

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 - 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 [1]: %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, roc_auc_score
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
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate

```

```

In [2]: # using SQLite Table to read data.
con = sqlite3.connect('/Users/puravshah/Downloads/amazon-fine-food-revi
ews/database.sqlite')

```

```

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
!= 3 LIMIT 50000""", con)

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

#changing reviews with score less than 3 to be positive and vice-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 (50000, 10)

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

```
In [3]: display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
In [4]: print(display.shape)
display.head()

(80668, 7)
```

```
Out[4]:
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
--	--------	-----------	-------------	------	-------	------	----------

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

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT
--	--------	-----------	-------------	------	-------	------	-------

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

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

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
--	----	-----------	--------	-------------	----------------------	----------

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2



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

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

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

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

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and 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 [8]: #Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True,
inplace=False, kind='quicksort', na_position='last')
```

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

```
Out[9]: (46072, 10)
```

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

```
Out[10]: 92.144
```

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

```
In [12]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

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

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

```
(46071, 10)
```

```
Out[13]: 1    38479  
         0     7592  
         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 [14]: # 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 very hard to find any chicken products made in the USA but they are out there, but this one isnt. Its too bad too bec ause its a good product but I wont take any chances till they know what is going on with the china imports.

=====

this is yummy, easy and unusual. it makes a quick, delicious pie, crisp or cobbler. home made is better, but a heck of a lot more work. this is great to have on hand for last minute dessert needs where you really wa nt to impress wih your creativity in cooking! recommended.

=====

Great flavor, low in calories, high in nutrients, high in protein! Usua lly protein powders are high priced and high in calories, this one is a great bargain and tastes great, I highly recommend for the lady gym rat s, probably not "macho" enough for guys since it is soy based...

=====

For those of you wanting a high-quality, yet affordable green tea, you should definitely give this one a try. Let me first start by saying tha t everyone is looking for something different for their ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through over 12 boxes of this tea myself, and highly recommend it for the following reasons:

-Quality: Fi rst, this tea offers a smooth quality without any harsh or bitter after tones, which often turns people off from many green teas. I've found m y ideal brewing time to be between 3-5 minutes, giving you a light but flavorful cup of tea. However, if you get distracted or forget about y

our tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still get a smooth but deeper flavor without the bad after taste. The leaves themselves are whole leaves (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon bags also include chunks of tropical fruit and other discernible ingredients. This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine powder, leaving you to wonder what it is you are actually drinking.

-Taste: This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored. You have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, yet the subtle hints of fruit make this a truly unique tea that I believe most will enjoy. If you want it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers an inherent warmth of flavor through it's ingredients.

-Price: This tea offers an excellent product at an exceptional price (especially when purchased at the prices Amazon offers). Compared to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding price. I have been purchasing this through Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.

Overall, this is a wonderful tea that is comparable, and even better than, other teas that are priced much higher. It offers a well-balanced cup of green tea that I believe many will enjoy. In terms of taste, quality, and price, I would argue you won't find a better combination that that offered by Revolution's Tropical Green Tea.

=====

```
In [15]: # 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 very hard to find any chicken products made in the USA but they are out there but this one isnt Its too bad too bad

the USA but they are out there, but this one isnt. Its 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 [16]: # 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 very hard to find any chicken products made in the USA but they are out there, but this one isnt. Its 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.

=====

this is yummy, easy and unusual. it makes a quick, delicious pie, crisp or cobbler. home made is better, but a heck of a lot more work. this is great to have on hand for last minute dessert needs where you really want to impress with your creativity in cooking! recommended.

=====

Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are high priced and high in calories, this one is a

great bargain and tastes great, I highly recommend for the lady gym rats, probably not "macho" enough for guys since it is soy based...

=====

For those of you wanting a high-quality, yet affordable green tea, you should definitely give this one a try. Let me first start by saying that everyone is looking for something different for their ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through over 12 boxes of this tea myself, and highly recommend it for the following reasons:-Quality: First, this tea offers a smooth quality without any harsh or bitter after tones, which often turns people off from many green teas. I've found my ideal brewing time to be between 3-5 minutes, giving you a light but flavorful cup of tea. However, if you get distracted or forget about your tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still get a smooth but deeper flavor without the bad after taste. The leaves themselves are whole leaves (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon bags also include chunks of tropical fruit and other discernible ingredients. This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine powder, leaving you to wonder what it is you are actually drinking.-Taste: This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored. You have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, yet the subtle hints of fruit make this a truly unique tea that I believe most will enjoy. If you want it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers an inherent warmth of flavor through its ingredients.-Price: This tea offers an excellent product at an exceptional price (especially when purchased at the prices Amazon offers). Compared to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding price. I have been purchasing this through Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.Overall, this is a wonderful tea that is comparable, and even better than, other teas that are priced much higher. It offers a well-balanced cup of green tea that I believe many will enjoy. In terms of taste, quality, and price, I would argue you wo

n't find a better combination than that offered by Revolution's Tropica

l Green Tea.

```
In [17]: # 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 [18]: sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are high priced and high in calories, this one is a great bargain and tastes great, I highly recommend for the lady gym rats, probably not "macho" enough for guys since it is soy based...

=====

```
In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub(r"\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 very hard to find any chicken products made in the USA but they are out there, but this one isnt. Its too bad too bec

ause its a good product but I wont take any chances till they know what is going on with the china imports.

```
In [20]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

Great flavor low in calories high in nutrients high in protein Usually protein powders are high priced and high in calories this one is a great bargain and tastes great I highly recommend for the lady gym rats probably not macho enough for guys since it is soy based

```
In [21]: # 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', "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', 'further', \
               'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', \
```

```

        'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
        's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
        've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn', \
        "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn', \
        "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
        'won', "won't", 'wouldn', "wouldn't"])

```

```

In [22]: # Combining all the above students
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)
    # https://gist.github.com/sebleier/554280
    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentence.strip())

```

```

100%|██████████| 46071/46071 [00:13<00:00, 3389.21it/s]

```

```

In [23]: preprocessed_reviews[1500]

```

```

Out[23]: 'great flavor low calories high nutrients high protein usually protein
powders high priced high calories one great bargain tastes great highly
recommend lady gym rats probably not macho enough guys since soy based'

```

[3.2] Preprocessing Review Summary

```
In [24]: ## Similarly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

```
In [25]: #Bow
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler(with_mean=False)
X_1, X_test, y_1, y_test = train_test_split(preprocessed_reviews, final
['Score'], test_size=0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(X_tr)

print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(X_tr)
final_counts=scaler.fit_transform(final_counts)
X_cv_bow=count_vect.transform(X_cv)
X_cv_bow=scaler.transform(X_cv_bow)
X_test_bow=count_vect.transform(X_test)
X_test_bow=scaler.transform(X_test_bow)
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])
print(y_test.shape)

some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaahhhhhh', 'aa
aaaawwwwwwww', 'aaf', 'aafco', 'aah', 'aahhs']
=====
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (22574, 28037)
the number of unique words  28037
(13822,)
```

[4.2] Bi-Grams and n-Grams.

