09_Amazon_Fine_Food_Reviews_Analysis_RF

March 4, 2021

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".

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

[ ]:

```
!wget --header 'Host: storage.googleapis.com' --user-agent 'Mozilla/5.0␣
↪(Windows NT 10.0; Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0' --header␣
↪'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
↪*;q=0.8' --header 'Accept-Language: en-US,en;q=0.5' --referer 'https://www.
↪kaggle.com/' --header 'Upgrade-Insecure-Requests: 1' 'https://storage.
↪googleapis.com/kaggle-data-sets/18/2157/bundle/archive.zip?
↪X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.
↪iam.gserviceaccount.
↪com%2F20210302%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210302T131925Z&X-Goog-Expires=
↪--output-document 'archive.zip'
```

```
--2021-03-04 04:11:50--  https://storage.googleapis.com/kaggle-data-
sets/18/2157/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-
Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210302%2Fa
uto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210302T131925Z&X-Goog-
Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=2dc38507fd18b59cfdae33
6980e38d0bf13c3681a102bff0bf78934dc50a3b1b036649976eefce9eb8bdae41877a0e84cf4834
b85424f0c466370b127a2b217cd537269e7f15bbc6b696c04a0915f7a6d856effcc9a0f23a726f1d
8c0a42e3f7643503777cda325bf54b42ee5a29cd94c3dd88a51fea894271bbfbf4ff3bd32832bb40
387fb0a7260a9d2084fc8962bd47ded7bd2d69ec21b77bc5834fc7ad5d86e1c3f65c41f95cb18959
c100b95c2295eacc4b91b2739c35596cff4e6d387235570de0b5a421392df24842a0e61f6bab58db
df82448fea57fd54592c237030df3eaff9808e2eabb2cfa3c22448617d71c769425283ed3c5e9e5b
e3509f5e0f
Resolving storage.googleapis.com (storage.googleapis.com)… 64.233.189.128,
108.177.97.128, 74.125.204.128, …
Connecting to storage.googleapis.com
(storage.googleapis.com)|64.233.189.128|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 253873708 (242M) [application/zip]
Saving to: ‘archive.zip’

archive.zip         100%[===================>] 242.11M  30.0MB/s    in 8.1s

2021-03-04 04:11:59 (30.0 MB/s) - ‘archive.zip’ saved [253873708/253873708]
```

[ ]:
```
# https://colab.research.google.com/drive/
↪1xinRwhXtlL-9YOKbPrTmTxNdcN-Hvq4m#scrollTo=O1_kc7HBeslm
# to extact zip or rar files
!unzip "archive.zip" -d "archive"
```

```
Archive:  archive.zip
  inflating: archive/Reviews.csv
  inflating: archive/database.sqlite
  inflating: archive/hashes.txt
```

[ ]:
```
# using SQLite Table to read data.
con = sqlite3.connect('archive/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""", 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 (525814, 10)
```

[ ]:

```
   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]
```

[ ]:

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

[ ]:

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

```
(80668, 7)
```

```
[ ]:                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]
```

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

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

     [1 rows x 7 columns]
```

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

```
[ ]: 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:

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

```
[ ]:        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.

[ ]:
```
#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')
```

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

[ ]:
```
(364173, 10)
```

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

[ ]:
```
69.25890143662969
```

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

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

display.head()
```

[ ]:
```
      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]
```

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

[ ]:
```
#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()
```

```
(364171, 10)
```

[ ]:
```
1    307061
0     57110
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

[ ]:
```
# 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]
```

```python
print(sent_4900)
print("="*50)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about
whales, India, drooping roses:  i love all the new words this book  introduces
and the silliness of it all.  this is a classic book i am  willing to bet my son
will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is
good, but I prefer bolder taste… imagine my surprise when I ordered 2 boxes -
both were expired! One expired back in 2005 for gosh sakes.  I admit that Amazon
agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!!
I'm hoping to find local San Diego area shoppe that carries pods so that I can
try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today's Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70's it was
poisonous until they figured out a way to fix that. I still like it but it could
be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the
excellence of this product.<br /><br />Thick, delicious.  Perfect.  3
ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No
garbage.<br /><br />Have numerous friends & family members hooked on this stuff.
My husband & son, who do NOT like "sugar free" prefer this over major label
regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes,
white brownies, muffins, pumpkin pies, etc… Unbelievably delicious…<br /><br
/>Can you tell I like it? :)
==================================================
```

[ ]:

```python
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about
whales, India, drooping roses:  i love all the new words this book  introduces
and the silliness of it all.  this is a classic book i am  willing to bet my son
will STILL be able to recite from memory when he is  in college
```

[ ]:
```python
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about
whales, India, drooping roses:  i love all the new words this book  introduces
and the silliness of it all.  this is a classic book i am  willing to bet my son
will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is
good, but I prefer bolder taste… imagine my surprise when I ordered 2 boxes -
both were expired! One expired back in 2005 for gosh sakes.  I admit that Amazon
agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!!
I'm hoping to find local San Diego area shoppe that carries pods so that I can
try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today's Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70's it was
poisonous until they figured out a way to fix that. I still like it but it could
be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the
excellence of this product.Thick, delicious.  Perfect.  3 ingredients: Water,
Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.Have
```

```
numerous friends & family members hooked on this stuff.  My husband & son, who
do NOT like "sugar free" prefer this over major label regular syrup.I use this
as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies,
etc… Unbelievably delicious…Can you tell I like it? :)
```

[ ]:
```python
# 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
```

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

```
Great ingredients although, chicken should have been 1st rather than chicken
broth, the only thing I do not think belongs in it is Canola oil. Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it, it would poison them. Today is Food industries
have convinced the masses that Canola oil is a safe and even better oil than
olive or virgin coconut, facts though say otherwise. Until the late 70 is it was
poisonous until they figured out a way to fix that. I still like it but it could
be better.
==================================================
```

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

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about
whales, India, drooping roses:  i love all the new words this book  introduces
and the silliness of it all.  this is a classic book i am  willing to bet my son
will STILL be able to recite from memory when he is  in college
```

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

```
Great ingredients although chicken should have been 1st rather than chicken
broth the only thing I do not think belongs in it is Canola oil Canola or
rapeseed is not someting a dog would ever find in nature and if it did find
rapeseed in nature and eat it it would poison them Today is Food industries have
convinced the masses that Canola oil is a safe and even better oil than olive or
virgin coconut facts though say otherwise Until the late 70 is it was poisonous
until they figured out a way to fix that I still like it but it could be better
```

[ ]:
```
# 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"])
```

[ ]:
```
# 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%|     | 364171/364171 [02:10<00:00, 2799.73it/s]
```

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

[ ]:
```
'great ingredients although chicken rather chicken broth thing not think belongs
canola oil canola rapeseed not someting dog would ever find nature find rapeseed
nature eat would poison today food industries convinced masses canola oil safe
even better oil olive virgin coconut facts though say otherwise late poisonous
figured way fix still like could better'
```

[3.2] Preprocessing Review Summary

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

5 [4] Featurization

[ ]:
```
# sampling the data of 50k from population for train test split, to avoid data␣
↪leakage issue
from sklearn.model_selection import train_test_split

n_samples= 50000
x_sampled_data= preprocessed_reviews[:n_samples]
y_sampled_data= final.Score[:n_samples]

X_train, x_test, Y_train, y_test= train_test_split(x_sampled_data,␣
↪y_sampled_data, test_size= .33)
```

5.1 [4.1] BAG OF WORDS

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

final_counts_train_bow = count_vect.transform(X_train)
print("the type of count vectorizer ",type(final_counts_train_bow))
print("the shape of out text BOW vectorizer ",final_counts_train_bow.
↪get_shape())
print("the number of unique words ", final_counts_train_bow.get_shape()[1])
print('='*50)

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

```
some feature names  ['aa', 'aaa', 'aaaaa', 'aaaaaa', 'aaaaaah',
'aaaaaahhhhhyaaaaaa', 'aaaallll', 'aaah', 'aachen', 'aafco']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (33500, 36706)
the number of unique words  36706
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (16500, 36706)
the number of unique words  36706
```

5.2 [4.2] Bi-Grams and n-Grams.

[ ]:
```
#bi-gram, tri-gram and n-gram

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

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

5.3 [4.3] TF-IDF

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

final_tf_idf_train = tf_idf_vect.transform(X_train)
print("the type of count vectorizer ",type(final_tf_idf_train))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_train.get_shape())
print("the number of unique words including both unigrams and bigrams ",␣
 ↪final_tf_idf_train.get_shape()[1])
print('='*50)

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy',
'able drink', 'able eat', 'able enjoy', 'able find', 'able get', 'able go',
'able keep']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (33500, 19537)
the number of unique words including both unigrams and bigrams  19537
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (16500, 19537)
the number of unique words including both unigrams and bigrams  19537
```

5.4 [4.4] Word2Vec

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

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

[ ]:
```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('wonderful', 0.8036404848098755), ('excellent', 0.7892823815345764),
('fantastic', 0.7885227799415588), ('good', 0.7805554866790771), ('perfect',
0.7570130825042725), ('awesome', 0.747679591178894), ('amazing',
0.718521237373352), ('terrific', 0.6882271766662598), ('fabulous',
0.6706435680389404), ('decent', 0.6495816707611084)]
==================================================
[('disgusting', 0.7586555480957031), ('nastiest', 0.7479588985443115), ('best',
0.7376443147659302), ('greatest', 0.7307570576667786), ('ashtray',
0.7128258943557739), ('tastiest', 0.696068286895752), ('closest',
0.6829813122749329), ('nicest', 0.6752094626426697), ('awful',
0.6700434684753418), ('horrible', 0.6635743379592896)]
```

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

```
number of words that occured minimum 5 times  11770
sample words  ['love', 'yogi', 'tea', 'paper', 'tab', 'attached', 'string',
'bag', 'wise', 'sayings', 'since', 'teach', 'clients', 'providing', 'learning',
'subtle', 'easy', 'drink', 'helps', 'shed', 'toxins', 'dog', 'absolutely',
'loved', 'bone', 'issues', 'getting', 'stuffing', 'middle', 'believe',
'enjoyed', 'challenge', 'hesitant', 'ordering', 'something', 'never', 'tasted',
'best', 'peach', 'definitely', 'buying', 'another', 'box', 'bars', 'good',
'texture', 'amazing', 'health', 'bar', 'really']
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

[ ]:
```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2v(list_of_sentance):
  sent_vectors = []; # the avg-w2v for each sentence/review is stored in this␣
 ↪list
  for sent in tqdm(list_of_sentance): # for each review/sentence
      sent_vec = np.zeros(50) # as word vectors are of zero length 50, you␣
 ↪might need to change this to 300 if you use google's w2v
      cnt_words =0; # num of words with a valid vector in the sentence/review
      for word in sent: # for each word in a review/sentence
          if word in w2v_words:
              vec = w2v_model.wv[word]
              sent_vec += vec
              cnt_words += 1
      if cnt_words != 0:
          sent_vec /= cnt_words
      sent_vectors.append(sent_vec)
  print(len(sent_vectors))
  print(len(sent_vectors[0]))
  return sent_vectors
```

[ ]:
```
avg_w2v_train = avg_w2v(list_of_sentance_train)
```

```
100%|          | 33500/33500 [01:08<00:00, 488.28it/s]
```

```
33500
50
```

[ ]:
```
avg_w2v_test = avg_w2v(list_of_sentance_test)
```

```
100%|          | 16500/16500 [00:36<00:00, 457.93it/s]
```

```
16500
50
```

[4.4.1.2] **TFIDF weighted W2v**

```
[ ]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

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

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

```
100%|      | 33500/33500 [13:23<00:00, 41.71it/s]
```

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

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is␣
↪stored in this list
row=0;
```

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

```
100%|      | 16500/16500 [06:38<00:00, 41.39it/s]
```

6 [5] Assignment 9: Random Forests

Apply Random Forests & GBDT on these feature sets

SET 1:Review text, preprocessed one converted into vectors using (BOW)

SET 2:Review text, preprocessed one converted into vectors using (TFIDF)

SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)

SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

```
</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 engine
        <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 <s
        <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.hea
<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 fo
    <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

[ ]:
```python
# importing libraries
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from wordcloud import WordCloud
```

```python
import seaborn as sns
import math
import warnings
warnings.filterwarnings("ignore")
```

