Bagging_and_Boosting

June 14, 2020

1 Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

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

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

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

[0]:
```
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
```

[2]:
```
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&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
```

ûûûûûûûûûû
Mounted at /content/drive

[3]:
```
# using SQLite Table to read data.

con = sqlite3.connect('drive/My Drive/FFRDB/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 100000""", con)

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

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

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

[3]:
```
   Id  ...                                               Text
0   1  ...  I have bought several of the Vitality canned d...
1   2  ...  Product arrived labeled as Jumbo Salted Peanut...
2   3  ...  This is a confection that has been around a fe...

[3 rows x 10 columns]
```

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

```
""", con)
```

[5]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

[5]:
```
                 UserId  ... COUNT(*)
0  #oc-R115TNMSPFT9I7  ...        2
1  #oc-R11D9D7SHXIJB9  ...        3
2  #oc-R11DNU2NBKQ23Z  ...        2
3  #oc-R11O5J5ZVQE25C  ...        3
4  #oc-R12KPBODL2B5ZD  ...        2

[5 rows x 7 columns]
```

[6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

[6]:
```
              UserId  ... COUNT(*)
80638  AZY10LLTJ71NX  ...        5

[1 rows x 7 columns]
```

[7]:
```
display['COUNT(*)'].sum()
```

[7]:
```
393063
```

3 [2] Exploratory Data Analysis

3.1 [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:

[8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

[8]:
```
       Id  ...                                               Text
0   78445  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  138317  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  138277  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3   73791  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  155049  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

```
[5 rows x 10 columns]
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

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

[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')
```

[10]:
```
#Deduplication of entries
final=sorted_data.
↪drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first',␣
↪inplace=False)
final.shape
```

[10]:
```
(87775, 10)
```

[11]:
```
final.sort_values('Time',inplace=True)
print(final.head(5))
```

```
          Id  ...                                               Text
70688  76882  ...  I bought a few of these after my apartment was...
1146    1245  ...  This was a really good idea and the final prod...
1145    1244  ...  I just received my shipment and could hardly w...
28086  30629  ...  Nothing against the product, but it does bothe...
28087  30630  ...  I love this stuff. It is sugar-free so it does...

[5 rows x 10 columns]
```

[12]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

[12]:
```
87.775
```

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

[13]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
```

```
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

[13]:
```
      Id  ...                                               Text
0  64422  ...  My son loves spaghetti so I didn't hesitate or...
1  44737  ...  It was almost a 'love at first bite' - the per...

[2 rows x 10 columns]
```

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

[15]:
```
#Before starting the next phase of preprocessing lets see the number of entries␣
 ↪left
print(final.shape)

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

```
(87773, 10)
```

[15]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

[16]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
```

```
print("="*50)

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

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

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

```
I bought a few of these after my apartment was infested with fruit flies. After
only a few hours, the trap had &quot;attracted&quot; many flies and within a few
days they were practically gone. This may not be a long term  solution, but if
flies are driving you crazy, consider buying this. One  caution- the surface is
very sticky, so try to avoid touching it.
==================================================
we use this as the base, then besides the chicken, we will also add pasta,
spices, veggies, or whatever we have around to make quick cheesy meals
==================================================
My dogs just love this food. The service is always fast and reliable.
==================================================
I am amazed by how well this tea works to relieve my chronic congestion and
recurring sinus problems. And it's not just a "quick" fix either -- its
therapeutic effects last for hours. I was a bit worried the tea would be a bit
too "licorice-y" since one of its main ingredients is licorice root, but the
fragrance and taste are mild and incredibly soothing. If you think this package
of six boxes is too much, you'll be happily proven wrong ... I would stock my
entire garage with this tea if I could!
==================================================
```

[17]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
I bought a few of these after my apartment was infested with fruit flies. After
only a few hours, the trap had &quot;attracted&quot; many flies and within a few
days they were practically gone. This may not be a long term  solution, but if
flies are driving you crazy, consider buying this. One  caution- the surface is
very sticky, so try to avoid touching it.
```

[18]:
```
# https://stackoverflow.com/questions/16206380/
↪python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

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

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

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

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

```
I bought a few of these after my apartment was infested with fruit flies. After
only a few hours, the trap had "attracted" many flies and within a few days they
were practically gone. This may not be a long term  solution, but if flies are
driving you crazy, consider buying this. One  caution- the surface is very
sticky, so try to avoid touching it.
==================================================
we use this as the base, then besides the chicken, we will also add pasta,
spices, veggies, or whatever we have around to make quick cheesy meals
==================================================
My dogs just love this food. The service is always fast and reliable.
==================================================
I am amazed by how well this tea works to relieve my chronic congestion and
recurring sinus problems. And it's not just a "quick" fix either -- its
therapeutic effects last for hours. I was a bit worried the tea would be a bit
too "licorice-y" since one of its main ingredients is licorice root, but the
fragrance and taste are mild and incredibly soothing. If you think this package
of six boxes is too much, you'll be happily proven wrong ... I would stock my
entire garage with this tea if I could!
```

[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
```

[20]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
My dogs just love this food. The service is always fast and reliable.
==================================================
```

[21]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
I bought a few of these after my apartment was infested with fruit flies. After
only a few hours, the trap had &quot;attracted&quot; many flies and within a few
days they were practically gone. This may not be a long term  solution, but if
flies are driving you crazy, consider buying this. One  caution- the surface is
very sticky, so try to avoid touching it.
```

[22]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
My dogs just love this food The service is always fast and reliable
```

[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 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', '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', "weren't", \
                'won', "won't", 'wouldn', "wouldn't"])
```

[24]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in␣
↪stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87773/87773 [00:30<00:00, 2869.57it/s]
```

[25]:
```
preprocessed_reviews[1500]
```

[25]: 'dogs love food service always fast reliable'

[3.2] Preprocessing Review Summary

[0]:
```python
## Similartly you can do preprocessing for review summary also.
```

5 [4] Featurization

5.1 [4.1] Loading 100k pickled data for Tfidf and Avg word to vec

[0]:
```python
dbfile1 = open('/content/drive/My Drive/FFRDB/tfidf.pkl', 'rb')
tfidf_sent_vectors = pickle.load(dbfile1)


dbfile2 = open('/content/drive/My Drive/FFRDB/sent_vectors.pkl', 'rb')
sent_vectors= pickle.load(dbfile2)
```

6 [5] Assignment 9: Random Forests

```
<li><strong>Apply Random Forests & GBDT on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vector
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engin
        <ul>
        <li>Taking length of reviews as another feature.</li>
```

```
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='3d_plot.JPG' width=500px> with X-axis as <strong>n_estimators</strong>, Y-axis as <st
        <p style="text-align:center;font-size:30px;color:red;"><strong>(or)</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='heat_map.JPG' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.heat
<li>You choose either of the plotting techniques out of 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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.

6.1 [5.1] Applying RF

6.1.1 [5.1.1] Applying Random Forests on BOW, SET 1

[0]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews[:
 ↪100000],final['Score'].values[:100000],test_size=0.3,random_state=0)

vectorizer = CountVectorizer(ngram_range=(1,2), min_df=10, max_features= 500)
vectorizer.fit(X_train)
X_train = vectorizer.transform(X_train)
```

```
X_test = vectorizer.transform(X_test)


print(X_train.shape)
print(X_test.shape)
```

```
(61441, 500)
(26332, 500)
```

[0]:
```
depth=[i for i in np.arange(9,15)]
estimators= [i for i in np.arange(100,200,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(class_weight='balanced',n_jobs=-1)
temp_gscv=␣
 ↪GridSearchCV(clf,param,cv=3,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  6.3min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 16.1min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 33.1min
[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed: 45.5min finished
```

[0]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=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=100, n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [9, 10, 11, 12, 13, 14],
```

```
                         'n_estimators': [100, 105, 110, 115, 120, 125, 130,
                                          135, 140, 145, 150, 155, 160, 165,
                                          170, 175, 180, 185, 190, 195]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

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

#code snippet from provided 3d scappter plot .ipynb file



import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[0]:
```
#finding the best CV scores that is maximas then using the one which is least␣
 ↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
```

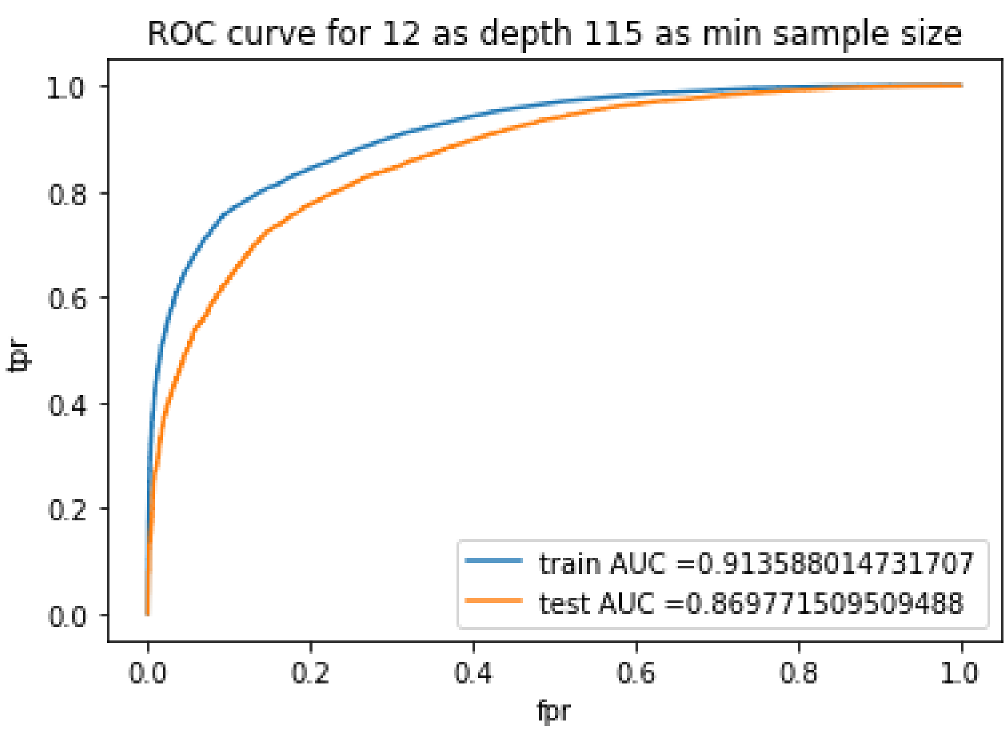
```python
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=1
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=k

