

Amazon Fine Food Reviews Analysis

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

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

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

Number of reviews: 568,454

Number of users: 256,059

Number of products: 74,258

Timespan: Oct 1999 - Oct 2012

Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 will be set to "positive". Otherwise, it will be set to "negative".

```
In [0]: 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.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
.....
Mounted at /content/drive

```
In [0]: %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

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

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

import xgboost as xgb
```

```
from datetime import datetime
import pytz

from wordcloud import WordCloud
```

```
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 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""", con)
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a 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 (5000, 10)

Out[0]:

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

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

(80668, 7)
```

Out[0]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIB9	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(*)
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="AR5J8UI46CURN"
ORDER BY ProductID
""", con)
display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sun
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFER
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFER
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFER
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFER
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFER



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 delete the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

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

```
In [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 calculations

```
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	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste wi crunchy almond inside

```
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 observed 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?
<http://www.amazon.com/VICTOR-FLY-MAGNET-B-AIT-REFILL/dp/B00004RBDY>

The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

I recently tried this flavor/brand and was surprised at how delicious these chips are. The best thing was that there were a lot of "brown" chips 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 flavor because they do not seem to be as salty, and the onion flavor is better. 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 they were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering.

These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let's also remember 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 cookie dough; however, I don't see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick 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 suggest Nabisco's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like a combination of chocolate 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.
This 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_150)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

Why is this \$[...] when the same product is available for \$[...] here?

The Victor M380 and M502 traps are unreal, of course -- total 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 genocide. Pretty stinky, but only right nearby.

I recently tried this flavor/brand and was surprised at how delicious these chips are. The best thing was that there were a lot of "brown" chips 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 flavor because they do not seem to be as salty, and the onion flavor is better. 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 they were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering. These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let's also remember 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 cookie dough; however, I don't see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick 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 suggest Nabisco's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like a combination of chocolate 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. This k cup 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)
    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]: sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("=*50)
```

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other wants crispy cookie s. Hey, I am sorry; but these reviews do nobody any good beyond reminding us to look before ordering.

Th ese are chocolate-oatmeal cookies. If you do not like that combination, do not order this type of cookie. I find th e combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coc onut-type consistency. Now let is also remember that tastes differ; so, I have given my opinion.

Then, th ese 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 cookie dough; however, I do not see where these taste like raw cookie dough. Bot h are soft, however, so is this the confusion? And, yes, they stick together. Soft cookies tend to do that. They a re not 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 suggest Nabisco is Ginger Snaps. If you want a cookie that is soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try. I am here to place my second o rder.

=====

```
In [0]: #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?
 />
The Victor and traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

```
In [0]: #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 were ordering the other wants crispy cookies Hey I am sorry but these reviews do nobody any good beyond reminding us to look before ordering br br These are chocolate oatmeal cookies If you do not like that combination 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 cookie sort of a coconut type consistency Now let 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 chewy I happen to like raw cookie dough however I do not see where these taste like raw cookie dough Both are soft however so is this the confusion And yes they stick together Soft cookies tend to 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 you want something hard and crisp I suggest Nabisco's Ginger Snaps If you want a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal give these a try I am here to place my second order

```
In [0]: # 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 removed in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'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', 'furthe \
    r', \
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor \
    e', \
    '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', "doesn'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', "were \
    n't", \
    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [0]: # Combining all the above students
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 crispy cookies hey sorry reviews nobody good beyond reminding us look ordering chocolate oatmeal cookies not like combination not order type cookie find combo quite nice really oatmeal sort calms rich chocolate flavor gives cookie sort coconut type consistency let also remember tastes differ given opinion soft chewy cookies advertised not crispy cookies blurb would say crispy rather chewy happen like raw cookie dough however not see taste like raw cookie dough soft however confusion yes stick together soft cookies tend not individually wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet want something hard crisp suggest nabisco ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

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

some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal', 'abiding', 'abilit
y']
=====
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[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-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 numbers 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_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.get_feature_names()[0:10])
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', 'able find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', '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]: # 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]: # 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 which contains a dict ,
# and it contains all our corpus 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 variable 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 occurred atleast 5 times
    w2v_model=Word2Vec(list_of_sentences,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 Google's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('especially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted', 0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]  
=====  
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.9992451071739197), ('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('america', 0.9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.9991567134857178)]
```

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

```
number of words that occurred minimum 5 times 3817  
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']
```

[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, 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 += 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 value
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_tfidf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = [] # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_senteance): # 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.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 parameter tuning (Consider two hyperparameters: n_estimators & max_depth)

- Find the best hyper parameter which will give the maximum [AUC](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) (<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/>) value
- Find the best hyper parameter 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 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](https://seaborn.pydata.org/generated/seaborn.heatmap.html) (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>) 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](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)

(<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/>) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#).

 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>).

(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>).

(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>).

6. Conclusion (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>).

- 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 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>) link (<http://zetcode.com/python/prettytable/>).



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. (<https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf>)

```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/preprocessed_reviews.pkl','rb') as f:
    preprocessed_reviews=pickle.load(f)
    with open('/content/drive/My Drive/Applied AI assignments/list_of_scores.pkl','rb') as f:
        list_of_scores=pickle.load(f)
```

[5.1] Applying RF

[5.1.1] Applying Random Forests on BOW, SET 1

```
In [0]: X=preprocessed_reviews  
y=list_of_scores
```

```
In [0]: #spliting the dataset into train and test datasets  
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
In [0]: #vectorizing using Bow  
bow_vectorizer=CountVectorizer()  
bow_vectorizer.fit(X_train)  
X_train_bow=bow_vectorizer.transform(X_train)  
X_test_bow=bow_vectorizer.transform(X_test)
```

```
In [0]: #GridSearch to find best hyperparamters  
rfclf=RandomForestClassifier()  
n_estimators_vals=[1,5,10,50,100,500,1000]  
max_depth_vals=[1,5,10,50,100,500,1000]  
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}  
clf=GridSearchCV(rfclf,param_grid,cv=3,scoring='roc_auc')  
clf.fit(X_train_bow,y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',  
estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',  
max_depth=None, max_features='auto', max_leaf_nodes=None,  
min_impurity_decrease=0.0, min_impurity_split=None,  
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,  
oob_score=False, random_state=None, verbose=0,  
warm_start=False),  
fit_params=None, iid='warn', n_jobs=None,  
param_grid={'n_estimators': [1, 5, 10, 50, 100, 500, 1000], 'max_depth': [1, 5, 10, 50, 100, 500, 1000]},  
pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',  
scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                               max_depth=1000, max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
                               oob_score=False, random_state=None, verbose=0,
                               warm_start=False)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 1000, 'n_estimators': 1000}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/bow_train_auc.pkl','wb') as f:
    pickle.dump(train_auc,f)

with open('/content/drive/My Drive/Applied AI assignments/Random Forest/bow_cv_auc.pkl','wb') as f:
    pickle.dump(cv_auc,f)
```

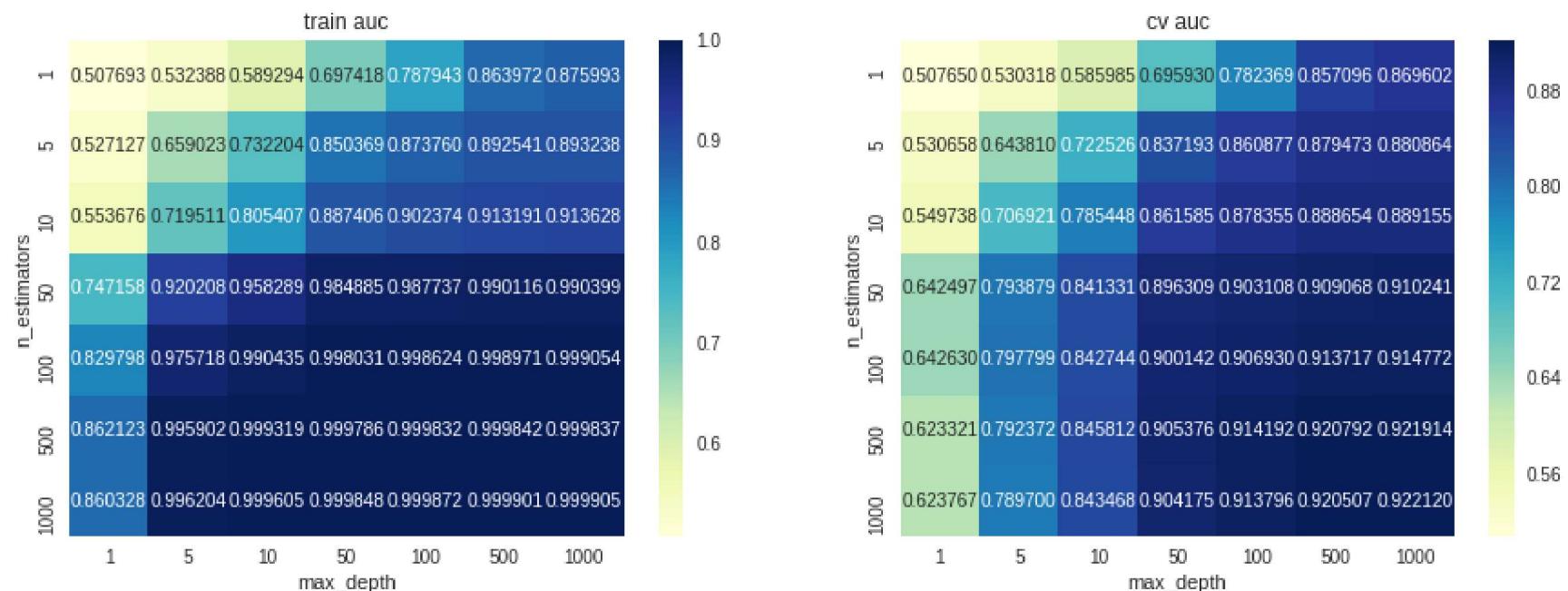
```
In [0]: n_estimators=[1,5,10,50,100,500,1000]
max_depth=[1,5,10,50,100,500,1000]
n_estimators_modified=[]
max_depth_modified=[]
for p in n_estimators:
    for q in range(len(max_depth)):
        n_estimators_modified.append(p)
for x in n_estimators:
    for y in max_depth:
        max_depth_modified.append(y)
```