```
In [26]: #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

count_vect_gram = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect_gram.fit_transform(X_tr)
final_bigram_counts = scaler.fit_transform(final_bigram_counts)
X_cv_ngram = count_vect_gram.transform(X_cv)
X_cv_ngram = scaler.transform(X_cv_ngram)
X_test_ngram=count_vect_gram.transform(X_test)
X_test_ngram = scaler.transform(X_test_ngram)
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 (22574, 5000)
the number of unique words including both unigrams and bigrams 5000
```

[4.3] TF-IDF

```
In [27]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_tr)

print("some sample features(unique words in the corpus)",tf_idf_vect.ge
```

```

t_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(X_tr)
final_tf_idf=scaler.fit_transform(final_tf_idf)
X_cv_tfidf=tf_idf_vect.transform(X_cv)
X_cv_tfidf=scaler.transform(X_cv_tfidf)
X_test_tfidf=tf_idf_vect.transform(X_test)
X_test_tfidf=scaler.transform(X_test_tfidf)
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 buy', 'able drink', 'able eat', 'able enjoy', 'able find', 'able get', 'able give', 'able make']

=====

the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer (22574, 13495)
the number of unique words including both unigrams and bigrams 13495

[4.4] Word2Vec

```

In [28]: i=0
list_of_sentence_train=[]
for sentence in X_tr:
    list_of_sentence_train.append(sentence.split())

```

```

In [29]: i=0
list_of_sentence_cv=[]
for sentence in X_cv:
    list_of_sentence_cv.append(sentence.split())

```

```

In [30]: i=0
list_of_sentence_test=[]

```

```
for sentence in X_test:
    list_of_sentence_test.append(sentence.split())
```

```
In [31]: # Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and model[word] as val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.

# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these 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 occured atleast 5 times
    w2v_model_train=Word2Vec(list_of_sentence_train,min_count=5,size=50
, workers=4)
    print(w2v_model_train.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_train.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
```

```

        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")

```

```

[('awesome', 0.8425315618515015), ('amazing', 0.8197296261787415), ('excellent', 0.8156817555427551), ('perfect', 0.7857245802879333), ('fantastic', 0.7851762771606445), ('good', 0.7816717028617859), ('wonderful', 0.7688904404640198), ('terrific', 0.7527984380722046), ('decent', 0.72575443983078), ('delicious', 0.6497917771339417)]
=====
[('nastiest', 0.8585181832313538), ('closest', 0.8497992157936096), ('repulsive', 0.7914880514144897), ('greatest', 0.7839831709861755), ('horrible', 0.7593116164207458), ('best', 0.7548462152481079), ('awful', 0.7345291376113892), ('terrible', 0.7300625443458557), ('hated', 0.7289366126060486), ('tastiest', 0.7265476584434509)]

```

```

In [32]: '''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 at least 5 times
    w2v_model_cv=Word2Vec(list_of_sentence_cv,min_count=5,size=50,workers=4)
    print(w2v_model_cv.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_cv.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")'''

```

```

Out[32]: 'is_your_ram_gt_16g=False\nwant_to_use_google_w2v = False\nwant_to_train_w2v = True\n\nif want_to_train_w2v:\n    # min_count = 5 considers only words that occurred at least 5 times\n    w2v_model_cv=Word2Vec(list_of_sentence_cv,min_count=5,size=50, workers=4)\n    print(w2v_model_cv.wv.most_similar('\great'))\n    print('\='*50)\n    print(w2v_model_cv.wv.most_similar('\worst'))\n\nelif want_to_use_google_w2v and is_your_ram_gt_16g:\n    if os.path.isfile('\GoogleNews-vectors-negative300.bin'):\n        w2v_model=KeyedVectors.load_word2vec_format('\GoogleNews-vectors-negative300.bin', binary=True)\n        print(w2v_model.wv.most_similar('\great'))\n        print(w2v_model.wv.most_similar('\worst'))\n    else:\n        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")'

```

```

In [33]: '''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 at least 5 times
    w2v_model_test=Word2Vec(list_of_sentence_test,min_count=5,size=50,
workers=4)
    print(w2v_model_test.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_test.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")'''

```

```

Out[33]: 'is_your_ram_gt_16g=False\nwant_to_use_google_w2v = False\nwant_to_train_w2v = True\n\nif want_to_train_w2v:\n    # min_count = 5 considers only words that occurred at least 5 times\n    w2v_model_test=Word2Vec(list_of_sentence_test,min_count=5,size=50, workers=4)\n    print(w2v_model_test.wv.most_similar('\great'))\n    print('\='*50)\n    print(w2v mo

```



```
del_test.wv.most_similar('\worst\'))\n    \nelif want_to_use_google_w2v
and is_your_ram_gt_16g:\n    if os.path.isfile('\GoogleNews-vectors-neg
ative300.bin\'):\n        w2v_model=KeyedVectors.load_word2vec_format
('\GoogleNews-vectors-negative300.bin\', binary=True)\n        print(w2
v_model.wv.most_similar('\great\'))\n        print(w2v_model.wv.most_si
milar('\worst\'))\n    else:\n        print("you don't have gogole's
word2vec file, keep want_to_train_w2v = True, to train your own w2v ")'
```

```
In [34]: w2v_words_train = list(w2v_model_train.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words_tra
in))
print("sample words ", w2v_words_train[0:50])
```

```
number of words that occurred minimum 5 times 9025
sample words ['feel', 'bad', 'alot', 'gf', 'bread', 'baking', 'flour',
'makers', 'discover', 'pamela', 'mix', 'would', 'go', 'business', 'sist
er', 'celiac', 'believe', 'tried', 'purchasing', 'every', 'different',
'could', 'find', 'came', 'pamelas', 'readily', 'available', 'local', 'g
rocery', 'stores', 'let', 'tell', 'blows', 'away', 'family', 'member',
'co', 'workers', 'time', 'pretty', 'scared', 'taste', 'testing', 'bake
d', 'goods', 'completly', 'surprised', 'even', 'said', 'preferred']
```

```
In [35]: '''w2v_words_cv = list(w2v_model_cv.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words_c
v))
print("sample words ", w2v_words_cv[0:50])'''
```

```
Out[35]: '\w2v_words_cv = list(w2v_model_cv.wv.vocab)\nprint("number of words t
hat occurred minimum 5 times ",len(w2v_words_cv))\nprint("sample words
", w2v_words_cv[0:50])'
```

```
In [36]: '''w2v_words_test = list(w2v_model_test.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words_tes
t))
print("sample words ", w2v_words_test[0:50])'''
```

```
Out[36]: '\w2v_words_test = list(w2v_model_test.wv.vocab)\nprint("number of wor
ds that occurred minimum 5 times ",len(w2v_words_test))\nprint("sample w
ords " w2v_words_test[0:50])'
```

```
w2v_words_test[0:50])
```

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

[4.4.1.1] Avg W2v

```
In [37]: sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentence_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 w2v_words_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
sent_vectors_train=scaler.fit_transform(sent_vectors_train)
```

```
100%|██████████| 22574/22574 [00:22<00:00, 997.87it/s]
```

```
22574
50
```

```
In [38]: sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentence_cv): # 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/re
```

```

view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
sent_vectors_cv=scaler.transform(sent_vectors_cv)

```

```
100%|██████████| 9675/9675 [00:11<00:00, 866.58it/s]
```

```
9675
```

```
50
```

In [39]: `sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list`

```

for sent in tqdm(list_of_sentence_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
    view
        for word in sent: # for each word in a review/sentence
            if word in w2v_words_train:
                vec = w2v_model_train.wv[word]
                sent_vec += vec
                cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
        sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
sent_vectors_test=scaler.transform(sent_vectors_test)

```

```
100%|██████████| 13822/13822 [00:13<00:00, 1056.23it/s]
```

```
13822
```