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

```
all local differences [0.036376282198051, 0.034730676446487085,
0.034925850462771746, 0.034925850462771746, 0.03672660780676584,
```

```
0.035837881218603984, 0.035837881218603984, 0.03617726512887609,
0.03690783417377608, 0.03690783417377608, 0.03529647756770771,
0.0360204987003635, 0.03483485108275042, 0.03582665865290868,
0.03591543122094509, 0.035783334007727374, 0.03601503615001522,
0.035308326585215544, 0.035308326585215544, 0.036650471561642584,
0.0359809667129442 6, 0.04176695380571649, 0.04151256871173925,
0.04177000074945214, 0.0423194558096587 9, 0.04164331850108971,
0.04098089623622814, 0.04098089623622814, 0.04235111107415057,
0.04132031764006028, 0.0410498799267609, 0.04218071396032652,
0.04118688287123473, 0.04109549675846891, 0.04150410682620287,
0.041928790728530285, 0.041928790728530285, 0.04037129364801728,
0.04120506454206696, 0.04120506454206696, 0.04218522217373333,
0.041688720209441144, 0.04823454642002778, 0.04628687473047122,
0.046844658605294076, 0.046844658605294076, 0.04691016814444604,
0.04801073874662487, 0.047845291364166154, 0.04751858209576765,
0.04651365642779215, 0.04651365642779215, 0.04797403578740733,
0.046463255133331494, 0.046463255133331494, 0.047111986806832706,
0.0473373100213957 4, 0.0473373100213957 4, 0.0475750597431267,
0.04689953573988337, 0.04689953573988337, 0.046754968370840144,
0.0473763126571618, 0.04674468193947001, 0.046934703387670695,
0.047463791200170524, 0.047463791200170524, 0.047789647286129444,
0.053055253063025165, 0.0536613487713824, 0.05243031036600565,
0.05332543280776714, 0.05378397207924934, 0.052773039585432,
0.05337499646541044, 0.052281406523750795, 0.05247109499057989,
0.05293975255268868, 0.05293975255268868, 0.05257526890140152,
0.05328210877793127, 0.05328210877793127, 0.051796091566501334,
0.052369119282190124, 0.053154199494873655, 0.05345199380110921,
0.059347891160457134, 0.059347891160457134, 0.05738195765097498,
0.05814923148755835, 0.05937137248027524, 0.05847622667590502,
0.05930370964648968, 0.05816717795118209, 0.05845286390017301,
0.05848475809987974, 0.05950031839121639, 0.05922120563053068,
0.05779823171923437, 0.05779823171923437, 0.057996689703797455,
0.057414498892739974, 0.057414498892739974, 0.058804761594071,
0.058366902428587664, 0.058366902428587664, 0.05840711095526219,
0.05821129967355587, 0.05821129967355587, 0.05778488948353877,
0.06429747654738771, 0.06303064021015492, 0.06429773187677756,
0.0641748080499921, 0.0641748080499921, 0.06482000425633538,
0.06417711757286826, 0.06417711757286826, 0.06442258462151651,
0.0637020203409843 8, 0.06311137318898374, 0.0637827619928849,
0.06425258264886169, 0.06387947085585377, 0.06377206743481012,
0.06374364606136762]
best cv score to use = 0.8633584747863492
best Parameters at index 3
best depth to use = 12
best sample split size to use = 115
```

[0]:
```
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,␣
 ↪auc
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(max_depth=best_depth,n_estimators=best_estm,class_weight='balanced'
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[0]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)


# %matplotlib inline
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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
 ↪min sample size')
plt.legend()
# plt.grid()
plt.show()
```



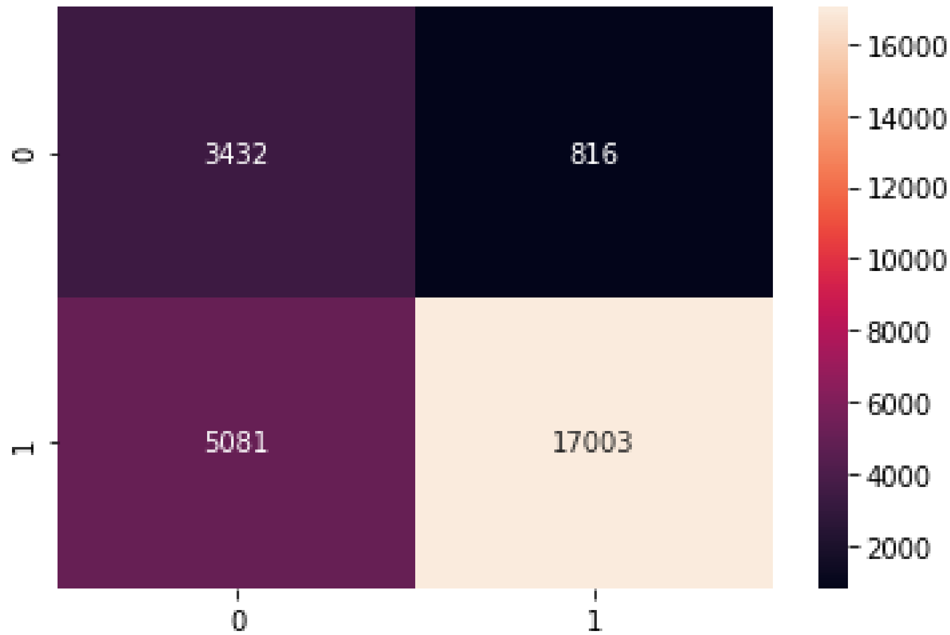
[0]:
```python
# This section of code where ever implemented is taken from sample kNN python
↪notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very
↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for
↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.6220289352313765 for threshold 0.521
```

[0]:
```python
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,
↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f313f9d6ef0>

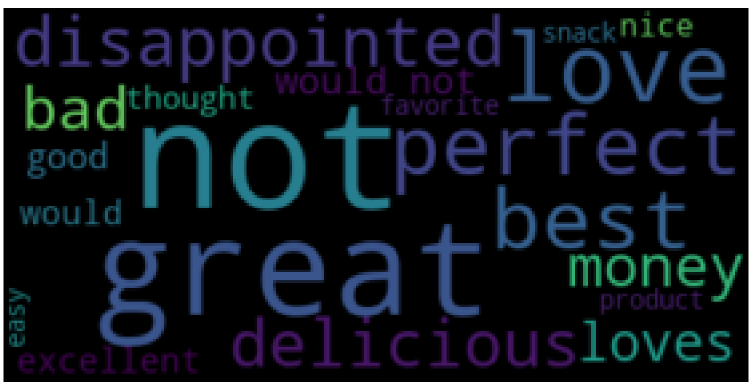


[0]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 77.61%
Precision on test set: 95.42%
recall score on test set: 76.99%
f1 score on test set: 85.22%
```

[0]:
```
count=0
value=[]
for i in clf.feature_importances_.reshape(-1,1):
  count+=1
  value.extend(i)

x=vectorizer.get_feature_names()

features= pd.DataFrame({'feature_name':x,'value':value})
Positive=features.sort_values(by = ['value'],␣
 ↪ascending=False,ignore_index=True).head(20)
```

[0]:
```
import matplotlib.pyplot as pPlot
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image

#taken from stack overflow
d = {}
for a, x in zip(Positive['feature_name'],Positive['value']):
    d[a] = x
wordcloud = WordCloud(stopwords=set(STOPWORDS))

wordcloud.generate_from_frequencies(frequencies=d)
plt.figure(figsize=(5,3) )
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```



6.1.2 [5.1.3] Applying Random Forests on TFIDF, SET 2

[0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =␣
↪train_test_split(preprocessed_reviews,final['Score'].values,test_size=0.
↪3,random_state=0)

vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features= 500)
vectorizer.fit(X_train)
X_train = vectorizer.transform(X_train)
X_test = vectorizer.transform(X_test)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(61441, 500)
(26332, 500)
```

[0]:
```
depth=[i for i in np.arange(9,15)]
estimators= [i for i in np.arange(100,200,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(class_weight='balanced',n_jobs=-1)
temp_gscv=␣
 ↪GridSearchCV(clf,param,cv=3,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  6.5min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 33.5min
[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed: 46.2min finished
```

[0]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=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=100, n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [9, 10, 11, 12, 13, 14],
                         'n_estimators': [100, 105, 110, 115, 120, 125, 130,
                                          135, 140, 145, 150, 155, 160, 165,
                                          170, 175, 180, 185, 190, 195]},
```

```
            pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
            scoring='roc_auc', verbose=5)
```

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

#code snippet from provided 3d scappter plot .ipynb file



import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[0]:
```
#finding the best CV scores that is maximas then using the one which is least␣
 ↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
```

```
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=1
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=k