```
In [0]: #creating a dataframe using hyperparamters and train, cv auc seperately to use with heatmaps
d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}
df_train_auc=pd.DataFrame(data=d_train_auc)
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')

d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}
df_cv_auc=pd.DataFrame(data=d_cv_auc)
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: #plotting heatmaps
plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt=' .6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt=' .6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



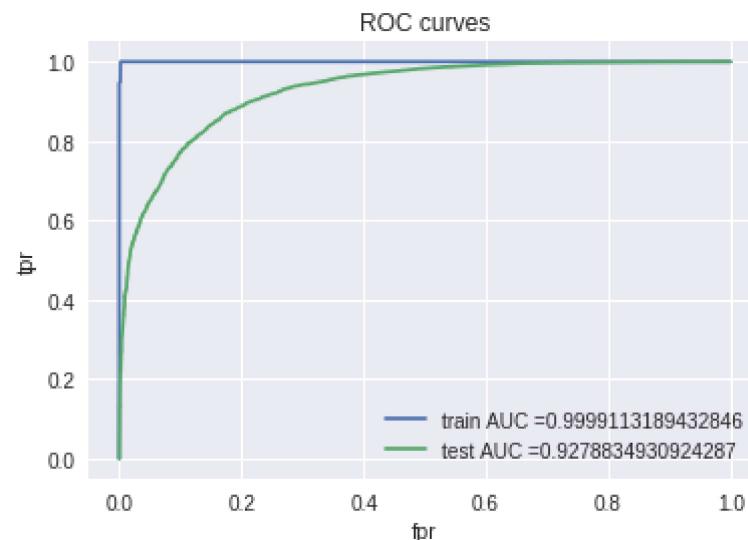
From the above heatmaps, we can see that CV AUC is max for max_depth=1000 and n_Estimators=10000

```
In [0]: #training the model with best hyperparameters
best_max_depth=1000
best_n_estimators=1000

rfclf=RandomForestClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth)
rfclf.fit(X_train_bow,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, rfclf.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfclf.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()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]: *#confusion matrix using seaborn heatmap*

```
y_pred=rfclf.predict(X_test_bow)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6ff417a20>



In [0]: *#get the feature importances*

```
feat_imp=rfclf.feature_importances_
```

In [0]: *with open('/content/drive/My Drive/Applied AI assignments/Random Forest/feat_imp_bow.pkl','wb') as f:*

```
pickle.dump(feat_imp,f)
```

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

In [0]: *# Please write all the code with proper documentation*

In [0]: *with open('/content/drive/My Drive/Applied AI assignments/Random Forest/feat_imp_bow.pkl','rb') as f:*

```
feat_imp=pickle.load(f)
```

```
In [0]: #feat_imp=dtclf.feature_importances_
feat_names=bow_vectorizer.get_feature_names()
feat_imp_list=feat_imp.tolist()
feat_name_imp_dict=dict(zip(feat_names,feat_imp_list))
feat_name_imp_dict_items=[(v,k) for k,v in feat_name_imp_dict.items()]
feat_name_imp_dict_items.sort()
feat_name_imp_dict_items.reverse()
top_20_imp_feat=[k for v,k in feat_name_imp_dict_items][:20]
```

```
In [81]: #https://stackoverflow.com/questions/16645799/how-to-create-a-word-cloud-from-a-corpus-in-python

wordcloud = WordCloud(width = 1500, height = 700,background_color ='black',min_font_size = 10).generate(str(top_20_imp_feat))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



[5.1.3] Applying Random Forests on TFIDF, SET 2

```
In [0]: # Please write all the code with proper documentation
```

```
In [0]: #tfidf vectorization
tfidf_vectorizer=TfidfVectorizer(ngram_range=(1,2),min_df=10)
tfidf_vectorizer.fit(X_train)
X_train_tfidf=tfidf_vectorizer.transform(X_train)
X_test_tfidf=tfidf_vectorizer.transform(X_test)
```

```
In [0]: #GridSearch to find best hyperparamters
rfclf=RandomForestClassifier()
n_estimators_vals=[1,5,10,50,100,500,1000]
max_depth_vals=[1,5,10,50,100,500,1000]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(rfclf,param_grid,cv=3,scoring='roc_auc')
clf.fit(X_train_tfidf,y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
                     estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False),
                     fit_params=None, iid='warn', n_jobs=None,
                     param_grid={'n_estimators': [1, 5, 10, 50, 100, 500, 1000], 'max_depth': [1, 5, 10, 50, 100, 500, 1000]},
                     pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                     scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                               max_depth=1000, max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
                               oob_score=False, random_state=None, verbose=0,
                               warm_start=False)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 1000, 'n_estimators': 1000}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/tfidf_train_auc.pkl','wb') as f:
pickle.dump(train_auc,f)

with open('/content/drive/My Drive/Applied AI assignments/Random Forest/tfidf_cv_auc.pkl','wb') as f:
pickle.dump(cv_auc,f)
```

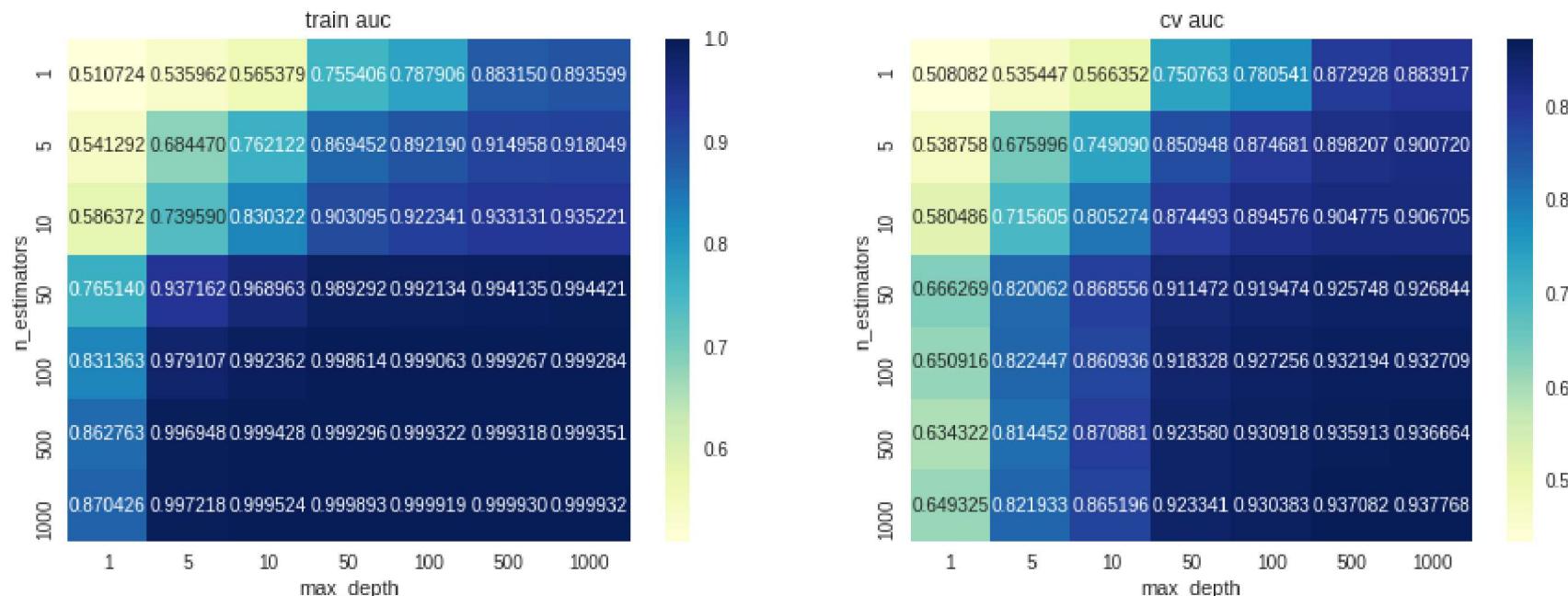
```
In [0]: n_estimators=[1,5,10,50,100,500,1000]
max_depth=[1,5,10,50,100,500,1000]
n_estimators_modified=[]
max_depth_modified=[]
for p in n_estimators:
    for q in range(len(max_depth)):
        n_estimators_modified.append(p)
for x in n_estimators:
    for y in max_depth:
        max_depth_modified.append(y)
```