[4.4.1.2] TFIDF weighted W2v

```
In [40]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model_train= TfidfVectorizer()
tf_idf_matrix_train = model_train.fit_transform(X_tr)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_train = dict(zip(model_train.get_feature_names(), list(model_train.idf_)))
```

```
In [41]: '''# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
#model_cv= TfidfVectorizer()
tf_idf_matrix_cv = model_train.transform(X_cv)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_cv= dict(zip(model_train.get_feature_names(), list(model_train.idf_)))'''
```

```
Out[41]: '# S = ["abc def pqr", "def def def abc", "pqr pqr def"]\n#model_cv= TfidfVectorizer()\n\ntf_idf_matrix_cv = model_train.transform(X_cv)\n\n# we are converting a dictionary with word as a key, and the idf as a value\n\ndictionary_cv= dict(zip(model_train.get_feature_names(), list(model_train.idf_)))'
```

```
In [42]: '''# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model_test= TfidfVectorizer()
tf_idf_matrix_test = model_test.fit_transform(X_test)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_test= dict(zip(model_test.get_feature_names(), list(model_test.idf_)))'''
```

```
Out[42]: '# S = ["abc def pqr", "def def def abc", "pqr pqr def"]\nmodel_test= TfidfVectorizer()\n\ntf_idf_matrix_test = model_test.fit_transform(X_test)\n\n# we are converting a dictionary with word as a key, and the idf as a value\n\ndictionary_test= dict(zip(model_test.get_feature_names(), list(model_test.idf_)))'
```

```
value(dictionary_test= dict(zip(model_test.get_feature_names(), list(model_test.idf_)))'
```

```
In [43]: # TF-IDF weighted Word2Vec
tfidf_feat_train = model_train.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(list_of_sentence_train): # for each review/sentence
    sent_vec_train = np.zeros(50) # as word vectors are of zero length
    weight_sum_train =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_train and word in tfidf_feat_train:
            vec_train = w2v_model_train.wv[word]
            # tfidf = tfidf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole corpus
            # sent.count(word) = tf value of word in this review
            tf_idf= dictionary_train[word]*(sent.count(word)/len(sent))
            sent_vec_train += (vec * tf_idf)
            weight_sum_train += tf_idf
        if weight_sum_train != 0:
            sent_vec_train /= weight_sum_train
            tfidf_sent_vectors_train.append(sent_vec_train)
    row += 1
tfidf_sent_vectors_train =scaler.fit_transform(tfidf_sent_vectors_train
)

100%|██████████| 22574/22574 [03:33<00:00, 105.55it/s]
```

```
In [44]: tfidf_feat_cv = model_train.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(list_of_sentence_cv): # for each review/sentence
    sent_vec_cv = np.zeros(50) # as word vectors are of zero length
    weight_sum_cv =0; # num of words with a valid vector in the sentenc
e/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train and word in tfidf_feat_cv:
            vec_cv = w2v_model_train.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_train[word]*(sent.count(word)/len(sent
))
            sent_vec_cv += (vec * tf_idf)
            weight_sum_cv += tf_idf
        if weight_sum_cv != 0:
            sent_vec_cv /= weight_sum_cv
            tfidf_sent_vectors_cv.append(sent_vec_cv)
            row += 1
tfidf_sent_vectors_cv =scaler.transform(tfidf_sent_vectors_cv)
100%|██████████| 9675/9675 [01:30<00:00, 106.62it/s]

```

```

In [45]: tfidf_feat_test = model_train.get_feature_names() # tfidf words/col-nam
es
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review
is stored in this list
row=0;
for sent in tqdm(list_of_sentence_test): # for each review/sentence
    sent_vec_test = np.zeros(50) # as word vectors are of zero length
    weight_sum_test =0; # num of words with a valid vector in the sente
nce/review
    for word in sent: # for each word in a review/sentence

```

```

        if word in w2v_words_train and word in tfidf_feat_test:
            vec_test = w2v_model_train.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 corpus
            # sent.count(word) = tf value of word in this review
            tf_idf = dictionary_train[word]*(sent.count(word)/len(sent
    ))

            sent_vec_test += (vec * tf_idf)
            weight_sum_test += tf_idf
    if weight_sum_test != 0:
        sent_vec_test /= weight_sum_test
        tfidf_sent_vectors_test.append(sent_vec_test)
    row += 1
tfidf_sent_vectors_test = scaler.transform(tfidf_sent_vectors_test)
100%|██████████| 13822/13822 [02:05<00:00, 109.98it/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](#) 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 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  with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#).



6. [Conclusion](#)

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



Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this [link](#).

[5.1] Applying RF

[5.1.1] Applying Random Forests on BOW, SET 1

```
In [46]: # Please write all the code with proper documentation
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
tuned_parameters={'n_estimators':[1,5,10,20,30,40], 'max_depth':[1,5,10,20,30,40]}
model_new=GridSearchCV(RandomForestClassifier(),tuned_parameters,scoring='roc_auc',cv=10,return_train_score=True)
model_new.fit(final_counts,y_tr)
```

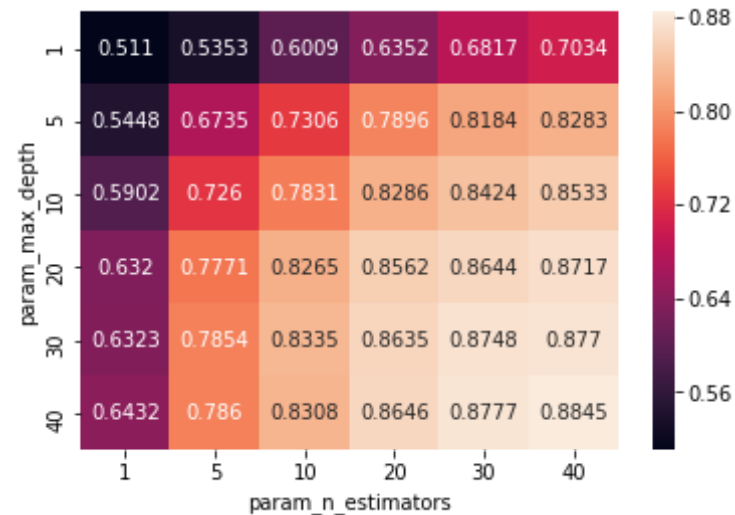
```
print(model_new.best_estimator_)
optimal_estimator=model_new.best_estimator_.n_estimators
optimal_depth=model_new.best_estimator_.max_depth
print(model_new.score(X_cv_bow,y_cv))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=40, 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=40,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)

0.8829977006770834
```

```
In [47]: scores=pd.DataFrame(model_new.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_score')
sns.heatmap(scores,annot=True,fmt='.4g',mask=scores.isnull())
```

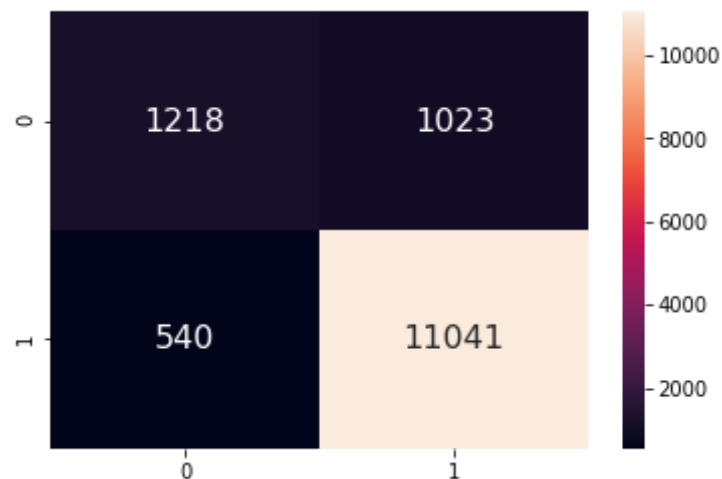
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1a3f7970b8>



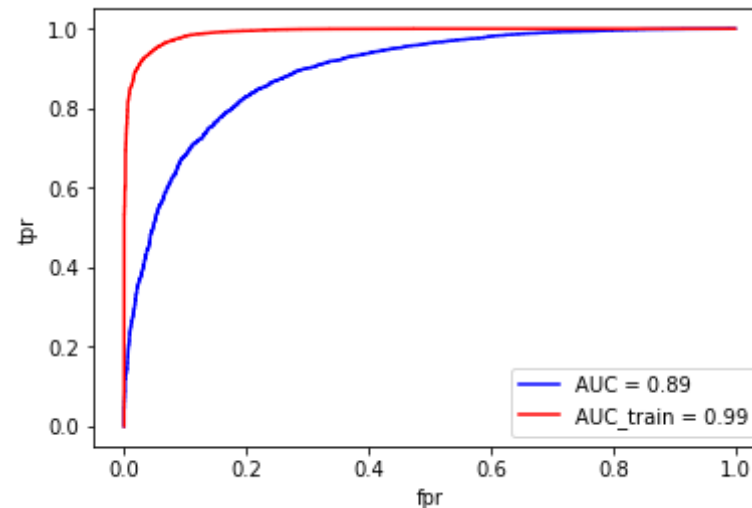
```