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

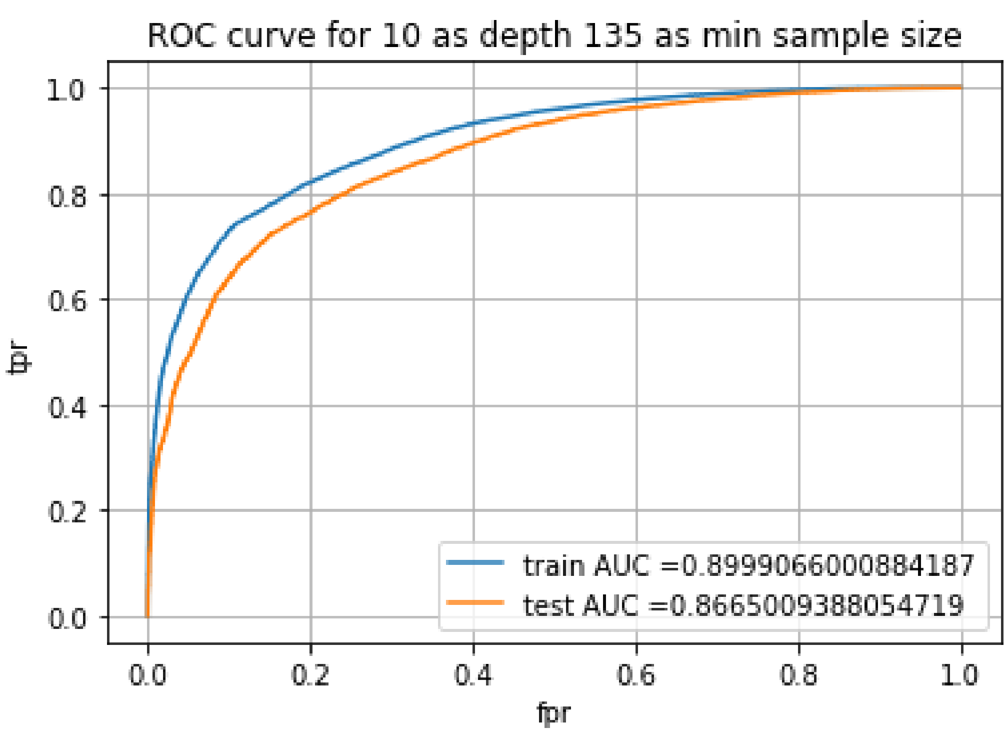
```
all local differences [0.0351919197894891, 0.03596554956658227,
0.035822853375630626, 0.03625191764723057, 0.03561170146353776,
0.03580619716843503, 0.03580619716843503, 0.0349039340939038,
0.03633432881283072, 0.03542343969321926, 0.03554898781030813,
0.03646261250297611, 0.03646261250297611, 0.036859155531381416,
```

```
0.035666757945348015, 0.03578053028440098, 0.035156352669036495,
0.03559250729313557, 0.03559250729313557, 0.035217919176834256,
0.03557184058863894, 0.03573480462848255, 0.04177778366684737,
0.040388860536362925, 0.040388860536362925, 0.04156480210765312,
0.041386646549183115, 0.041386646549183115, 0.041840538623983514,
0.04138583849322108, 0.04059493657758029, 0.04152603201118843,
0.040722671882578276, 0.040722671882578276, 0.04181023466614864,
0.04168447519886853, 0.041577752732621875, 0.04204141747980905,
0.04089503441648179, 0.04120566005102255, 0.04089954645637761,
0.04160839968816121, 0.04163423080133988, 0.042007871918772755,
0.047015554058738696, 0.046528929646384776, 0.04661576872123607,
0.04789676267108123, 0.04728041677884498, 0.046902428635551274,
0.04681426906261699, 0.04681426906261699, 0.047298322456389985,
0.048068950448377445, 0.048068950448377445, 0.04767552189821567,
0.047690894917739035, 0.047690894917739035, 0.04584550617887151,
0.0474126490390032, 0.046807353680596187, 0.04728960319360409,
0.047070768881536496, 0.04804088422343045, 0.0466564329332555,
0.0528355303886725, 0.0528355303886725, 0.05330209830803767,
0.053560446236684434, 0.053560446236684434, 0.05270290891864016,
0.053166875087143795, 0.053166875087143795, 0.05192625559635,
0.052134804971897486, 0.0524233778693729, 0.05320713478005934,
0.05204740052568313, 0.05204740052568313, 0.053828552286370734,
0.05285370024688629, 0.05176571159274086, 0.052286008245261195,
0.05278675669020627, 0.052793259337826415, 0.05177240467816446,
0.052447134850453714, 0.05841204854200166, 0.0581369090152517,
0.05871873018483609, 0.05871873018483609, 0.05759352611387358,
0.05815019536269683, 0.0582677172513999, 0.05880655151635039,
0.057291435464244755, 0.05927229876536033, 0.058723603084238296,
0.05949078352145509, 0.05949078352145509, 0.058770676058693794,
0.059006763643618565, 0.05828629026344567, 0.05847156616070226,
0.05855744722803946, 0.05855744722803946, 0.05805807189614498,
0.05855017710788746, 0.05855017710788746, 0.06455477055392134,
0.06489021546523899, 0.06423422747480723, 0.06384145374709038,
0.06485688862436745, 0.06452734802883175, 0.06409845219440147,
0.06326959814928379, 0.06326959814928379, 0.06311666374653091,
0.06297369058275493, 0.06297369058275493, 0.0634751925974929,
0.06336722698524666, 0.06336722698524666, 0.06413288472216361,
0.06323057955025091, 0.06323057955025091, 0.0640085584839335,
0.06401286604066958]
best cv score to use = 0.8638699785103636
best Parameters at index 7
best depth to use = 10
best sample split size to use = 135
```

[0]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve, auc
```

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(max_depth=best_depth,n_estimators=best_estm,class_weight='balanced'
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[0]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)

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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
 ↪min sample size')
plt.legend()
plt.grid()
plt.show()
```



[0]:
```
# This section of code where ever implemented is taken from sample kNN python␣
 ↪notebook
```

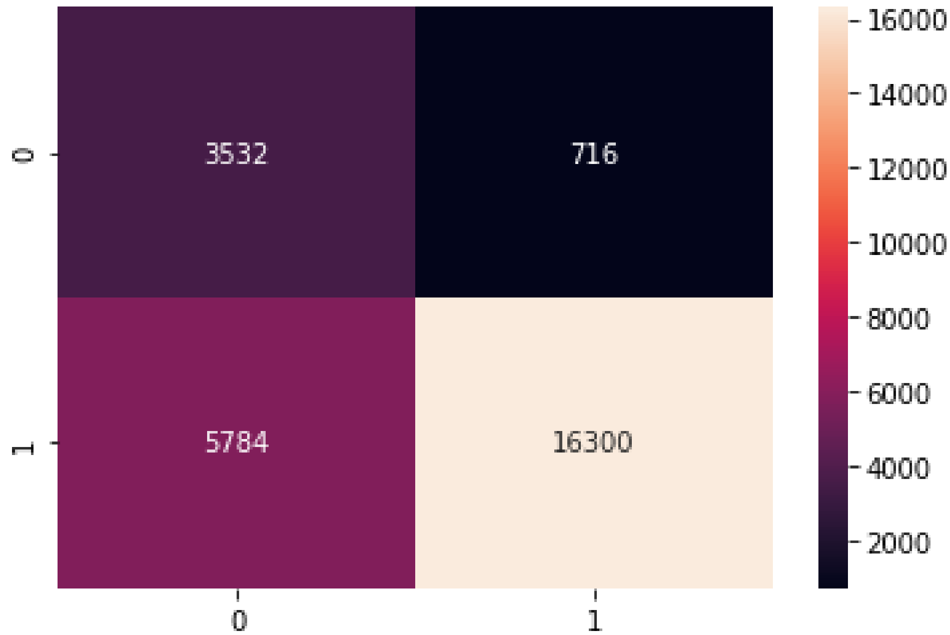
```python
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
 ↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.6136857695544251 for threshold 0.521
```

[0]:
```python
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
 ↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f313b8e3780>
```



[0]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 75.32%
Precision on test set: 95.79%
recall score on test set: 73.81%
f1 score on test set: 83.38%
```

[0]:
```
count=0
value=[]
for i in clf.feature_importances_.reshape(-1,1):
  count+=1
  value.extend(i)

x=vectorizer.get_feature_names()

features= pd.DataFrame({'feature_name':x,'value':value})
```

```
Positive=features.sort_values(by = ['value'],␣
↪ascending=False,ignore_index=True).head(20)
```

[0]:
```
import matplotlib.pyplot as pPlot
from wordcloud import WordCloud
import numpy as np
from PIL import Image

#taken from Stckpverflow
d = {}
for a, x in zip(Positive['feature_name'],Positive['value']):
    d[a] = x
wordcloud = WordCloud()

wordcloud.generate_from_frequencies(frequencies=d)
plt.figure( figsize=(10,5) )
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

6.1.3 [5.1.5] Applying Random Forests on AVG W2V, SET 3

[0]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(sent_vectors,final['Score'].
↪values,test_size=0.3,random_state=0)
```

[0]:
```
depth=[i for i in np.arange(9,15)]
estimators= [i for i in np.arange(100,200,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(class_weight='balanced',n_jobs=-1)
temp_gscv=␣
 ↪GridSearchCV(clf,param,cv=3,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 28.0min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 68.1min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 129.9min
[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed: 170.2min finished
```

[0]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=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=100, n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [9, 10, 11, 12, 13, 14],
                         'n_estimators': [100, 105, 110, 115, 120, 125, 130,
                                          135, 140, 145, 150, 155, 160, 165,
                                          170, 175, 180, 185, 190, 195]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

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

#code snippet from provided 3d scappter plot .ipynb file
```