```
In [0]: #creating a dataframe for use with heatmap
d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}
df_train_auc=pd.DataFrame(data=d_train_auc)
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')

d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}
df_cv_auc=pd.DataFrame(data=d_cv_auc)
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

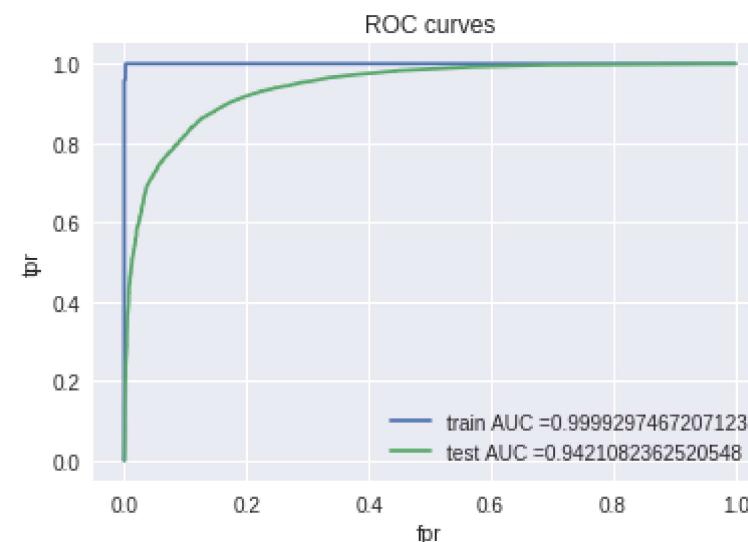
```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



From the above heatmaps, it is observed that CV AUC is max for max_depth=1000 and n_estimators=1000

```
In [0]: #training the model with best hyperparameters  
best_max_depth=clf.best_params_['max_depth']  
best_n_estimators=clf.best_params_['n_estimators']  
  
rfclf=RandomForestClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth)  
rfclf.fit(X_train_tfidf,y_train)  
  
train_fpr, train_tpr, thresholds = roc_curve(y_train, rfclf.predict_proba(X_train_tfidf)[:,1])  
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfclf.predict_proba(X_test_tfidf)[:,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()  
plt.xlabel("fpr")  
plt.ylabel("tpr")  
plt.title("ROC curves")  
plt.show()
```



In [0]: *#confusion matrix using seaborn heatmap*

```
y_pred=rfclf.predict(X_test_tfidf)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6f75d7048>



In [0]: *#get feature importances*
feat_imp=rfclf.feature_importances_

In [0]: *with open('/content/drive/My Drive/Applied AI assignments/Random Forest/feat_imp_tfidf.pkl','wb') as f:*
pickle.dump(feat_imp,f)

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

In [0]: *# Please write all the code with proper documentation*

In [0]: *with open('/content/drive/My Drive/Applied AI assignments/Random Forest/feat_imp_tfidf.pkl','rb') as f:*
feat_imp=pickle.load(f)

```
In [0]: #feat_imp=dtclf.feature_importances_
feat_names=tfidf_vectorizer.get_feature_names()
feat_imp_list=feat_imp.tolist()
feat_name_imp_dict=dict(zip(feat_names,feat_imp_list))
feat_name_imp_dict_items=[(v,k) for k,v in feat_name_imp_dict.items()]
feat_name_imp_dict_items.sort()
feat_name_imp_dict_items.reverse()
top_20_imp_feat=[k for v,k in feat_name_imp_dict_items][:20]
```

```
In [78]: #https://stackoverflow.com/questions/16645799/how-to-create-a-word-cloud-from-a-corpus-in-python

wordcloud = WordCloud(width = 1500, height = 700,background_color ='black',min_font_size = 10).generate(str(top_20_imp
_feat))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



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

```
In [0]: # Please write all the code with proper documentation
```

```
In [0]: # Please write all the code with proper documentation\
# List of tokenized sentences
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())

y=list_of_scores
```

```
In [0]: X_train,X_test,y_train,y_test=train_test_split(list_of_sentance,y,test_size=0.3,random_state=0)
```

```
In [0]: #W2V model on train data
X_train_w2v_model=Word2Vec(X_train,min_count=5,size=50,workers=4)
```

```
In [0]: #vectorizing X_train using Avg W2V
X_train_w2v_words = list(X_train_w2v_model.wv.vocab)
X_train_sent_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_train_sent_vectors.append(sent_vec)

#vectorizing X_test using Avg W2V
#X_test_w2v_words = list(X_test_w2v_model.wv.vocab)
X_test_sent_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
oogle'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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_test_sent_vectors.append(sent_vec)
```

100% |██████████| 61441/61441 [02:11<00:00, 466.63it/s]
100% |██████████| 26332/26332 [00:55<00:00, 471.00it/s]

```
In [0]: #gridsearch to find the best hyperparameters
rfclf=RandomForestClassifier()
n_estimators_vals=[1,5,10,50,100,500,1000]
max_depth_vals=[1,5,10,50,100,500,1000]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(rfclf,param_grid,cv=3,scoring='roc_auc')
clf.fit(X_train_sent_vectors,y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False),
fit_params=None, iid='warn', n_jobs=None,
param_grid={'n_estimators': [1, 5, 10, 50, 100, 500, 1000], 'max_depth': [1, 5, 10, 50, 100, 500, 1000]}, 
pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=1000, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 1000, 'n_estimators': 1000}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

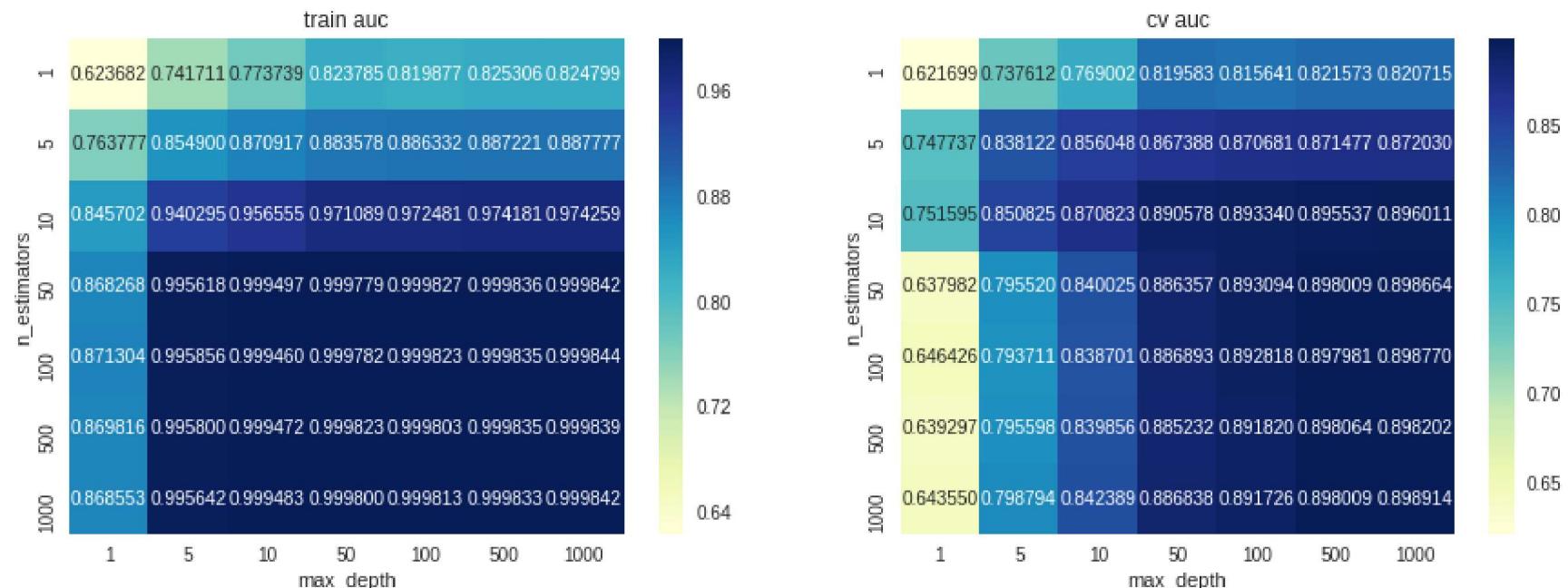
```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/avgw2v_train_auc.pkl','wb') as f:  
    pickle.dump(train_auc,f)  
  
with open('/content/drive/My Drive/Applied AI assignments/Random Forest/avgw2v_cv_auc.pkl','wb') as f:  
    pickle.dump(cv_auc,f)
```

```
In [0]: n_estimators=[1,5,10,50,100,500,1000]  
max_depth=[1,5,10,50,100,500,1000]  
n_estimators_modified=[]  
max_depth_modified=[]  
for p in n_estimators:  
    for q in range(len(max_depth)):  
        n_estimators_modified.append(p)  
for x in n_estimators:  
    for y in max_depth:  
        max_depth_modified.append(y)
```

```
In [0]: #creating a dataframe for use with heatmaps  
d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}  
df_train_auc=pd.DataFrame(data=d_train_auc)  
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')  
  
d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}  
df_cv_auc=pd.DataFrame(data=d_cv_auc)  
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



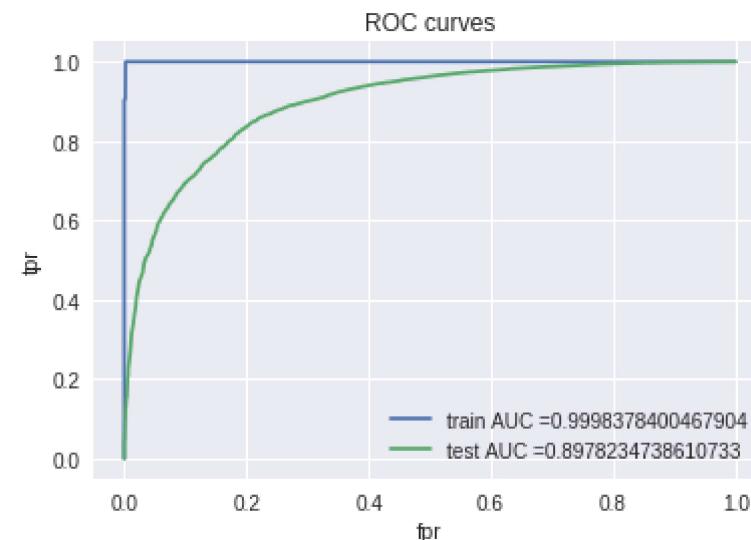
From the above heatmaps, it is observed that CV AUC is max for max_depth=1000 and n_estimators=1000