In [48]: model_new=RandomForestClassifier(n_estimators=optimal_estimator,max_dep
th=optimal_depth,class_weight='balanced')
model_new.fit(final_counts,y_tr)
pred=model_new.predict(X_test_bow)
probab=model_new.predict_proba(X_test_bow)[:,:1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
pred2=model_new.predict(final_counts)
probab2=model_new.predict_proba(final_counts)[:,:1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatmap for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')

```



Out[48]: Text(0,0.5,'tpr')



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

```
In [49]: # Please write all the code with proper documentation
feature_names=model_new.feature_importances_
bow=count_vect.get_feature_names()
df=pd.DataFrame([feature_names],columns=bow)
df=df.T
print('The top twenty positive features are-\n',df[0].sort_values(ascending=False)[0:20])
print('The top twenty negative features are-\n',df[0].sort_values(ascending=True)[0:20])
```

The top twenty positive features are-

not	0.038586
great	0.029372
disappointed	0.013207
love	0.011029
perfect	0.009849
best	0.008987
would	0.008720
delicious	0.008196
bad	0.007721

```

like          0.007649
product       0.007560
away          0.005689
loves         0.005513
favorite      0.004989
horrible      0.004975
highly        0.004880
stale         0.004716
worst         0.004659
excellent     0.004658
money         0.004494
Name: 0, dtype: float64
The top twenty negative features are-
licoricey     0.0
officials     0.0
officially    0.0
offices       0.0
officer       0.0
officemate    0.0
offerr        0.0
offensive     0.0
offense       0.0
offpopcorn    0.0
offends       0.0
offender      0.0
offended      0.0
offend        0.0
offa          0.0
ofcourse      0.0
ofbottom      0.0
ofbang        0.0
oevr          0.0
offending     0.0
Name: 0, dtype: float64

```

```

In [50]: from wordcloud import WordCloud, STOPWORDS

features = count_vect.get_feature_names()

```

```

coef = model_new.feature_importances_
coef_df = pd.DataFrame({'word': features, 'coefficient': coef}, index =
None)
df = coef_df.sort_values("coefficient", ascending = False)[:100]
cloud = "".join(word for word in df.word)
stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 1000, height = 600, background_color = 'wh
ite', stopwords = stopwords).generate(cloud)

# plot the WordCloud image
plt.figure(figsize = (10, 8))
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Top 100 most important features")
plt.tight_layout(pad = 0)

plt.show()

```



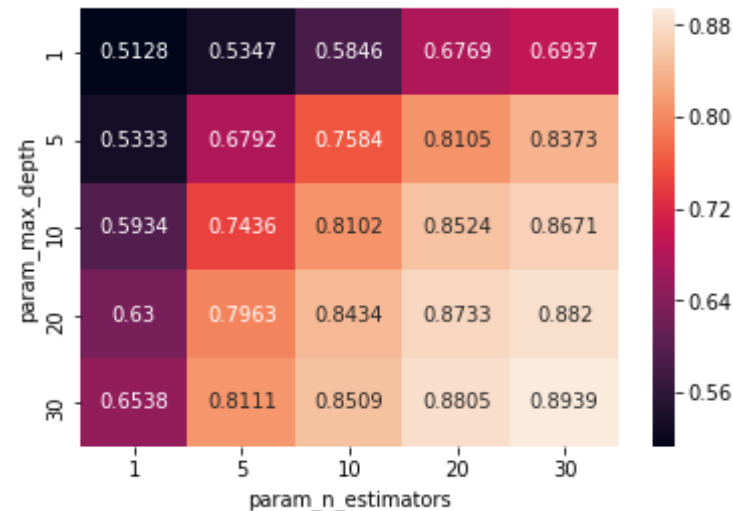
[5.1.3] Applying Random Forests on TFIDF, SET 2

```
In [51]: # Please write all the code with proper documentation
# Please write all the code with proper documentation
#from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,30]}
model=GridSearchCV(RandomForestClassifier(),tuned_parameters,scoring='roc_auc',cv=10,return_train_score=True)
model.fit(final_tf_idf,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(X_cv_tfidf,y_cv))
```

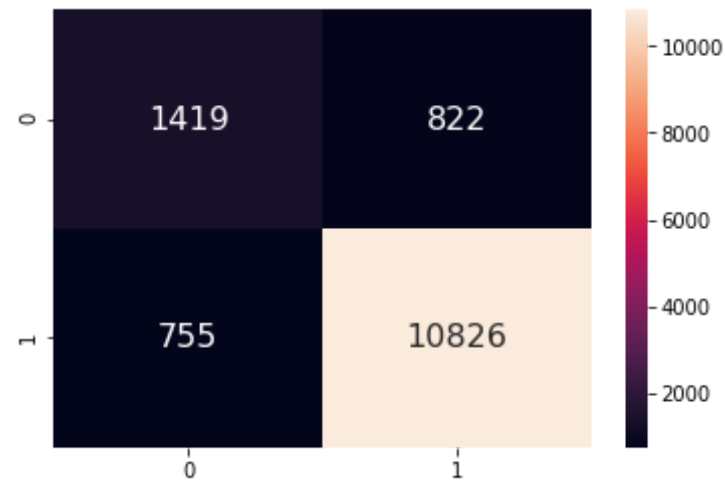
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=30, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=30,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
0.8907382545537286
```

```
In [52]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_score')
sns.heatmap(scores,annot=True,fmt='.4g')
```

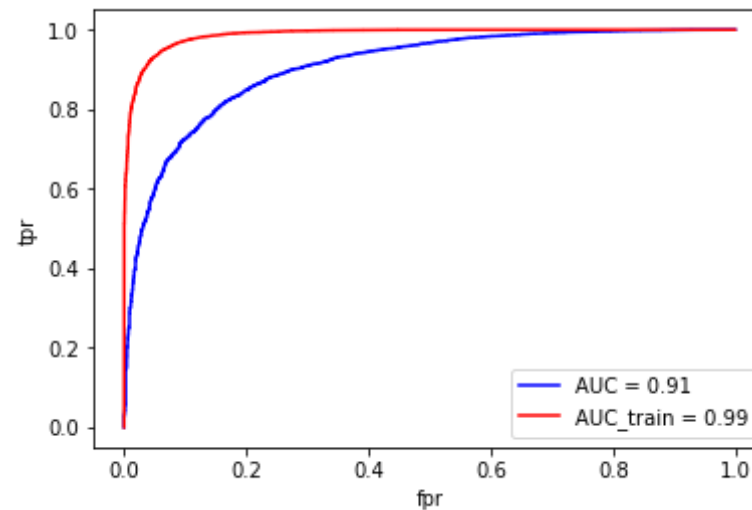
```
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4a69e0f0>
```



```
In [53]: model_new=RandomForestClassifier(n_estimators=optimal_estimator,max_dep
th=optimal_depth,class_weight='balanced')
model_new.fit(final_tf_idf,y_tr)
pred=model_new.predict(X_test_tfidf)
probab=model_new.predict_proba(X_test_tfidf)[:,:1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
pred2=model_new.predict(final_tf_idf)
probab2=model_new.predict_proba(final_tf_idf)[:,:1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
#heatmap for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```

Out[53]: Text(0,0.5,'tpr')



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

```
In [54]: from wordcloud import WordCloud, STOPWORDS
```

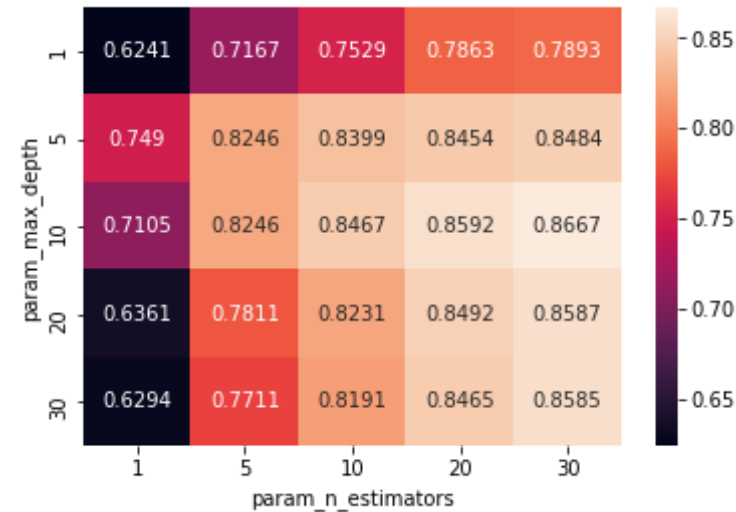

[5.1.5] Applying Random Forests on AVG W2V, SET 3