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[0]:

```
#finding the best CV scores that is maximas then using the one which is least␣
↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
```

```
k=[]
neighbor=1
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=list(set(k))

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

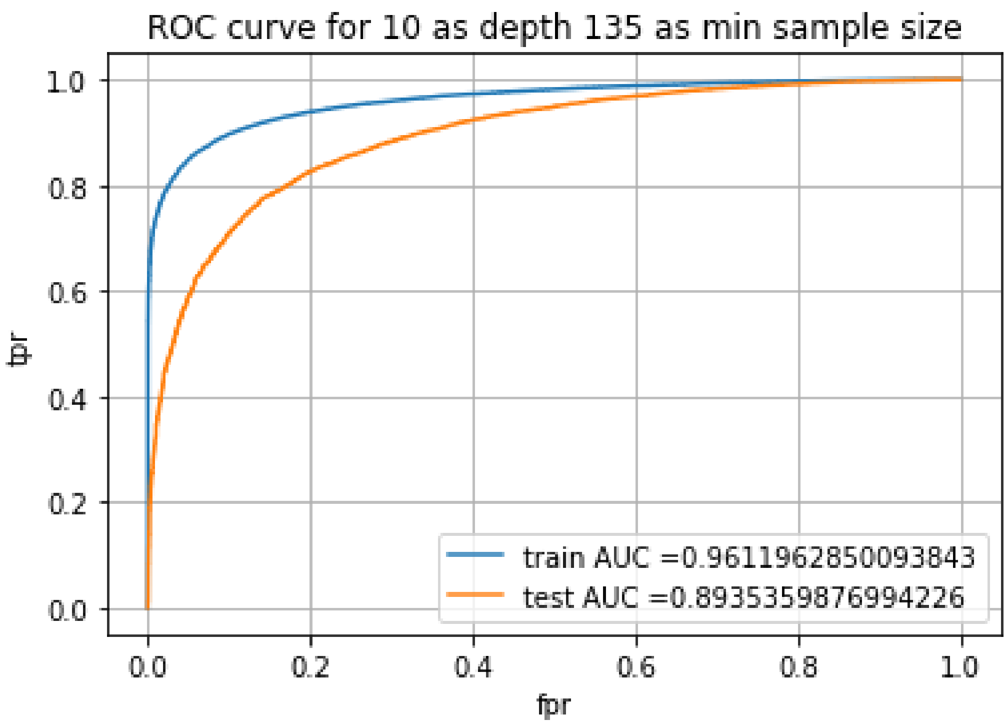
```
all local differences [0.06218463924990281, 0.06187116687205674,
0.062352292962120304, 0.06223994749428963, 0.06251681116572194,
0.06251681116572194, 0.06171029871838418, 0.06226804292651833,
0.06226804292651833, 0.06139255230448848, 0.06176924611519519,
0.06190878864042948, 0.06139593678968214, 0.0622523180216531,
0.06183122706661415, 0.0616591417885165, 0.06189197538777069,
0.07579858881801249, 0.07583629266857583, 0.07617416741057348,
0.07617416741057348, 0.07561922873706461, 0.07588395802333459,
0.07561827245389807, 0.07559779047093496, 0.075563274372215,
0.07545998429131495, 0.07536372799986457, 0.07535505982406843,
0.07535505982406843, 0.07533139697120239, 0.075249030342325,
```

```
0.07543035466746573, 0.07542126594161658, 0.07528155676557624,
0.07528155676557624, 0.07520536153526203, 0.07551794422197722,
0.08677474904358307, 0.08575013709567791, 0.08612103069185473,
0.08612103069185473, 0.08536081977137833, 0.0861813513139651,
0.08572572444886928, 0.08588281788807028, 0.08600282828558914,
0.08575641306476067, 0.0855227490614695, 0.0860008293716582, 0.0860008293716582,
0.0852887813917459, 0.08588462305357425, 0.08540774475619883,
0.08546447046181616, 0.08558165696471709, 0.08613934026174908,
0.08565270773111755, 0.09290930425953703, 0.09310619872966708,
0.09263250710649373, 0.09307554230100523, 0.09307554230100523,
0.09232922488775908, 0.09226231887399283, 0.09226231887399283,
0.0921846114018882 7, 0.0924075728767837, 0.0922963796746199, 0.0915515016677575,
0.09243767890917132, 0.09257846457497987, 0.09205320710609366,
0.09255706681284948, 0.09255706681284948, 0.0918444391941835,
0.09189378631293565, 0.09265656210208573, 0.09214924610142217,
0.09210976339328147, 0.09706103908402874, 0.09633356972584473,
0.09684508004559811, 0.09684508004559811, 0.09634853546223521,
0.0968440533770194, 0.09672839136611056, 0.09660703623502098,
0.09662843884524819, 0.09662843884524819, 0.09565133954749638,
0.09584502243395132, 0.09617086074024883, 0.09617749237544493,
0.09632171461494454, 0.09583232913788386, 0.09548099503036123,
0.09626948664613477, 0.09586930671674332, 0.09539765850703075,
0.0960748221670874, 0.09845956824736601, 0.0980539403908075,
0.09879114777095799, 0.09879114777095799, 0.09828941604552732,
0.09852025143535748, 0.09861022307861034, 0.09817323597239558,
0.0982975310496349, 0.09833146455187192, 0.09793284492564613,
0.09831286541475648, 0.09791681226811022, 0.09756319504649213,
0.09816103858728609]
best cv score to use = 0.8939216194872338
best Parameters at index 7
best depth to use = 10
best sample split size to use = 135
```

[0]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,auc
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(max_depth=best_depth,n_estimators=best_estm,class_weight='balanced'
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[0]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)
```

```
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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
 ↪min sample size')
plt.legend()
plt.grid()
plt.show()
```



[0]:
```
# This section of code where ever implemented is taken from sample kNN python␣
 ↪notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
 ↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
```

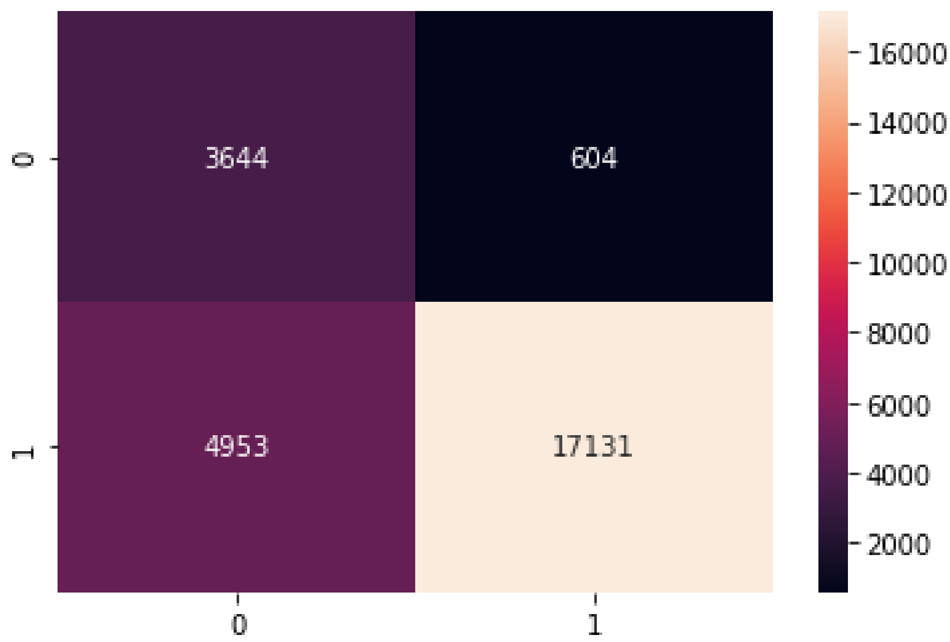
```
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.665424576458794 for threshold 0.607
```

[0]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
 ↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f313a38cf60>
```



[0]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
```

```
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 78.90%
Precision on test set: 96.59%
recall score on test set: 77.57%
f1 score on test set: 86.04%
```

6.1.4 [5.1.6] Applying Random Forests on TFIDF W2V, SET 4

[0]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =␣
 ↪train_test_split(tfidf_sent_vectors,final['Score'].values,test_size=0.
 ↪3,random_state=0)
```

[0]:
```
depth=[i for i in np.arange(9,15)]
estimators= [i for i in np.arange(100,200,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(class_weight='balanced',n_jobs=-1)
temp_gscv=␣
 ↪GridSearchCV(clf,param,cv=3,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 28.1min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 68.4min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 130.8min
[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed: 171.3min finished
```

[0]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
```

```
                                                  max_samples=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=100, n_jobs=-1,
                                                  oob_score=False,
                                                  random_state=None, verbose=0,
                                                  warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [9, 10, 11, 12, 13, 14],
                         'n_estimators': [100, 105, 110, 115, 120, 125, 130,
                                          135, 140, 145, 150, 155, 160, 165,
                                          170, 175, 180, 185, 190, 195]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

[0]:

```
train_auc=temp_gscv.cv_results_['mean_train_score']
cv_auc=temp_gscv.cv_results_['mean_test_score']

#code snippet from provided 3d scappter plot .ipynb file


import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
```

```
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

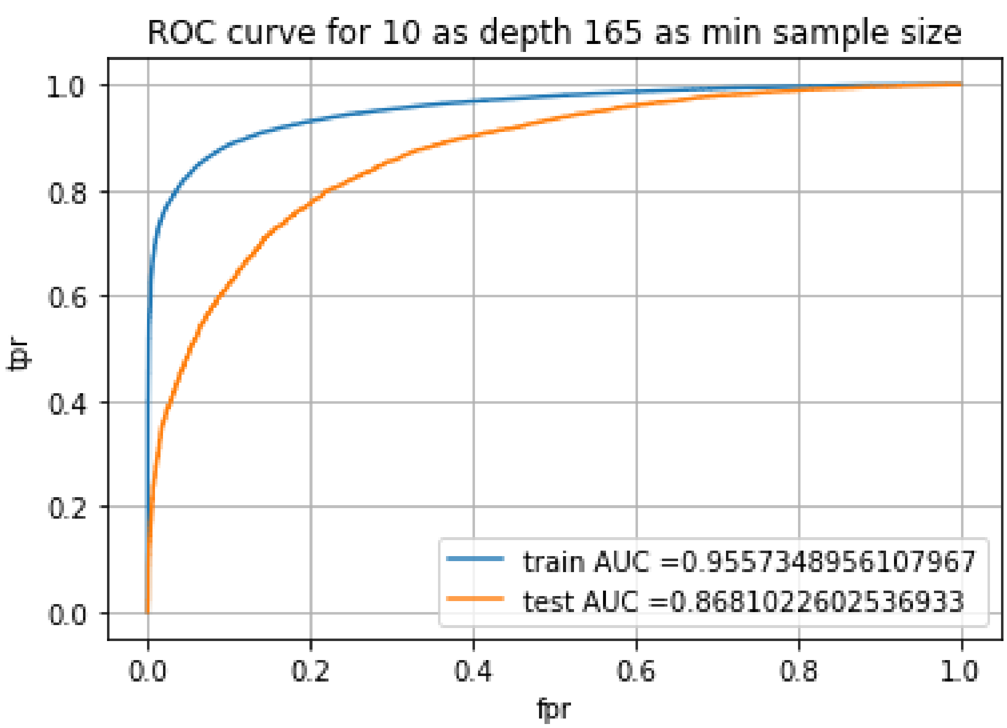
[0]:
```
#finding the best CV scores that is maximas then using the one which is least␣
↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=1
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=list(set(k))

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))
```

```python
# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