[ ]:
```python
# Defining a function for hyper parameter tuning based on Randomized Search CV
def hyper_param_tuning(X_train):
  depth = [1, 10, 50, 100, 500, 1000]
  n_estimators = [100, 200, 300, 400, 500]
  tuned_parameters = [{'max_depth':depth, 'n_estimators': n_estimators}]

  # Applying RandomizedSearchCV with k folds = 5 and taking 'roc_auc' as score␣
 ↪metric
  clf = RandomizedSearchCV(RandomForestClassifier(), param_distributions=␣
 ↪tuned_parameters, scoring = 'roc_auc', cv=5, return_train_score= True,␣
 ↪verbose = 10, n_jobs= -1)
  clf.fit(X_train, Y_train)

  # Plotting the hyper parameters CV results wrt. AUC i.e. mean_score where␣
 ↪scoring is done using 'roc_auc'
  max_depth_list = list(clf.cv_results_['param_max_depth'].data)
  n_estimators_list = list(clf.cv_results_['param_n_estimators'].data)
  sns.set_style("whitegrid")
  plt.figure(figsize=(16,6))
  plt.subplot(1,2,1)
  data = pd.DataFrame(data={'Number of Estimators':n_estimators_list, 'Max␣
 ↪Depth':max_depth_list, 'AUC':clf.cv_results_['mean_train_score']})
  data = data.pivot(index='Number of Estimators', columns='Max Depth',␣
 ↪values='AUC')
  sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Training␣
 ↪data')
  plt.subplot(1,2,2)
  data = pd.DataFrame(data={'Number of Estimators':n_estimators_list, 'Max␣
 ↪Depth':max_depth_list, 'AUC':clf.cv_results_['mean_test_score']})
  data = data.pivot(index='Number of Estimators', columns='Max Depth',␣
 ↪values='AUC')
  sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for CV data')
  plt.show()

  print('Best hyper parameter: ', clf.best_params_)
  print('Model Score: ', clf.best_score_)
  print('Model estimator: ', clf.best_estimator_)
```

[ ]:
```python
# Defining a function so to plot area under roc for both the train and test data
def plot_auc(X_train, x_test, max_depth, n_estimators):
```

```
    model = RandomForestClassifier(max_depth= max_depth, n_estimators=␣
↪n_estimators, class_weight='balanced', n_jobs= -1)
    model.fit(X_train, Y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
    # not the predicted outputs

    train_fpr, train_tpr, thresholds = roc_curve(Y_train, model.
↪predict_proba(X_train)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, model.
↪predict_proba(x_test)[:,1])

    plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr,␣
↪train_tpr)))
    plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("False Positive Rate(FPR)")
    plt.ylabel("True Positive Rate(TPR)")
    plt.title("Area under ROC Plots")
    plt.show()
    return model
```

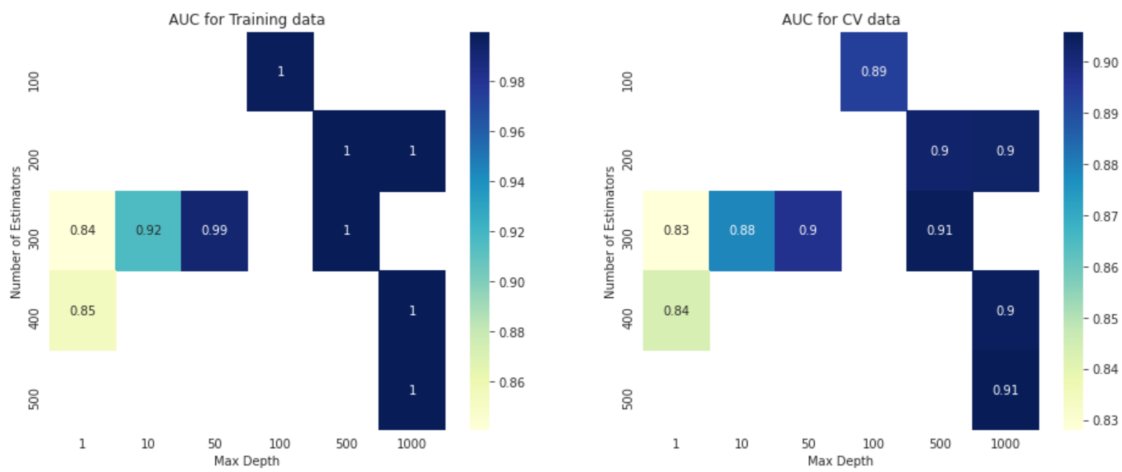
[ ]:
```
def confusion_matrix_plot(dataType, x_data, y_data, model):
    print("{} confusion matrix".format(dataType))
    # Ref: https://datatofish.com/confusion-matrix-python/
    df_= pd.DataFrame(confusion_matrix(y_data, model.predict(x_data)))
    ax = sns.heatmap(df_, annot=True,annot_kws={"size": 16}, fmt='g')
    ax.set(ylabel="Actual Label", xlabel="Predicted Label")
```

[ ]:
```
def word_cloud_image(model, vectorizer):
    w = vectorizer.get_feature_names()
    coef = model.feature_importances_
    coeff_df = pd.DataFrame({'Word' : w, 'Coefficient' : coef})
    coeff_df = coeff_df.sort_values(['Coefficient', 'Word'], ascending=[0, 1])

    # Ref. Link: https://www.datacamp.com/community/tutorials/wordcloud-python
    text = coeff_df['Word'][:20]

    # Create and generate a word cloud image:
    wordcloud = WordCloud().generate(text.to_string())

    # Display the generated image:
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

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

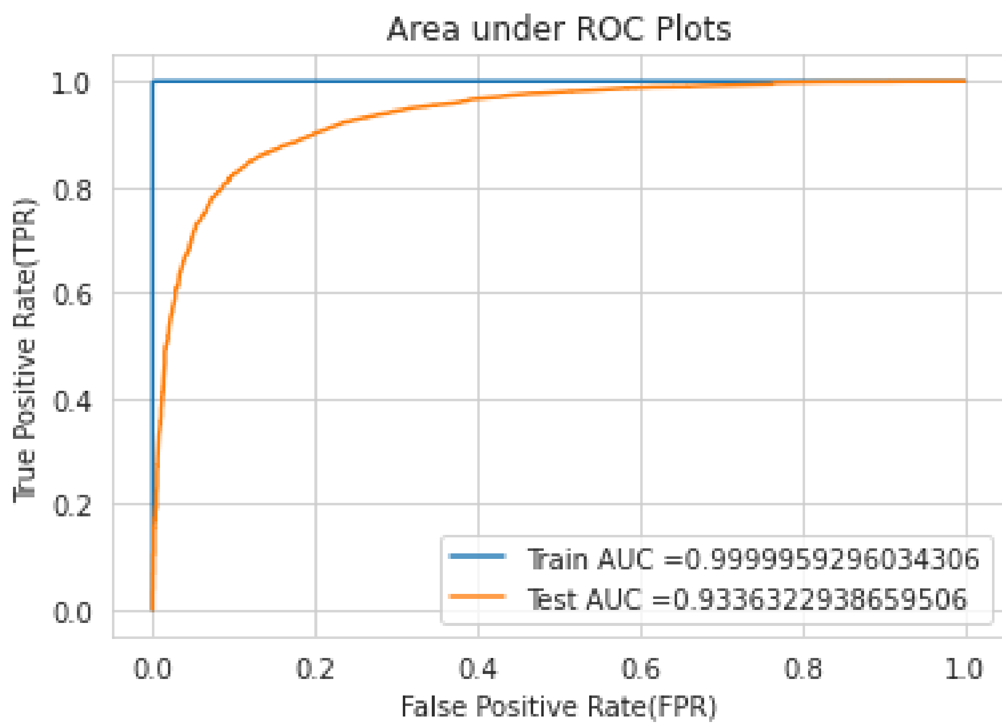
[ ]:
```
hyper_param_tuning(final_counts_train_bow)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   25.6s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   49.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 26.4min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 49.1min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 69.5min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 89.3min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 96.6min finished
```



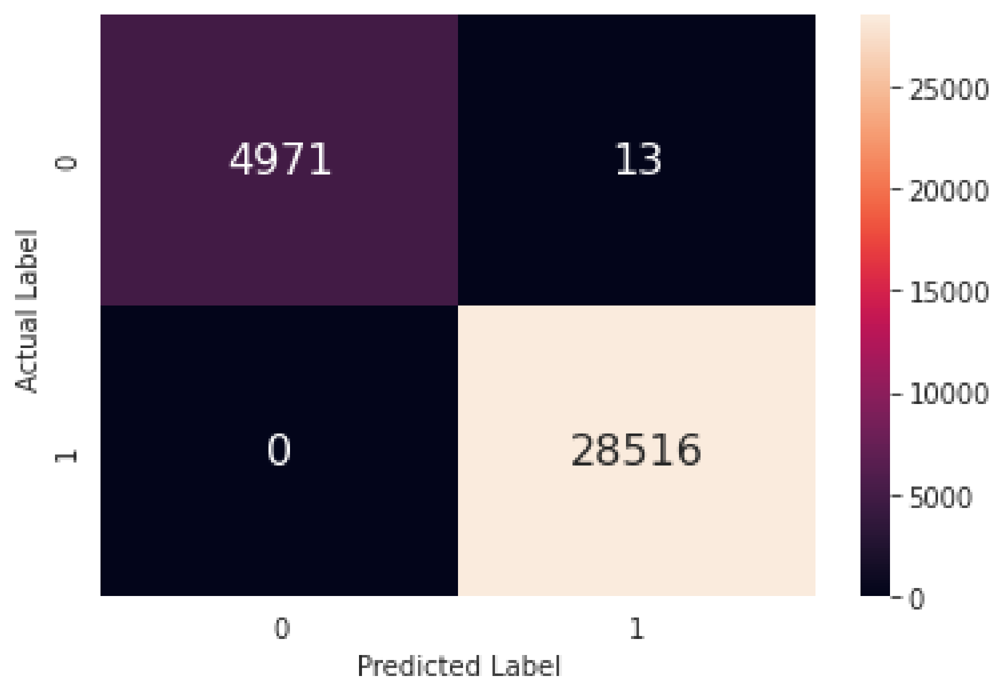

```
Best hyper parameter:  {'n_estimators': 500, 'max_depth': 1000}
Model Score:  0.9058819205797249
Model estimator:  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=1000, max_features='auto',
                       max_leaf_nodes=None, 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=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[ ]:
```
clf = plot_auc(final_counts_train_bow, final_counts_test_bow, max_depth= 1000,␣
↪n_estimators= 500)
```



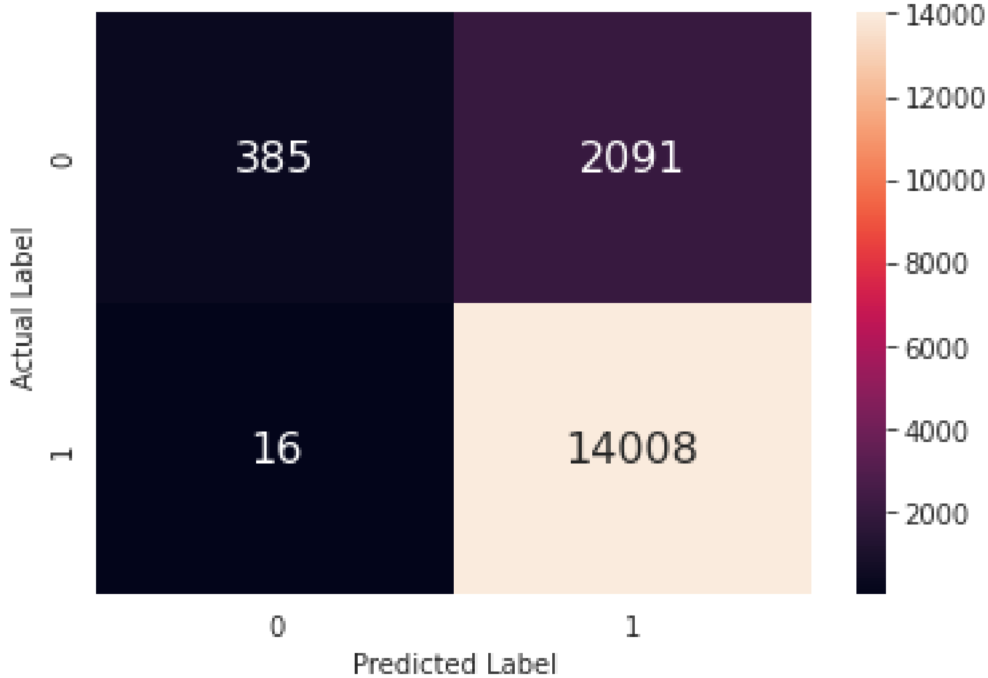

```
[ ]: confusion_matrix_plot(dataType= 'Train', x_data= final_counts_train_bow,␣
      ↪y_data= Y_train, model = clf)
```

```
Train confusion matrix
```



```
[ ]: confusion_matrix_plot(dataType= 'Test', x_data= final_counts_test_bow, y_data=␣
     ↪y_test, model= clf)
```

