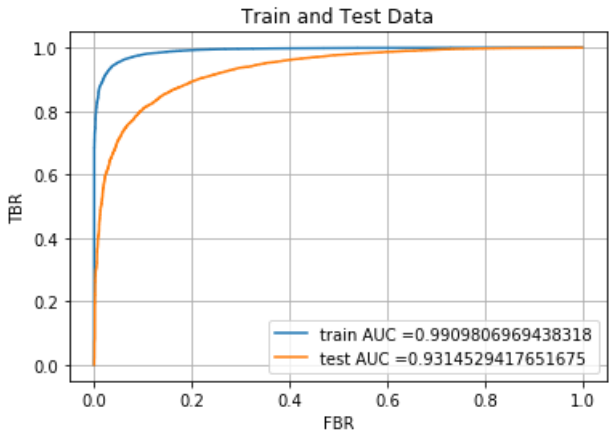
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:  7.8min finished
```






```
Max depth is =  6  Optimal value of n_estimator : 200
--------------------------
Cv auc scores
```

```
[0.80276124 0.81893761 0.89012039 0.9086701  0.92025787 0.83224843
 0.84998513 0.90345532 0.91879895 0.92648852 0.8564634  0.87089055
 0.9122201  0.92362004 0.92868181 0.87083482 0.88245507 0.91754852
 0.92657015 0.9299722  0.87675099 0.88902157 0.92004502 0.92771255
 0.93022946 0.88213594 0.89413641 0.92224985 0.92815172 0.93017106]
Maximun Auc value : 0.930229463915462
```



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

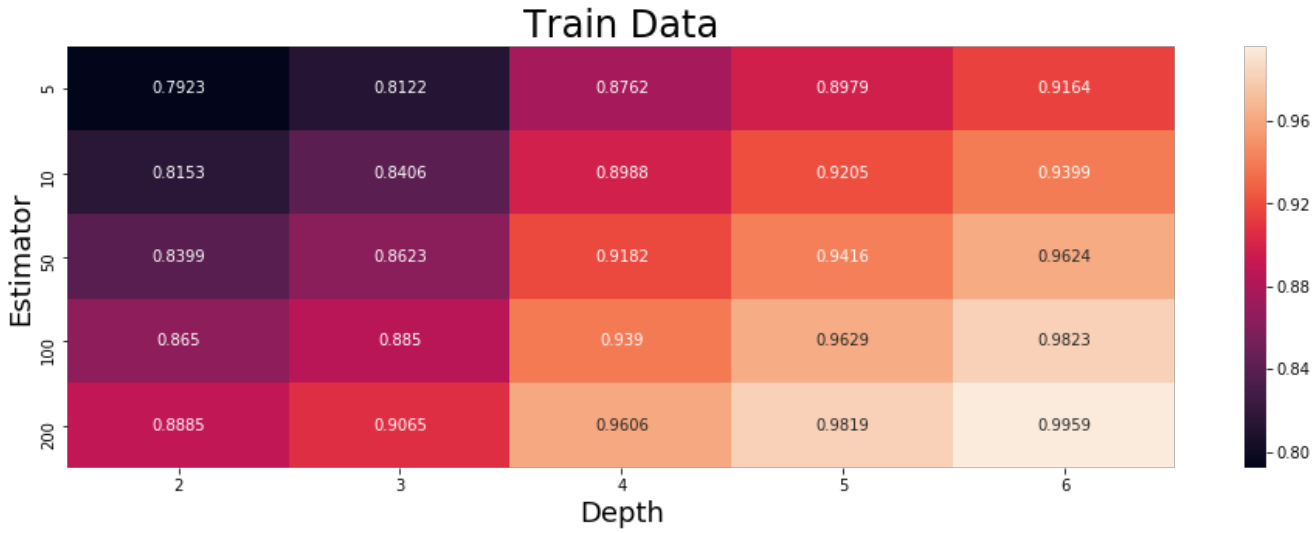
In [52]:

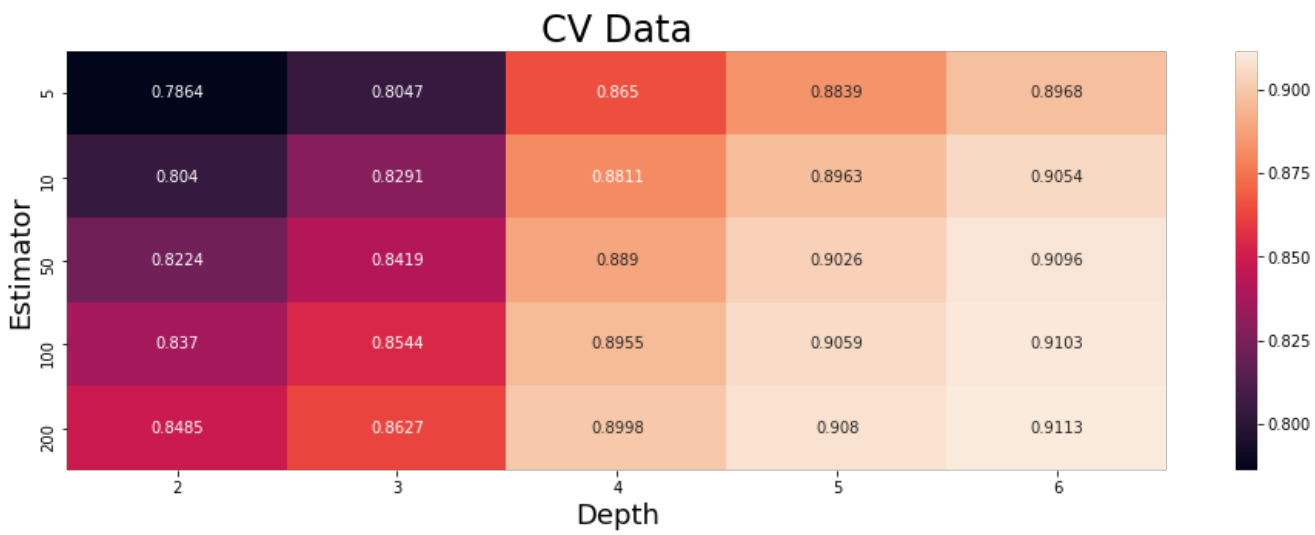
```
n_estimators = [5, 10, 50, 100, 200]
depth = [2, 3, 4, 5, 6]
param = {'n_estimators':n_estimators, 'max_depth':depth}