```
all local differences [0.07931785535627478, 0.07921020970675363,
0.07912567359576794, 0.07910854820252178, 0.07855726506873384,
0.0786498011116179, 0.07888378900717141, 0.07910800219315739,
0.07862491039698505, 0.0785959747512639, 0.078503895477529, 0.07895740332520895,
0.09609877386659416, 0.0958850725213739, 0.09593016284062539,
0.0958082965157947, 0.09570730231455715, 0.09550034225327642,
0.09586222529593647, 0.09594881056966287, 0.09585940960110573,
0.09565881564781464, 0.09632767563956035, 0.09561183051850808,
0.09593258421526596, 0.09593719496836661, 0.09514302805627262,
0.09547673716822003, 0.09559886240785176, 0.0952462462349889,
0.10852993257314014, 0.10853918833538345, 0.1079765016239076,
0.10860245746423591, 0.1081294790278916, 0.10850745201161471,
0.10766325886826955, 0.10893364000782468, 0.10848183069562811,
0.10775170306613668, 0.10792660742688853, 0.10821725894267575,
0.10753342777853836, 0.10793236170817755, 0.107867309016758,
0.10707145392504858, 0.10779645883509525, 0.10816840075193945,
0.10763771311182524, 0.11613994367693936, 0.11600172803664988,
0.11613421223698706, 0.11615395124215067, 0.1156629331108886,
0.11630360071669177, 0.11595222267214345, 0.11543300853955496,
0.1155585405838665, 0.11516135831505925, 0.11537069199538719,
0.11507682048377199, 0.11522077639037864, 0.1151949964640554,
0.11481706431102268, 0.11540531335497539, 0.11535451898158211,
0.11458807159398754, 0.12103845335259145, 0.11995405910317847,
0.11977283997543287, 0.12005563200112657, 0.11876559433642642,
0.11944259276105162, 0.11854832658964809, 0.11941123190019498,
0.11935789623183479, 0.11874246456523196, 0.11920246483150565,
0.11923748334254991, 0.11922370137454985, 0.12251514459194812,
0.12281360625946913, 0.12167734297713417, 0.12195412986187526,
0.12197089196517963, 0.12102071002825243, 0.12171135998604832,
0.12158995937119499, 0.12201920384702958, 0.1216278626429188,
0.12067692527030394, 0.12163367381079604, 0.12129690538723037,
0.12168873924378865]
best cv score to use = 0.8703076716550645
best Parameters at index 13
best depth to use = 10
best sample split size to use = 165
```

[0]:
```
from sklearn.metrics import␣
  ↪accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,␣
  ↪auc
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(max_depth=best_depth,n_estimators=best_estm,class_weight='balanced'
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[0]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)

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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
  ↪min sample size')
plt.legend()
plt.grid()
plt.show()
```



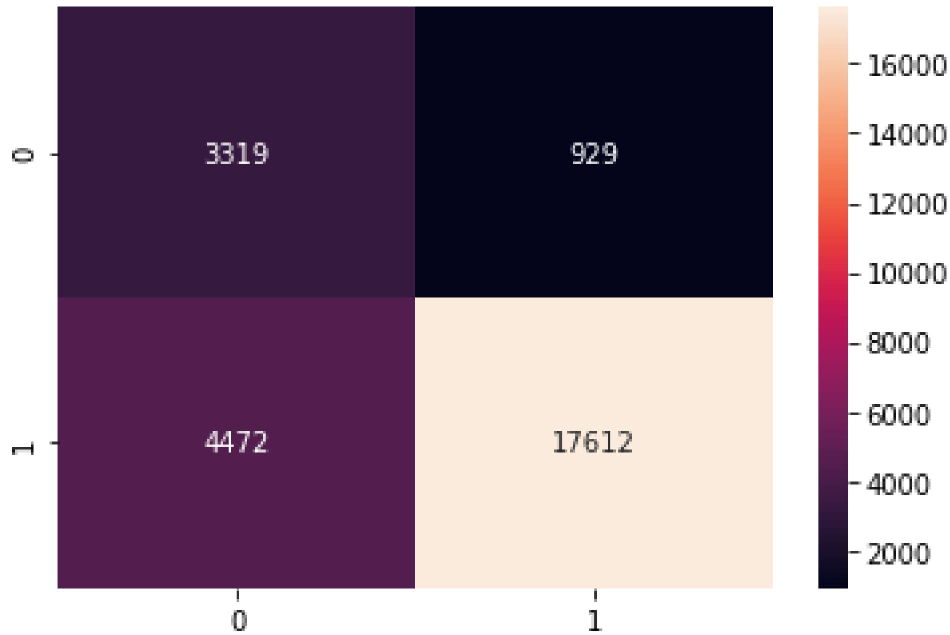
[0]:
```python
# This section of code where ever implemented is taken from sample kNN python␣
↪notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.6230941626407781 for threshold 0.572
```

[0]:
```python
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3138ead898>
```



[0]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 79.49%
Precision on test set: 94.99%
recall score on test set: 79.75%
f1 score on test set: 86.71%
```

6.2 [5.2] Applying GBDT using XGBOOST

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

[0]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =␣
↪train_test_split(preprocessed_reviews,final['Score'].values,test_size=0.
↪3,random_state=0)

vectorizer = CountVectorizer(ngram_range=(1,2), min_df=10, max_features= 500)
```

```
vectorizer.fit(X_train)
X_train = vectorizer.transform(X_train)
X_test = vectorizer.transform(X_test)


print(X_train.shape)
print(X_test.shape)
```

```
(61441, 500)
(26332, 500)
```

[0]:
```
# !wget https://s3-us-west-2.amazonaws.com/xgboost-wheels/xgboost-0.81-py2.
 ↪py3-none-manylinux1_x86_64.whl
```

[0]:
```
!pip install xgboost
!pip install --upgrade xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.6/dist-packages
(1.1.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages
(from xgboost) (1.4.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
(from xgboost) (1.18.5)
Requirement already up-to-date: xgboost in /usr/local/lib/python3.6/dist-
packages (1.1.1)
Requirement already satisfied, skipping upgrade: scipy in
/usr/local/lib/python3.6/dist-packages (from xgboost) (1.4.1)
Requirement already satisfied, skipping upgrade: numpy in
/usr/local/lib/python3.6/dist-packages (from xgboost) (1.18.5)
```

[0]:
```
depth=[i for i in np.arange(1,3)]
estimators= [i for i in np.arange(300,500,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier as xgbc
clf=xgbc(n_jobs=-1,tree_method='exact')
temp_gscv=␣
 ↪GridSearchCV(clf,param,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  7.4min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 19.3min
```

```
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 39.9min
[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed: 64.8min finished
```

[0]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_es...
                                     subsample=None, tree_method='exact',
                                     validate_parameters=None, verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 2],
                         'n_estimators': [300, 305, 310, 315, 320, 325, 330,
                                          335, 340, 345, 350, 355, 360, 365,
                                          370, 375, 380, 385, 390, 395, 400,
                                          405, 410, 415, 420, 425, 430, 435,
                                          440, 445, ...]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

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

#code snippet from provided 3d scappter plot .ipynb file


import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]
```

```
layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[0]:
```
#finding the best CV scores that is maximas then using the one which is least␣
 ↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=0
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=list(set(k))

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
```

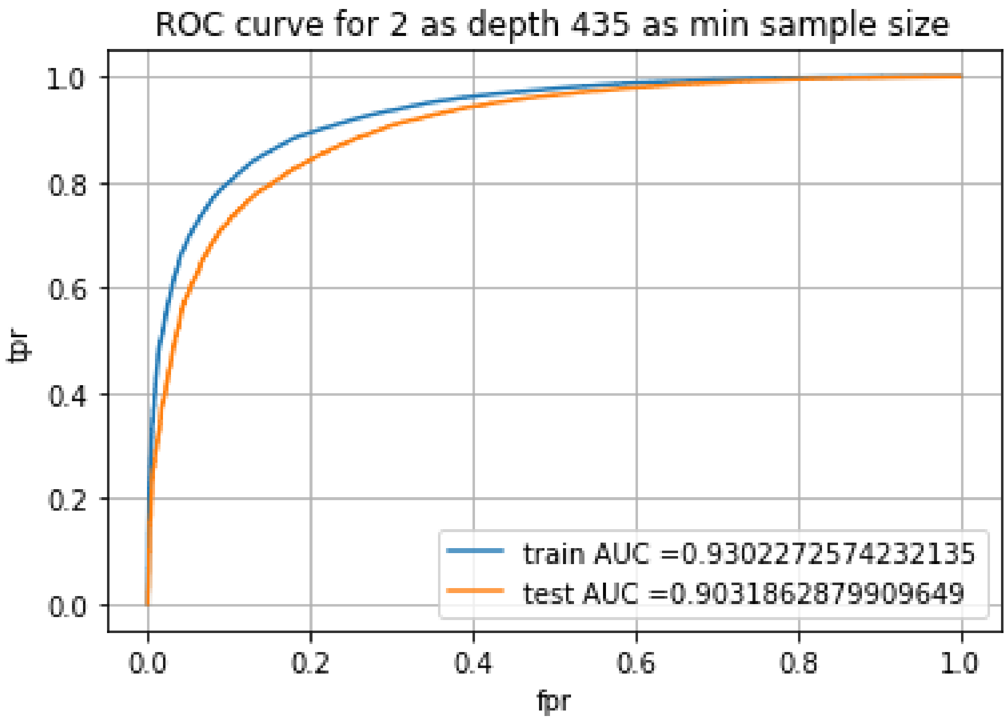
```
    break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

```
all local differences [0.026191047733764017]
best cv score to use = 0.9074864171563398
best Parameters at index 67
best depth to use = 2
best sample split size to use = 435
```