Test confusion matrix



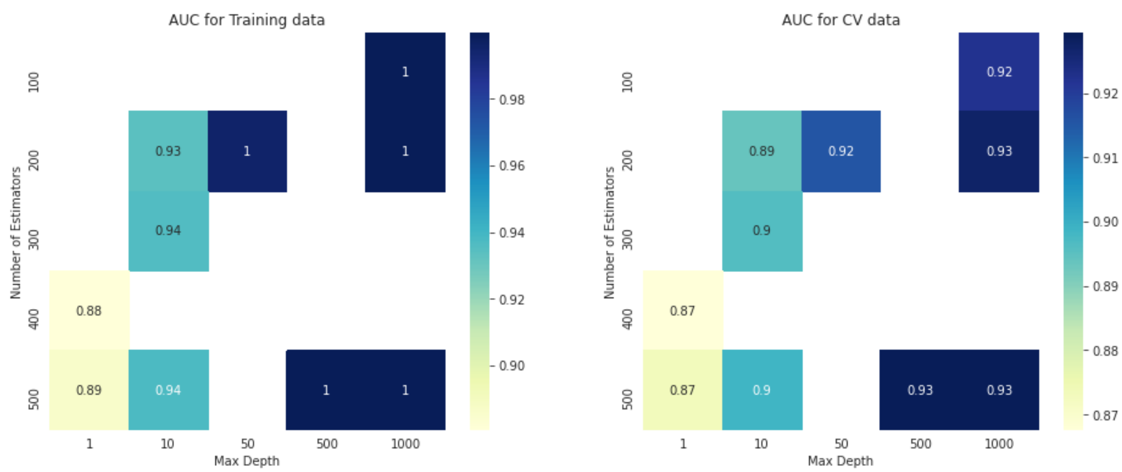
6.1.2 [5.1.2] Wordcloud of top 20 important features from SET 1

```
[ ]: word_cloud_image(model = clf, vectorizer= count_vect)
```

6.1.3 [5.1.3] Applying Random Forests on TFIDF, SET 2

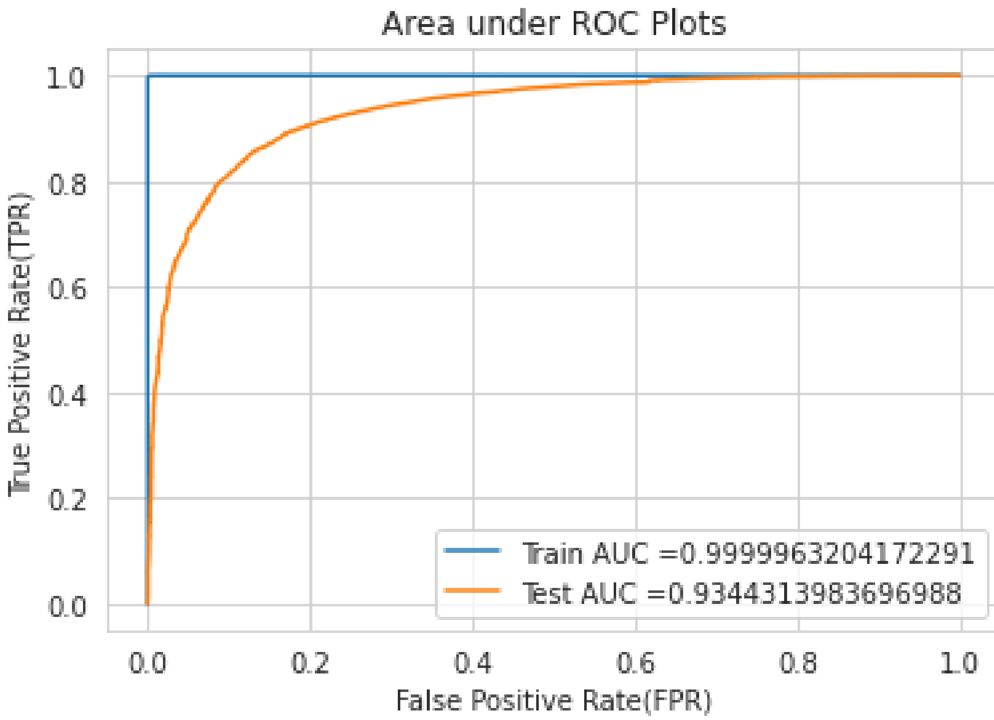
[ ]:
```
hyper_param_tuning(final_tf_idf_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    7.7s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   13.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   36.0s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 17.5min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 24.0min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 32.3min finished
```



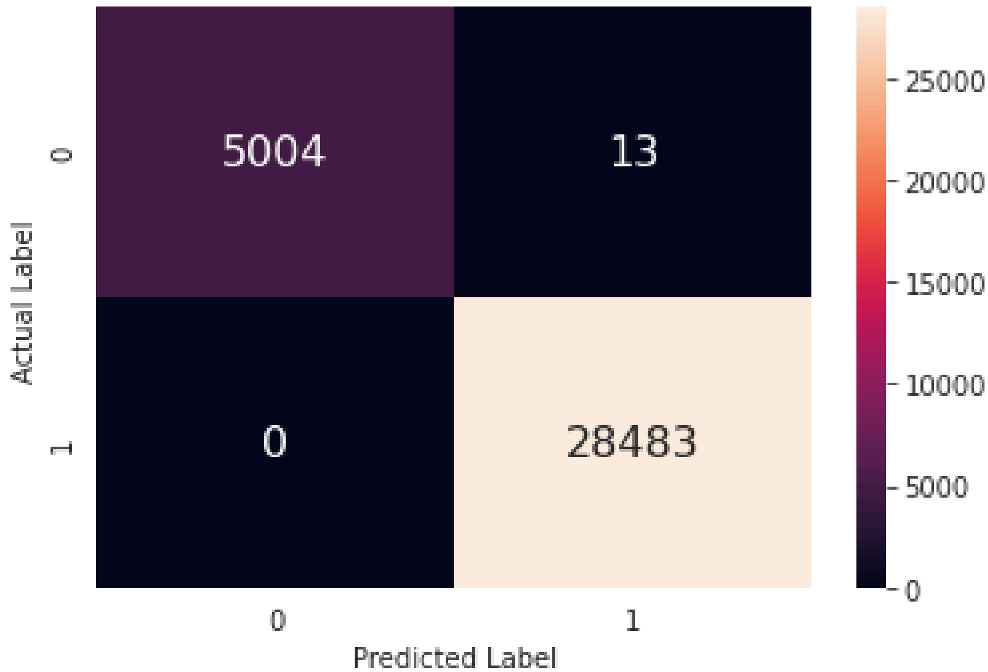

```
Best hyper parameter:  {'n_estimators': 500, 'max_depth': 500}
Model Score:  0.929465962863319
Model estimator:  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=500, max_features='auto',
                       max_leaf_nodes=None, 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=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[ ]:
```
clf = plot_auc(final_tf_idf_train, final_tf_idf_test, max_depth= 500,␣
↪n_estimators= 500)
```



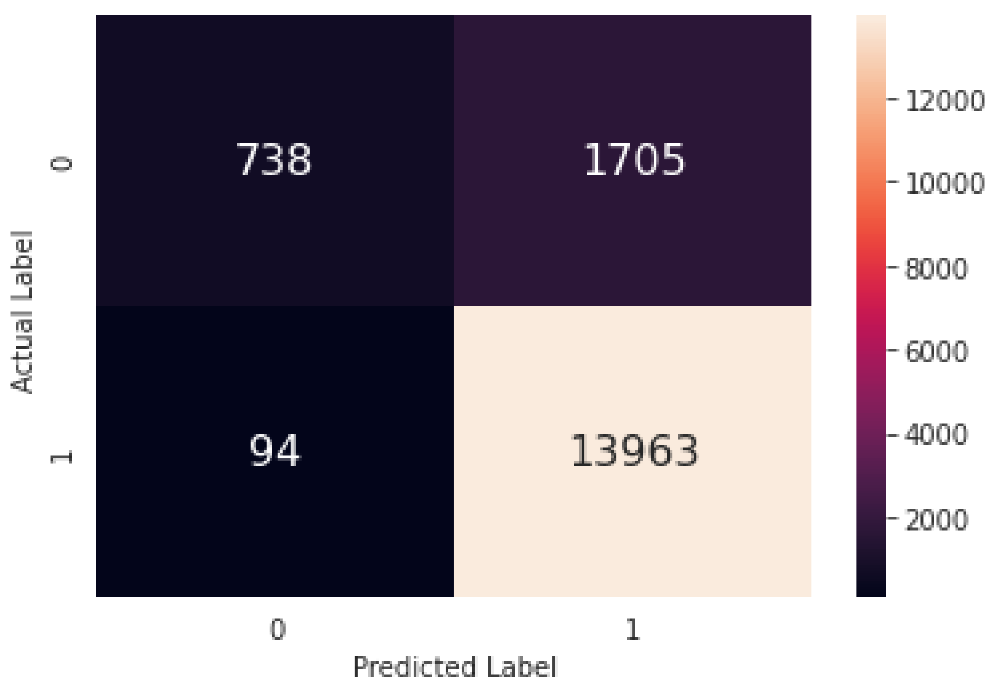
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= final_tf_idf_train, y_data=␣
↪Y_train, model = clf)
```

```
Train confusion matrix
```



[ ]:
```
confusion_matrix_plot(dataType= 'Test', x_data= final_tf_idf_test, y_data=␣
↪y_test, model= clf)
```

Test confusion matrix



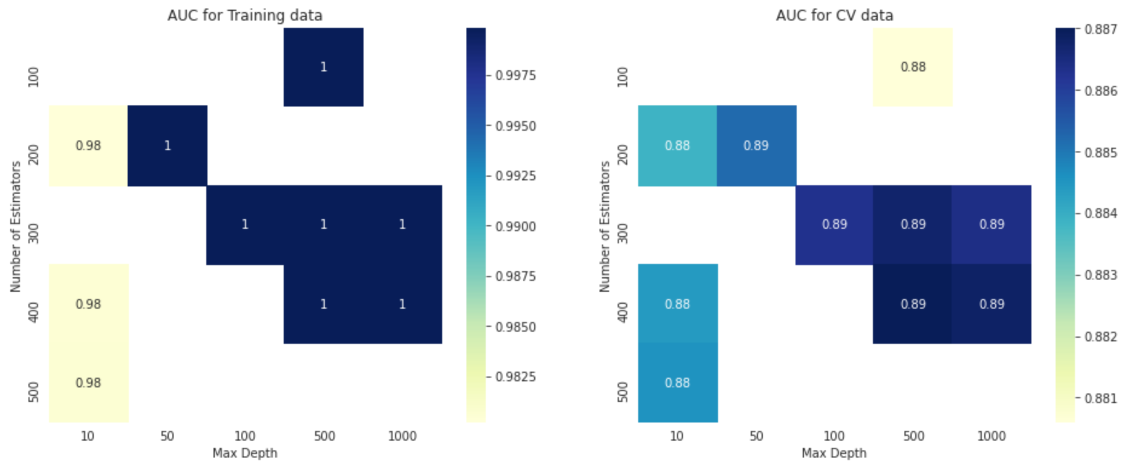
6.1.4 [5.1.4] Wordcloud of top 20 important features from SET 2