```
In [55]: # Please write all the code with proper documentation
# Please write all the code with proper documentation
# Please write all the code with proper documentation
#from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,
30]}
model=GridSearchCV(RandomForestClassifier(),tuned_parameters,scoring='r
oc_auc',cv=10,)
model.fit(sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(sent_vectors_cv,y_cv))
```

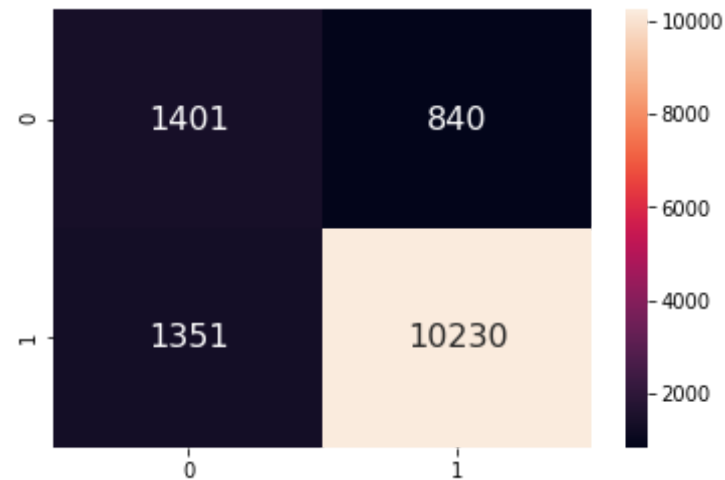
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
                        max_depth=10, max_features='auto', max_leaf_node
s=None,
                        min_impurity_decrease=0.0, min_impurity_split=No
ne,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=30,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
0.8652644019304738
```

```
In [56]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_s
core')
sns.heatmap(scores,annot=True,fmt='.4g')
```

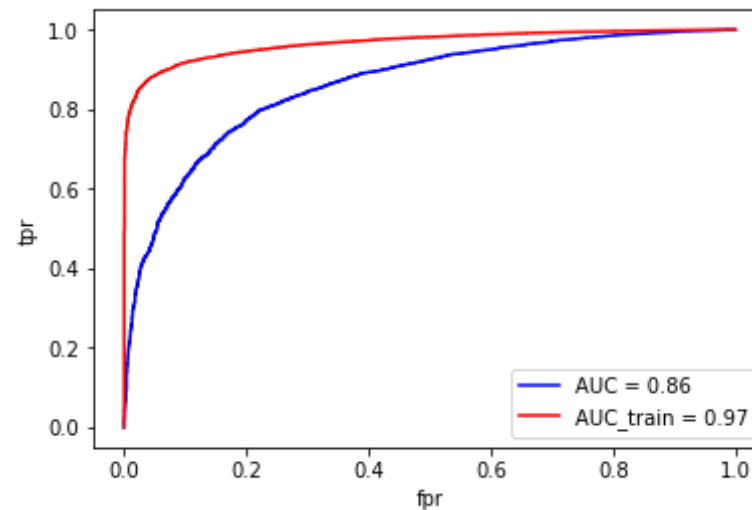
```
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4a6267f0>
```



```
In [57]: model_new=RandomForestClassifier(n_estimators=optimal_estimator,max_dep
th=optimal_depth,class_weight='balanced')
model_new.fit(sent_vectors_train,y_tr)
pred=model_new.predict(sent_vectors_test)
probab=model_new.predict_proba(sent_vectors_test)[:,:1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
pred2=model_new.predict(sent_vectors_train)
probab2=model_new.predict_proba(sent_vectors_train)[:,:1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
#heatmap for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```



Out[57]: Text(0,0.5,'tpr')



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

```
In [58]: # Please write all the code with proper documentation
```

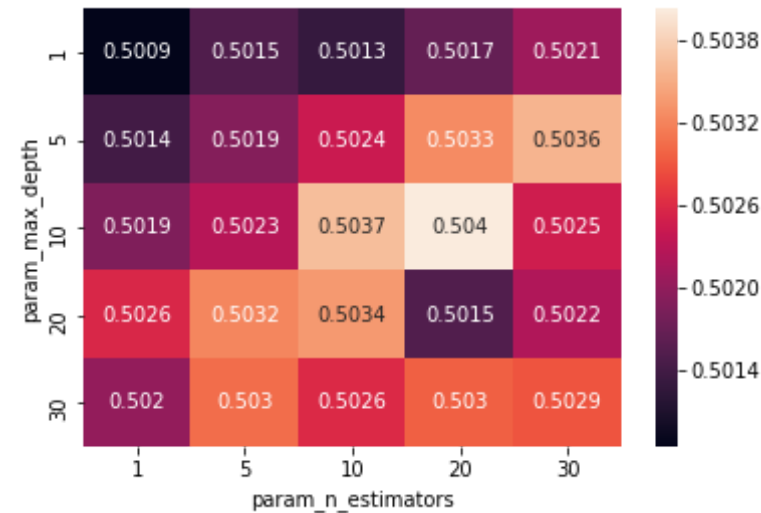
```
#from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,
30]}
model=GridSearchCV(RandomForestClassifier(),tuned_parameters,scoring='r
oc_auc',cv=10,return_train_score=True)
model.fit(tfidf_sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(tfidf_sent_vectors_cv,y_cv))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
                        max_depth=10, max_features='auto', max_leaf_node
s=None,
                        min_impurity_decrease=0.0, min_impurity_split=No
ne,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=20,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)

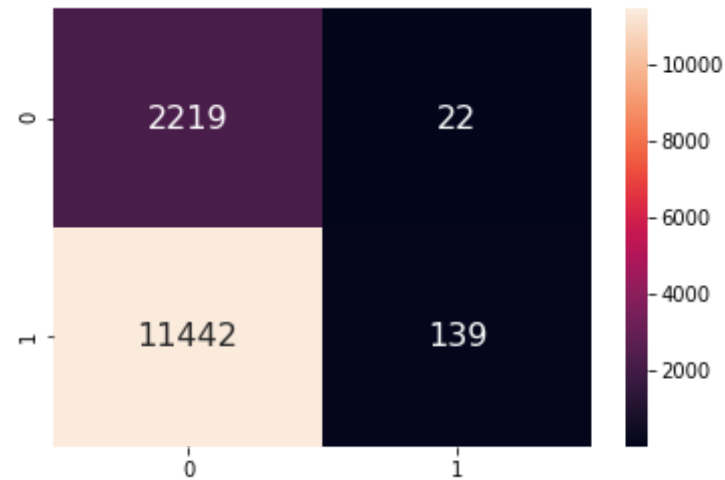
0.4981690044307894
```

```
In [59]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_s
core')
sns.heatmap(scores,annot=True,fmt='.4g')
```

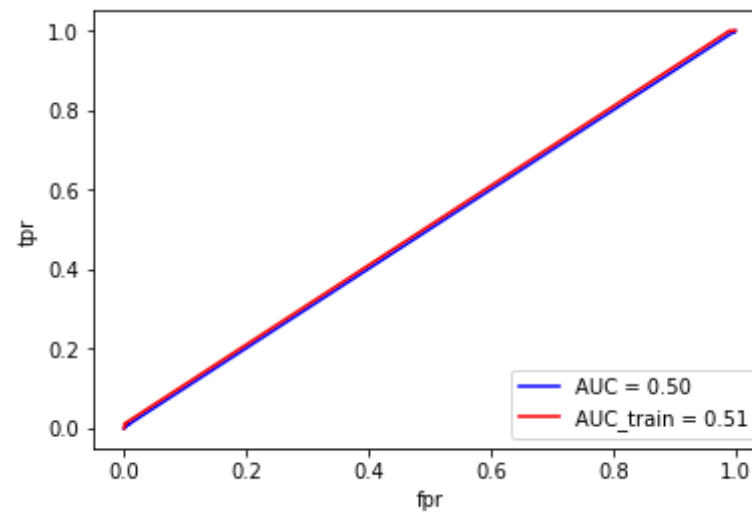
```
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x1a49b3a588>
```