[0]:
```
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,␣
↪auc
from xgboost import XGBClassifier
clf=XGBClassifier(tree_method='exact',n_jobs=-1,max_depth=best_depth,n_estimators=best_estm)
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[0]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)


# %matplotlib inline
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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
↪min sample size')
```

```
plt.legend()
plt.grid()
plt.show()
```



[0]:
```
# This section of code where ever implemented is taken from sample kNN python␣
↪notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

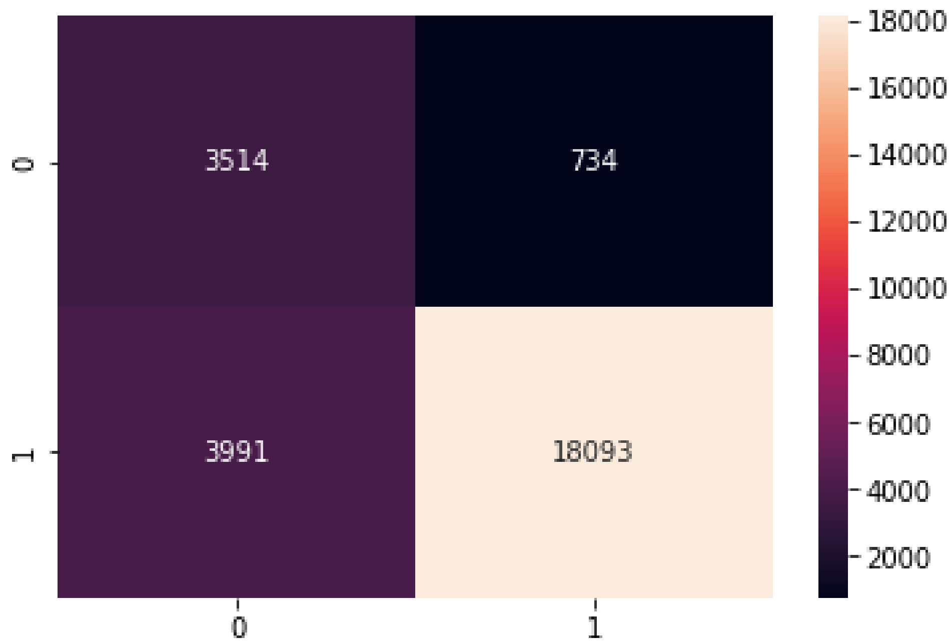
```
print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.6777196748521568 for threshold 0.823
```

[0]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f85904230b8>
```



[0]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 82.06%
Precision on test set: 96.10%
recall score on test set: 81.93%
f1 score on test set: 88.45%
```

[0]:

```
count=0
value=[]
for i in clf.feature_importances_.reshape(-1,1):
  count+=1
  value.extend(i)

x=vectorizer.get_feature_names()

features= pd.DataFrame({'feature_name':x,'value':value})
Positive=features.sort_values(by = ['value'],␣
 ↪ascending=False,ignore_index=True).head(20)
```

[0]:

```
import matplotlib.pyplot as pPlot
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image

#taken from stack overflow
d = {}
for a, x in zip(Positive['feature_name'],Positive['value']):
    d[a] = x
wordcloud = WordCloud(background_color="white",stopwords=set(STOPWORDS))

wordcloud.generate_from_frequencies(frequencies=d)
plt.figure(figsize=(10,5) )
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

6.2.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

[28]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =␣
↪train_test_split(preprocessed_reviews,final['Score'].values,test_size=0.
↪3,random_state=0)

vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features= 500)
vectorizer.fit(X_train)
X_train = vectorizer.transform(X_train)
X_test = vectorizer.transform(X_test)


print(X_train.shape)
print(X_test.shape)
```

```
(61441, 500)
(26332, 500)
```

[0]:
```
depth=[i for i in np.arange(1,5)]
estimators= [i for i in np.arange(450,500,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier as xgbc
```

```
clf=xgbc(n_jobs=-1,tree_method='exact')
temp_gscv=␣
 ↪GridSearchCV(clf,param,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 29.3min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 107.0min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed: 159.1min finished
```

[0]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_es...
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method='exact',
                                     validate_parameters=None, verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 4],
                         'n_estimators': [450, 455, 460, 465, 470, 475, 480,
                                          485, 490, 495]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

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

#code snippet from provided 3d scappter plot .ipynb file


import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc
```

```
x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[0]:
```
#finding the best CV scores that is maximas then using the one which is least␣
 ↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=3
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
```

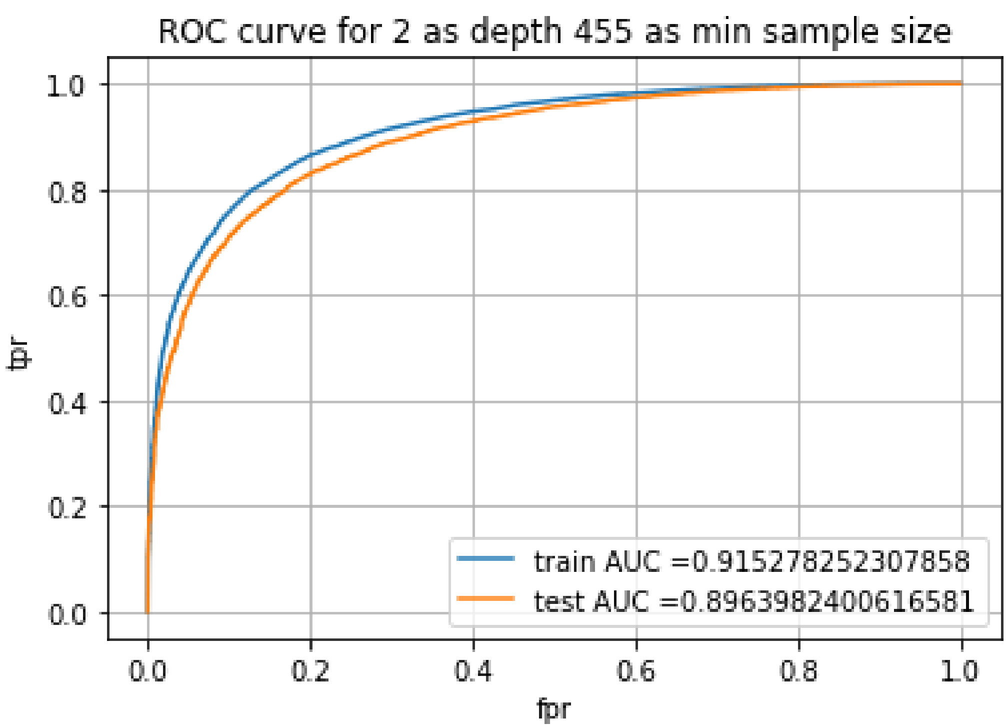
```
l=list(set(k))

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

```
all local differences [0.01820956296224241, 0.0182891052239611,
0.01833475474454771, 0.01843990580067001, 0.01856734612574562,
0.0186769558587081, 0.03464348647488025, 0.03486976991056456,
0.0350265637568794, 0.0352351311012038, 0.035446411879398676,
0.03564592824647905, 0.03585043210579042, 0.03611011197760439,
0.036291135608055414, 0.0364987250356289, 0.05529292027067667,
0.05559715363968731, 0.05583318789427594, 0.05613970757866282,
0.056456808045819984, 0.05670007095984386, 0.056964165259752564,
0.05722206084252224, 0.05752115937665092]
best cv score to use = 0.9134932228603286
best Parameters at index 14
best depth to use = 2
best sample split size to use = 455
```

[0]:
```
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,␣
↪auc
from xgboost import XGBClassifier
clf=XGBClassifier(tree_method='exact',n_jobs=-1,max_depth=best_depth,n_estimators=best_estm)
clf.fit(X_train,y_train)
```

```
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[32]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)


# %matplotlib inline
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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
 ↪min sample size')
plt.legend()
plt.grid()
plt.show()
```



[33]:
```
# This section of code where ever implemented is taken from sample kNN python␣
 ↪notebook
```

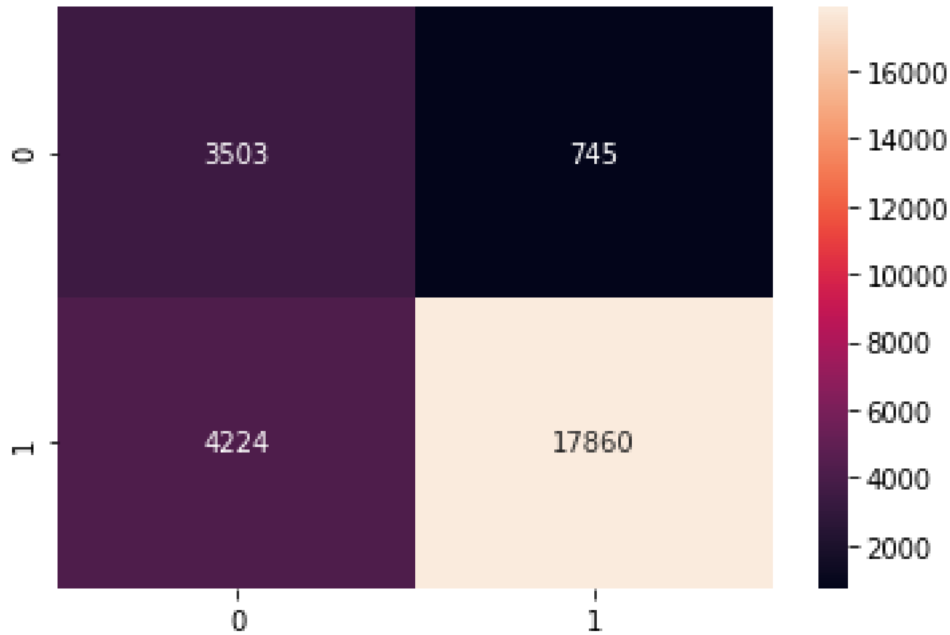
```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
 ↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.6668978930302413 for threshold 0.812
```