[ ]:
```
word_cloud_image(model = clf, vectorizer= tf_idf_vect)
```

6.1.5 [5.1.5] Applying Random Forests on AVG W2V, SET 3

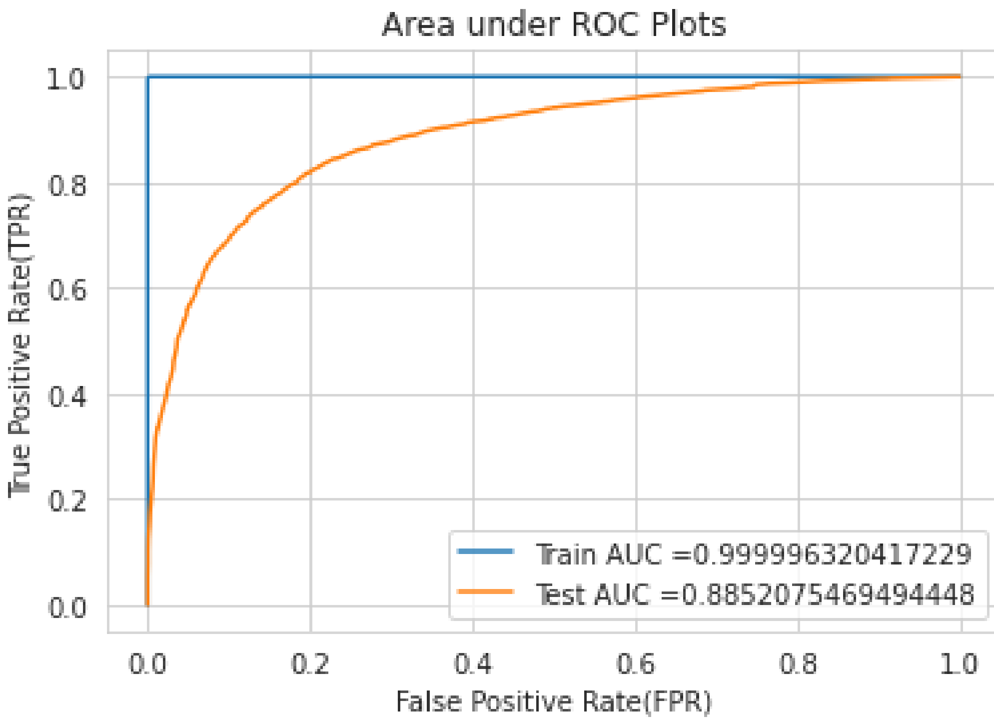
[ ]:
```
hyper_param_tuning(avg_w2v_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  7.7min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 10.8min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 13.9min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 18.8min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 26.8min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 33.3min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 36.1min finished
```



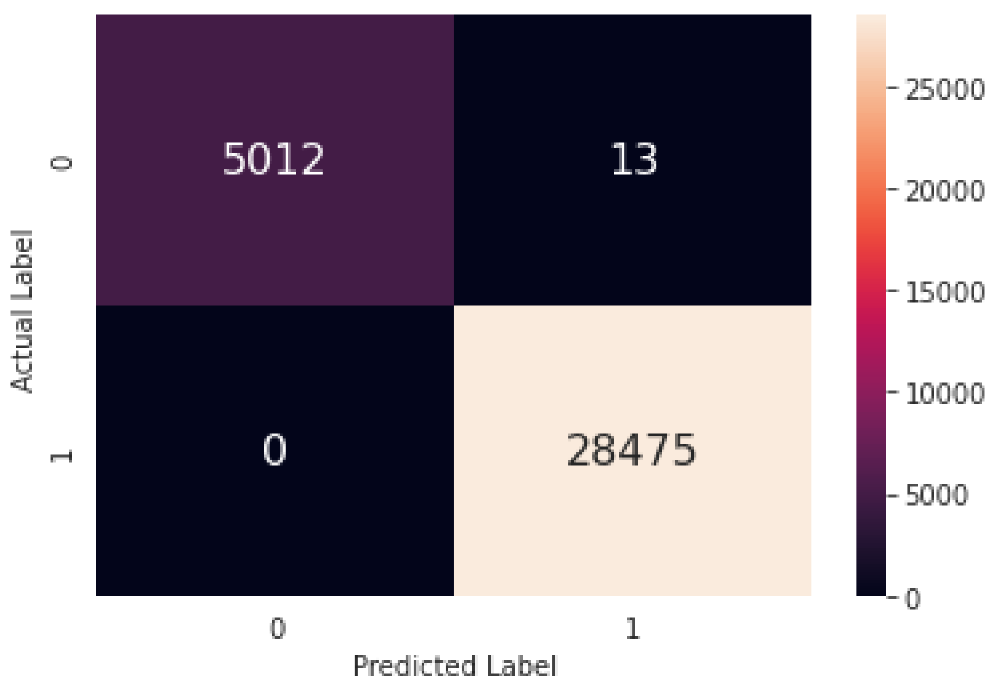

```
Best hyper parameter:  {'n_estimators': 400, 'max_depth': 500}
Model Score:  0.8870140551626374
Model estimator:  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=500, max_features='auto',
                       max_leaf_nodes=None, 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=400,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[ ]:
```
clf = plot_auc(avg_w2v_train, avg_w2v_test, max_depth= 500, n_estimators= 400)
```



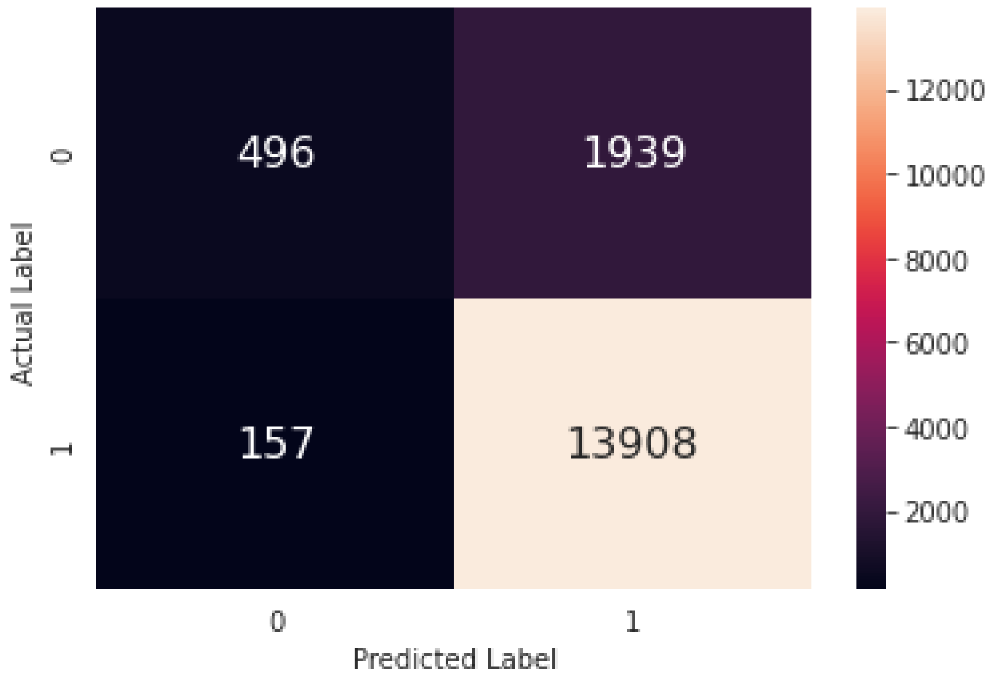
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= avg_w2v_train, y_data=␣
↪Y_train, model = clf)
```

```
Train confusion matrix
```



```
[ ]: confusion_matrix_plot(dataType= 'Test', x_data= avg_w2v_test, y_data= y_test,␣
      ↪model= clf)
```

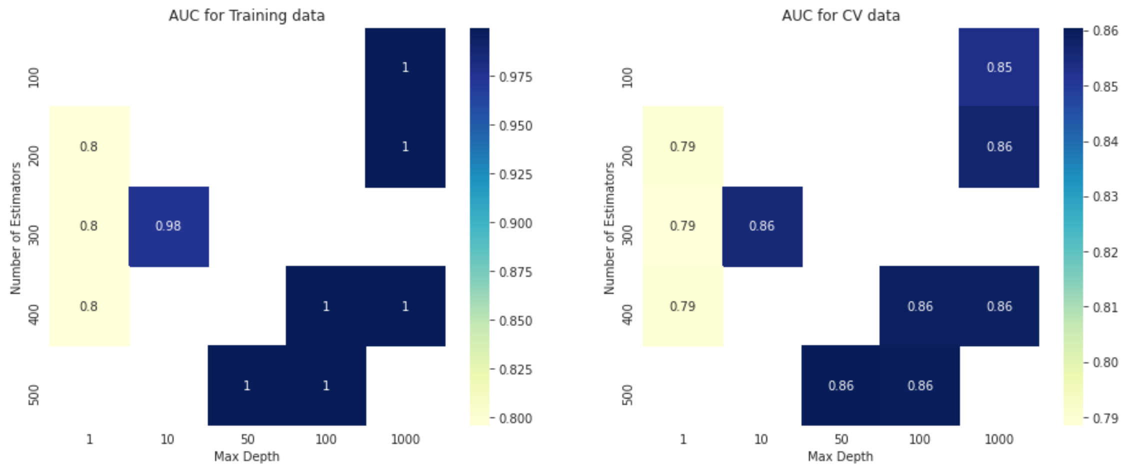
Test confusion matrix



6.1.6 [5.1.6] Applying Random Forests on TFIDF W2V, SET 4

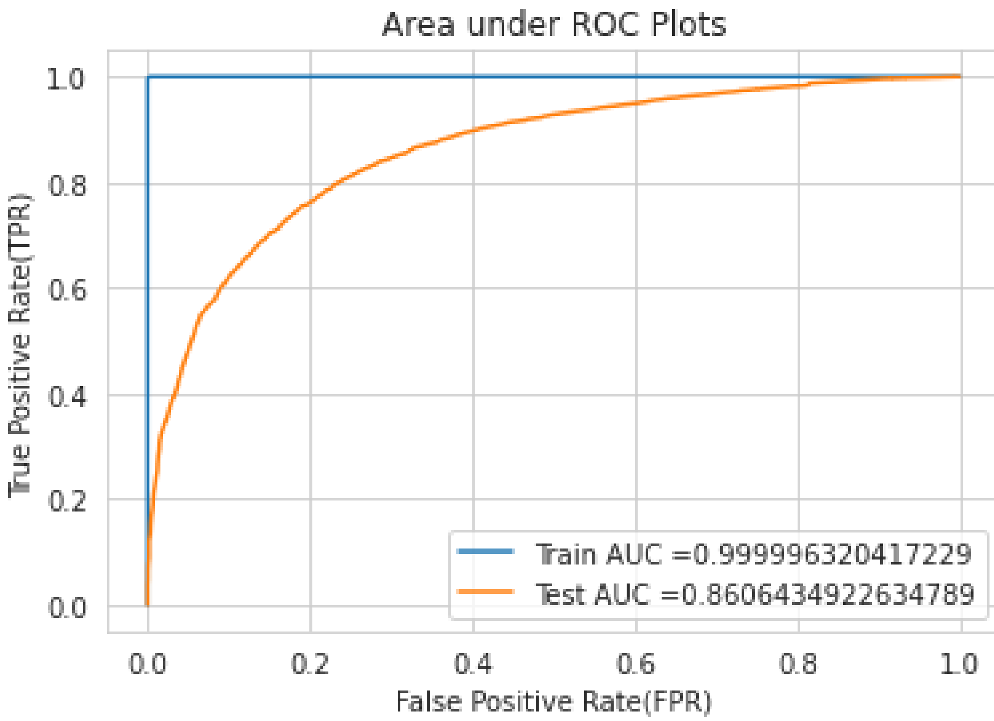
```
[ ]: hyper_param_tuning(tfidf_sent_vectors_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  7.3min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 13.5min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 20.3min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 22.7min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 24.0min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 30.6min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 32.1min finished
```