```
In [60]: model_new=RandomForestClassifier(n_estimators=optimal_estimator,max_depth=optimal_depth,class_weight='balanced')
model_new.fit(tfidf_sent_vectors_train,y_tr)
pred=model_new.predict(tfidf_sent_vectors_test)
probab=model_new.predict_proba(tfidf_sent_vectors_test)[: ,1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
pred2=model_new.predict(tfidf_sent_vectors_train)
probab2=model_new.predict_proba(tfidf_sent_vectors_train)[: ,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
#heatmap for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```



Out[60]: Text(0,0.5,'tpr')



[5.2] Applying GBDT using XGBOOST

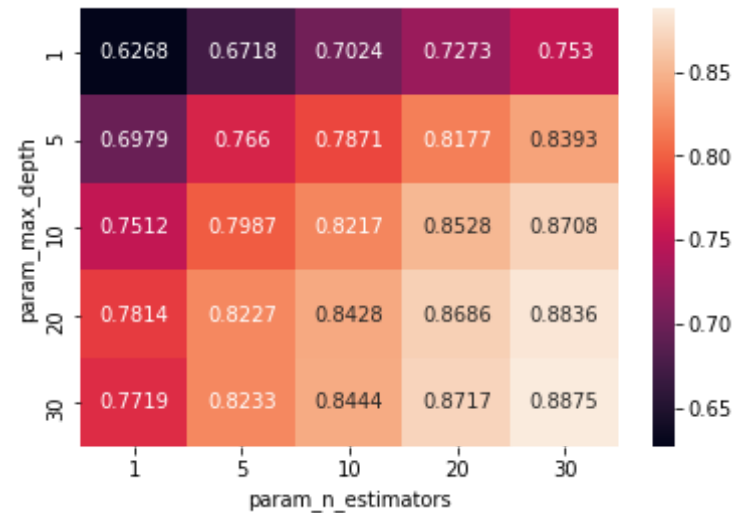
[5.2.1] Applying XGBOOST on BOW, SET 1


```
In [61]: # Please write all the code with proper documentation
import xgboost as xgb
#from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,30]}
model=GridSearchCV(xgb.XGBClassifier(),tuned_parameters,scoring='roc_auc',cv=10,return_train_score=True)
model.fit(final_counts,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(X_cv_bow,y_cv))

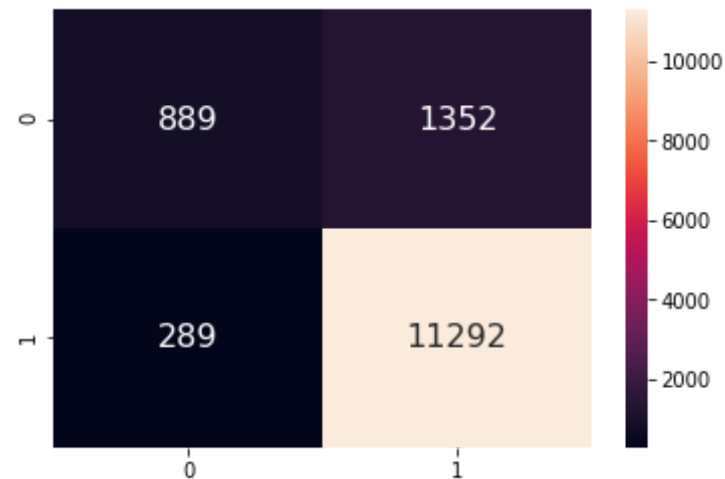
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=30,
              min_child_weight=1, missing=None, n_estimators=30, n_jobs
=1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.883349305602614
```

```
In [62]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_score')
sns.heatmap(scores,annot=True,fmt='.4g')
```

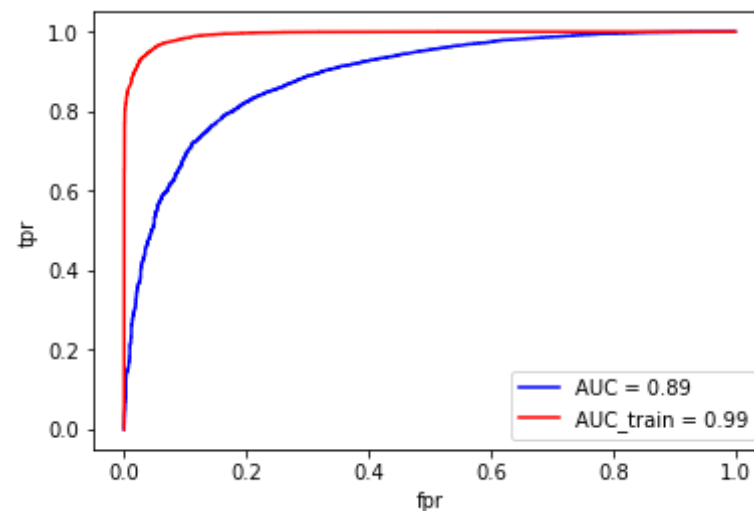
```
Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4a6194e0>
```



```
In [63]: model_new=xgb.XGBClassifier(n_estimators=optimal_estimator,max_depth=op
        timal_depth,class_weight='balanced')
        model_new.fit(final_counts,y_tr)
        pred=model_new.predict(X_test_bow)
        probab=model_new.predict_proba(X_test_bow)[: ,1]
        acc=accuracy_score(y_test,pred,normalize=True)*float(100)
        fpr,tpr,thresholds=roc_curve(y_test,probab)
        auc=roc_auc_score(y_test,probab)
        pred2=model_new.predict(final_counts)
        probab2=model_new.predict_proba(final_counts)[: ,1]
        fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
        auc2=roc_auc_score(y_tr,probab2)
        df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
        #heatmap for visualization of matrix
        sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
        plt.show()
        plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
        plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
        plt.legend(loc='lower right')
        plt.xlabel('fpr')
        plt.ylabel('tpr')
```



Out[63]: Text(0,0.5,'tpr')



[5.2.2] Applying XGBOOST on TFIDF, SET 2

```
In [64]: # Please write all the code with proper documentation
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,
```

```

30]}
model=GridSearchCV(xgb.XGBClassifier(),tuned_parameters,scoring='roc_auc',cv=10)
model.fit(final_tf_idf,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(X_cv_tfidf,y_cv))

```

```

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=30,
              min_child_weight=1, missing=None, n_estimators=30, n_jobs
=1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8900304208783618

```

```

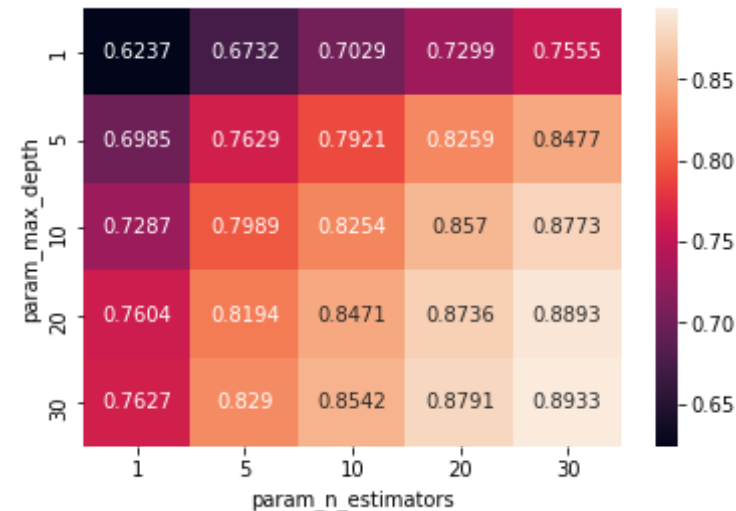
In [65]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_score')
sns.heatmap(scores,annot=True,fmt='.4g')