[34]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
 ↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2892cbfeb8>



[35]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 81.13%
Precision on test set: 96.00%
recall score on test set: 80.87%
f1 score on test set: 87.79%
```

[36]:
```
count=0
value=[]
for i in clf.feature_importances_.reshape(-1,1):
  count+=1
  value.extend(i)

x=vectorizer.get_feature_names()

features= pd.DataFrame({'feature_name':x,'value':value})
Positive=features.sort_values(by = ['value'],␣
 ↪ascending=False,ignore_index=True).head(20)
```

```
import matplotlib.pyplot as pPlot
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image

#taken from stack overflow
d = {}
for a, x in zip(Positive['feature_name'],Positive['value']):
    d[a] = x
wordcloud = WordCloud(background_color="white",stopwords=set(STOPWORDS))

wordcloud.generate_from_frequencies(frequencies=d)
plt.figure(figsize=(10,5) )
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

6.2.3 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

[0]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(sent_vectors,final['Score'].
 ↪values,test_size=0.3,random_state=0)
X_train, X_test, y_train, y_test=np.array(X_train),np.array(X_test), np.
 ↪array(y_train),np.array(y_test)
```

[0]:
```
depth=[i for i in np.arange(1,5)]
estimators= [i for i in np.arange(500,550,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier as xgbc
clf=xgbc(n_jobs=-1,tree_method='exact')
temp_gscv=␣
 ↪GridSearchCV(clf,param,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  7.0min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 42.3min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 159.0min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed: 238.3min finished
```

[0]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=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=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=None,
                                     subsample=1, tree_method='exact',
                                     verbosity=1),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 4],
                         'n_estimators': [500, 505, 510, 515, 520, 525, 530,
                                          535, 540, 545]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

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

#code snippet from provided 3d scappter plot .ipynb file


import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

```
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[0]:

```
#finding the best CV scores that is maximas then using the one which is least␣
 ↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=1
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
```

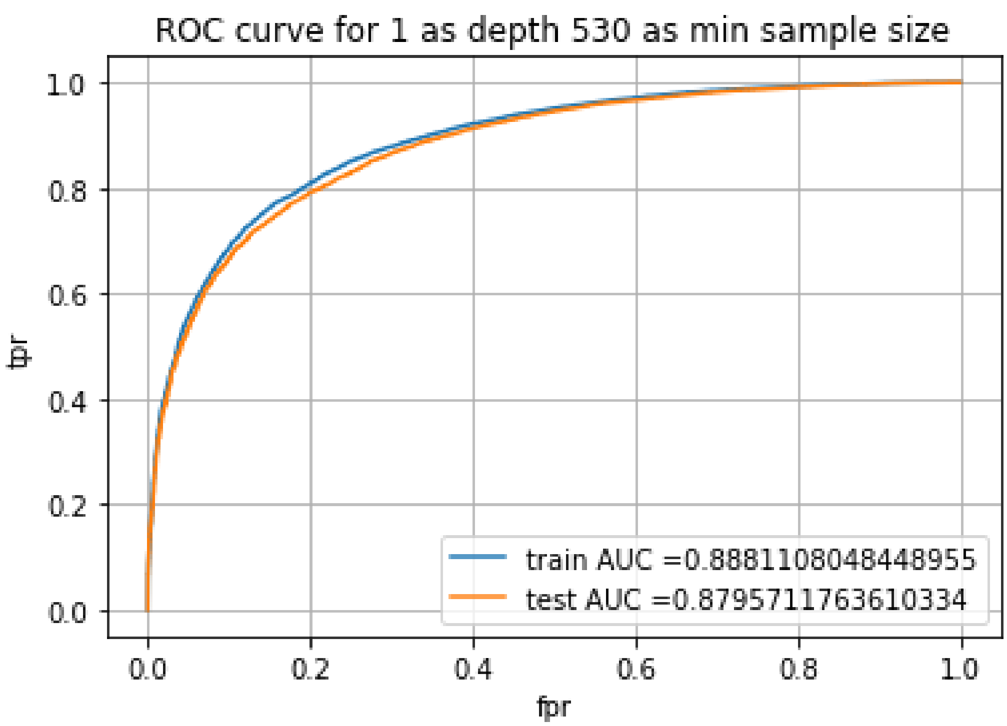
```
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=list(set(k))

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

```
all local differences [0.05583318789427594, 0.05613970757866282,
0.056456808045819984, 0.05670007095984386, 0.056964165259752564,
0.01833475474454771, 0.01843990580067001, 0.01856734612574562,
0.0186769558587081, 0.03464348647488025, 0.03486976991056456,
0.0350265637568794, 0.0352351311012038, 0.03564592824647905,
0.03585043210579042, 0.03611011197760439, 0.036291135608055414,
0.0364987250356289, 0.05529292027067667, 0.05559715363968731]
best cv score to use = 0.9136165854366058
best Parameters at index 16
best depth to use = 1
best sample split size to use = 530
```

[0]:
```
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,␣
↪auc
from xgboost import XGBClassifier
clf=XGBClassifier(tree_method='exact',n_jobs=-1,max_depth=best_depth,n_estimators=best_estm)
```

```
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[0]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)


# %matplotlib inline
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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
 ↪min sample size')
plt.legend()
plt.grid()
plt.show()
```



[0]:
```
# This section of code where ever implemented is taken from sample kNN python␣
 ↪notebook
```

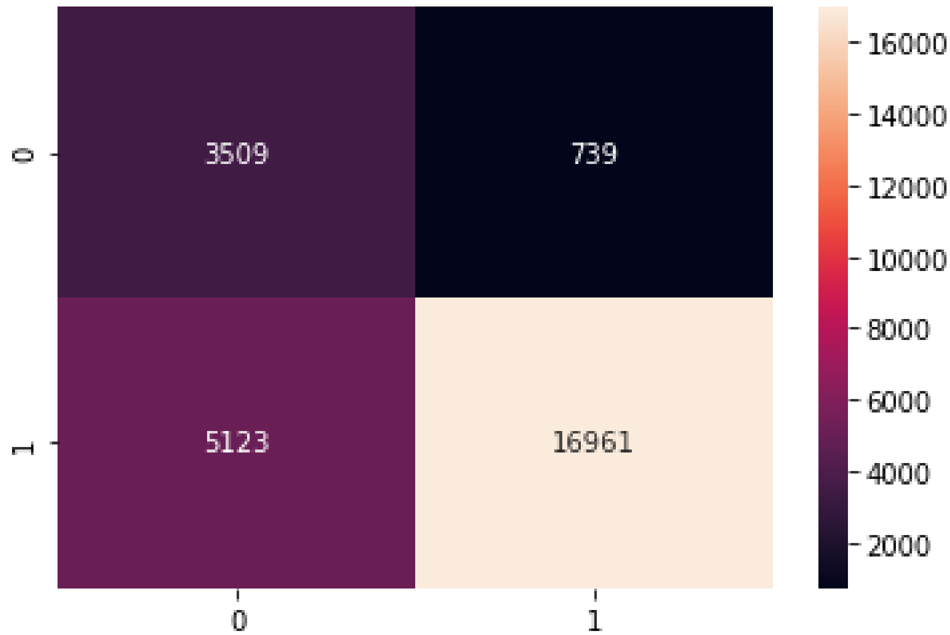
```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
 ↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.6344137335071602 for threshold 0.821
```

[0]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
 ↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdf47bd6ef0>
```



[0]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 77.74%
Precision on test set: 95.82%
recall score on test set: 76.80%
f1 score on test set: 85.27%
```

6.2.4 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

[0]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =␣
↪train_test_split(tfidf_sent_vectors,final['Score'].values,test_size=0.
↪3,random_state=0)
X_train, X_test, y_train, y_test=np.array(X_train),np.array(X_test), np.
↪array(y_train),np.array(y_test)
```

[46]:
```
depth=[i for i in np.arange(1,5)]
estimators= [i for i in np.arange(490,540,5)]
param = {'max_depth':depth,'n_estimators':estimators}

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier as xgbc
clf=xgbc(n_jobs=-1,tree_method='hist')
temp_gscv=␣
↪GridSearchCV(clf,param,verbose=5,n_jobs=-1,scoring='roc_auc',return_train_score=True)
temp_gscv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 19.5min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed: 28.1min finished
```

[46]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=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=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=None,
                                     subsample=1, tree_method='hist',
                                     verbosity=1),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 4],
                         'n_estimators': [490, 495, 500, 505, 510, 515, 520,
                                          525, 530, 535]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

[47]:
```
train_auc=temp_gscv.cv_results_['mean_train_score']
cv_auc=temp_gscv.cv_results_['mean_test_score']

#code snippet from provided 3d scappter plot .ipynb file


import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