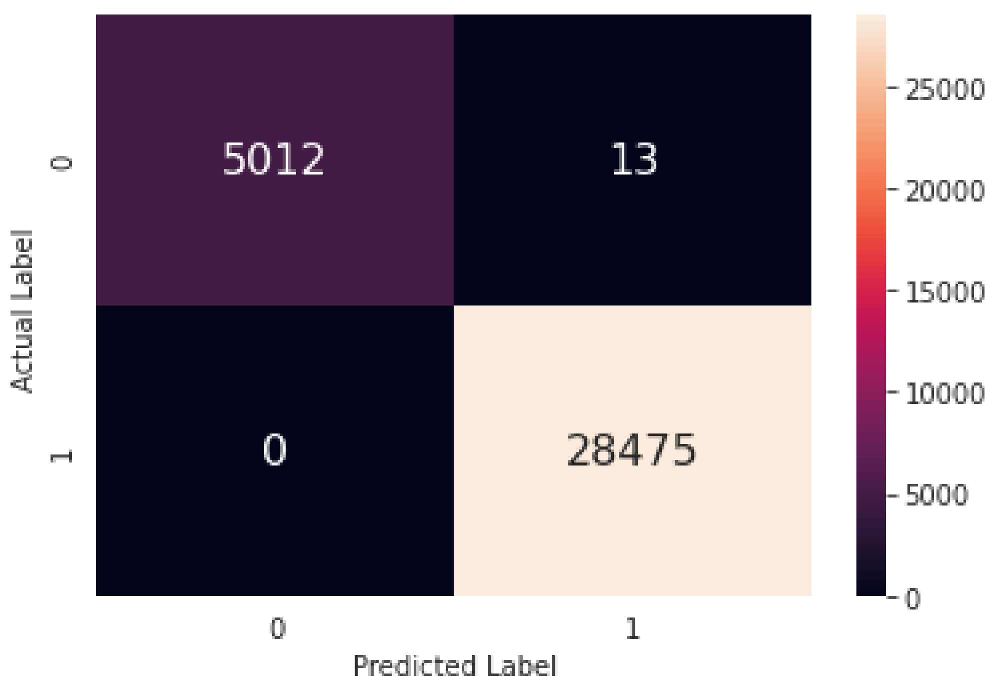

```
Best hyper parameter:  {'n_estimators': 500, 'max_depth': 50}
Model Score:  0.860505777988003
Model estimator:  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=50, max_features='auto',
                       max_leaf_nodes=None, 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=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[ ]:
```
clf = plot_auc(tfidf_sent_vectors_train, tfidf_sent_vectors_test, max_depth=␣
 ↪50, n_estimators= 500)
```



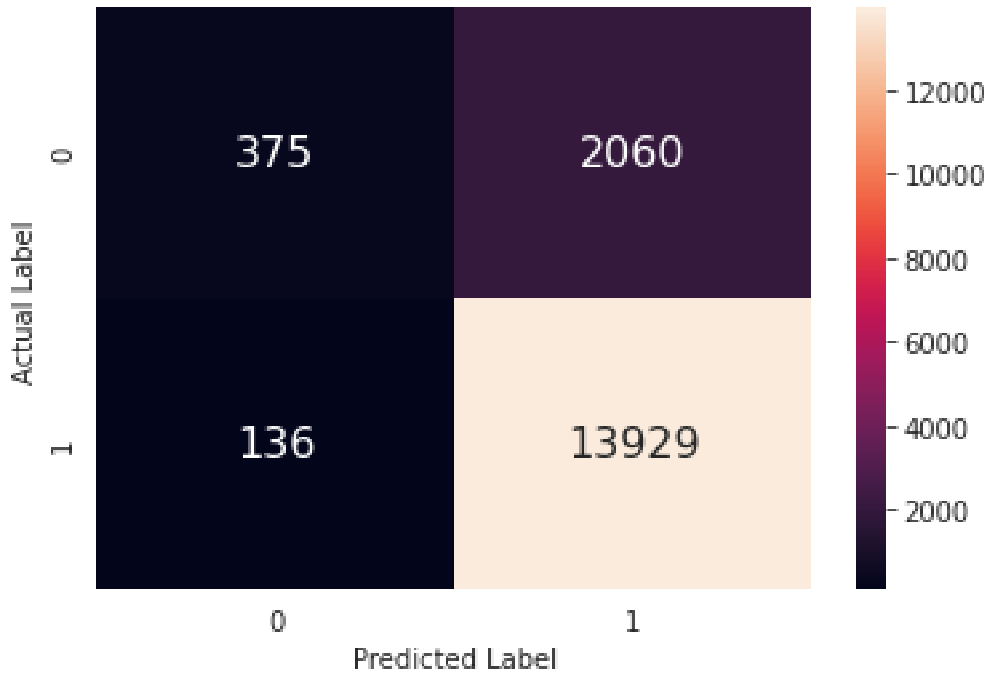
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= tfidf_sent_vectors_train,␣
↪y_data= Y_train, model = clf)
```

```
Train confusion matrix
```



```
[ ]: confusion_matrix_plot(dataType= 'Test', x_data= tfidf_sent_vectors_test,␣
     ↪y_data= y_test, model= clf)
```

```
Test confusion matrix
```



6.2 [5.2] Applying GBDT using XGBOOST

```
[ ]: # Official Doc: https://xgboost.readthedocs.io/en/latest/#
     # Learning Ref:https://machinelearningmastery.com/
     ↪develop-first-xgboost-model-python-scikit-learn/
     from xgboost import XGBClassifier
```

```
[ ]: # Defining a function for hyper parameter tuning based on Randomized Search CV
     def hyper_param_tuning_xgb(X_train):
       depth = [1, 2, 5, 10, 15, 20, 30, 40, 50]
       n_estimators = [100, 200, 300, 400, 500, 1000]
       tuned_parameters = [{'max_depth':depth, 'n_estimators': n_estimators}]

       # Applying RandomizedSearchCV with k folds = 5 and taking 'roc_auc' as score␣
     ↪metric
```

```
clf = RandomizedSearchCV(XGBClassifier(), param_distributions=␣
↪tuned_parameters, scoring = 'roc_auc', cv=5, return_train_score= True,␣
↪verbose = 10, n_jobs= -1)
clf.fit(X_train, Y_train)

# Plotting the hyper parameters CV results wrt. AUC i.e. mean_score where␣
↪scoring is done using 'roc_auc'
max_depth_list = list(clf.cv_results_['param_max_depth'].data)
n_estimators_list = list(clf.cv_results_['param_n_estimators'].data)
sns.set_style("whitegrid")
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
data = pd.DataFrame(data={'Number of Estimators':n_estimators_list, 'Max␣
↪Depth':max_depth_list, 'AUC':clf.cv_results_['mean_train_score']})
data = data.pivot(index='Number of Estimators', columns='Max Depth',␣
↪values='AUC')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Training␣
↪data')
plt.subplot(1,2,2)
data = pd.DataFrame(data={'Number of Estimators':n_estimators_list, 'Max␣
↪Depth':max_depth_list, 'AUC':clf.cv_results_['mean_test_score']})
data = data.pivot(index='Number of Estimators', columns='Max Depth',␣
↪values='AUC')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for CV data')
plt.show()

print('Best hyper parameter: ', clf.best_params_)
print('Model Score: ', clf.best_score_)
print('Model estimator: ', clf.best_estimator_)
```

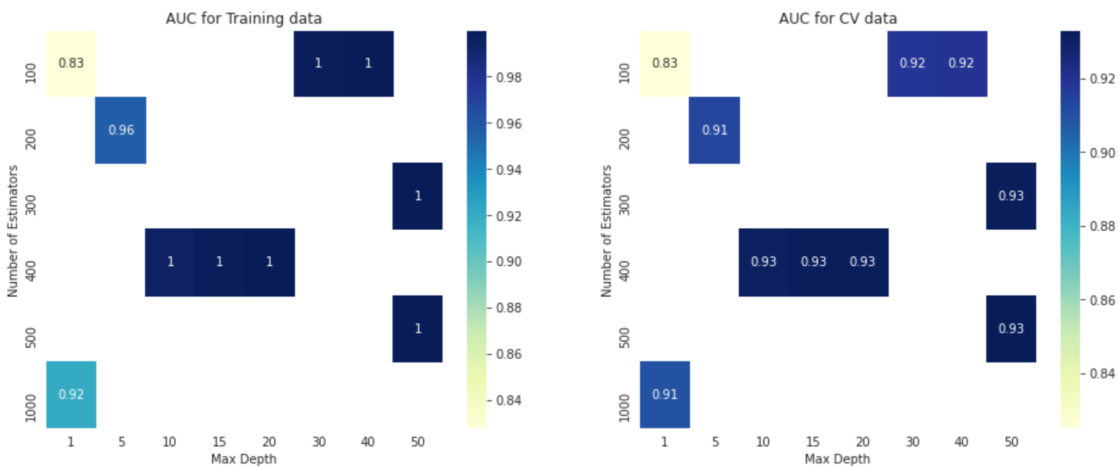
[ ]:
```
# Defining a function so to plot area under roc for both the train and test data
def plot_auc_xgb(X_train, x_test, max_depth, n_estimators):
  model = XGBClassifier(max_depth= max_depth, n_estimators= n_estimators,␣
↪n_jobs= -1)
  model.fit(X_train, Y_train)
  # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
  # not the predicted outputs

  train_fpr, train_tpr, thresholds = roc_curve(Y_train, model.
↪predict_proba(X_train)[:,1])
  test_fpr, test_tpr, thresholds = roc_curve(y_test, model.
↪predict_proba(x_test)[:,1])

  plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr,␣
↪train_tpr)))
```

```
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("Area under ROC Plots")
plt.show()
return model
```

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

[ ]:
```
hyper_param_tuning_xgb(X_train= final_counts_train_bow)
```

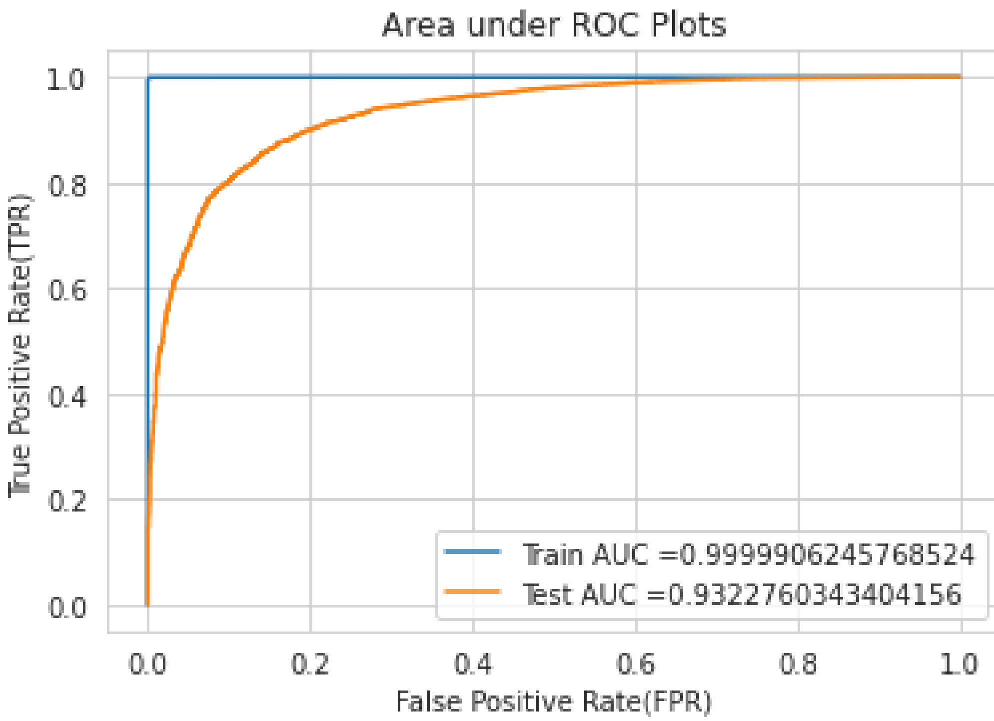
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   56.2s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  9.9min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 19.0min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 70.8min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 84.0min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 116.6min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 127.6min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 135.7min finished
```