```

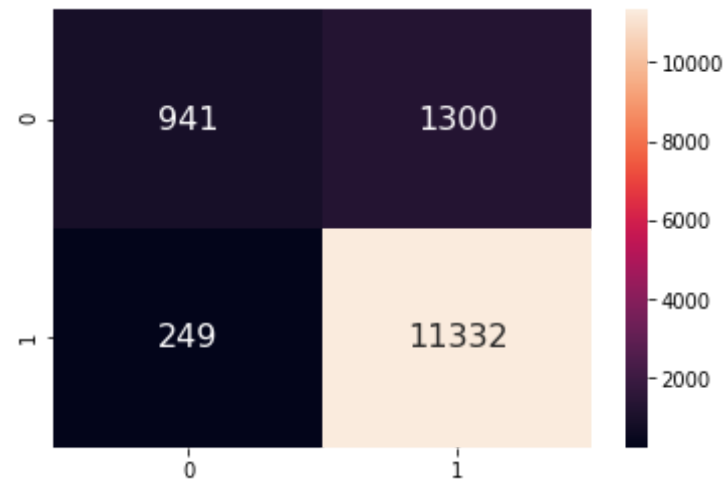
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4d46a278>

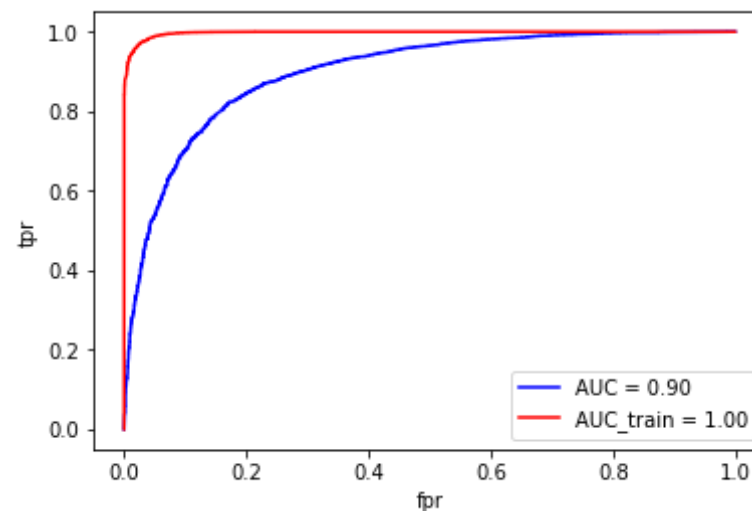
```



```
In [66]: model_new=xgb.XGBClassifier(n_estimators=optimal_estimator,max_depth=optimal_depth,class_weight='balanced')
model_new.fit(final_tf_idf,y_tr)
pred=model_new.predict(X_test_tfidf)
probab=model_new.predict_proba(X_test_tfidf)[:,:1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
pred2=model_new.predict(final_tf_idf)
probab2=model_new.predict_proba(final_tf_idf)[:,:1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
#heatmap for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```



Out[66]: Text(0,0.5,'tpr')



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

```
In [67]: # Please write all the code with proper documentation
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,
```

```

30]}
model=GridSearchCV(xgb.XGBClassifier(),tuned_parameters,scoring='roc_auc',cv=10)
model.fit(sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(sent_vectors_cv,y_cv))

```

```

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=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=30, n_jobs
=1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.877416977874651

```

```

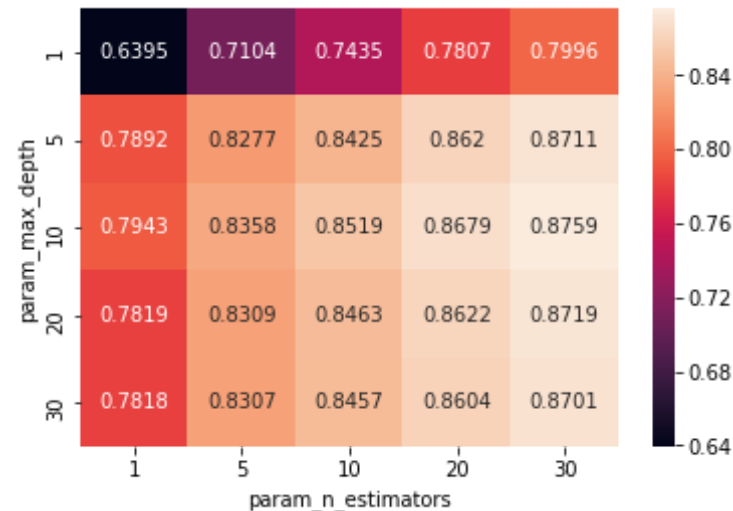
In [68]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_score')
sns.heatmap(scores,annot=True,fmt='.4g')

```

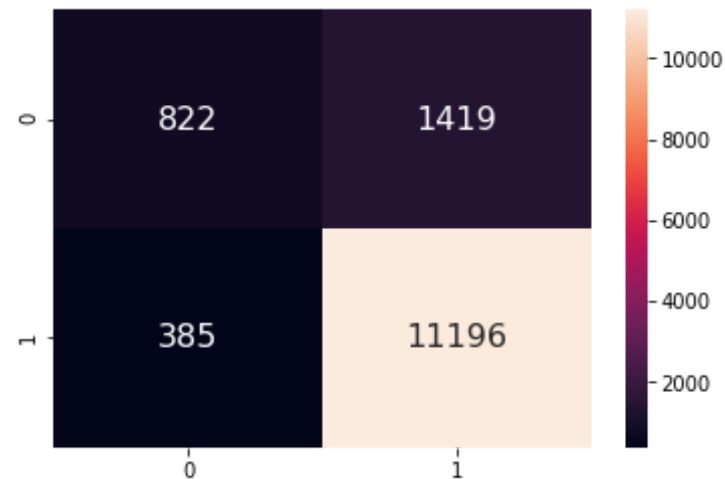
```

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4b193278>

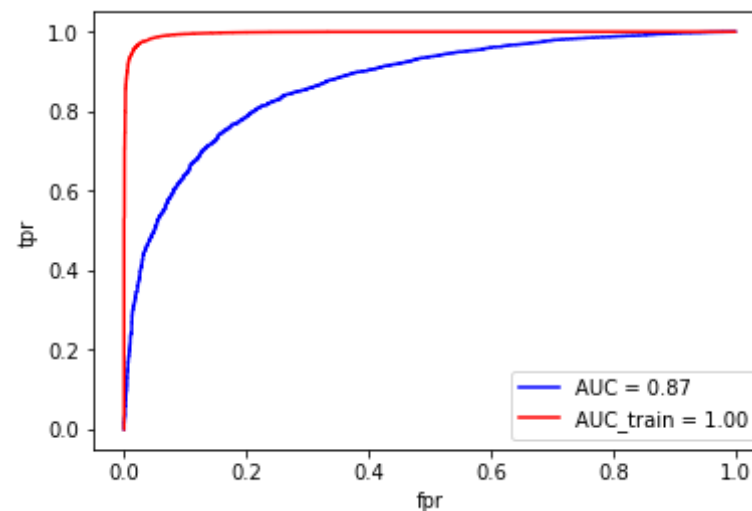
```



```
In [69]: model_new=xgb.XGBClassifier(n_estimators=optimal_estimator,max_depth=optimal_depth,class_weight='balanced')
model_new.fit(sent_vectors_train,y_tr)
pred=model_new.predict(sent_vectors_test)
probab=model_new.predict_proba(sent_vectors_test)[:,:1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
pred2=model_new.predict(sent_vectors_train)
probab2=model_new.predict_proba(sent_vectors_train)[:,:1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
#heatmap for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```

Out[69]: Text(0,0.5,'tpr')



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

```
In [70]: # Please write all the code with proper documentation
tuned_parameters={'n_estimators':[1,5,10,20,30], 'max_depth':[1,5,10,20,
```

```

30]}
model=GridSearchCV(xgb.XGBClassifier(),tuned_parameters,scoring='roc_auc',cv=10)
model.fit(tfidf_sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_estimator=model.best_estimator_.n_estimators
optimal_depth=model.best_estimator_.max_depth
print(model.score(tfidf_sent_vectors_cv,y_cv))

```

```

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=1,
              min_child_weight=1, missing=None, n_estimators=1, n_jobs=
1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.5009183148204216

```

```

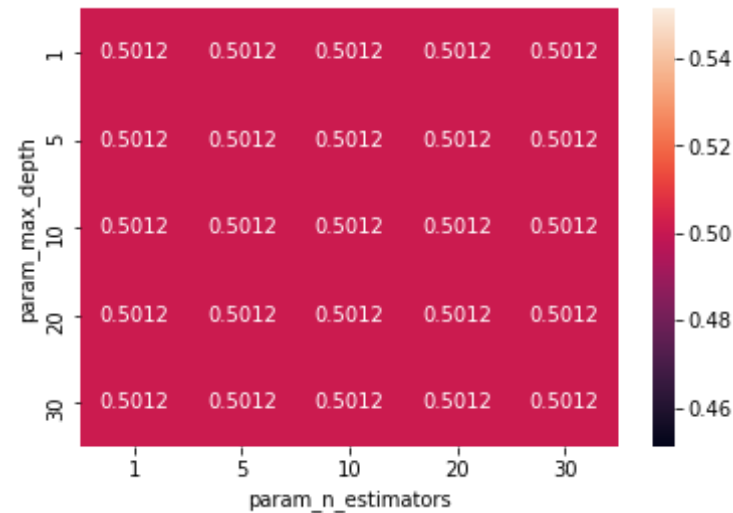
In [71]: scores=pd.DataFrame(model.cv_results_)
scores=scores.pivot('param_max_depth','param_n_estimators','mean_test_score')
sns.heatmap(scores,annot=True,fmt='.4g')

```

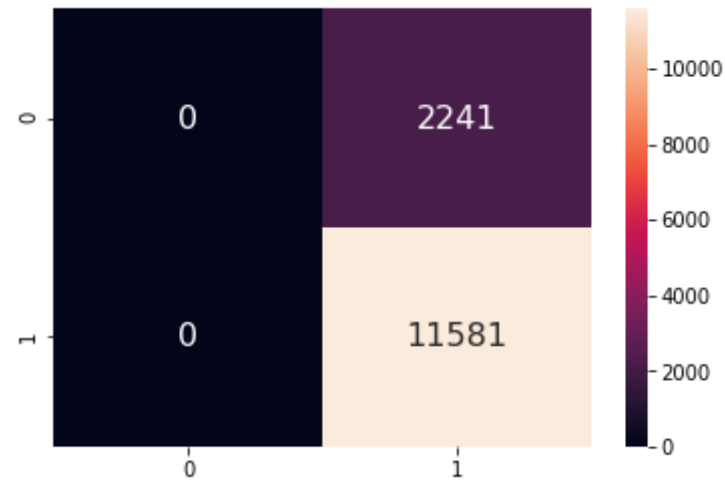
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4d5a3240>

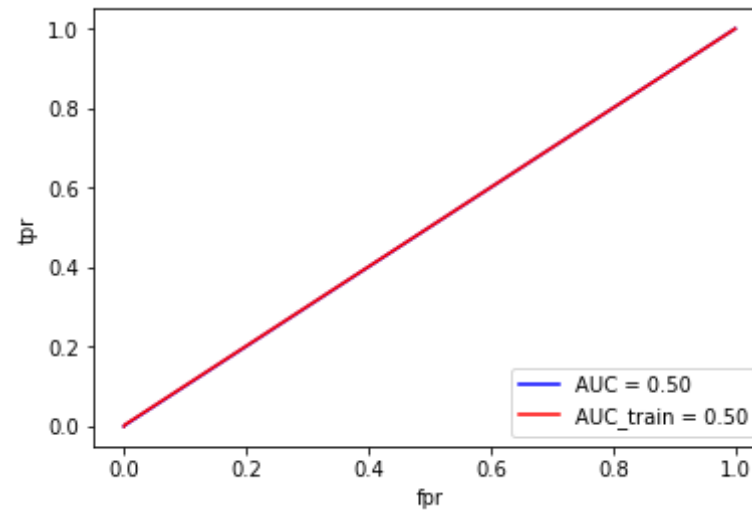
```



```
In [72]: model_new=xgb.XGBClassifier(n_estimators=optimal_estimator,max_depth=op
        timal_depth,class_weight='balanced')
        model_new.fit(tfidf_sent_vectors_train,y_tr)
        pred=model_new.predict(tfidf_sent_vectors_test)
        probab=model_new.predict_proba(tfidf_sent_vectors_test)[: ,1]
        acc=accuracy_score(y_test,pred,normalize=True)*float(100)
        fpr,tpr,thresholds=roc_curve(y_test,probab)
        auc=roc_auc_score(y_test,probab)
        pred2=model_new.predict(tfidf_sent_vectors_train)
        probab2=model_new.predict_proba(tfidf_sent_vectors_train)[: ,1]
        fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
        auc2=roc_auc_score(y_tr,probab2)
        df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
        #heatmap for visualization of matrix
        sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
        plt.show()
        plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
        plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
        plt.legend(loc='lower right')
        plt.xlabel('fpr')
        plt.ylabel('tpr')
```



Out[72]: Text(0,0.5,'tpr')



[6] Conclusions

In [73]: *# Please compare all your models using Prettytable library*

```

from prettytable import PrettyTable
x=PrettyTable()
x.field_names=(['Vectorizer','Algorithm','max_depth','min samples split', 'AUC'])
x.add_row(['BOW','Random Forest Classifier',40,40,0.90])
x.add_row(['TFIDF','Random Forest Classifier',30,30,0.91])
x.add_row(['Avg W2V','Random Forest Classifier',10,30,0.86])
x.add_row(['TFIDF weighted W2V','Random Forest Classifier',20,10,0.50])
x.add_row(['BOW','XGBoost',30,30,0.89])
x.add_row(['TFIDF','XGBoost',30,30,0.90])
x.add_row(['Avg W2V','XGboost',10,30,0.87])
x.add_row(['TFIDF weighted W2V','XGBoost',1,1,0.50])
print(x)

```

```

+-----+-----+-----+-----+
-----+-----+
|      Vectorizer      |      Algorithm      | max_depth | min samples split | AUC |
+-----+-----+-----+-----+
-----+-----+
|      BOW      | Random Forest Classifier |      40      |      4      | 0.9 | | |
|      | 0.9 |      |      |      |      |      |
|      TFIDF      | Random Forest Classifier |      30      |      3      | 0.91 |
|      | 0.91 |      |      |      |      |      |
|      Avg W2V      | Random Forest Classifier |      10      |      3      | 0.86 |
|      | 0.86 |      |      |      |      |      |
| TFIDF weighted W2V | Random Forest Classifier |      20      |      1      | 0.5 |
|      | 0.5 |      |      |      |      |      |
|      BOW      |      XGBoost      |      30      |      3      | 0.89 |
|      | 0.89 |      |      |      |      |      |
|      TFIDF      |      XGBoost      |      30      |      3      | 0.9 |
|      | 0.9 |      |      |      |      |      |
|      Avg W2V      |      XGboost      |      10      |      3      | 0.87 |
|      | 0.87 |      |      |      |      |      |
| TFIDF weighted W2V |      XGBoost      |      1      |      1      | 0.5 |
|      | 0.5 |      |      |      |      |      |
+-----+-----+-----+-----+
-----+-----+

```