clf = GridSearchCV(xgb.XGBClassifier(booster='gbtree',class_weight = 'balanced'),param,verbose=1,sc
oring='roc_auc',n_jobs=-1,pre_dispatch=2,cv=3)
train_tfidfw2v = np.array(tfidf_sent_vectors_train)
clf.fit(train_tfidfw2v,Y_train)
opt_estimator_tfidfw2v_xg, opt_depth_tfidfw2v_xg = clf.best_params_.get('n_estimators'), clf.best_p
arams_.get('max_depth')

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']

df_heatmap = pd. DataFrame(train_auc.reshape(len(n_estimators), len(depth)), index=n_estimators,
columns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("Train Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

df_heatmap = pd. DataFrame(cv_auc.reshape(len(n_estimators), len(depth)), index=n_estimators, colum
ns=depth )
fig = plt. figure(figsize=(16,5))
heatmap = sns. heatmap(df_heatmap, annot=True, fmt='.4g')
plt. title("CV Data", size=24)
plt. xlabel('Depth' , size=18)
plt. ylabel('Estimator' , size=18)
plt. show()

print("Max depth is = ", opt_depth_tfidfw2v_xg , " Optimal value of n_estimator :",
opt_estimator_tfidfw2v_xg)

#Cv auc scores
print("---------------------------")
print("Cv auc scores")
print(cv_auc)
print("Maximun Auc value :",max(cv_auc))

#test data

clf = xgb.XGBClassifier(max_depth=opt_depth_tfidfw2v_xg,
n_estimators=opt_estimator_tfidfw2v_xg,random_state=0 )
clf.fit(train_tfidfw2v,Y_train)
```