```
Best hyper parameter:  {'n_estimators': 500, 'max_depth': 50}
Model Score:  0.9329617164680786
Model 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=50,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
```

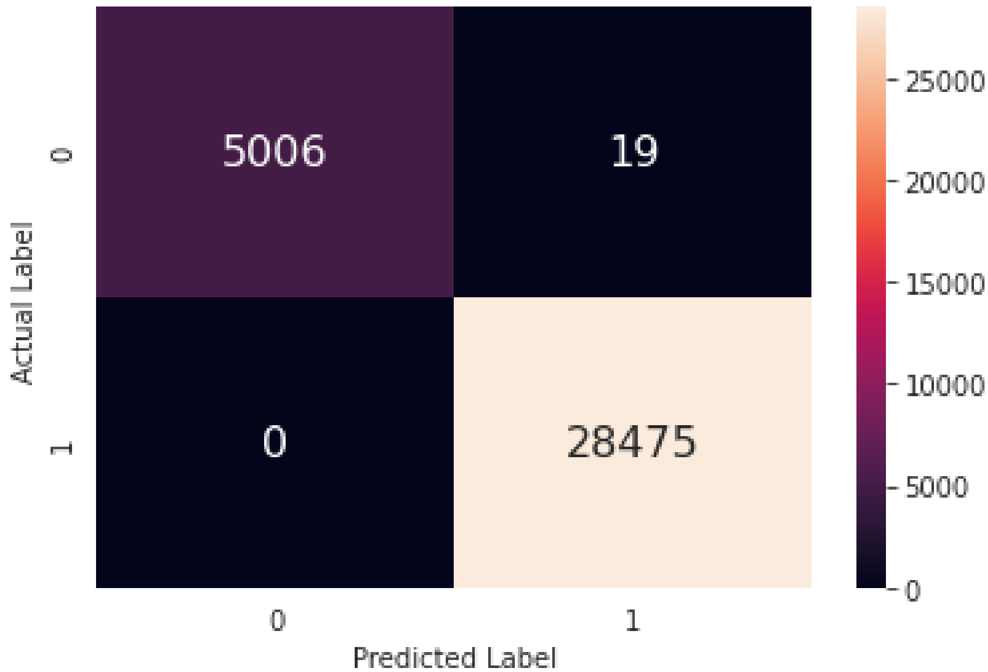
```
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[ ]:
```
clf = plot_auc_xgb(final_counts_train_bow, final_counts_test_bow, max_depth=␣
 ↪50, n_estimators= 500)
```



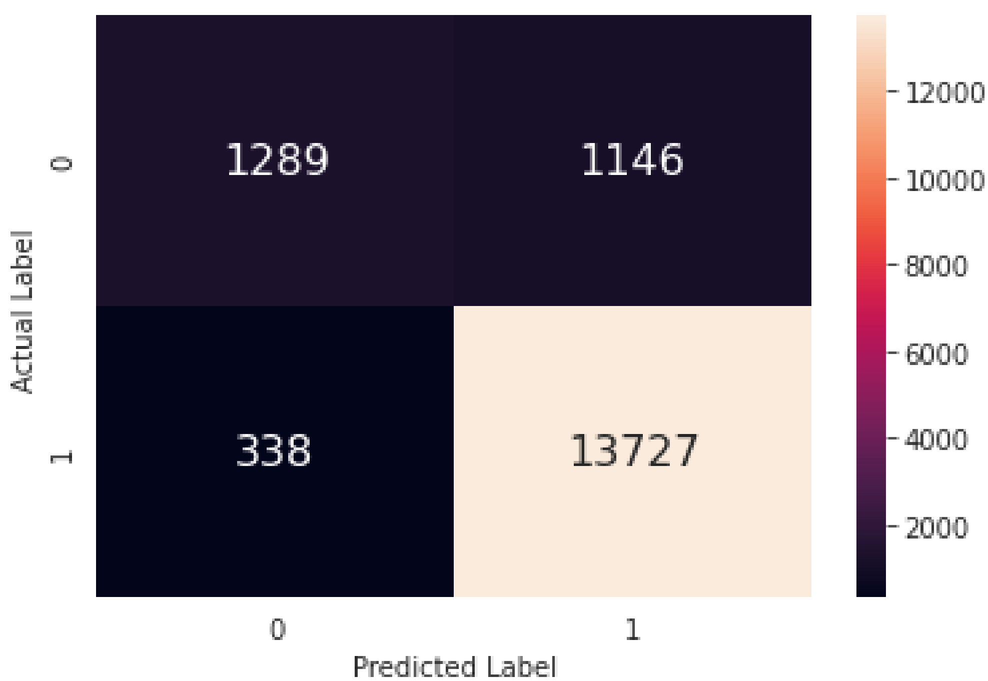
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= final_counts_train_bow,␣
 ↪y_data= Y_train, model = clf)
```

```
Train confusion matrix
```



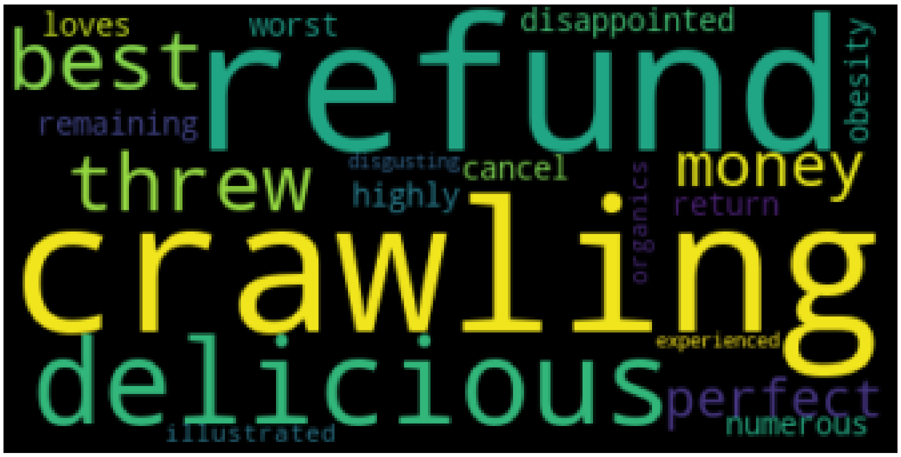
[ ]:
```
confusion_matrix_plot(dataType= 'Test', x_data= final_counts_test_bow, y_data=
↪y_test, model= clf)
```

```
Test confusion matrix
```



6.2.2 Wordcloud of top 20 important features

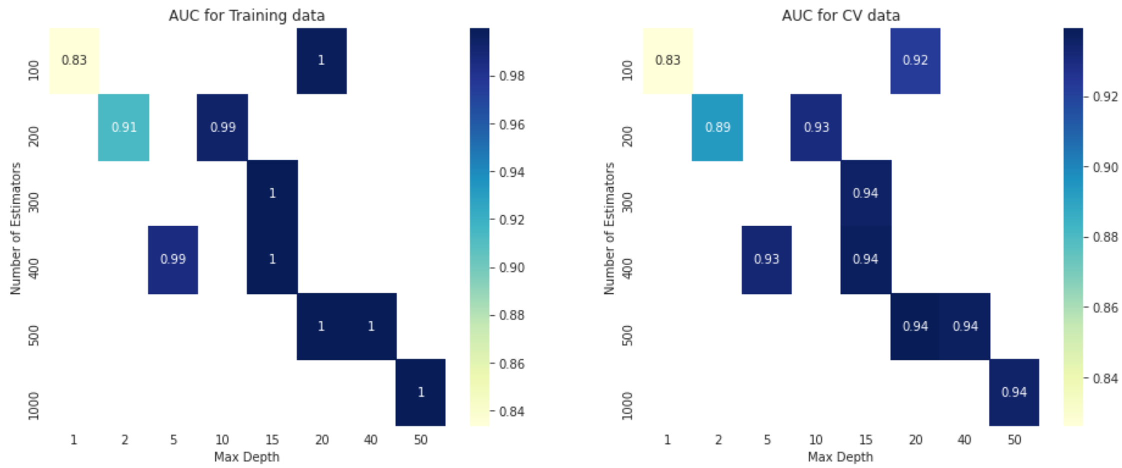
[ ]:
```
word_cloud_image(model = clf, vectorizer= count_vect)
```



6.2.3 [5.2.2] Applying XGBOOST on TFIDF, SET 2

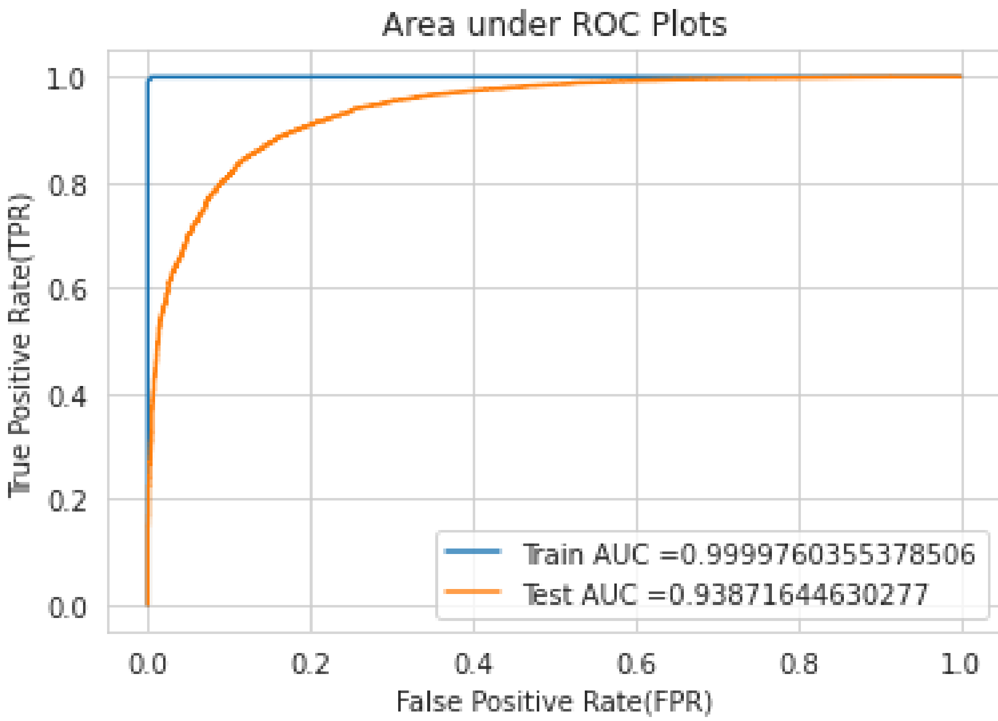
[ ]:
```
hyper_param_tuning_xgb(X_train= final_tf_idf_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   2.6min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   5.2min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  81.8min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 116.6min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 163.5min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 172.1min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 211.0min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 220.8min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 225.2min finished
```



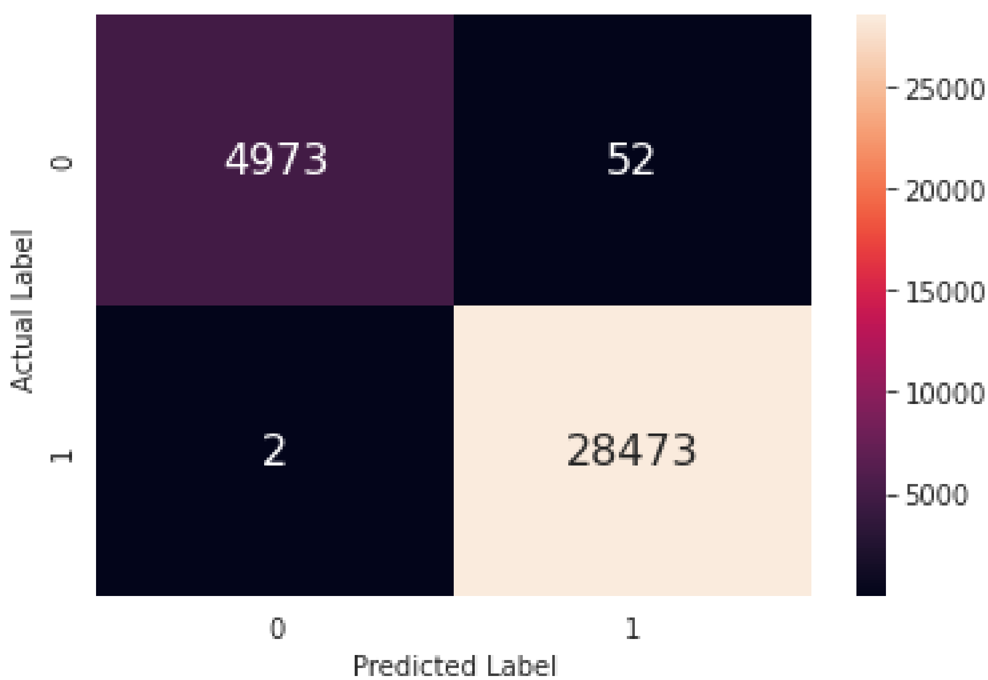