```
x1 = estimators*len(depth)
y1 = depth*len(estimators)
z1 = train_auc

x2 = estimators*len(depth)
y2 = depth*len(estimators)
z2 = cv_auc

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Sample_size'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
fig.show(renderer='colab')
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[48]:
```
#finding the best CV scores that is maximas then using the one which is least␣
 ↪distant then its AUC counter part to derive
# C and gamma to avoid using Dumb model.

from scipy.signal import argrelextrema
import numpy as np
x = np.array(train_auc)
y = np.array(cv_auc)

local_max=()
diff=x-y

#finding index of maximas of CV scores
local_max_i=argrelextrema(y, np.greater)

#generating a list of indexs for maximas
l=list(i for i in local_max_i[0])

#generating list of indices in neighbor of maximas to check
k=[]
neighbor=2
for i in l:
  if i >neighbor and i<len(y):
    k.extend(range(i-neighbor,i+neighbor+1))
  elif i<neighbor and i < len(y):
```

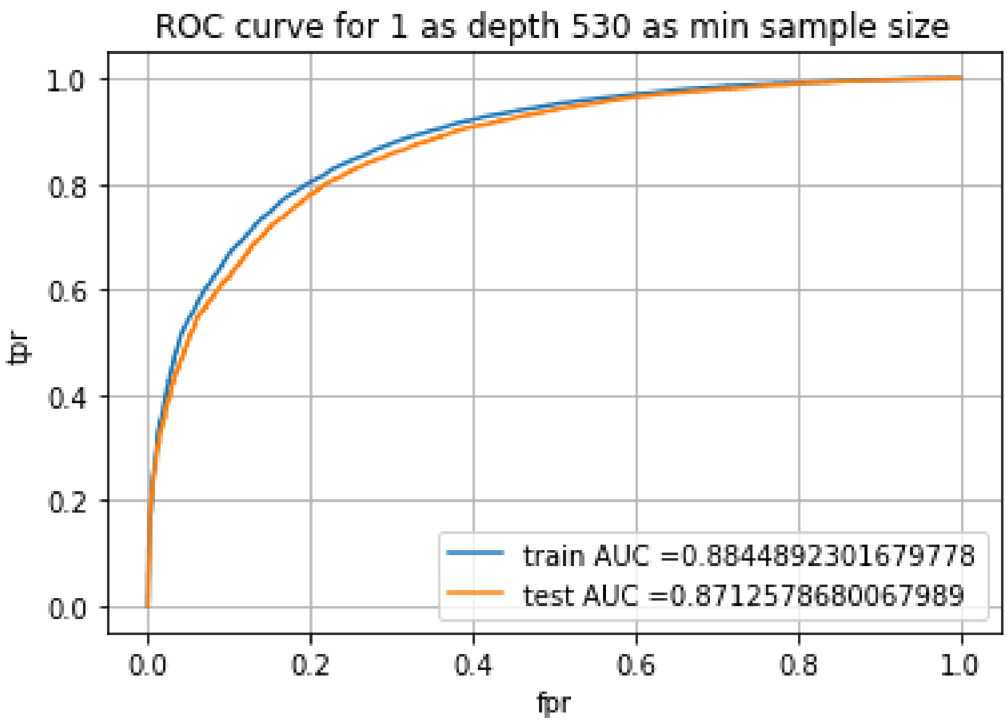
```
    k.extend(range(i,i+neighbor+1))
  else:
    k.extend(range(i-neighbor,i+1))
l=list(set(k))

# diff between CV and Test AUC at the local maximas
local_diff=list(diff[i] for i in l)
print(f'all local differences {local_diff}')

#fetching the index where local diff is min
for i in np.nditer(np.argmin(local_diff)):
  v=i
  break
print(f'best cv score to use = {y[l[v]]}')


best_index= l[v]

print('best Parameters at index {}'.format(best_index))

# as index are in range of 0 to hundread
# for differnt permutation of C and gamma
# fetching the C and Gamma index from them

best_depth=depth[best_index%len(depth)]
print(f'best depth to use = {best_depth}')

best_estm=estimators[best_index%len(estimators)]
print('best sample split size to use = {}'.format(best_estm))
```

```
all local differences [0.06501542808148031, 0.06536666192994744,
0.06574348072139469, 0.06604553450321604, 0.06634845125668587,
0.06673091563544487, 0.06705878286582467, 0.04017414966866262,
0.040350937580183155, 0.06430808983022396, 0.06468100457436976]
best cv score to use = 0.8965455183274557
best Parameters at index 28
best depth to use = 1
best sample split size to use = 530
```

[0]:
```
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,f1_score,precision_score,recall_score,roc_curve,␣
↪auc
from xgboost import XGBClassifier
clf=XGBClassifier(tree_method='exact',n_jobs=-1,max_depth=best_depth,n_estimators=best_estm)
clf.fit(X_train,y_train)
y_pred_tr = clf.predict_proba(X_train)
y_pred_ts = clf.predict_proba(X_test)
```

```
y_pred_tr=y_pred_tr[:,1]
y_pred_ts=y_pred_ts[:,1]
```

[50]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_ts)


# %matplotlib inline
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.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (best_depth)+' as depth '+str(best_estm)+ ' as␣
 ↪min sample size')
plt.legend()
plt.grid()
plt.show()
```



[51]:
```
# This section of code where ever implemented is taken from sample kNN python␣
 ↪notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
```

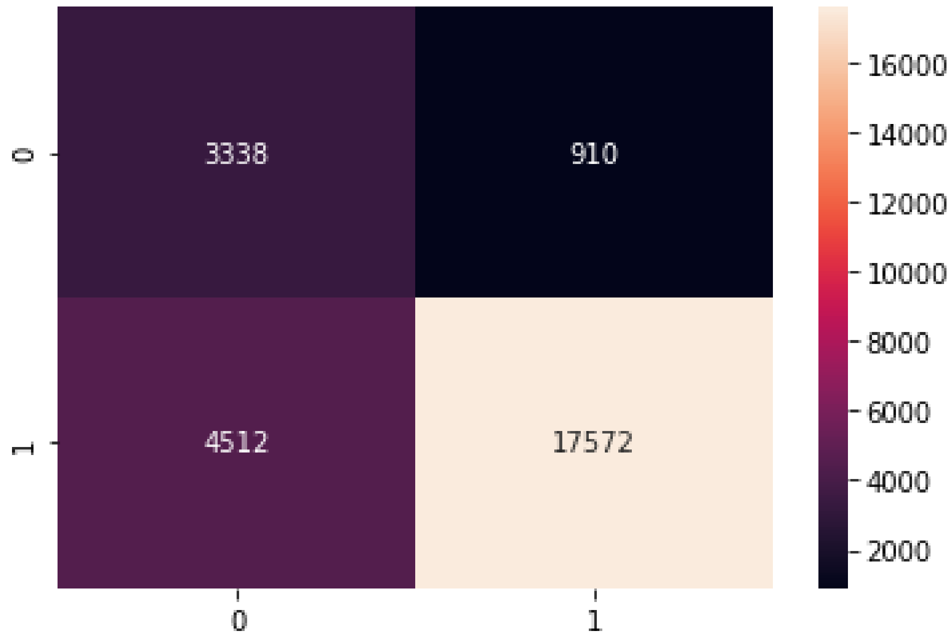
```
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
 ↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.625237877905658 for threshold 0.818
```

[52]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_pred_ts,␣
 ↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f288a6ee390>
```



[53]:
```
acc=accuracy_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
ps=precision_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
rc=recall_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100
f1=f1_score(y_test, predict_with_best_t(y_pred_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 79.41%
Precision on test set: 95.08%
recall score on test set: 79.57%
f1 score on test set: 86.63%
```

7 [6] Conclusions

[56]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["S.NO.", "Vectorization",  "Max Depth", "Max Sample Split␣
↪size","Test AUC","Precision Score"]
x.add_row(["1", "BOW", "12", "115","0.8697","95.42%"])
x.add_row(["2", "TFIDF","10", "135","0.8665","95.79%"])
x.add_row(["3", "AVG W2V", "10", "135","0.8935","96.59%"])
x.add_row(["4", "TFIDF W2V", "10", "165","0.8681","94.99%"])
print('Random forest results')
```

```
print(x)
```

```
Random forest results
+-------+---------------+-----------+-----------------------+----------+--------
---------+
| S.NO. | Vectorization | Max Depth | Max Sample Split size | Test AUC |
Precision Score |
+-------+---------------+-----------+-----------------------+----------+--------
---------+
|   1   |      BOW      |     12    |          115          |  0.8697  |
95.42%     |
|   2   |     TFIDF     |     10    |          135          |  0.8665  |
95.79%     |
|   3   |    AVG W2V    |     10    |          135          |  0.8935  |
96.59%     |
|   4   |   TFIDF W2V   |     10    |          165          |  0.8681  |
94.99%     |
+-------+---------------+-----------+-----------------------+----------+--------
---------+
```

[55]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["S.NO.", "Vectorization",  "Max Depth", "Max Sample Split␣
↪size","Test AUC","Precision Score"]
x.add_row(["1", "BOW", "2", "435","0.9031","96.10%"])
x.add_row(["2", "TFIDF","2", "455","0.8963","96.00%"])
x.add_row(["3", "AVG W2V", "1", "530","0.8795","95.82%"])
x.add_row(["4", "TFIDF W2V", "1", "530","0.8712","95.08%"])
print('XGBoost results')
print(x)
```

```
XGBoost results
+-------+---------------+-----------+-----------------------+----------+--------
---------+
| S.NO. | Vectorization | Max Depth | Max Sample Split size | Test AUC |
Precision Score |
+-------+---------------+-----------+-----------------------+----------+--------
---------+
|   1   |      BOW      |     2     |          435          |  0.9031  |
96.10%     |
|   2   |     TFIDF     |     2     |          455          |  0.8963  |
96.00%     |
|   3   |    AVG W2V    |     1     |          530          |  0.8795  |
95.82%     |
|   4   |   TFIDF W2V   |     1     |          530          |  0.8712  |
95.08%     |
+-------+---------------+-----------+-----------------------+----------+--------
---------+
```

```
[0]:
```