```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(tfidf_sent_vectors_train)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(tfidf_sent_vectors_test)[:,1])



plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.legend()
plt.xlabel("FBR")
plt.ylabel("TBR")
plt.title("Train and Test Data")
plt.show()
```

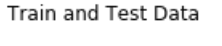
```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
```

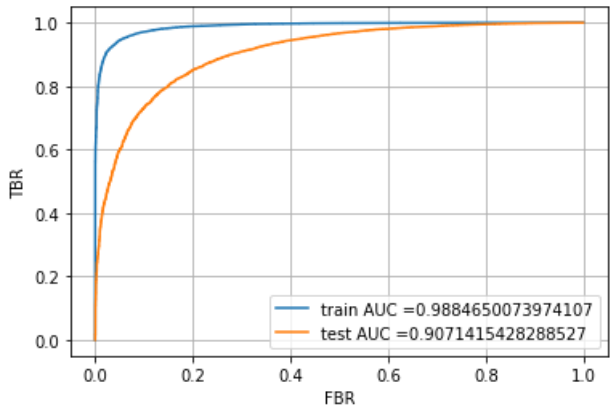
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done  75 out of  75 | elapsed:  5.9min finished
```






```
Max depth is =  6  Optimal value of n_estimator : 200
--------------------------
Cv auc scores
[0.78635987 0.80469426 0.86501315 0.8838856  0.89683494 0.80399093
 0.82908119 0.88110455 0.89627201 0.90536226 0.82242231 0.84187183
 0.88903347 0.90256719 0.90958466 0.83700497 0.85440374 0.89553572
 0.90594707 0.91026479 0.84853307 0.86266867 0.89983192 0.90797666
 0.91130283]
Maximun Auc value : 0.91130283179647
```





In [56]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.NO","VECTORIZER","MODEL", "MAX DEPTH","ESTIMATOR"]

auc = [0.94,0.94,0.90,0.88,0.94,0.95,0.93,0.90]

x.add_row(["1","BAG OF WORDS","RANDOM FOREST",opt_depth_bow, opt_estimator_bow])
x.add_row(["2","TFIDF","RANDOM FOREST",opt_depth_tfidf, opt_estimator_tfidf])
x.add_row(["3","AVG W2V","RANDOM FOREST",opt_depth_avgw2v, opt_estimator_avgw2v])
x.add_row(["4","TFIDF W2V","RANDOM FOREST",opt_depth_tfidfw2v, opt_estimator_tfidfw2v])

x.add_row(["5","BAG OF WORDS","XGBOOST",opt_depth_bow, opt_estimator_bow])
x.add_row(["6","TFIDF","XGBOOST",opt_depth_tfidf, opt_estimator_tfidf])
x.add_row(["7","AVG W2V","XGBOOST",opt_depth_avgw2v, opt_estimator_avgw2v])
x.add_row(["8","TFIDF W2V","XGBOOST",opt_depth_tfidfw2v, opt_estimator_tfidfw2v])


x.add_column("AUC",auc)

# Printing the Table
print(x)
```

```
+------+--------------+---------------+-----------+-----------+------+
| S.NO |  VECTORIZER  |     MODEL     | MAX DEPTH | ESTIMATOR | AUC  |
+------+--------------+---------------+-----------+-----------+------+
|  1   | BAG OF WORDS | RANDOM FOREST |     9     |    1000   | 0.94 |
|  2   |    TFIDF     | RANDOM FOREST |     10    |    1000   | 0.94 |
|  3   |   AVG W2V    | RANDOM FOREST |     10    |    1000   | 0.9  |
|  4   |  TFIDF W2V   | RANDOM FOREST |     10    |    1000   | 0.88 |
|  5   | BAG OF WORDS |    XGBOOST    |     9     |    1000   | 0.94 |
|  6   |    TFIDF     |    XGBOOST    |     10    |    1000   | 0.95 |
|  7   |   AVG W2V    |    XGBOOST    |     10    |    1000   | 0.93 |
|  8   |  TFIDF W2V   |    XGBOOST    |     10    |    1000   | 0.9  |
+------+--------------+---------------+-----------+-----------+------+
```

In [ ]:

```
```