```
Best hyper parameter:  {'n_estimators': 500, 'max_depth': 20}
Model Score:  0.9394453568356299
Model estimator:  XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=20,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[ ]:
```
clf = plot_auc_xgb(final_tf_idf_train, final_tf_idf_test, max_depth= 20,␣
↪n_estimators= 500)
```



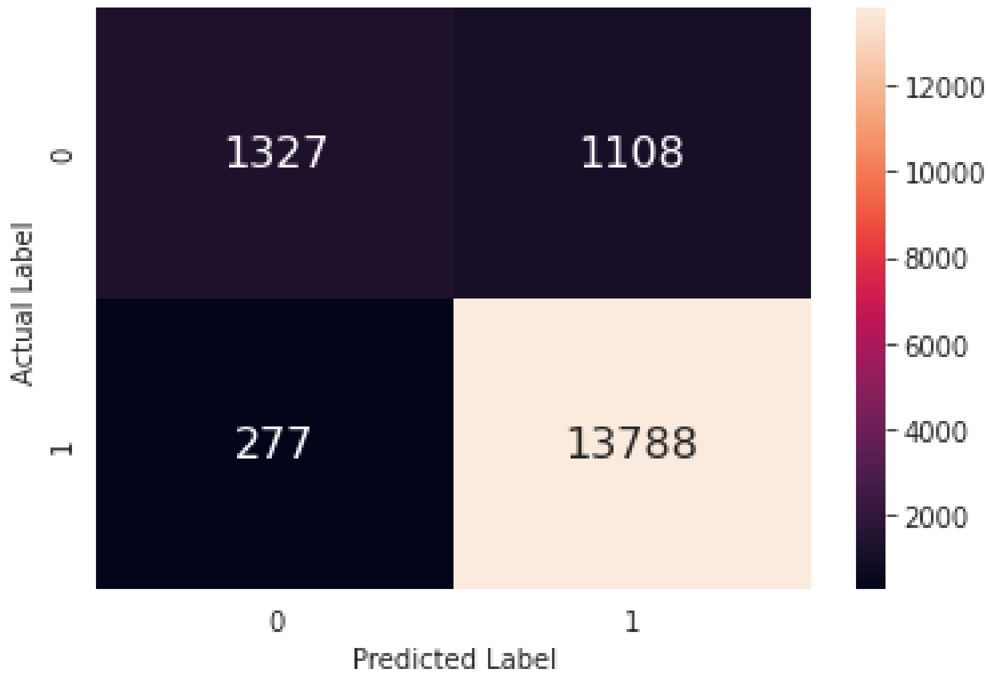
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= final_tf_idf_train, y_data=␣
 ↪Y_train, model = clf)
```

Train confusion matrix



[ ]:
```
confusion_matrix_plot(dataType= 'Test', x_data= final_tf_idf_test, y_data=
↪y_test, model= clf)
```

Test confusion matrix



6.2.4 Wordcloud of top 20 important features

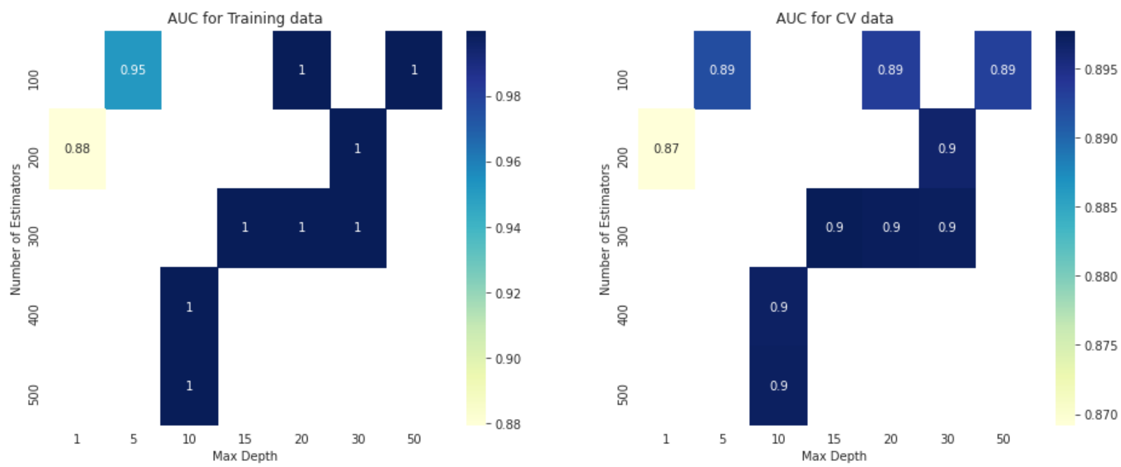
[ ]:
```
word_cloud_image(model = clf, vectorizer= tf_idf_vect)
```

6.2.5 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

[ ]:
```
avg_w2v_tr_ = np.array(avg_w2v_train)
avg_w2v_ts_ = np.array(avg_w2v_test)
```

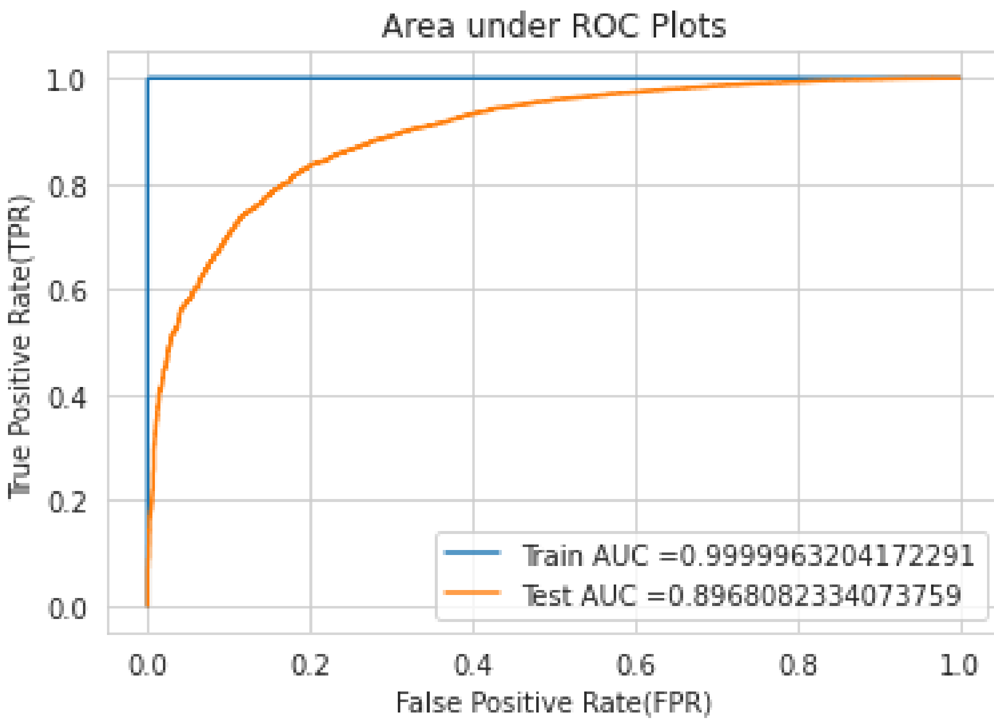
[ ]:
```
hyper_param_tuning_xgb(X_train= avg_w2v_tr_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   26.3s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   52.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  9.2min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 10.1min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 21.1min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 31.3min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 44.3min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 60.8min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 63.8min finished
```



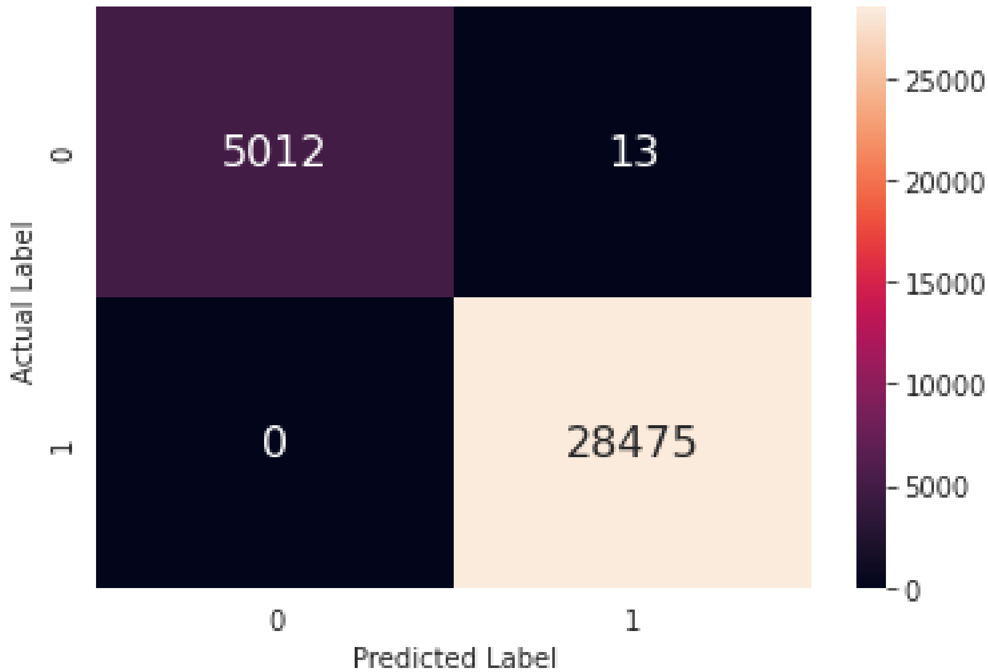

```
Best hyper parameter:  {'n_estimators': 300, 'max_depth': 15}
Model Score:  0.8977599447887865
Model estimator:  XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=None, n_estimators=300, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[ ]:
```
clf = plot_auc_xgb(avg_w2v_tr_, avg_w2v_ts_, max_depth= 15, n_estimators= 300)
```



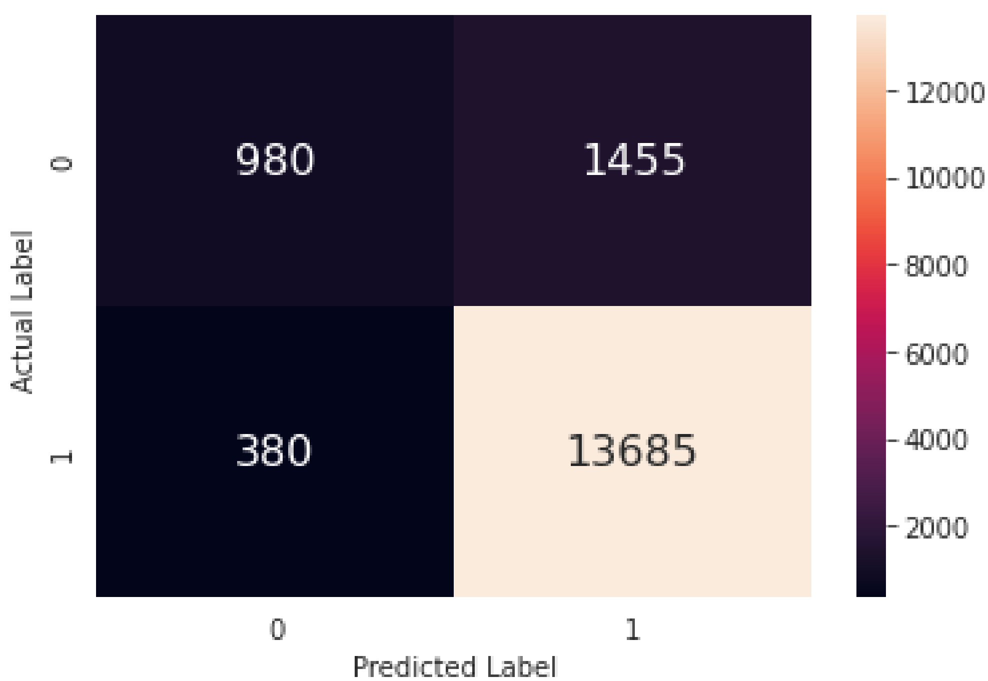
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= avg_w2v_tr_, y_data= Y_train,␣
↪model = clf)
```