```
In [0]: #training the model with best hyperparameters
best_max_depth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']

rfclf=RandomForestClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth)
rfclf.fit(X_train_sent_vectors,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, rfclf.predict_proba(X_train_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfclf.predict_proba(X_test_sent_vectors)[:,1])

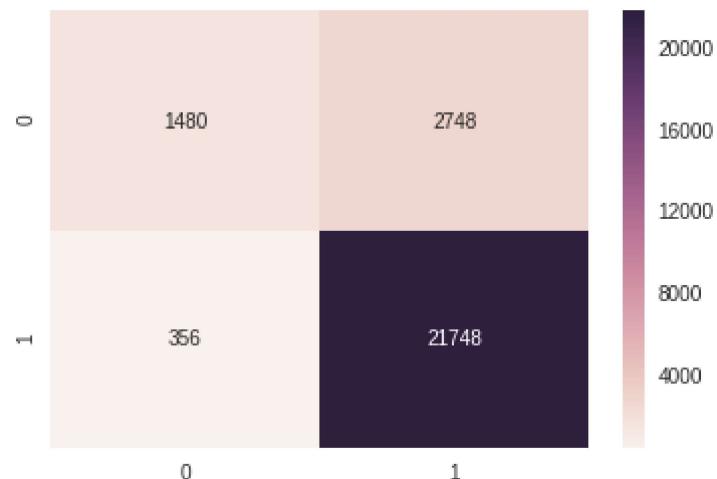
plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



```
In [0]: #confusion matrix using seaborn heatmap
```

```
y_pred=rfclf.predict(X_test_sent_vectors)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

```
Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f45e351d6d8>
```



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

```
In [0]: # Please write all the code with proper documentation
```

```
In [0]: # Please write all the code with proper documentation
X=preprocessed_reviews
y=list_of_scores
```

```
In [0]: X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
In [0]: model=TfidfVectorizer()  
model.fit(X_train)  
X_train_tfidf=model.transform(X_train)  
X_test_tfidf=model.transform(X_test)
```

```
In [0]: dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: #vectorizing X_train using tfidf w2v
X_train_list_of_sents=[]
for sent in X_train:
    X_train_list_of_sents.append(sent.split())

#build w2v model for X_train
X_train_w2v_model=Word2Vec(X_train_list_of_sents,min_count=5,size=50,workers=4)
words=list(X_train_w2v_model.wv.vocab)

# 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

X_train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_train_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_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_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

100% |██████████| 61441/61441 [27:26<00:00, 37.32it/s]

In [0]: #vectorizing test dataset

```
X_test_list_of_sents=[]
for sent in X_test:
    X_test_list_of_sents.append(sent.split())

#build w2v model for X_test
#X_test_w2v_model=Word2Vec(X_test_list_of_sents,min_count=5,size=50,workers=4)
#X_test_words=list(X_test_w2v_model.wv.vocab)

# 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

X_test_tfidf_sent_vectors = [] # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_test_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_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 values 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_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

100% |██████████| 26332/26332 [12:18<00:00, 32.13it/s]

```
In [0]: #GridSearch to find the best hyperparamters
rfclf=RandomForestClassifier()
n_estimators_vals=[1,5,10,50,100,500,1000]
max_depth_vals=[1,5,10,50,100,500,1000]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(rfclf,param_grid,cv=3,scoring='roc_auc')
clf.fit(X_train_tfidf_sent_vectors,y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
                     estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False),
                     fit_params=None, iid='warn', n_jobs=None,
                     param_grid={'n_estimators': [1, 5, 10, 50, 100, 500, 1000], 'max_depth': [1, 5, 10, 50, 100, 500, 1000]},
                     pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                     scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                               max_depth=500, max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
                               oob_score=False, random_state=None, verbose=0,
                               warm_start=False)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 500, 'n_estimators': 1000}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

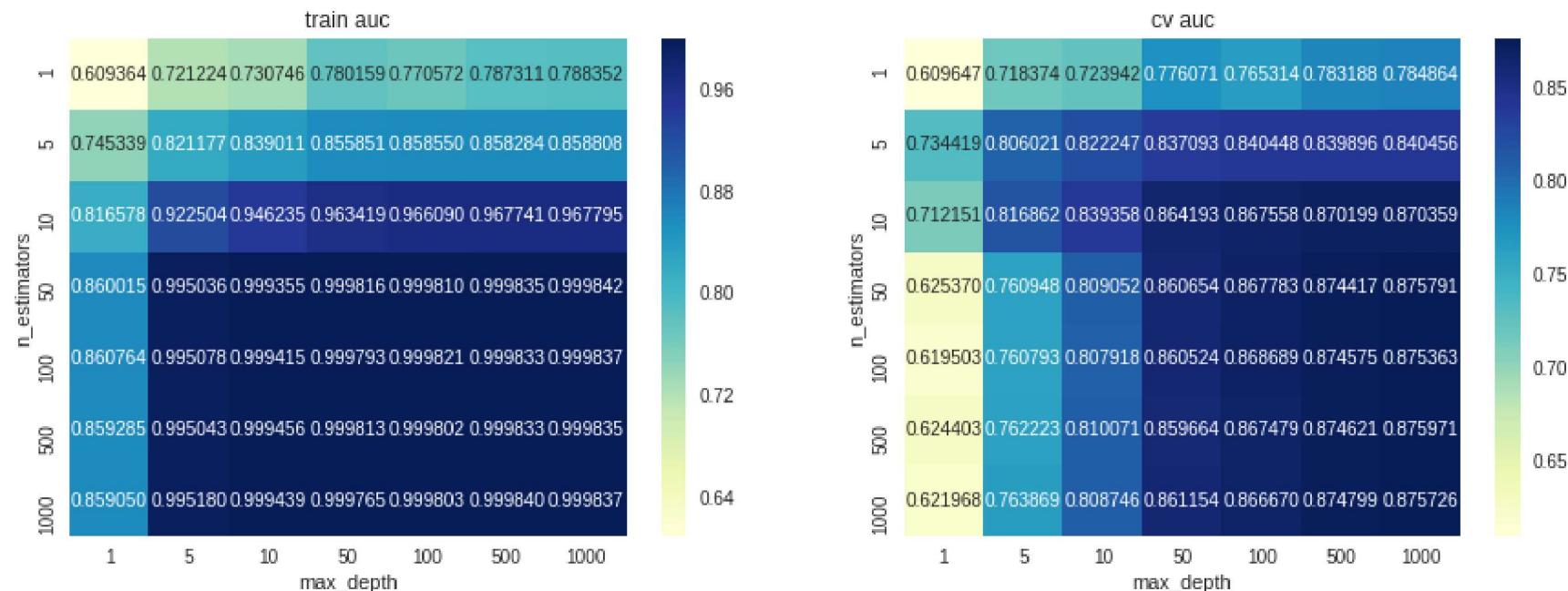
```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/tfidf2v_train_auc.pkl','wb') as f:  
    pickle.dump(train_auc,f)  
  
with open('/content/drive/My Drive/Applied AI assignments/Random Forest/tfidf2v_cv_auc.pkl','wb') as f:  
    pickle.dump(cv_auc,f)
```

```
In [0]: n_estimators=[1,5,10,50,100,500,1000]  
max_depth=[1,5,10,50,100,500,1000]  
n_estimators_modified=[]  
max_depth_modified=[]  
for p in n_estimators:  
    for q in range(len(max_depth)):  
        n_estimators_modified.append(p)  
for x in n_estimators:  
    for y in max_depth:  
        max_depth_modified.append(y)
```

```
In [0]: #creating a datafram for use with heatmaps  
d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}  
df_train_auc=pd.DataFrame(data=d_train_auc)  
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')  
  
d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}  
df_cv_auc=pd.DataFrame(data=d_cv_auc)  
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



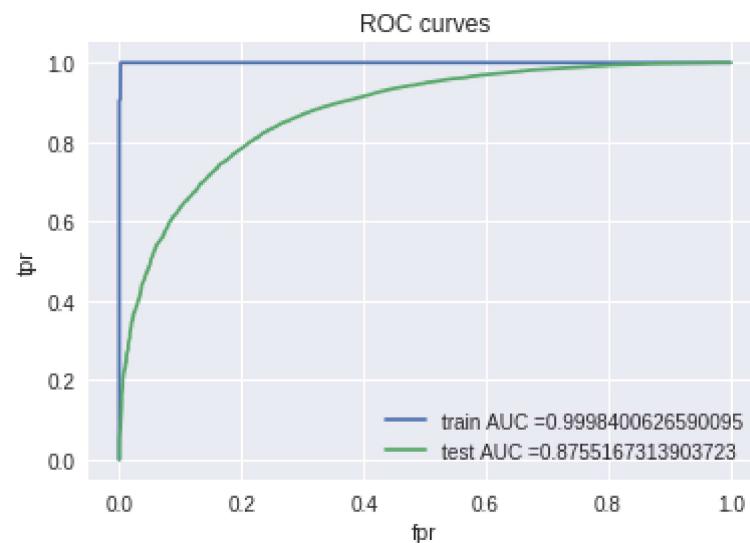
From the above heatmaps, it is observed that CV AUC is maximum for max_Depth=500 and n_estimators=1000

```
In [0]: best_max_depth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']

rfclf=RandomForestClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth)
rfclf.fit(X_train_tfidf_sent_vectors,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, rfclf.predict_proba(X_train_tfidf_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfclf.predict_proba(X_test_tfidf_sent_vectors)[:,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()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



```
In [0]: #confusion matrix using seaborn heatmap
```

```
y_pred=rfclf.predict(X_test_tfidf_sent_vectors)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

```
Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f45c1281390>
```



[5.2] Applying GBDT using XGBOOST

[5.2.1] Applying XGBOOST on BOW, SET 1

```
In [0]: tz = pytz.timezone('Asia/Kolkata')
dt_obj=datetime.now(tz)
start_time=str(dt_obj.hour)+str(':')+str(dt_obj.minute)+str(':')+str(dt_obj.second)
```

```
In [0]: X=preprocessed_reviews
y=list_of_scores
```

```
In [0]: #spliting the dataset into train and test datasets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
In [0]: bow_vectorizer=CountVectorizer()
bow_vectorizer.fit(X_train)
X_train_bow=bow_vectorizer.transform(X_train)
X_test_bow=bow_vectorizer.transform(X_test)
```

```
In [0]: xgbclf=xgb.XGBClassifier(booster='gbtree')
#n_estimators_vals=[1,5,10,50,100,500]
#max_depth_vals=[1,5,10,50,100,500]
n_estimators_vals=[1,5,10,50,100]
max_depth_vals=[1,5,10,50,100]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(xgbclf,param_grid, cv=3, scoring='roc_auc')
clf.fit(X_train_bow,y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_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_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1),
fit_params=None, iid='warn', n_jobs=None,
param_grid={'n_estimators': [1, 5, 10, 50, 100], 'max_depth': [1, 5, 10, 50, 100]}, 
pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
max_depth=100, min_child_weight=1, missing=None, n_estimators=100,
n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 100, 'n_estimators': 100}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_bow_train_auc.pkl','wb') as f:
    pickle.dump(train_auc,f)

with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_bow_cv_auc.pkl','wb') as f:
    pickle.dump(cv_auc,f)
```

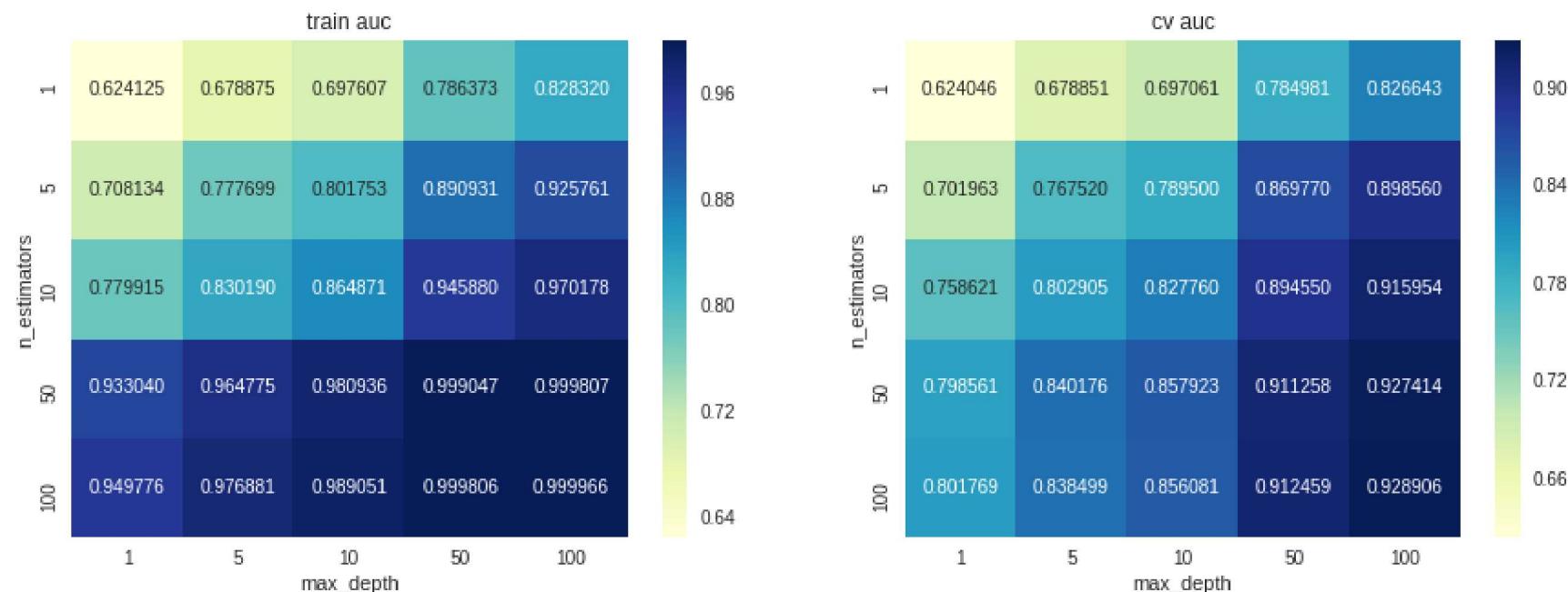
```
In [0]: n_estimators=[1,5,10,50,100]
max_depth=[1,5,10,50,100]
n_estimators_modified=[]
max_depth_modified=[]
for p in n_estimators:
    for q in range(len(max_depth)):
        n_estimators_modified.append(p)
for x in n_estimators:
    for y in max_depth:
        max_depth_modified.append(y)
```

```
In [0]: d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}
df_train_auc=pd.DataFrame(data=d_train_auc)
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')

d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}
df_cv_auc=pd.DataFrame(data=d_cv_auc)
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



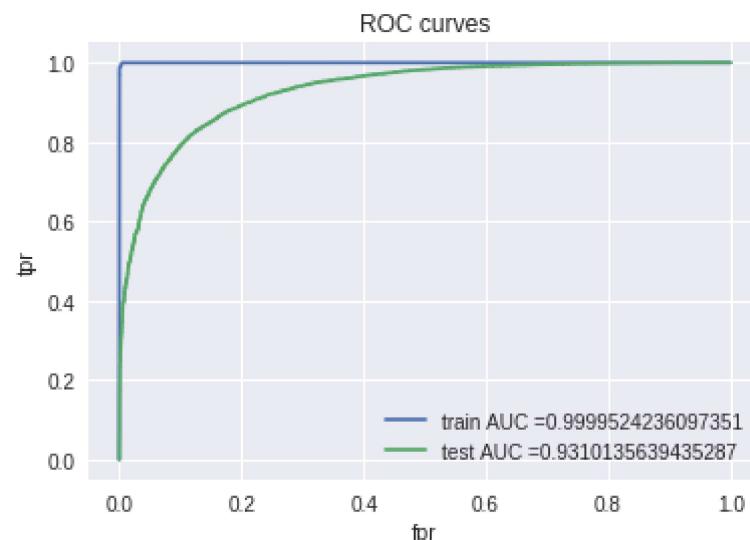
From the above heatmaps, it is observed that CV AUC is maximum for max_depth=100 and n_estimators=100

```
In [0]: best_max_depth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']

xgbclf=xgb.XGBClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth,booster='gbtree')
xgbclf.fit(X_train_bow,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, xgbclf.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, xgbclf.predict_proba(X_test_bow)[:,1])

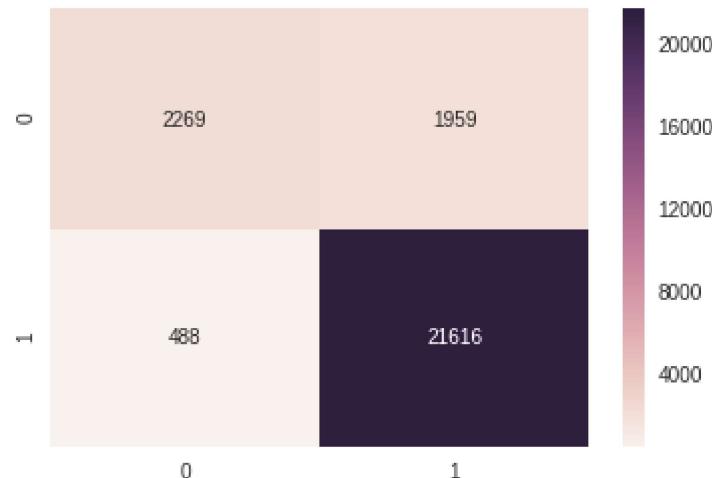
plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]: *#confusion matrix using seaborn heatmap*

```
y_pred=xgbclf.predict(X_test_bow)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f02fe249080>



[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [0]: *# Please write all the code with proper documentation*

```
tfidf_vectorizer=TfidfVectorizer(ngram_range=(1,2),min_df=10)
tfidf_vectorizer.fit(X_train)
X_train_tfidf=tfidf_vectorizer.transform(X_train)
X_test_tfidf=tfidf_vectorizer.transform(X_test)
```

```
In [0]: xgbclf=xgb.XGBClassifier(booster='gbtree')
n_estimators_vals=[1,5,10,50,100]
max_depth_vals=[1,5,10,50,100]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(xgbclf,param_grid,cv=3,scoring='roc_auc')
clf.fit(X_train_tfidf,y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_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_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1),
fit_params=None, iid='warn', n_jobs=None,
param_grid={'n_estimators': [1, 5, 10, 50, 100], 'max_depth': [1, 5, 10, 50, 100]}, 
pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
max_depth=100, min_child_weight=1, missing=None, n_estimators=100,
n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 100, 'n_estimators': 100}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

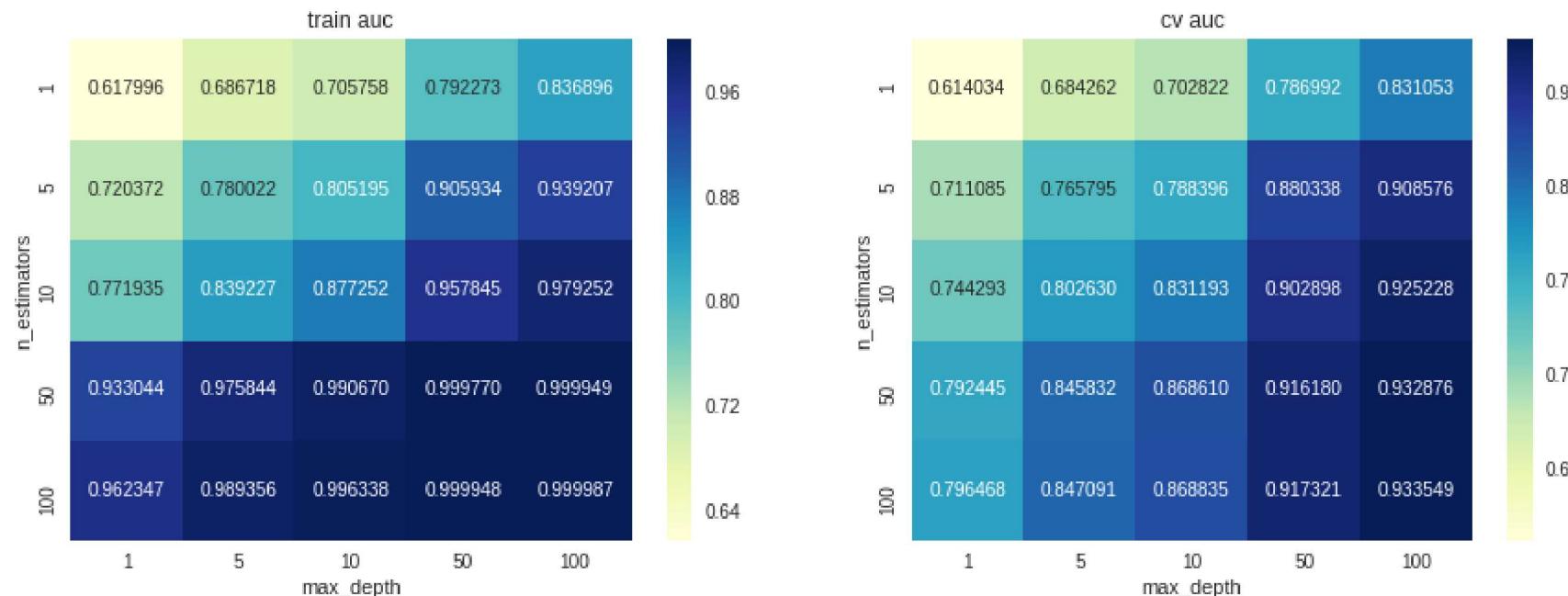
```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_tfidf_train_auc.pkl','wb') as f:  
    pickle.dump(train_auc,f)  
  
with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_tfidf_cv_auc.pkl','wb') as f:  
    pickle.dump(cv_auc,f)
```

```
In [0]: n_estimators=[1,5,10,50,100]  
max_depth=[1,5,10,50,100]  
n_estimators_modified=[]  
max_depth_modified=[]  
for p in n_estimators:  
    for q in range(len(max_depth)):  
        n_estimators_modified.append(p)  
for x in n_estimators:  
    for y in max_depth:  
        max_depth_modified.append(y)
```

```
In [0]: d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}  
df_train_auc=pd.DataFrame(data=d_train_auc)  
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')  
  
d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}  
df_cv_auc=pd.DataFrame(data=d_cv_auc)  
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



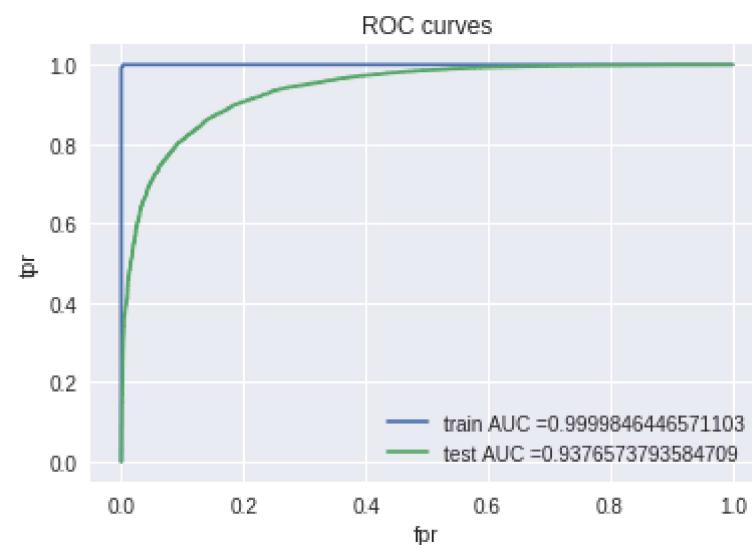
From the above heatmaps, it is observed that CV AUC is maximum for max_depth=100 and n_estimators=100

```
In [0]: best_max_depth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']

xgbclf=xgb.XGBClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth,booster='gbtree')
xgbclf.fit(X_train_tfidf,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, xgbclf.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, xgbclf.predict_proba(X_test_tfidf)[:,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()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



```
In [0]: #confusion matrix using seaborn heatmap
```

```
y_pred=xgbclf.predict(X_test_tfidf)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

```
Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f02fd2abda0>
```



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

```
In [0]: # Please write all the code with proper documentation
```

```
In [0]: # Please write all the code with proper documentation\
# List of tokenized sentences
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())

y=list_of_scores
```

```
In [0]: X_train,X_test,y_train,y_test=train_test_split(list_of_sentance,y,test_size=0.3,random_state=0)
```

```
In [0]: #W2V model on train data
X_train_w2v_model=Word2Vec(X_train,min_count=5,size=50,workers=4)
```

```
In [0]: #vectorizing X_train using Avg W2V
X_train_w2v_words = list(X_train_w2v_model.wv.vocab)
X_train_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_train): # for each review/sentence
    sent_vec = 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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_train_sent_vectors.append(sent_vec)

#vectorizing X_test using Avg W2V
#X_test_w2v_words = list(X_test_w2v_model.wv.vocab)
X_test_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_test): # for each review/sentence
    sent_vec = 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 X_train_w2v_words:
            vec = X_train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_test_sent_vectors.append(sent_vec)
```

100% |██████████| 61441/61441 [02:28<00:00, 413.61it/s]
100% |██████████| 26332/26332 [01:04<00:00, 411.18it/s]

```
In [0]: xgbclf=xgb.XGBClassifier(booster='gbtree')
n_estimators_vals=[1,5,10,50,100]
max_depth_vals=[1,5,10,50,100]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(xgbclf,param_grid,cv=3,scoring='roc_auc')
clf.fit(np.array(X_train_sent_vectors),y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_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_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1),
fit_params=None, iid='warn', n_jobs=None,
param_grid={'n_estimators': [1, 5, 10, 50, 100], 'max_depth': [1, 5, 10, 50, 100]},
pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
max_depth=10, min_child_weight=1, missing=None, n_estimators=100,
n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 10, 'n_estimators': 100}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

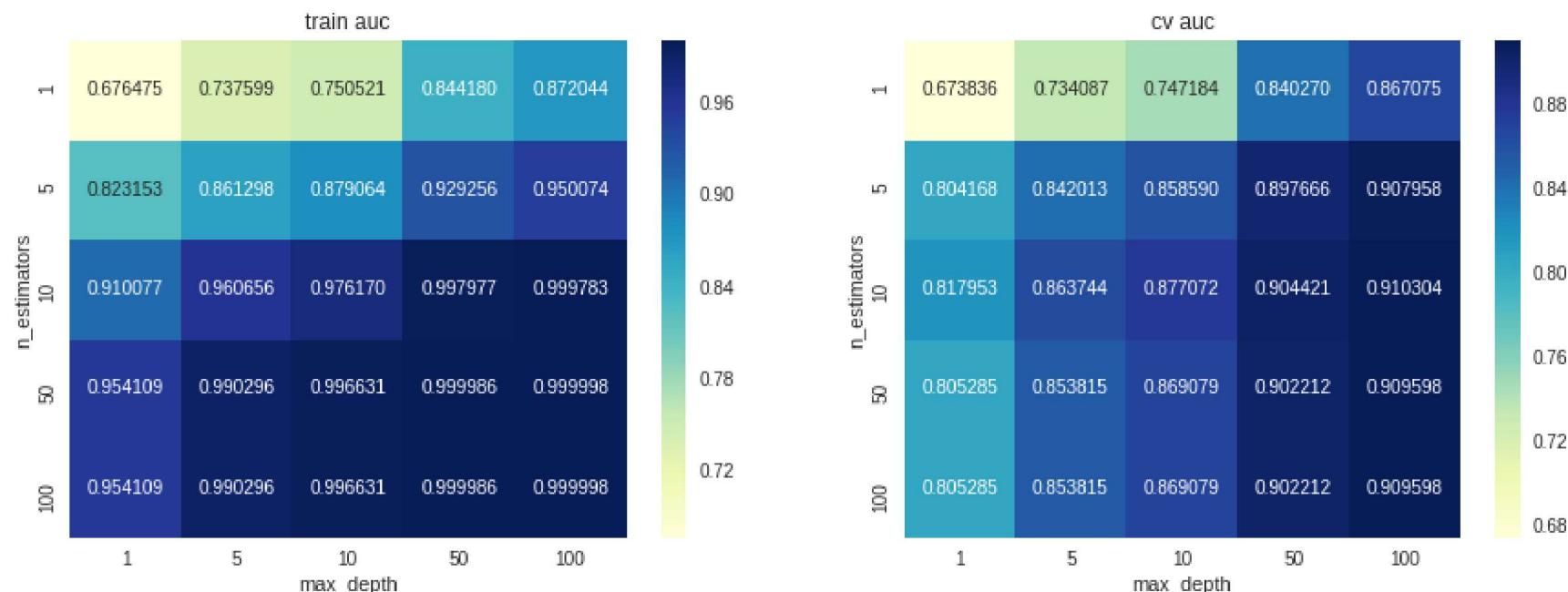
```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_avgw2v_train_auc.pkl','wb') as f:  
    pickle.dump(train_auc,f)  
  
with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_avgw2v_cv_auc.pkl','wb') as f:  
    pickle.dump(cv_auc,f)
```

```
In [0]: n_estimators=[1,5,10,50,100]  
max_depth=[1,5,10,50,100]  
n_estimators_modified=[]  
max_depth_modified=[]  
for p in n_estimators:  
    for q in range(len(max_depth)):  
        n_estimators_modified.append(p)  
for x in n_estimators:  
    for y in max_depth:  
        max_depth_modified.append(y)
```

```
In [0]: d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}  
df_train_auc=pd.DataFrame(data=d_train_auc)  
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')  
  
d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}  
df_cv_auc=pd.DataFrame(data=d_cv_auc)  
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



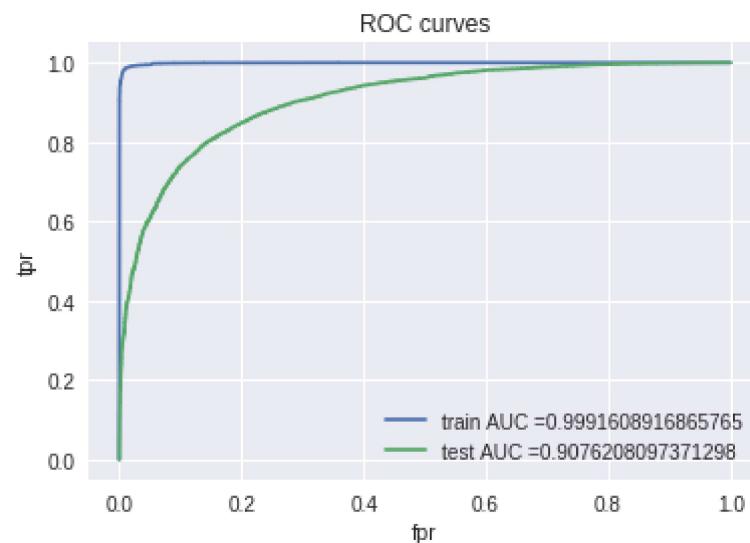
From the above heatmaps, it is observed that CV AUC is maximum for max_depth=10 and n_estimators=100

```
In [0]: best_max_depth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']

xgbclf=xgb.XGBClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth,booster='gbtree')
xgbclf.fit(np.array(X_train_sent_vectors),y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, xgbclf.predict_proba(X_train_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, xgbclf.predict_proba(X_test_sent_vectors)[:,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()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



```
In [0]: #confusion matrix using seaborn heatmap
```

```
y_pred=xgbclf.predict(X_test_sent_vectors)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

```
Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3bb2577b8>
```



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

```
In [0]: # Please write all the code with proper documentation
```

```
In [0]: # Please write all the code with proper documentation
X=preprocessed_reviews
y=list_of_scores
```

```
In [0]: X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
In [0]: model=TfidfVectorizer()
model.fit(X_train)
X_train_tfidf=model.transform(X_train)
X_test_tfidf=model.transform(X_test)

In [0]: dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

In [0]: #vectorizing X_train using tfidf w2v
X_train_list_of_sents=[]
for sent in X_train:
    X_train_list_of_sents.append(sent.split())

#build w2v model for X_train
X_train_w2v_model=Word2Vec(X_train_list_of_sents,min_count=5,size=50,workers=4)
words=list(X_train_w2v_model.wv.vocab)

# 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

X_train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_train_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_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_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [0]: #vectorizing test dataset

```
X_test_list_of_sents=[]
for sent in X_test:
    X_test_list_of_sents.append(sent.split())

#build w2v model for X_test
#X_test_w2v_model=Word2Vec(X_test_list_of_sents,min_count=5,size=50,workers=4)
#X_test_words=list(X_test_w2v_model.wv.vocab)

# 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

X_test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_test_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_train_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 values 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_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [0]: with open('/content/drive/My Drive/Applied AI assignments/X_train_tfidf_sent_vectors.pkl','rb') as f:
 X_train_tfidf_sent_vectors=pickle.load(f)
with open('/content/drive/My Drive/Applied AI assignments/X_test_tfidf_sent_vectors.pkl','rb') as f:
 X_test_tfidf_sent_vectors=pickle.load(f)

```
In [0]: xgbclf=xgb.XGBClassifier(booster='gbtree')
n_estimators_vals=[1,5,10,50,100]
max_depth_vals=[1,5,10,50,100]
param_grid={'n_estimators':n_estimators_vals,'max_depth':max_depth_vals}
clf=GridSearchCV(xgbclf,param_grid,cv=3,scoring='roc_auc')
clf.fit(np.array(X_train_tfidf_sent_vectors),y_train)
```

```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',
estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_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_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1),
fit_params=None, iid='warn', n_jobs=None,
param_grid={'n_estimators': [1, 5, 10, 50, 100], 'max_depth': [1, 5, 10, 50, 100]},
pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
scoring='roc_auc', verbose=0)
```

```
In [0]: clf.best_estimator_
```

```
Out[0]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
max_depth=10, min_child_weight=1, missing=None, n_estimators=100,
n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1)
```

```
In [0]: clf.best_params_
```

```
Out[0]: {'max_depth': 10, 'n_estimators': 100}
```

```
In [0]: train_auc=clf.cv_results_['mean_train_score']
cv_auc=clf.cv_results_['mean_test_score']
```

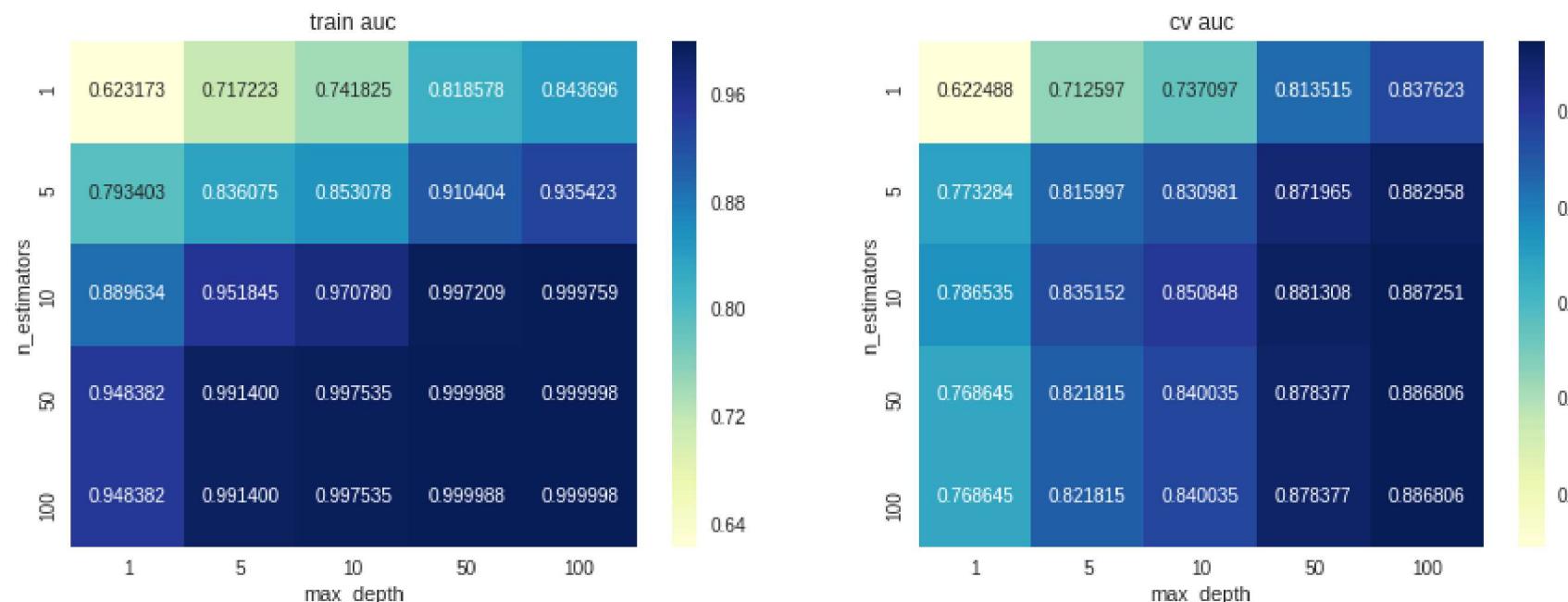
```
In [0]: with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_tfidf2v_train_auc.pkl','wb') as f:  
    pickle.dump(train_auc,f)  
  
with open('/content/drive/My Drive/Applied AI assignments/Random Forest/xgb_tfidf2v_cv_auc.pkl','wb') as f:  
    pickle.dump(cv_auc,f)
```

```
In [0]: n_estimators=[1,5,10,50,100]  
max_depth=[1,5,10,50,100]  
n_estimators_modified=[]  
max_depth_modified=[]  
for p in n_estimators:  
    for q in range(len(max_depth)):  
        n_estimators_modified.append(p)  
for x in n_estimators:  
    for y in max_depth:  
        max_depth_modified.append(y)
```

```
In [0]: d_train_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'train_auc':train_auc}  
df_train_auc=pd.DataFrame(data=d_train_auc)  
df_train_auc=df_train_auc.pivot('n_estimators','max_depth','train_auc')  
  
d_cv_auc = {'n_estimators': n_estimators_modified, 'max_depth': max_depth_modified,'cv_auc':cv_auc}  
df_cv_auc=pd.DataFrame(data=d_cv_auc)  
df_cv_auc=df_cv_auc.pivot('n_estimators','max_depth','cv_auc')
```

```
In [0]: plt.figure(figsize=(15,5))
plt.subplot(121)
ax1 = sns.heatmap(df_train_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('train auc')
plt.subplot(122)
ax2 = sns.heatmap(df_cv_auc, annot=True, fmt='.6f', cmap='YlGnBu')
plt.title('cv auc')
```

Out[0]: Text(0.5, 1.0, 'cv auc')



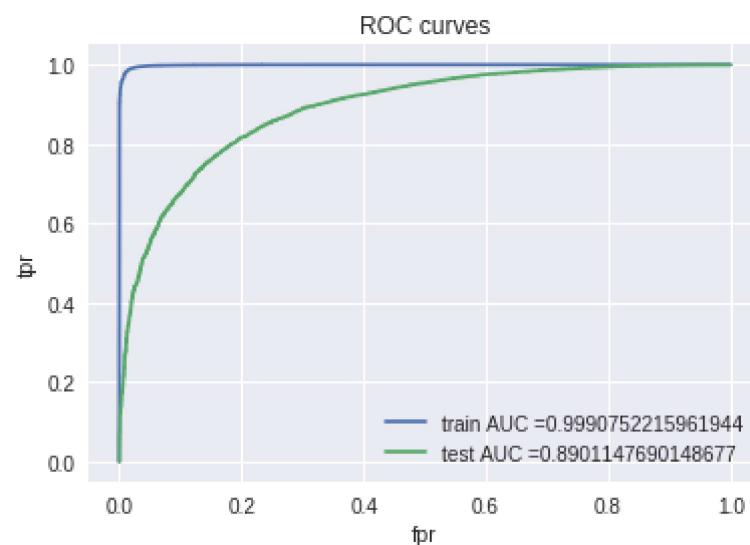
From the above heatmaps, it is observed that CV AUC is maximum for max_depth=10 and n_estimators=100

```
In [0]: best_max_depth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']

xgbclf=xgb.XGBClassifier(n_estimators=best_n_estimators,max_depth=best_max_depth)
xgbclf.fit(np.array(X_train_tfidf_sent_vectors),y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, xgbclf.predict_proba(X_train_tfidf_sent_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, xgbclf.predict_proba(X_test_tfidf_sent_vectors)[:,1])

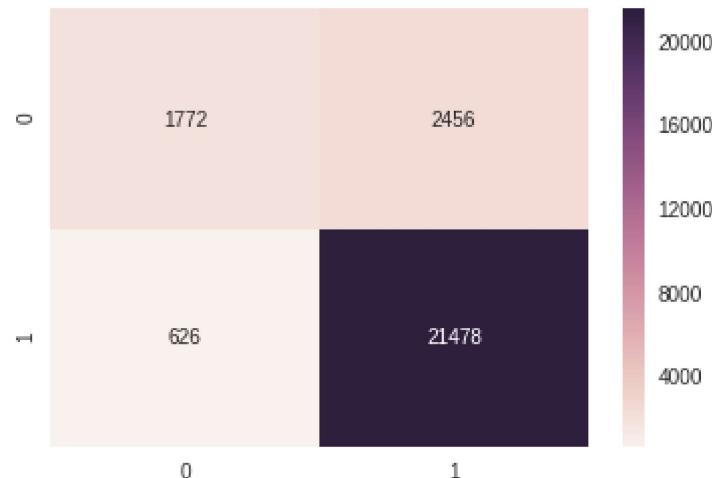
plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curves")
plt.show()
```



In [0]: #confusion matrix using seaborn heatmap

```
y_pred=xgbclf.predict(X_test_tfidf_sent_vectors)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3bb1fbc88>



[6] Conclusions

```
In [73]: from prettytable import PrettyTable

x = PrettyTable()

print('-----RF-----')

x.field_names = ["vectorization", "max_depth", "n_estimators", "test auc"]

x.add_row(["BoW", 1000, 1000, 0.92])
x.add_row(["Tfidf", 1000, 1000, 0.94])
x.add_row(["AvgW2V", 1000, 1000, 0.89])
x.add_row(["TfidfW2V", 500, 1000, 0.87])

print(x)

print('\n\n-----XGBoost-----')

x1 = PrettyTable()

x1.field_names = ["vectorization", "max_depth", "n_estimators", "test auc"]

x1.add_row(["BoW", 100, 100, 0.93])
x1.add_row(["Tfidf", 100, 100, 0.93])
x1.add_row(["AvgW2V", 10, 100, 0.90])
x1.add_row(["TfidfW2V", 10, 100, 0.89])

print(x)
```

-----RF-----

vectorization	max_depth	n_estimators	test auc
BoW	1000	1000	0.92
Tfidf	1000	1000	0.94
AvgW2V	1000	1000	0.89
TfidfW2V	500	1000	0.87

-----XGBoost-----

vectorization	max_depth	n_estimators	test auc
BoW	1000	1000	0.92
Tfidf	1000	1000	0.94
AvgW2V	1000	1000	0.89
TfidfW2V	500	1000	0.87