```
Train confusion matrix
```




```
[ ]: confusion_matrix_plot(dataType= 'Test', x_data= avg_w2v_ts_, y_data= y_test,␣
     ↪model= clf)
```

Test confusion matrix

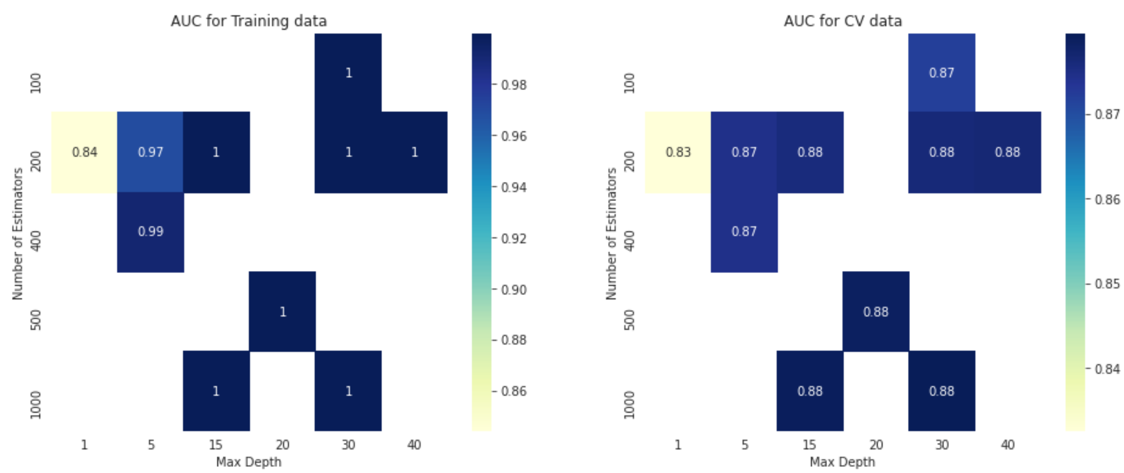


6.2.6 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

[ ]:
```
tfidf_sent_vectors_tr_ = np.array(tfidf_sent_vectors_train)
tfidf_sent_vectors_ts_ = np.array(tfidf_sent_vectors_test)
```

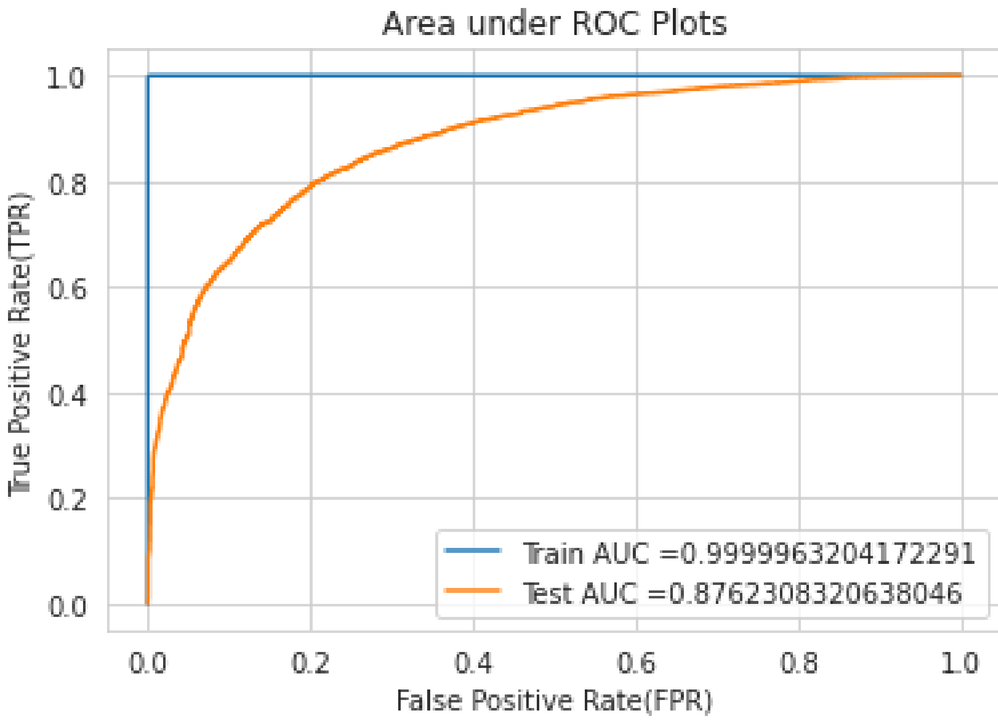
[ ]:
```
hyper_param_tuning_xgb(X_train= tfidf_sent_vectors_tr_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  3.2min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 18.6min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 47.5min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 51.7min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 67.5min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 92.5min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 97.6min finished
```



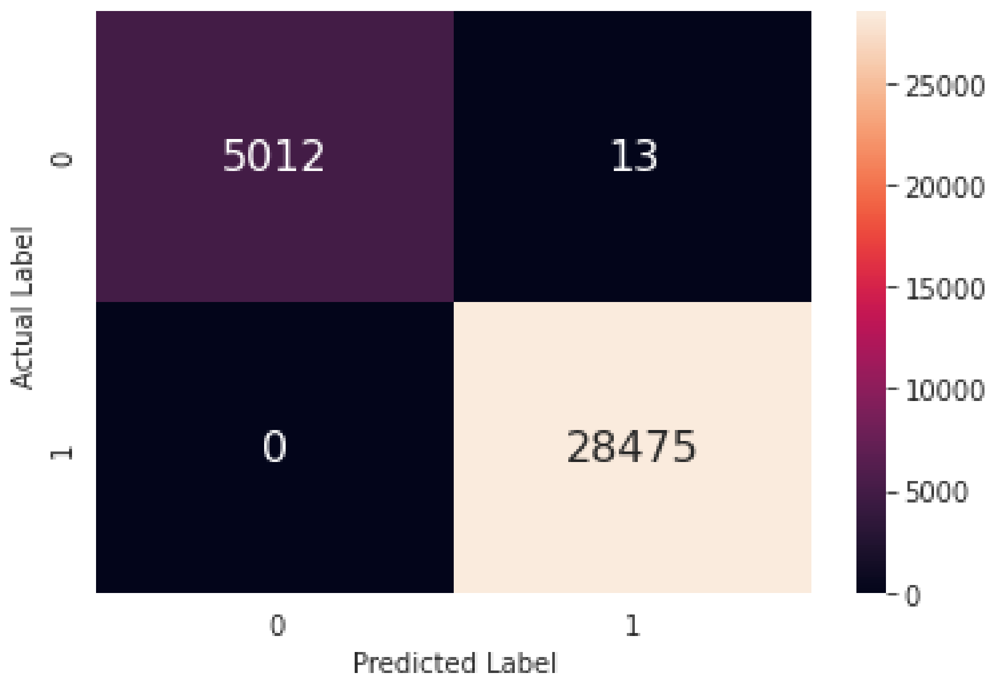

```
Best hyper parameter:  {'n_estimators': 1000, 'max_depth': 30}
Model Score:  0.8794988883501718
Model estimator:  XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=30,
              min_child_weight=1, missing=None, n_estimators=1000, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[ ]:
```
clf = plot_auc_xgb(tfidf_sent_vectors_tr_, tfidf_sent_vectors_ts_, max_depth=
↪30, n_estimators= 1000)
```



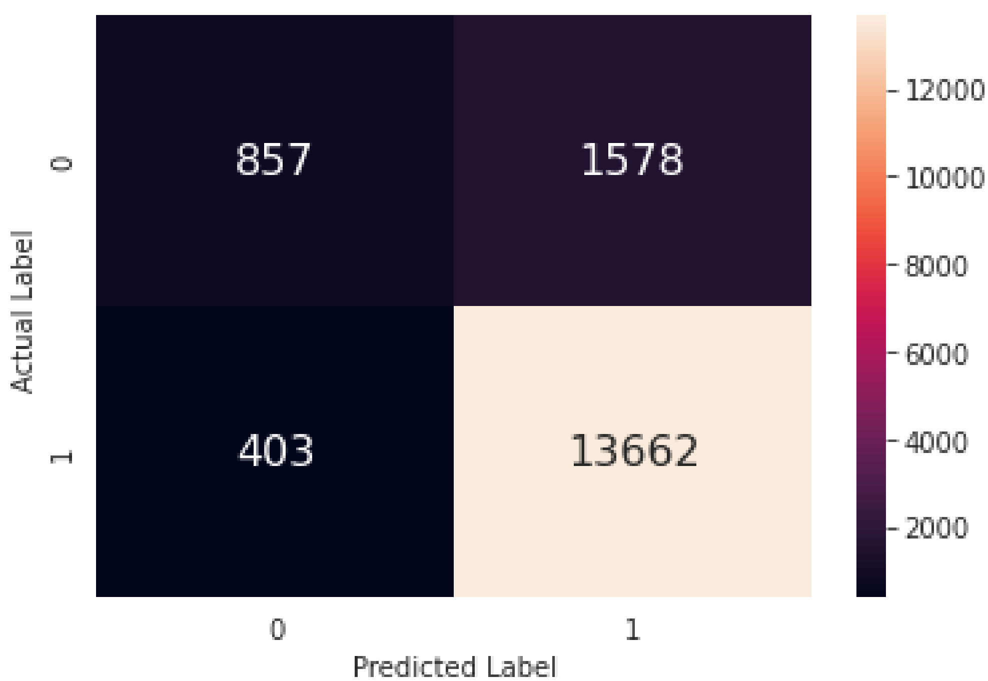
[ ]:
```
confusion_matrix_plot(dataType= 'Train', x_data= tfidf_sent_vectors_tr_,
↪y_data= Y_train, model = clf)
```

```
Train confusion matrix
```




```
[ ]: confusion_matrix_plot(dataType= 'Test', x_data= tfidf_sent_vectors_ts_, y_data=
     ↪y_test, model= clf)
```

Test confusion matrix



7 [6] Conclusions

```
[ ]: # Documetation: https://pypi.org/project/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
y = PrettyTable()

x.field_names= ["Vectorizer", "Hyper parameter(max_depth)", "Hyper␣
 ↪parameter(n_estimator)", "AUC"]
y.field_names= ["Vectorizer", "Hyper parameter(max_depth)", "Hyper␣
 ↪parameter(n_estimator)", "AUC"]

print('************* Random Forest *************','\n')
x.add_row(["BOW", 1000, 500, 0.93])
x.add_row(["TFIDF", 500, 500, 0.93])
x.add_row(["Avg W2V", 500, 400, .89])
x.add_row(["TFIDF Avg W2V", 50, 500, .86])
print(x,'\n')


print("************* GBDT *************",'\n')
y.add_row(["BOW",  50, 500, .93])
y.add_row(["TFIDF",  20, 500, .94])
y.add_row(["Avg W2V",  15, 300, .89])
y.add_row(["TFIDF Avg W2V", 30, 1000, .88])
print(y)
```

```
************* Random Forest *************

+---------------+----------------------------+-----------------------------+---
---+
|   Vectorizer  | Hyper parameter(max_depth) | Hyper parameter(n_estimator) |
AUC  |
+---------------+----------------------------+-----------------------------+---
---+
|      BOW      |            1000            |             500             |
0.93 |
|     TFIDF     |            500             |             500             |
0.93 |
|    Avg W2V    |            500             |             400             |
0.89 |
| TFIDF Avg W2V |             50             |             500             |
0.86 |
```

```
+----------------+-----------------------------+-------------------------------+---
---+

************* GBDT *************

+----------------+-----------------------------+-------------------------------+---
---+
|   Vectorizer   | Hyper parameter(max_depth) | Hyper parameter(n_estimator) |
AUC  |
+----------------+-----------------------------+-------------------------------+---
---+
|      BOW       |             50              |              500              |
0.93 |
|     TFIDF      |             20              |              500              |
0.94 |
|    Avg W2V     |             15              |              300              |
0.89 |
| TFIDF Avg W2V |             30              |              1000             |
0.88 |
+----------------+-----------------------------+-------------------------------+---
---+
```

[ ]: