09 Amazon Fine Food Reviews Analysis_RF

March 25, 2019

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".

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, 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
import sys

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from wordcloud import WordCloud, STOPWORDS
```

```
/usr/local/lib/python3.5/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarn
```

```
  from numpy.core.umath_tests import inner1d
```

```
In [2]: # using SQLite Table to read data.
        con = sqlite3.connect(os.path.join( os.getcwd(), '..', '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 point
        # 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 5
        # for tsne assignment you can take 5k data points

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

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

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

```
Number of data points in our data (5000, 10)
```

```
Out[2]:    Id   ProductId          UserId                      ProfileName  \
        0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
        1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
        2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     1                       1      1  1303862400
        1                     0                       0      0  1346976000
        2                     1                       1      1  1219017600

                         Summary                                              Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
```

```
In [3]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
```

```
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

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

```
(80668, 7)
```

```
Out[4]:                 UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

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

```
Out[5]:               UserId   ProductId                       ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

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

```
Out[6]: 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:

```
In [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
```

```
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:
```
       Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                               Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

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

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

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

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

```
Out[9]: (4986, 10)
```

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

```
Out[10]: 99.72
```

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

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

         display.head()
```

```
Out[11]:       Id   ProductId          UserId              ProfileName  \
         0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
         1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

            HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
         0                     3                       1      5  1224892800
         1                     3                       2      4  1212883200

                                               Summary  \
         0            Bought This for My Son at College
         1  Pure cocoa taste with crunchy almonds inside

                                                         Text
         0  My son loves spaghetti so I didn't hesitate or...
         1  It was almost a 'love at first bite' - the per...
```

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

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

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

```
(4986, 10)
```

Out[13]:
```
1    4178
0     808
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

In [14]:
```python
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

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

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

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

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.c
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
from bs4 import BeautifulSoup

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

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

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

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

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M5
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dca
```

In [17]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

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

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
```

```
In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victor
```

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

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wan
```

```
In [21]: # https://gist.github.com/sebleier/554280
         # we are removing the words from the stop words list: 'no', 'nor', 'not'
         # <br /><br /> ==> after the above steps, we are getting "br br"
         # we are including them into stop words list
         # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

         stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
                     "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
                     'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 
                     'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
```

```
                'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Sampling the data
final = final.sample(n=100000, replace=True)
```

In [23]:
```
# 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 stopwo
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 100000/100000 [00:37<00:00, 2665.66it/s]
```

In [24]:
```
preprocessed_reviews[1500]
```

Out[24]:
```
'subscribe several hormel compleats definitely best flavored dinner tried planning dou
```

[3.2] Preprocessing Review Summary

In [25]:
```
## Similartly you can do preprocessing for review summary also.
```

In [26]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for summary in tqdm(final['Summary'].values):
    summary = re.sub(r"http\S+", "", summary)
    summary = BeautifulSoup(summary, 'lxml').get_text()
    summary = decontracted(summary)
    summary = re.sub("\S*\d\S*", "", summary).strip()
    summary = re.sub('[^A-Za-z]+', ' ', summary)
    # https://gist.github.com/sebleier/554280
    summary = ' '.join(e.lower() for e in summary.split() if e.lower() not in stopword

    preprocessed_summary.append(summary.strip())
```

```
100%|| 100000/100000 [00:24<00:00, 4072.49it/s]
```

```
In [27]: final['CleanedText'] = preprocessed_reviews #adding a column of CleanedText which dis
         final['CleanedText'] = final['CleanedText'].astype('str')

         final['CleanedSummary'] = preprocessed_summary #adding a column of CleanedSummary whi
         final['CleanedSummary'] = final['CleanedSummary'].astype('str')

         final['Text_Summary'] = final['CleanedSummary'] + final['CleanedText']

         # # store final table into an SQlLite table for future.
         # conn = sqlite3.connect('final.sqlite')
         # c=conn.cursor()
         # conn.text_factory = str
         # final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
         #               index=True, index_label=None, chunksize=None, dtype=None)
         # conn.close()
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [28]: # #BoW
         # count_vect = CountVectorizer() #in scikit-learn
         # count_vect.fit(preprocessed_reviews)
         # print("some feature names ", count_vect.get_feature_names()[:10])
         # print('='*50)

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

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

```
In [29]: # #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/m

         # # 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_bigr
```

5.3 [4.3] TF-IDF

In [30]:
```
# tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
# tf_idf_vect.fit(preprocessed_reviews)
# print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_na
# print('='*50)

# final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
# print("the type of count vectorizer ",type(final_tf_idf))
# print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
# print("the number of unique words including both unigrams and bigrams ", final_tf_i
```

5.4 [4.4] Word2Vec

In [31]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [32]:
```
# # 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,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
#         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
```

```
In [33]: # 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])
```

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

[4.4.1.1] Avg W2v

```
In [34]: # # average Word2Vec
         # # compute average word2vec for each review.
         # sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         # for sent in tqdm(list_of_sentance): # for each review/sentence
         #     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need
         #     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]))
```

[4.4.1.2] TFIDF weighted W2v

```
In [35]: # # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
         # model = TfidfVectorizer()
         # tf_idf_matrix = model.fit_transform(preprocessed_reviews)
         # # we are converting a dictionary with word as a key, and the idf as a value
         # dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [36]: # # 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 = tfi

         # tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
         # row=0;
         # for sent in tqdm(list_of_sentance): # for each review/sentence
         #     sent_vec = np.zeros(50) # as word vectors are of zero length
         #     weight_sum =0; # num of words with a valid vector in the sentence/review
         #     for word in sent: # for each word in a review/sentence
```

```
#         if word in w2v_words and word in tfidf_feat:
#             vec = w2v_model.wv[word]
# #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
#             # to reduce the computation we are
#             # dictionary[word] = idf value of word in whole courpus
#             # sent.count(word) = tf valeus of word in this review
#             tf_idf = dictionary[word]*(sent.count(word)/len(sent))
#             sent_vec += (vec * tf_idf)
#             weight_sum += tf_idf
#     if weight_sum != 0:
#         sent_vec /= weight_sum
#     tfidf_sent_vectors.append(sent_vec)
#     row += 1
```

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)
The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)
Find the best hyper parameter which will give the maximum AUC value
Find the best hyper paramter using k-fold cross validation or simple cross validation data
Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

```
</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

In [37]:
```
# Source: https://docs.python.org/3/library/pickle.html

# Saving data to pickle file
def topicklefile(obj, file_name):
    pickle.dump(obj,open(file_name+'.pkl', 'wb'))
```

In [38]:
```
# Data from pickle file
def frompicklefile(file_name):
    data = pickle.load(open(file_name+'.pkl', 'rb'))
    return data
```

In [39]:
```
# Sort 'Time' column
final = final.sort_values(by='Time', ascending=True)
```

In [40]:
```
# Source: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.t

# Train Test split for train and test data
```

```
def data_split(X,y):
    # split the data set into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_st
    topicklefile(X_train, 'X_train')
    topicklefile(X_test, 'X_test')
    topicklefile(y_train, 'y_train')
    topicklefile(y_test, 'y_test')
```

In [41]:
```
def apply_avgw2v_train_test(X_train, X_test):

    # Training own Word2Vec model using your own text corpus
    list_of_sent_train = []
    for sent in X_train:#final['Text_Summary'].values:
        list_of_sent_train.append(sent.split())
    list_of_sent_test = []
    for sent in X_test:#final['Text_Summary'].values:
        list_of_sent_test.append(sent.split())

    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=8)

    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])

    # compute average word2vec for each review for train data
    avgw2v_train = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent_train): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        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
        avgw2v_train.append(sent_vec)
#     print(len(avgw2v_train))
#     print(len(avgw2v_train[0]))

    # compute average word2vec for each review for test data
    avgw2v_test = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent_test): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        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
        avgw2v_test.append(sent_vec)
#     print(len(avgw2v_test))
#     print(len(avgw2v_test[0]))

    return avgw2v_train, avgw2v_test
```

```
In [42]: def apply_tfidfw2v_train_test(X_train, X_test):

             # Training own Word2Vec model using your own text corpus
             list_of_sent_train = []
             for sent in X_train:#final['Text_Summary'].values:
                 list_of_sent_train.append(sent.split())
             list_of_sent_test = []
             for sent in X_test:#final['Text_Summary'].values:
                 list_of_sent_test.append(sent.split())

             # min_count = 5 considers only words that occured atleast 5 times
             w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=16)

             w2v_words = list(w2v_model.wv.vocab)

             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 = t

             tfidfw2v_train = []; # the tfidf-w2v for each sentence/review is stored in this l
             row=0;
             for sent in tqdm(list_of_sent_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
        tfidfw2v_train.append(sent_vec)
        row += 1


    tf_idf_matrix = model.transform(X_test)
    # 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 = t

    tfidfw2v_test = []; # the tfidf-w2v for each sentence/review is stored in this li
    row=0;
    for sent in tqdm(list_of_sent_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
        tfidfw2v_test.append(sent_vec)
        row += 1


    return tfidfw2v_train, tfidfw2v_test
```

In [43]:
```
# Applying BOW on train and test data and creating the
from sklearn.preprocessing import StandardScaler
from scipy.sparse import hstack


def apply_vectorizers_train_test(model_name, train_data, test_data):

    if model_name == 'BOW':
```

```
        #Applying BoW on Train data
        count_vect = CountVectorizer()

        #Applying BoW on Test data
        train_vect = count_vect.fit_transform(train_data)

        #Applying BoW on Test data similar to the bow_train data
        test_vect = count_vect.transform(test_data)

        topicklefile(train_vect, 'train_vect')
        topicklefile(test_vect, 'test_vect')

        print("'train_vect' and 'test_vect' are the pickle files.")
        return count_vect

    elif model_name == 'TF-IDF':
        #Applying TF-IDF on Train data
        count_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)

        #Applying BoW on Test data
        train_vect = count_vect.fit_transform(train_data)

        #Applying BoW on Test data similar to the bow_train data
        test_vect = count_vect.transform(test_data)

        topicklefile(train_vect, 'train_vect')
        topicklefile(test_vect, 'test_vect')

        print("'train_vect' and 'test_vect' are the pickle files.")
        return count_vect

    elif model_name == 'AvgW2V':
        train_vect, test_vect = apply_avgw2v_train_test(train_data, test_data)

        topicklefile(train_vect, 'train_vect')
        topicklefile(test_vect, 'test_vect')
        print("'train_vect' and 'test_vect' are the pickle files.")

    elif model_name == 'TF-IDF W2V':
        train_vect, test_vect = apply_tfidfw2v_train_test(train_data, test_data)

        topicklefile(train_vect, 'train_vect')
        topicklefile(test_vect, 'test_vect')
        print("'train_vect' and 'test_vect' are the pickle files.")

    else:
        #Error Message
        print('Model specified is not valid! Please check.')
```

In [44]:
```
def applying_rf_xgb(model_name, parameters, train_data, y_train):

    if model_name == 'RandomForest':
        rf_clf = RandomForestClassifier(n_jobs=-1, class_weight='balanced')
        clf = GridSearchCV(rf_clf, parameters, cv=10, scoring= 'roc_auc', n_jobs=-1,re
        clf.fit(train_data, y_train)
        clf_cv_results = pd.DataFrame(clf.cv_results_)
        max_depth_optimal = clf.best_params_.get('max_depth')
        n_estimators_optimal = clf.best_params_.get('n_estimators')
    elif model_name == 'XGBoost':
        xgb_clf = xgb.XGBClassifier(n_jobs=-1)
        clf = GridSearchCV(xgb_clf, parameters, cv=10, scoring= 'roc_auc', n_jobs=-1,r
        clf.fit(train_data, y_train)
        clf_cv_results = pd.DataFrame(clf.cv_results_)
        max_depth_optimal = clf.best_params_.get('max_depth')
        n_estimators_optimal = clf.best_params_.get('n_estimators')

    return clf_cv_results, max_depth_optimal, n_estimators_optimal
```

In [45]:
```
#Source: https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot
def train_cv_error_plot(cv_results, values_param):

    pvt = pd.pivot_table(cv_results, values=values_param, index='param_max_depth', co
    sns.set(font_scale=1.4)
    ax = sns.heatmap(pvt, annot=True, cmap='mako_r', fmt='.2g')
```

In [46]:
```
#Source: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFor
#https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifie

def rf_xgb_optimal(model_name, max_depth_optimal, n_estimators_optimal,train_vec, y_t
    if model_name == 'RandomForest':
        rf_optimal = RandomForestClassifier(max_depth = max_depth_optimal, n_estimato

        # fitting the model with optimal K for training data
        rf_optimal.fit(train_vec, y_train)

        return rf_optimal

    elif model_name == 'XGBoost':
        xgb_optimal = xgb.XGBClassifier(max_depth = max_depth_optimal, n_estimators =

        # fitting the model with optimal K for training data
        xgb_optimal.fit(train_vec, y_train)

        return xgb_optimal
```

In [47]:
```
# Confusion Matrix
def cm_fig(model_optimal, y_test, test_vec):
```

```
    cm = pd.DataFrame(confusion_matrix(y_test, model_optimal.predict(test_vec)))
    # print(confusion_matrix(y_test, y_pred))

    plt.figure(1, figsize=(18,5))
    plt.subplot(121)
    plt.title("Confusion Matrix")
    sns.set(font_scale=1.4)
    sns.heatmap(cm, cmap= 'gist_earth', annot=True, annot_kws={'size':15}, fmt='g')
```

In [48]:
```
#Reference: https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-
def error_plot(model_optimal, train_vec, y_train, test_vec, y_test):
    train_fpr, train_tpr, thresholds = roc_curve(y_train, model_optimal.predict_proba
    test_fpr, test_tpr, thresholds = roc_curve(y_test, model_optimal.predict_proba(tes

    plt.plot(train_fpr, train_tpr, label="train AUC = %0.3f" %auc(train_fpr, train_tp
    plt.plot(test_fpr, test_tpr, label="train AUC = %0.3f" %auc(test_fpr, test_tpr))
    plt.plot([0.0, 1.0], [0.0, 1.0],'k--')
    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC Curve")
    plt.show()

    return auc(test_fpr, test_tpr)
```

In [49]:
```
#Source: https://www.datacamp.com/community/tutorials/wordcloud-python
#https://stackoverflow.com/questions/16645799/how-to-create-a-word-cloud-from-a-corpu
# https://stackoverflow.com/questions/43606339/generate-word-cloud-from-single-column

def get_features_top(count_vect, model_optimal):
    features=count_vect.get_feature_names()
    feature_prob=model_optimal.feature_importances_.ravel()
    df_feature_proba = pd.DataFrame({'features':features, 'probabilities':feature_prob
    df_feature_proba = df_feature_proba.sort_values(by=['probabilities'],ascending=Fal

    wordcloud = WordCloud(max_font_size=100, max_words=100, background_color="black")
    plt.figure()
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

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

In [50]:
```
# Please write all the code with proper documentation
```

In [51]:
```
X = np.array(final['Text_Summary'])
y = np.array(final['Score'])
data_split(X,y)
X_train = frompicklefile('X_train')
```

```
X_test = frompicklefile('X_test')
y_train = frompicklefile('y_train')
y_test = frompicklefile('y_test')
count_vect = apply_vectorizers_train_test('BOW', X_train, X_test)
```

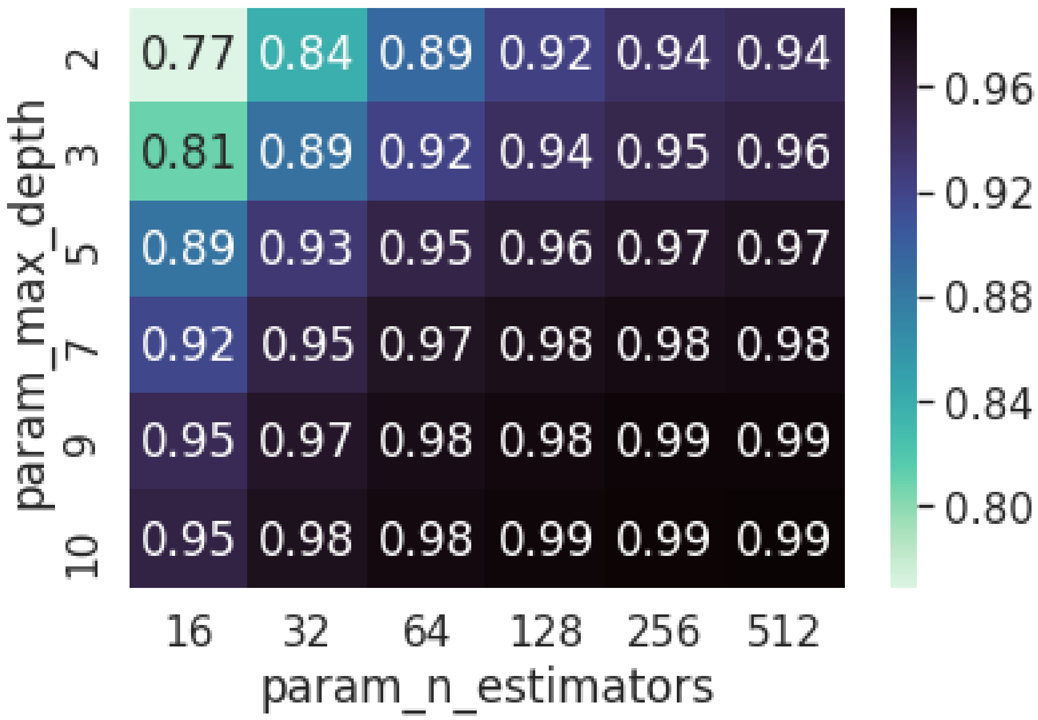
'train_vect' and 'test_vect' are the pickle files.

```
In [52]: train_vect = frompicklefile('train_vect')
         test_vect = frompicklefile('test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
```

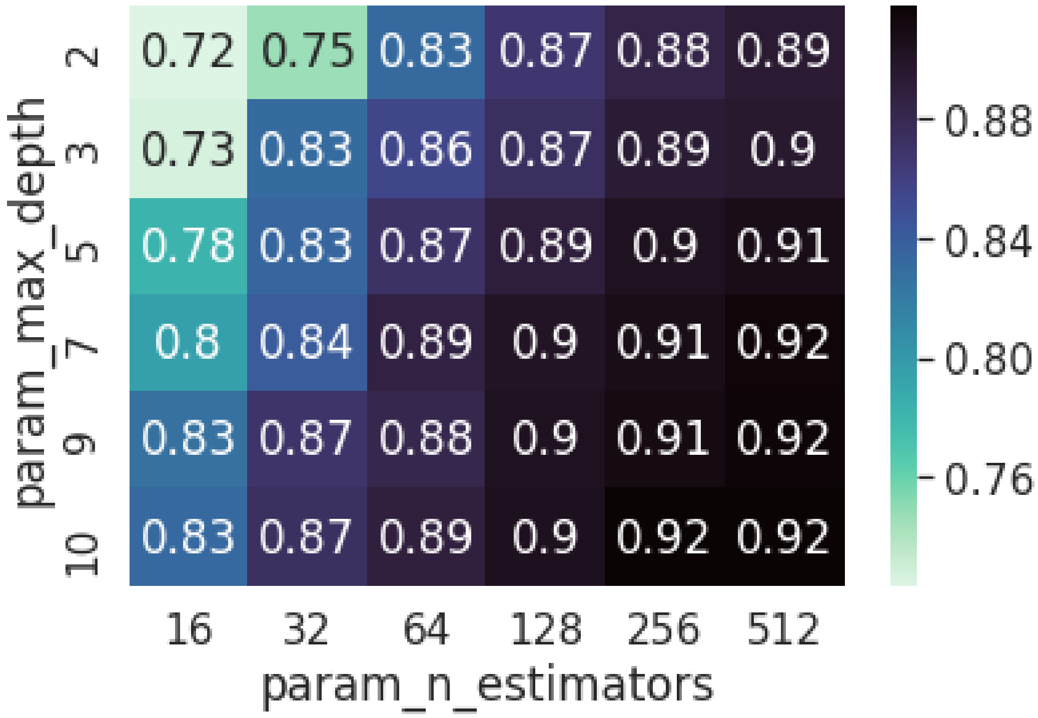
```
In [53]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in

         tree_max_depth = [2, 3, 5, 7, 9, 10]
         estimators = [16, 32, 64, 128, 256, 512]

         parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

         cv_results, bow_max_depth_optimal1, bow_n_estimators_optimal1 = applying_rf_xgb('Rand

         print('bow_max_depth_optimal1, bow_n_estimators_optimal1 :',bow_max_depth_optimal1, bo
```

```
bow_max_depth_optimal1, bow_n_estimators_optimal1 : 10 512
```

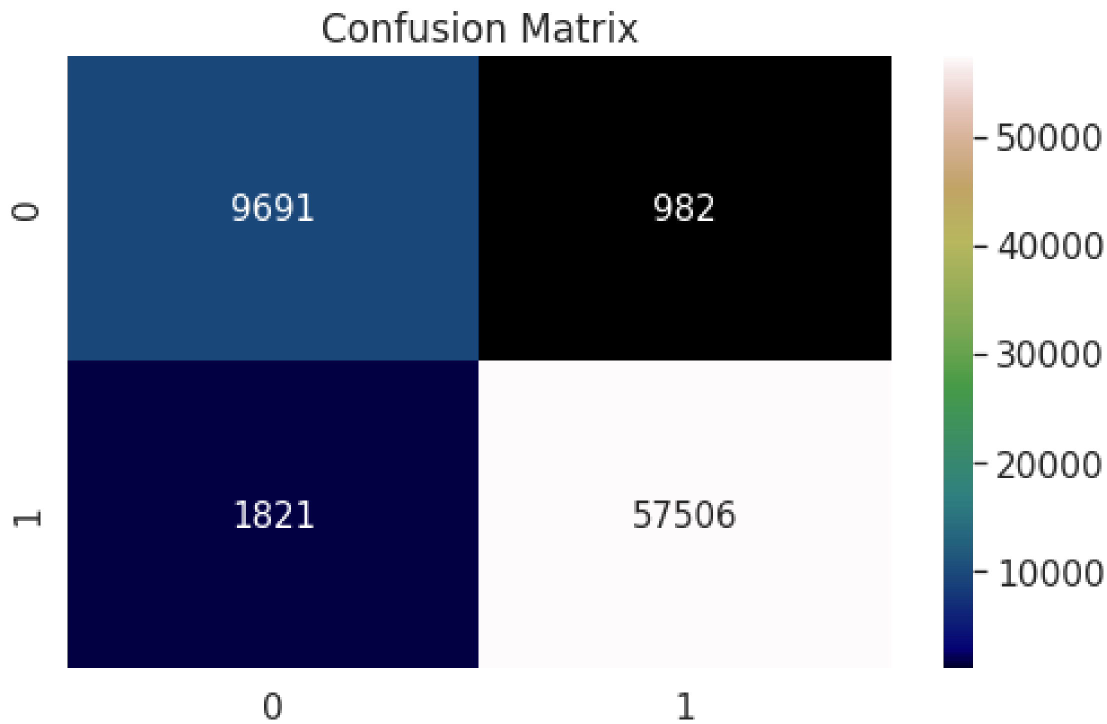
```
In [54]: # print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_train_score')
```



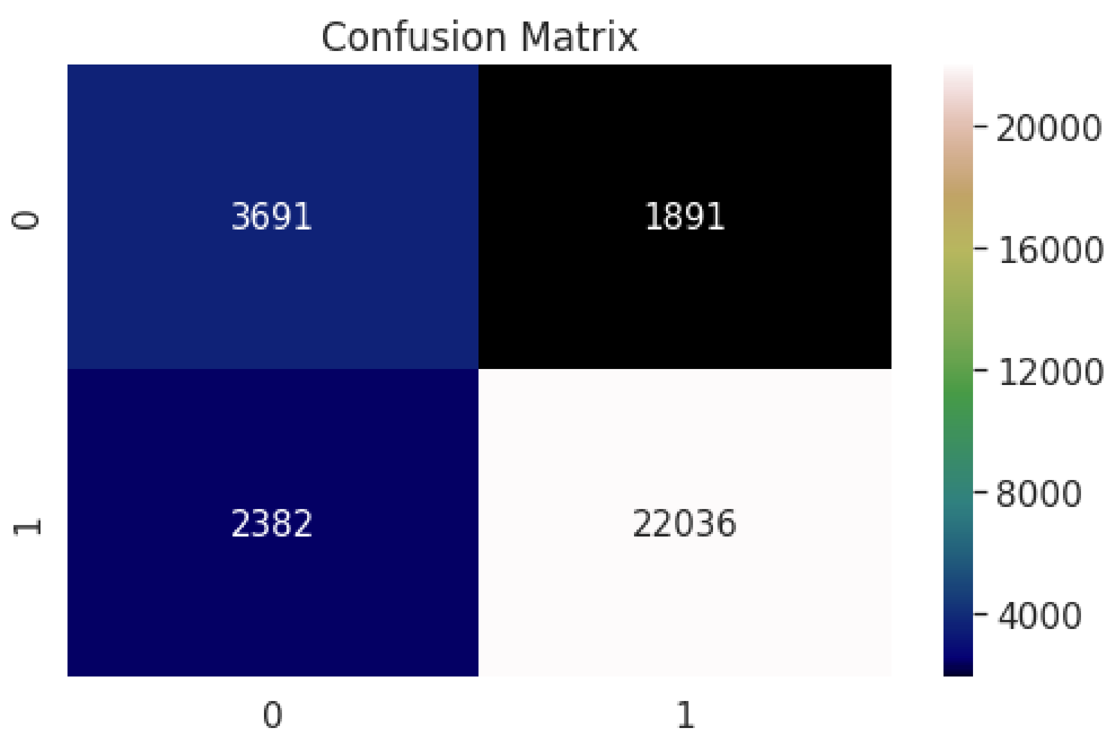
```
In [55]: # print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_test_score')
```



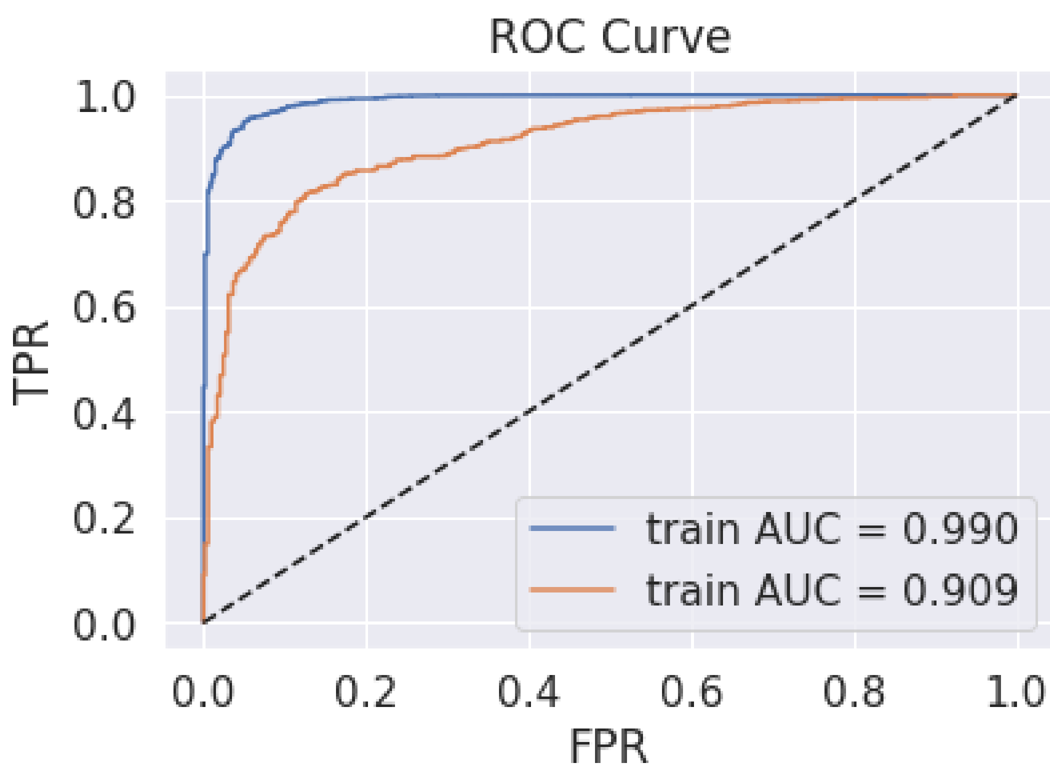

```
In [56]: rf_optimal = rf_xgb_optimal('RandomForest', bow_max_depth_optimal1, bow_n_estimators_o

         cm_fig(rf_optimal, y_train, train_vect)
```



In [57]: `cm_fig(rf_optimal, y_test, test_vect)`



```
In [58]: bow_auc1 = error_plot(rf_optimal, train_vect, y_train, test_vect, y_test)
```



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

```
In [59]: # Please write all the code with proper documentation
```

```
In [60]: get_features_top(count_vect, rf_optimal)
```

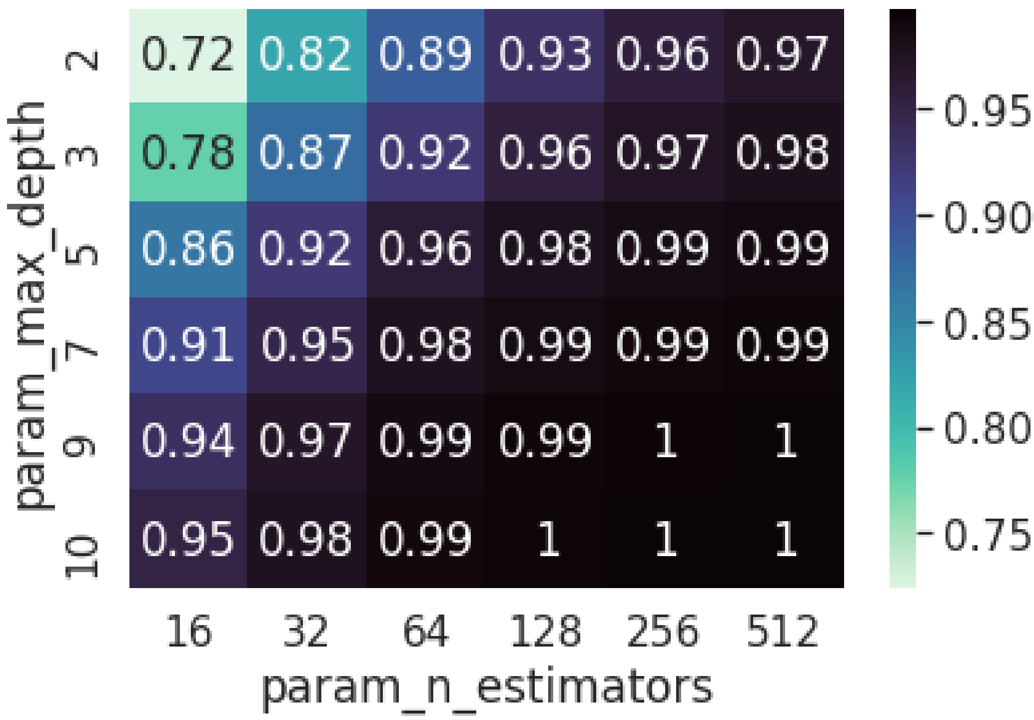
6.1.3 [5.1.3] Applying Random Forests on TFIDF, SET 2

```
In [61]: # Please write all the code with proper documentation

In [62]: X = np.array(final['Text_Summary'])
         y = np.array(final['Score'])
         data_split(X,y)
         X_train = frompicklefile('X_train')
         X_test = frompicklefile('X_test')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
         count_vect = apply_vectorizers_train_test('TF-IDF', X_train, X_test)

'train_vect' and 'test_vect' are the pickle files.


In [63]: train_vect = frompicklefile('train_vect')
         test_vect = frompicklefile('test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')

In [64]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in

         tree_max_depth = [2, 3, 5, 7, 9, 10]
         estimators = [16, 32, 64, 128, 256, 512]

         parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

         cv_results, tfidf_max_depth_optimal1, tfidf_n_estimators_optimal1 = applying_rf_xgb('
```

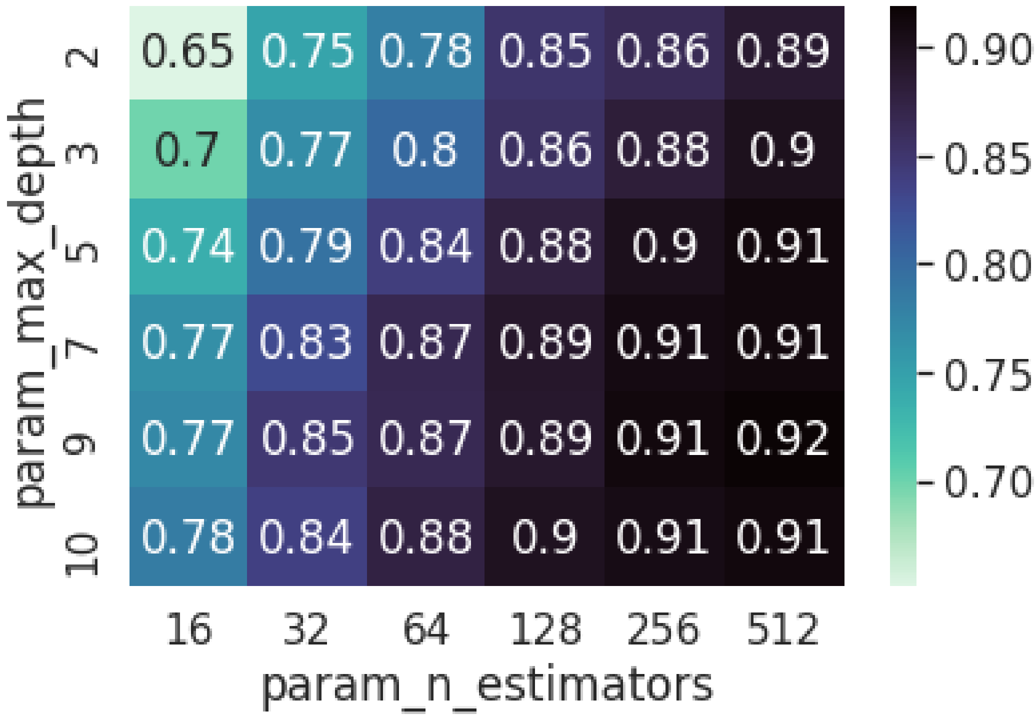
```
print('tfidf_max_depth_optimal1, tfidf_n_estimators_optimal1 :',tfidf_max_depth_optima
```

```
tfidf_max_depth_optimal1, tfidf_n_estimators_optimal1 : 9 512
```

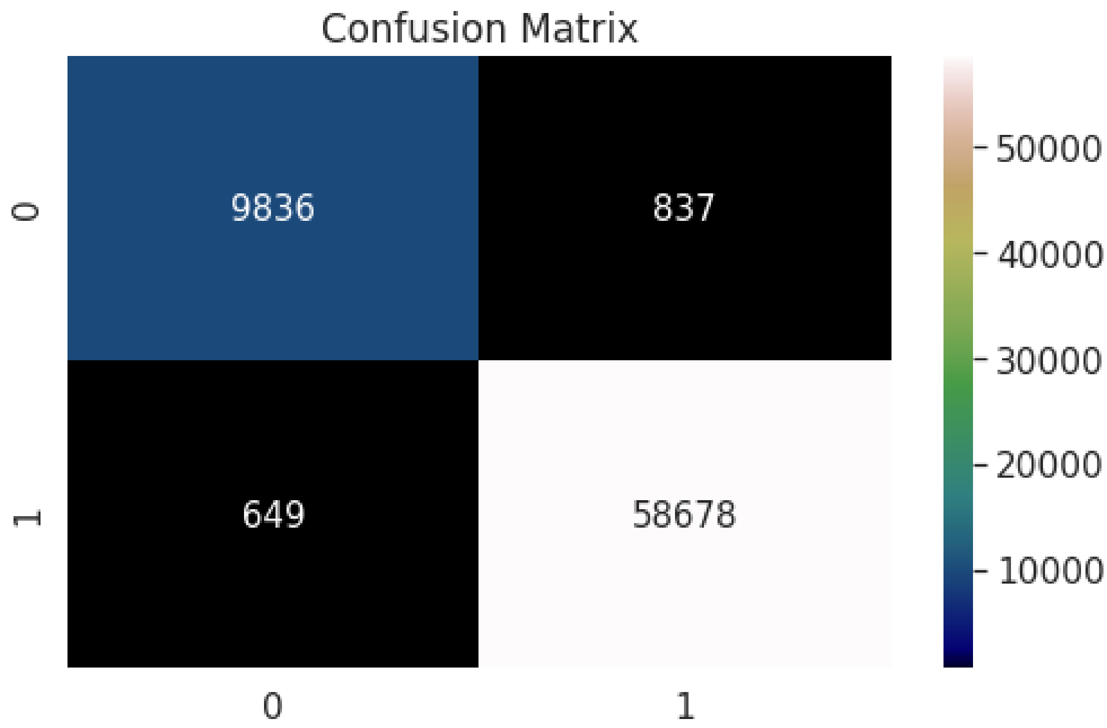
```
In [65]: # print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_train_score')
```



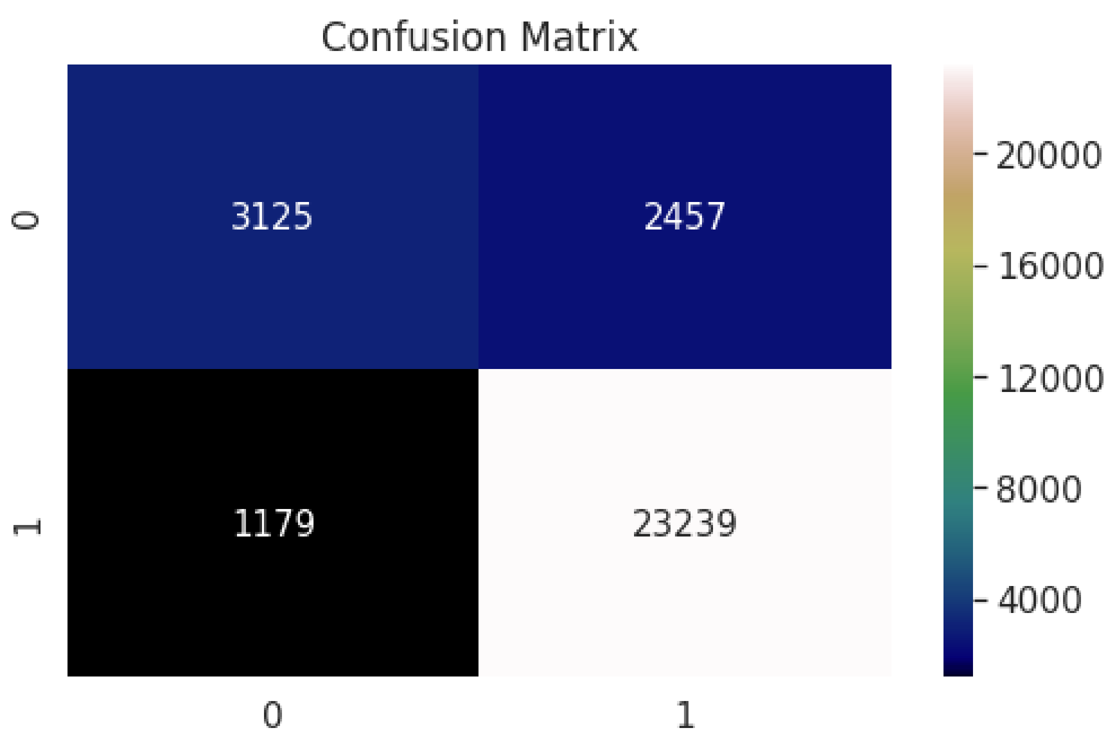

```
In [66]: # print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_test_score')
```



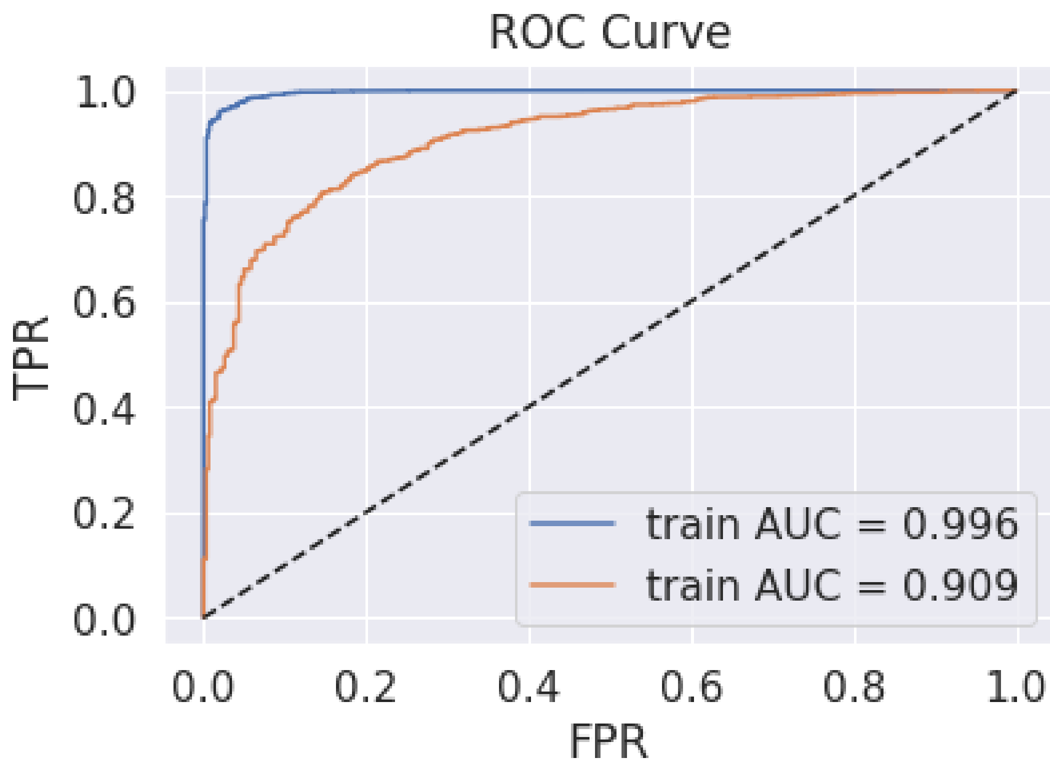

```
In [67]: rf_optimal = rf_xgb_optimal('RandomForest', tfidf_max_depth_optimal1, tfidf_n_estimato
         cm_fig(rf_optimal, y_train, train_vect)
```



In [68]: `cm_fig(rf_optimal, y_test, test_vect)`



```
In [69]: tfidf_auc1 = error_plot(rf_optimal, train_vect, y_train, test_vect, y_test)
```



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

```
In [70]: # Please write all the code with proper documentation
```

```
In [71]: get_features_top(count_vect, rf_optimal)
```

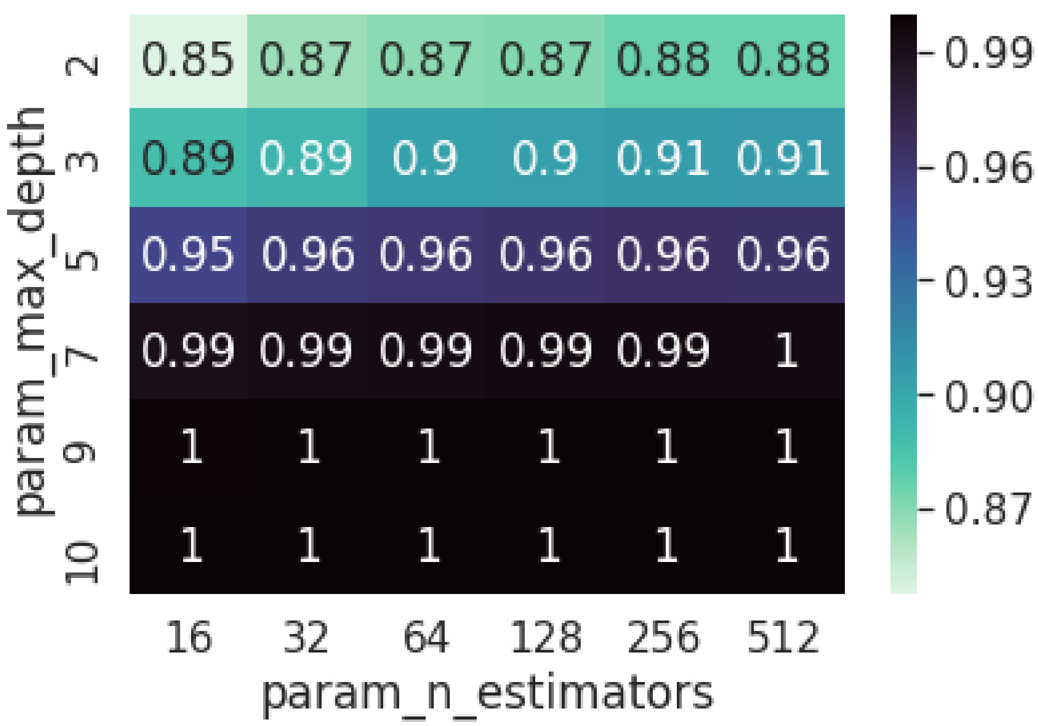
6.1.5 [5.1.5] Applying Random Forests on AVG W2V, SET 3

```
In [72]: # Please write all the code with proper documentation

In [73]: X = np.array(final['Text_Summary'])
         y = np.array(final['Score'])
         data_split(X,y)
         X_train = frompicklefile('X_train')
         X_test = frompicklefile('X_test')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
         count_vect = apply_vectorizers_train_test('AvgW2V', X_train, X_test)

100%|| 70000/70000 [12:42<00:00, 91.76it/s]
100%|| 30000/30000 [05:30<00:00, 90.82it/s]


'train_vect' and 'test_vect' are the pickle files.


In [74]: train_vect = frompicklefile('train_vect')
         test_vect = frompicklefile('test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')

In [75]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in

         tree_max_depth = [2, 3, 5, 7, 9, 10]
         estimators = [16, 32, 64, 128, 256, 512]
```

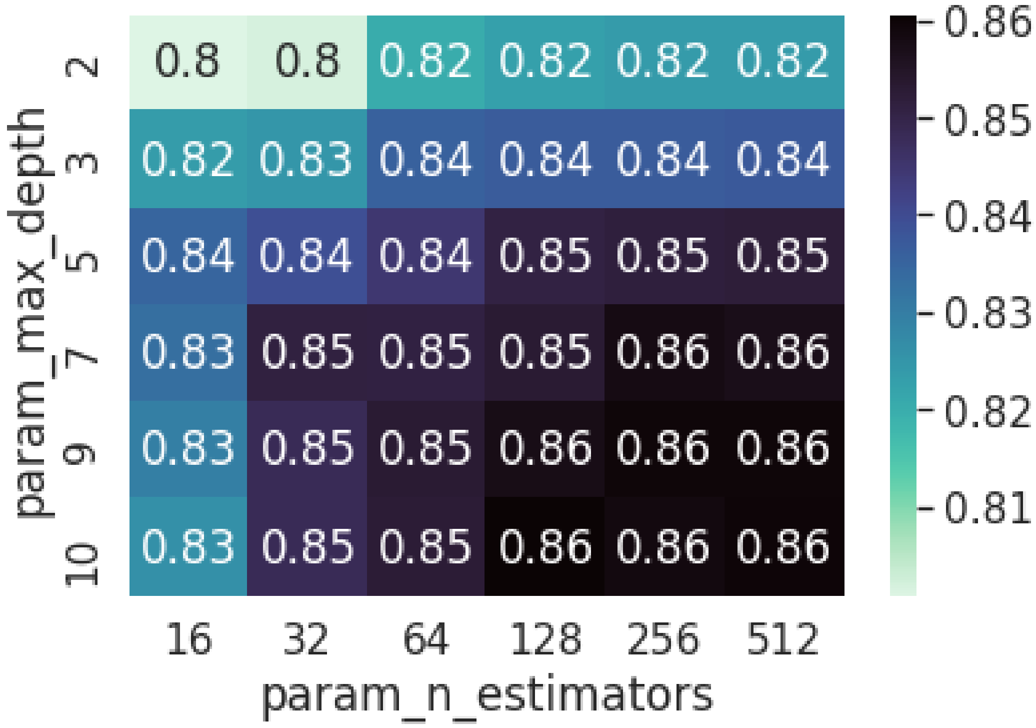
```
parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

cv_results, avgw2v_max_depth_optimal1, avgw2v_n_estimators_optimal1 = applying_rf_xgb

print('avgw2v_max_depth_optimal1, avgw2v_n_estimators_optimal1 :',avgw2v_max_depth_op
```

```
avgw2v_max_depth_optimal1, avgw2v_n_estimators_optimal1 : 10 128
```

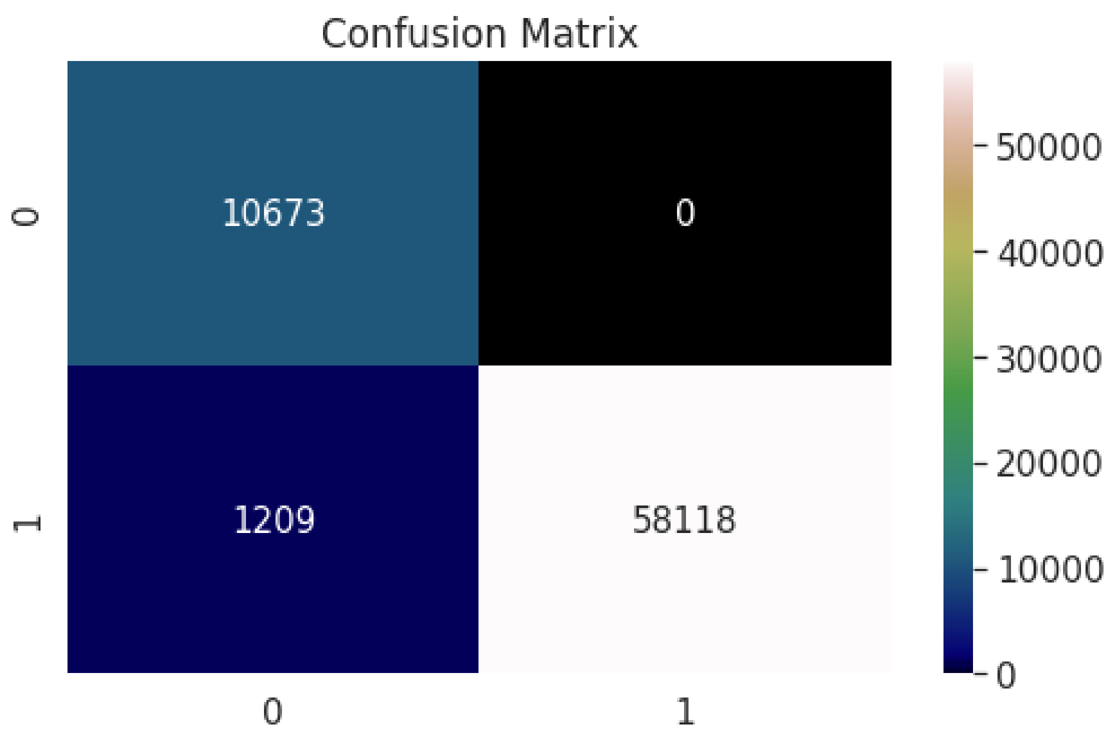
In [76]:
```
# print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']])
train_cv_error_plot(cv_results, 'mean_train_score')
```



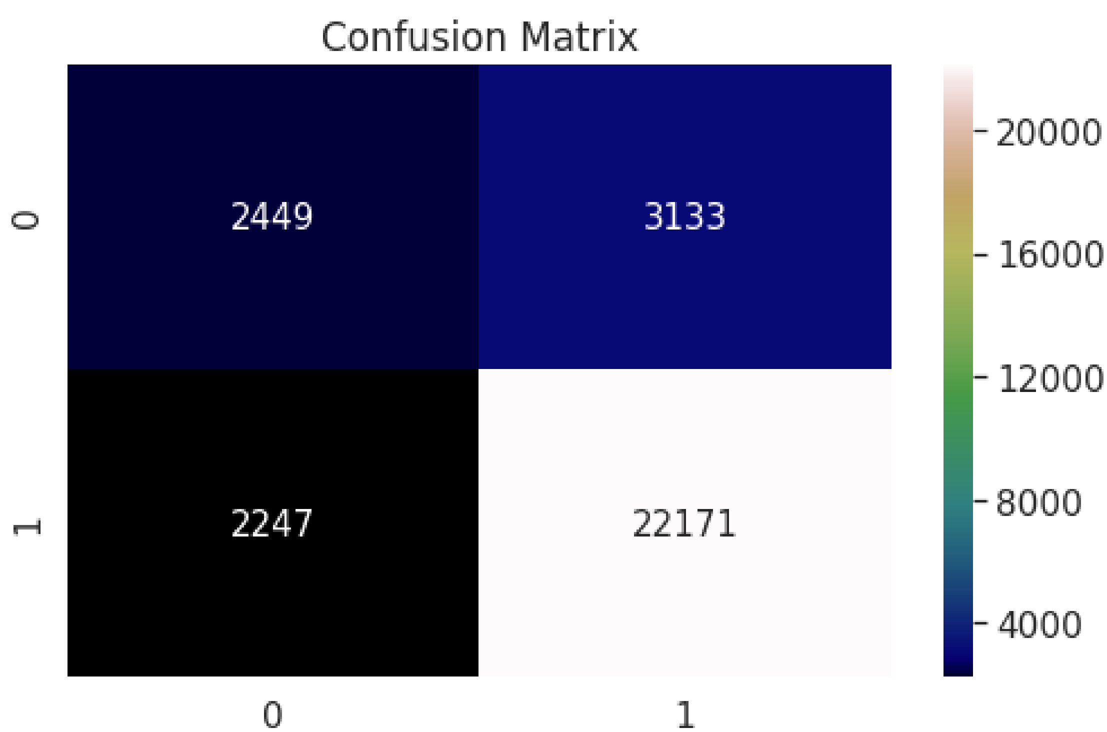
In [77]:
```
# print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
train_cv_error_plot(cv_results, 'mean_test_score')
```



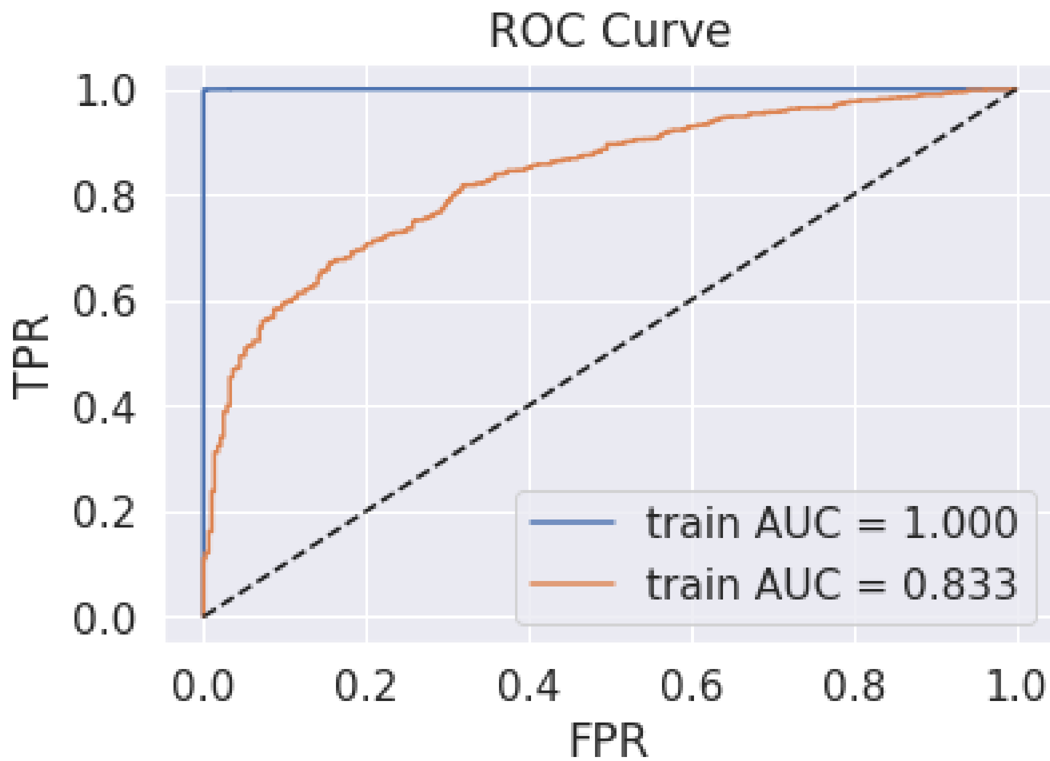
In [78]:
```
rf_optimal = rf_xgb_optimal('RandomForest', avgw2v_max_depth_optimal1, avgw2v_n_estima

cm_fig(rf_optimal, y_train, train_vect)
```



In [79]: cm_fig(rf_optimal, y_test, test_vect)



```
In [80]: avgw2v_auc1 = error_plot(rf_optimal, train_vect, y_train, test_vect, y_test)
```



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

```
In [81]: # Please write all the code with proper documentation
```

```
In [82]: X = np.array(final['Text_Summary'])
         y = np.array(final['Score'])
         data_split(X,y)
         X_train = frompicklefile('X_train')
         X_test = frompicklefile('X_test')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
         count_vect = apply_vectorizers_train_test('TF-IDF W2V', X_train, X_test)
```

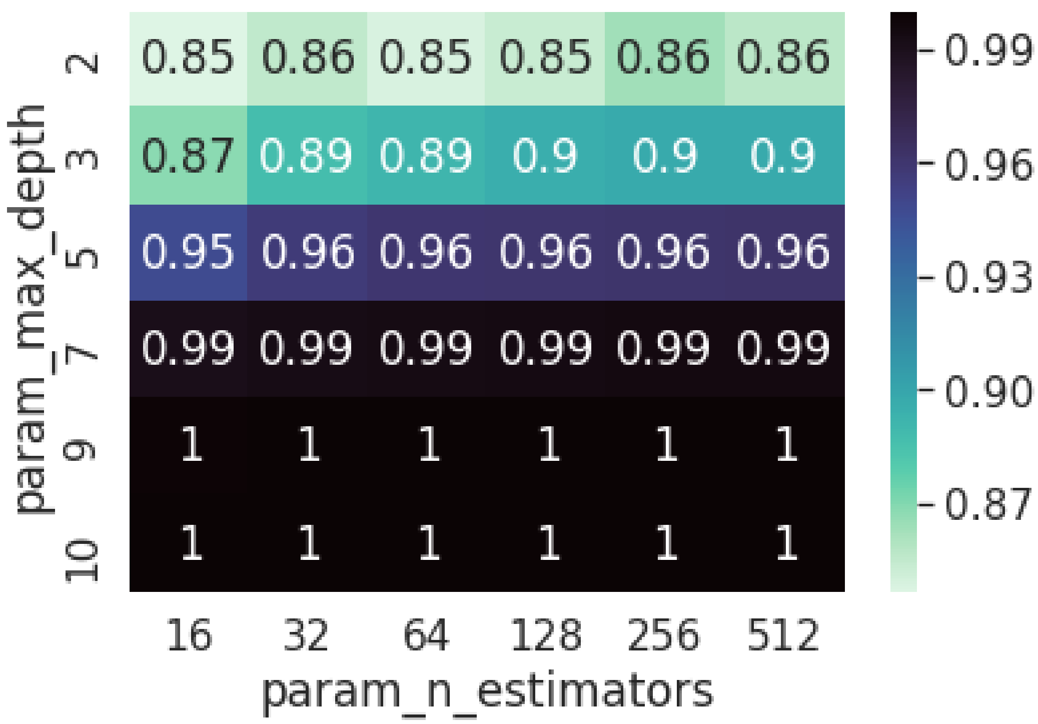
```
100%|| 70000/70000 [23:17<00:00, 50.10it/s]
100%|| 30000/30000 [09:35<00:00, 52.09it/s]


'train_vect' and 'test_vect' are the pickle files.
```

```
In [83]: train_vect = frompicklefile('train_vect')
         test_vect = frompicklefile('test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
```

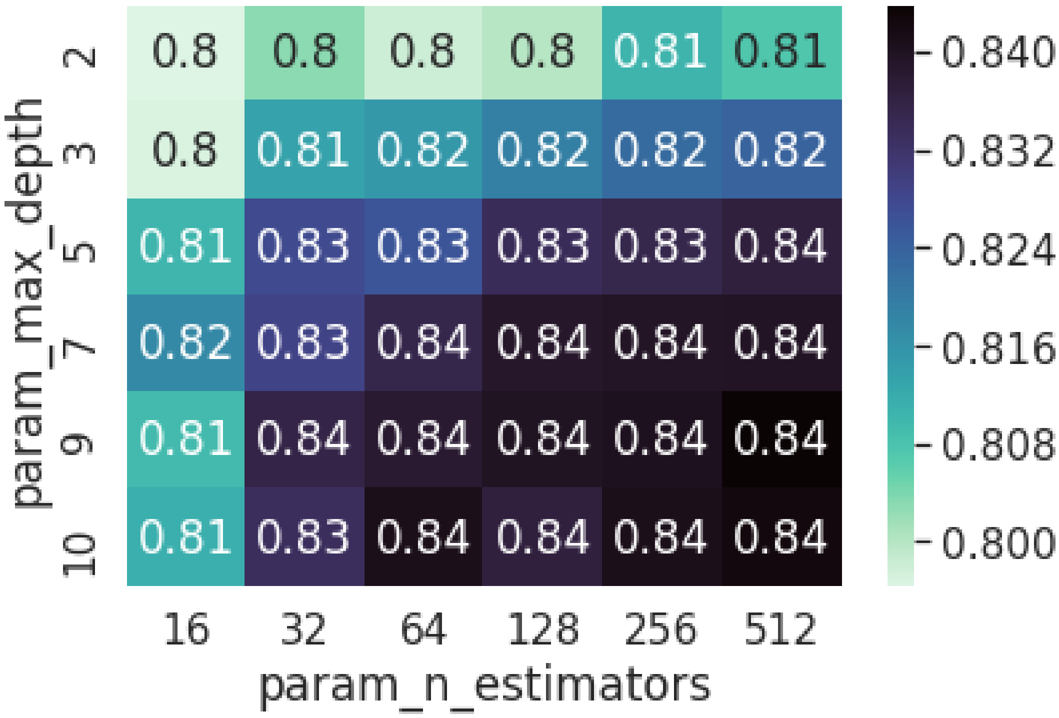
```
In [84]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in

         tree_max_depth = [2, 3, 5, 7, 9, 10]
         estimators = [16, 32, 64, 128, 256, 512]

         parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

         cv_results, tfidfw2v_max_depth_optimal1, tfidfw2v_n_estimators_optimal1 = applying_rf_

         print('tfidfw2v_max_depth_optimal1, tfidfw2v_n_estimators_optimal1 :',tfidfw2v_max_dep
```

```
tfidfw2v_max_depth_optimal1, tfidfw2v_n_estimators_optimal1 : 9 512
```

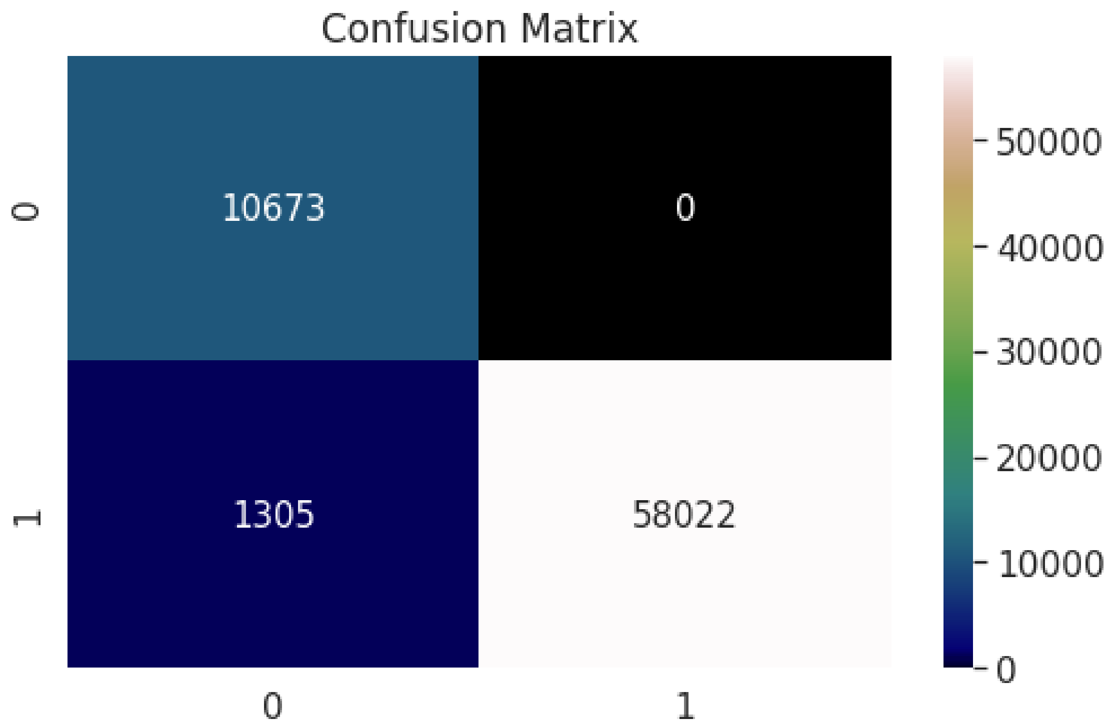
```
In [85]: # print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_train_score')
```



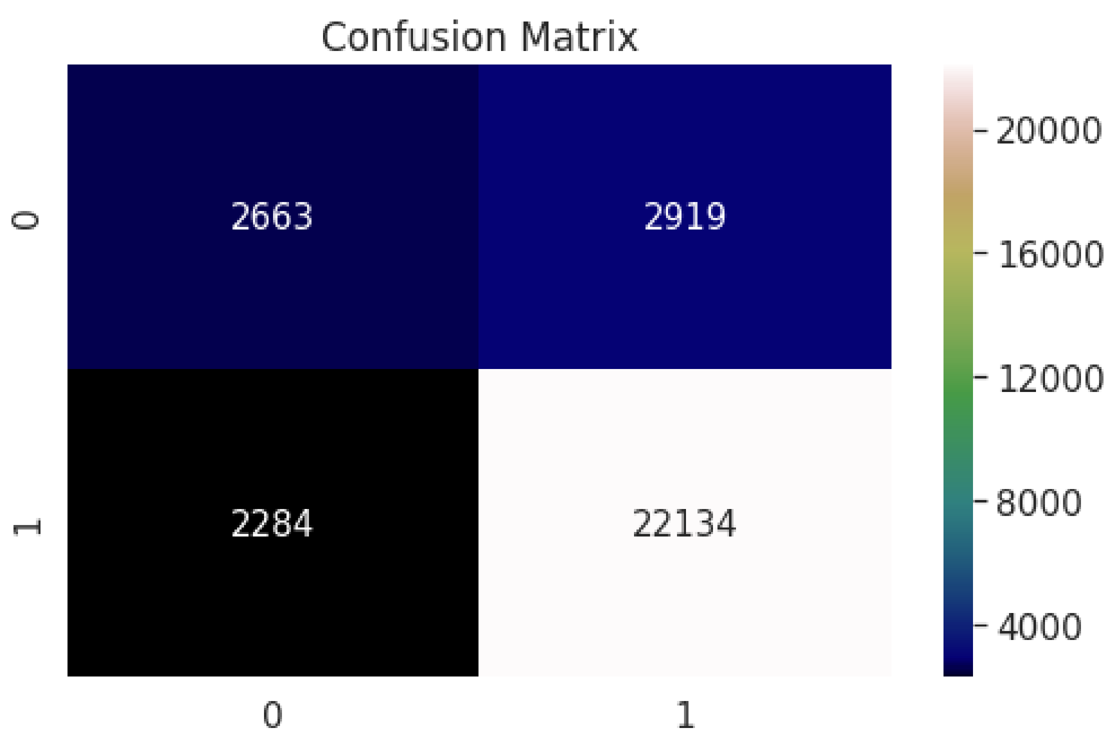

```
In [86]: # print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_test_score')
```



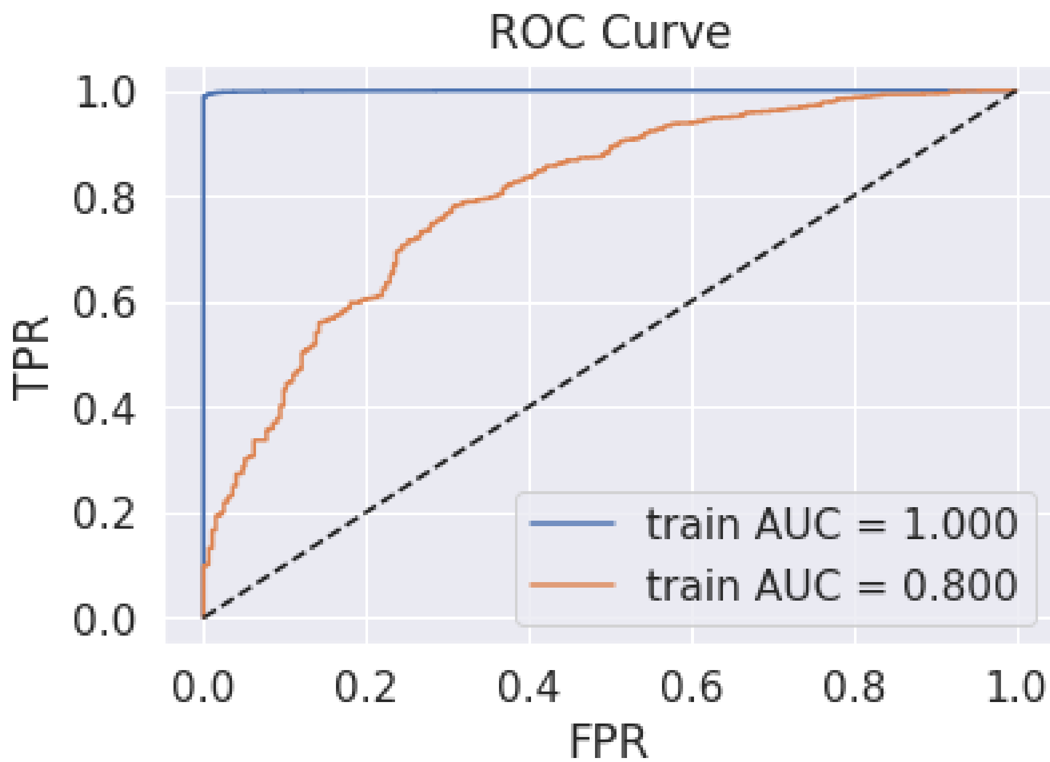

```
In [87]: rf_optimal = rf_xgb_optimal('RandomForest', tfidfw2v_max_depth_optimal1, tfidfw2v_n_es

         cm_fig(rf_optimal, y_train, train_vect)
```



In [88]: `cm_fig(rf_optimal, y_test, test_vect)`



```
In [89]: tfidfw2v_auc1 = error_plot(rf_optimal, train_vect, y_train, test_vect, y_test)
```



6.2 [5.2] Applying GBDT using XGBOOST

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

```
In [90]: # Please write all the code with proper documentation
```

```
In [91]: X = np.array(final['Text_Summary'])
         y = np.array(final['Score'])
         data_split(X,y)
         X_train = frompicklefile('X_train')
         X_test = frompicklefile('X_test')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
         count_vect = apply_vectorizers_train_test('BOW', X_train, X_test)
```

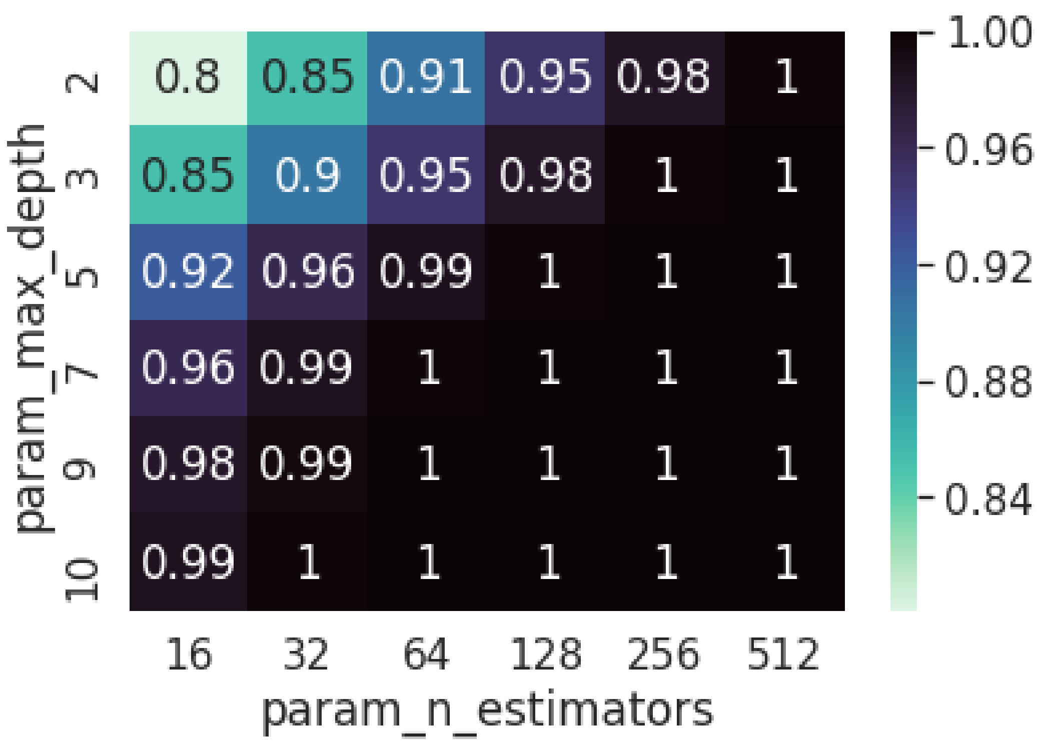
```
'train_vect' and 'test_vect' are the pickle files.
```

```
In [92]: train_vect = frompicklefile('train_vect')
         test_vect = frompicklefile('test_vect')
         y_train = frompicklefile('y_train')
         y_test = frompicklefile('y_test')
```

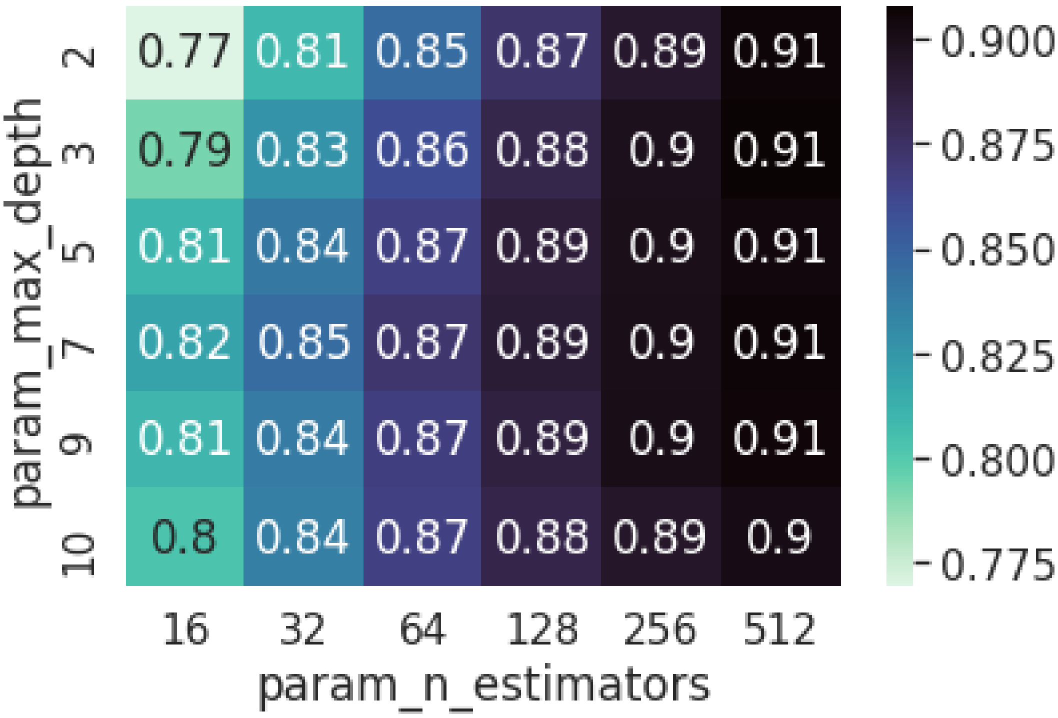
```
In [93]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in

         tree_max_depth = [2, 3, 5, 7, 9, 10]
         estimators = [16, 32, 64, 128, 256, 512]

         parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

         cv_results, bow_max_depth_optimal2, bow_n_estimators_optimal2 = applying_rf_xgb('XGBoo

         print('bow_max_depth_optimal2, bow_n_estimators_optimal2 :',bow_max_depth_optimal2, bo
```

```
bow_max_depth_optimal2, bow_n_estimators_optimal2 : 3 512
```

```
In [94]: # print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_train_score')
```

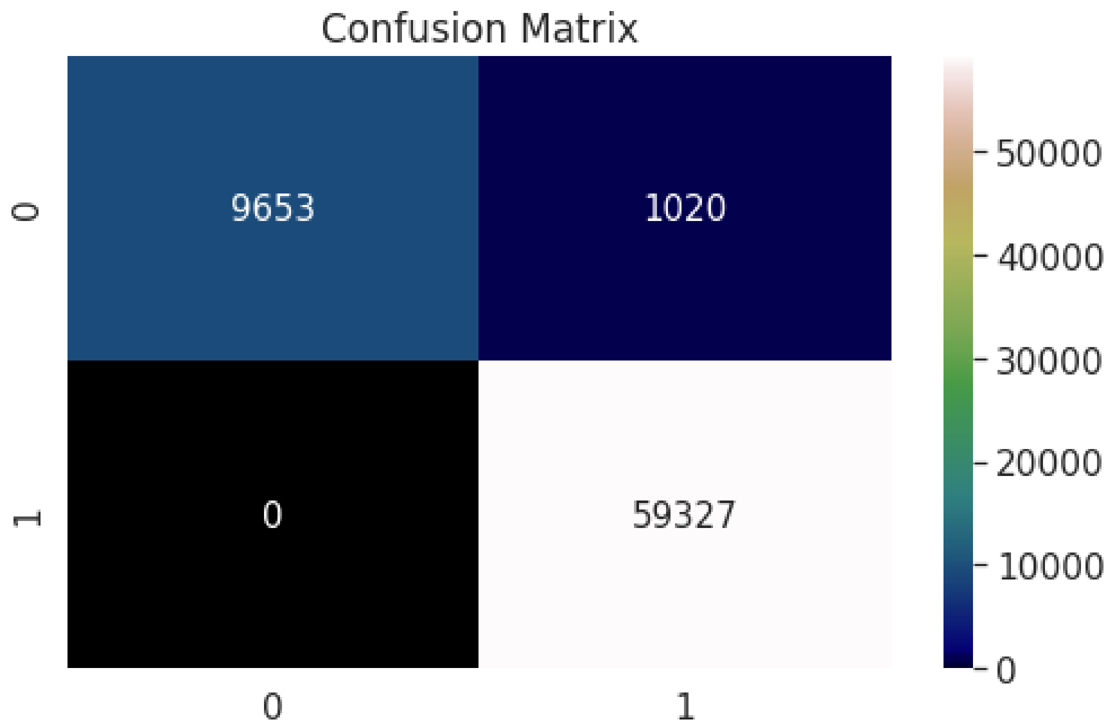



```
In [95]: # print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
         train_cv_error_plot(cv_results, 'mean_test_score')
```

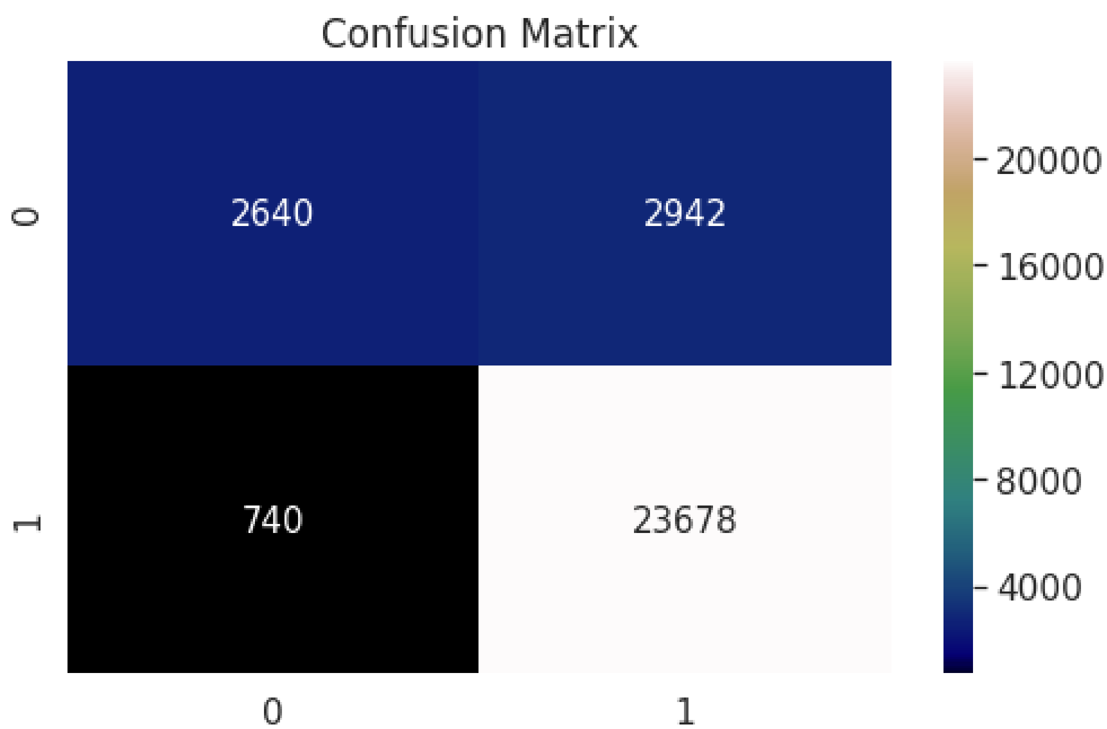



```
In [96]: xgb_optimal = rf_xgb_optimal('XGBoost', bow_max_depth_optimal2, bow_n_estimators_optim

         cm_fig(xgb_optimal, y_train, train_vect)
```

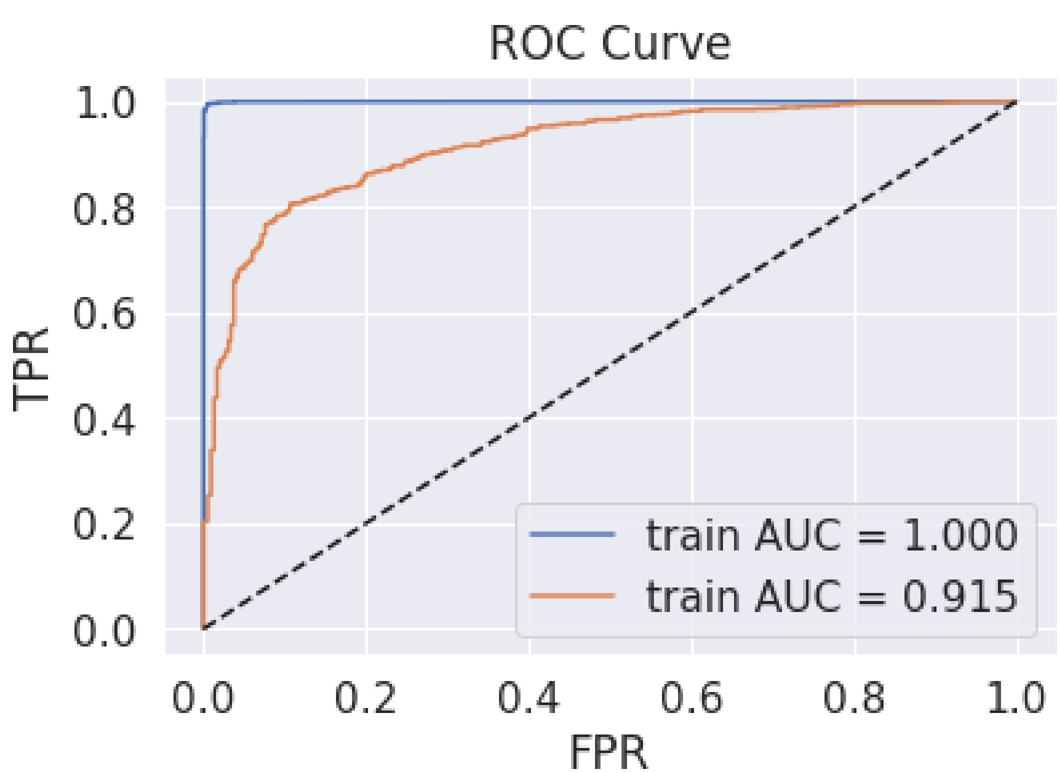
```
/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```



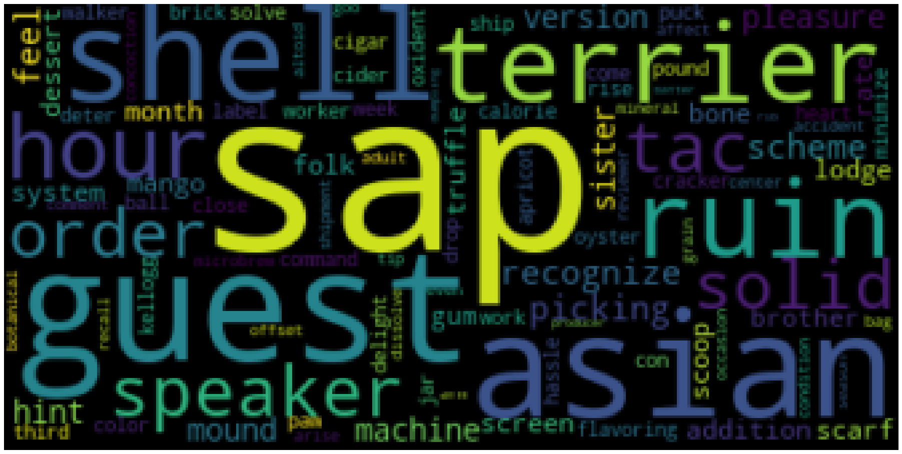

```
In [97]: cm_fig(xgb_optimal, y_test, test_vect)

/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```




```
In [98]: bow_auc2 = error_plot(xgb_optimal, train_vect, y_train, test_vect, y_test)
```



```
In [99]: get_features_top(count_vect, xgb_optimal)
```



6.2.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

```
In [100]: # Please write all the code with proper documentation
```

```
In [101]: X = np.array(final['Text_Summary'])
          y = np.array(final['Score'])
          data_split(X,y)
          X_train = frompicklefile('X_train')
          X_test = frompicklefile('X_test')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
          count_vect = apply_vectorizers_train_test('TF-IDF', X_train, X_test)
```

```
'train_vect' and 'test_vect' are the pickle files.
```

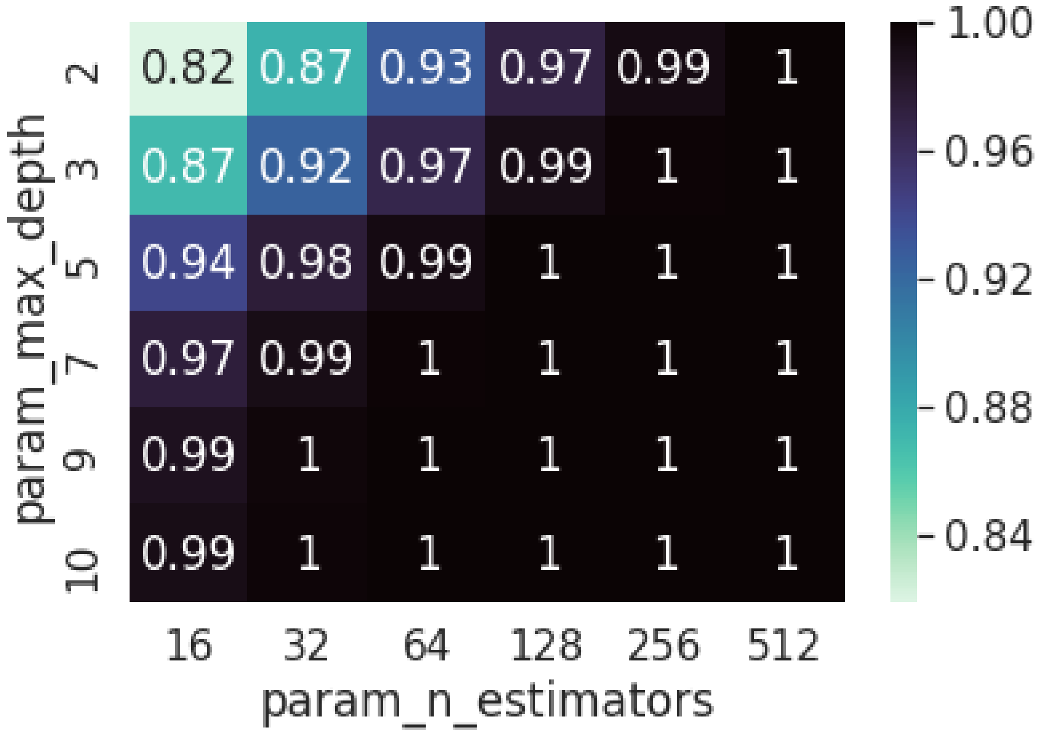
```
In [102]: train_vect = frompicklefile('train_vect')
          test_vect = frompicklefile('test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

```
In [103]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` i

          tree_max_depth = [2, 3, 5, 7, 9, 10]
```

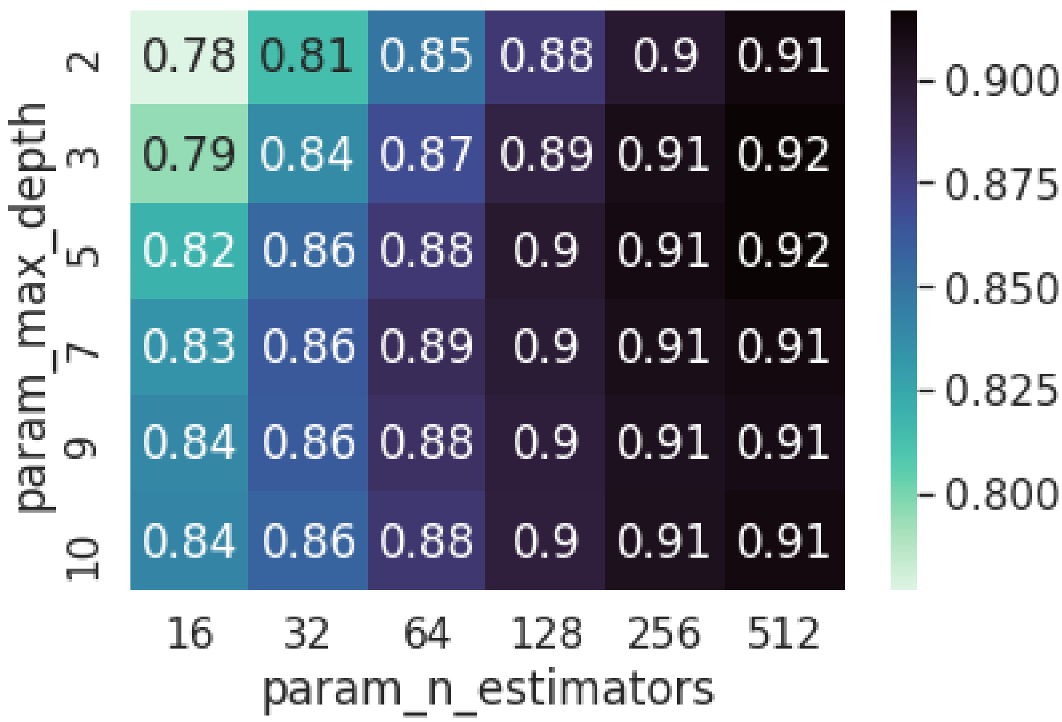
```
estimators = [16, 32, 64, 128, 256, 512]

parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

cv_results, tfidf_max_depth_optimal2, tfidf_n_estimators_optimal2 = applying_rf_xgb(

print('tfidf_max_depth_optimal2, tfidf_n_estimators_optimal2 :',tfidf_max_depth_opti
```

```
tfidf_max_depth_optimal2, tfidf_n_estimators_optimal2 : 3 512
```

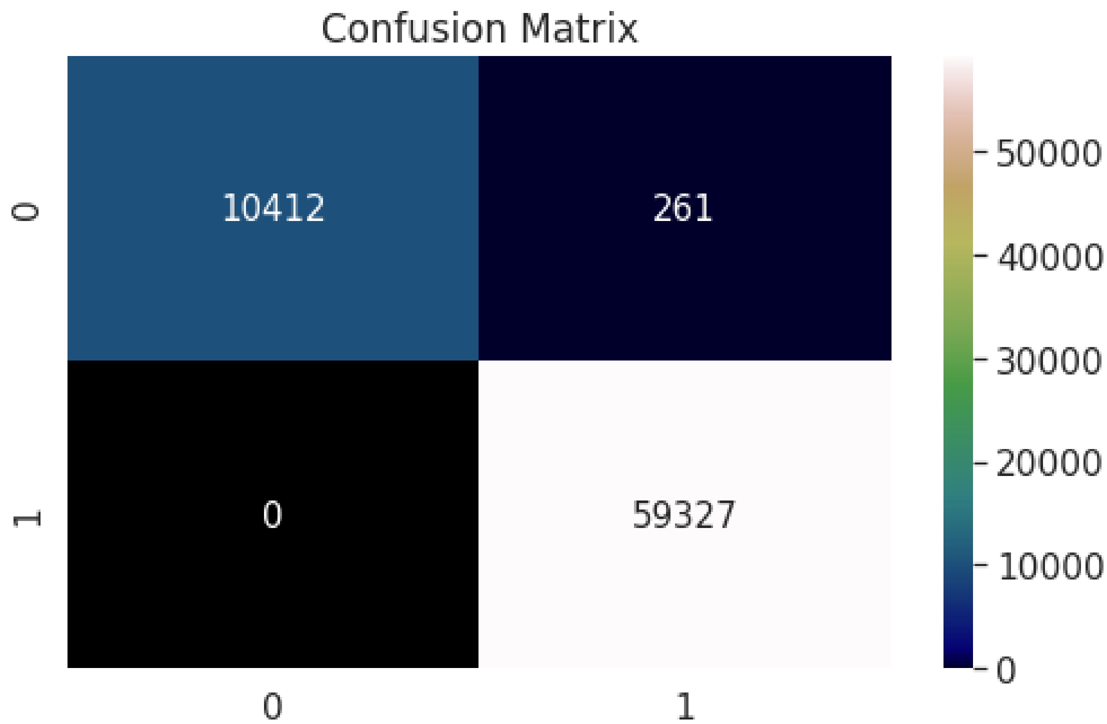
In [104]:
```
# print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']].
train_cv_error_plot(cv_results, 'mean_train_score')
```



In [105]:
```
# print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
train_cv_error_plot(cv_results, 'mean_test_score')
```

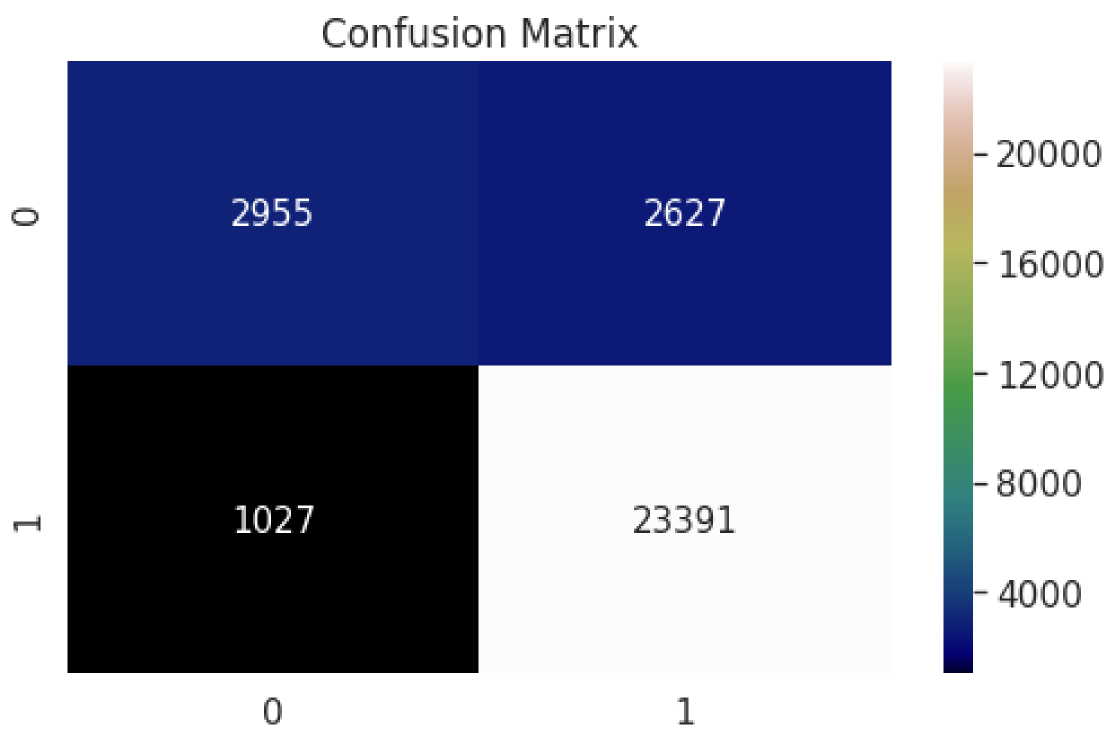



```
In [106]: xgb_optimal = rf_xgb_optimal('XGBoost', tfidf_max_depth_optimal2, tfidf_n_estimators_

          cm_fig(xgb_optimal, y_train, train_vect)

/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```

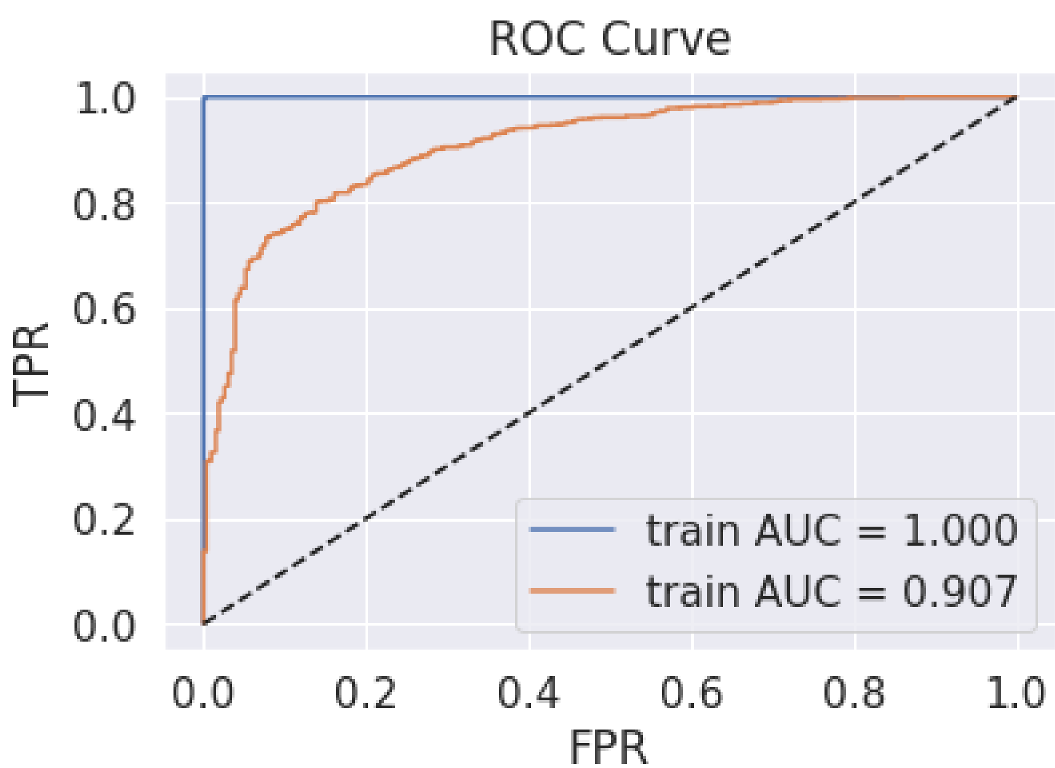



```
In [107]: cm_fig(xgb_optimal, y_test, test_vect)
```

```
/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```




```
In [108]: tfidf_auc2 = error_plot(xgb_optimal, train_vect, y_train, test_vect, y_test)
```



```
In [109]: get_features_top(count_vect, xgb_optimal)
```

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

```
In [110]: # Please write all the code with proper documentation
```

```
In [111]: X = np.array(final['Text_Summary'])
          y = np.array(final['Score'])
          data_split(X,y)
          X_train = frompicklefile('X_train')
          X_test = frompicklefile('X_test')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
          count_vect = apply_vectorizers_train_test('AvgW2V', X_train, X_test)
```

```
100%|| 70000/70000 [13:23<00:00, 87.11it/s]
100%|| 30000/30000 [05:44<00:00, 87.18it/s]
```

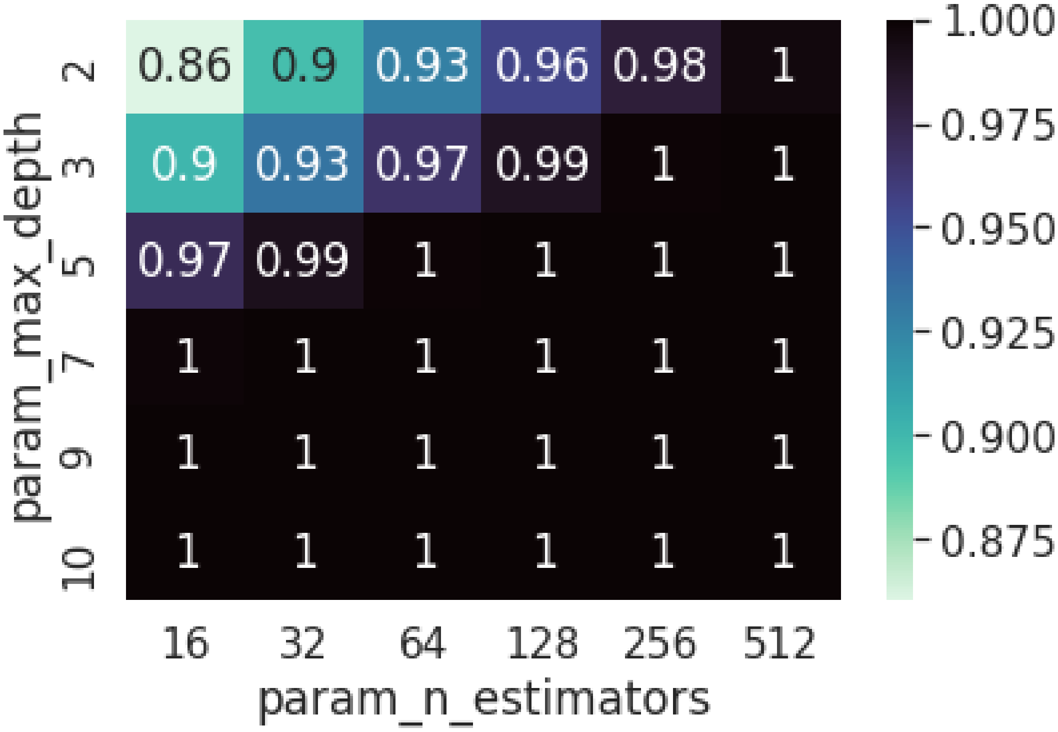
```
'train_vect' and 'test_vect' are the pickle files.
```

```
In [112]: train_vect = frompicklefile('train_vect')
          test_vect = frompicklefile('test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

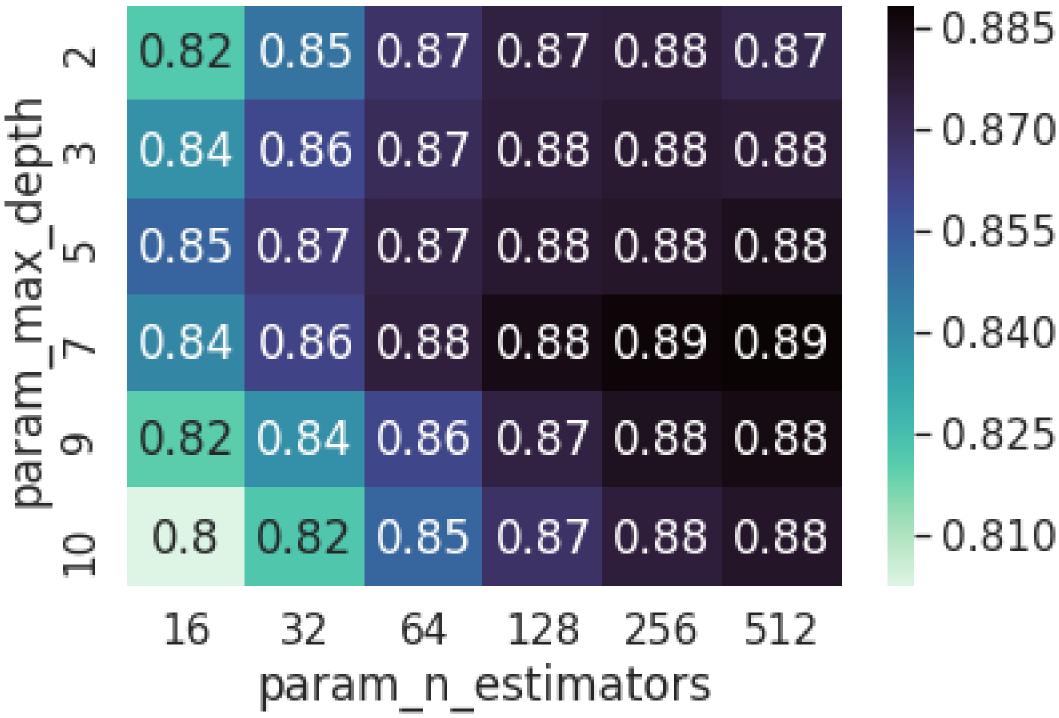
```
In [113]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` i

          tree_max_depth = [2, 3, 5, 7, 9, 10]
          estimators = [16, 32, 64, 128, 256, 512]

          parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

          cv_results, avgw2v_max_depth_optimal2, avgw2v_n_estimators_optimal2 = applying_rf_xgb

          print('avgw2v_max_depth_optimal2, avgw2v_n_estimators_optimal2 :',avgw2v_max_depth_op
```

```
avgw2v_max_depth_optimal2, avgw2v_n_estimators_optimal2 : 7 512
```

```
In [114]: # print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']].
          train_cv_error_plot(cv_results, 'mean_train_score')
```

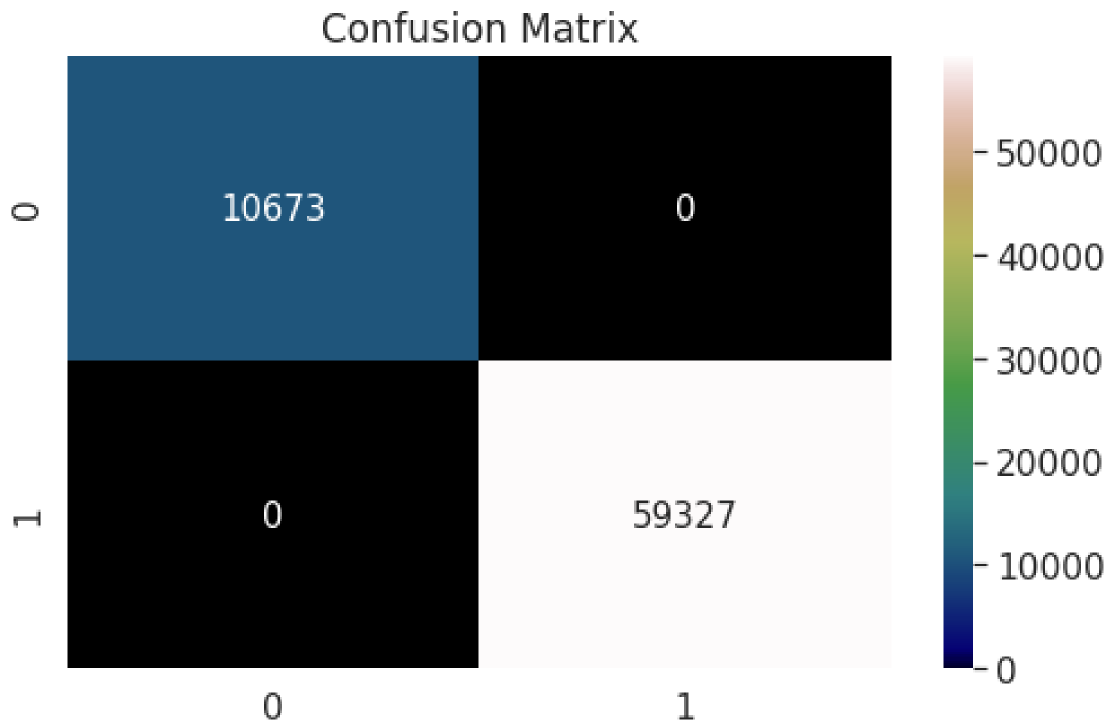



```
In [115]: # print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
          train_cv_error_plot(cv_results, 'mean_test_score')
```

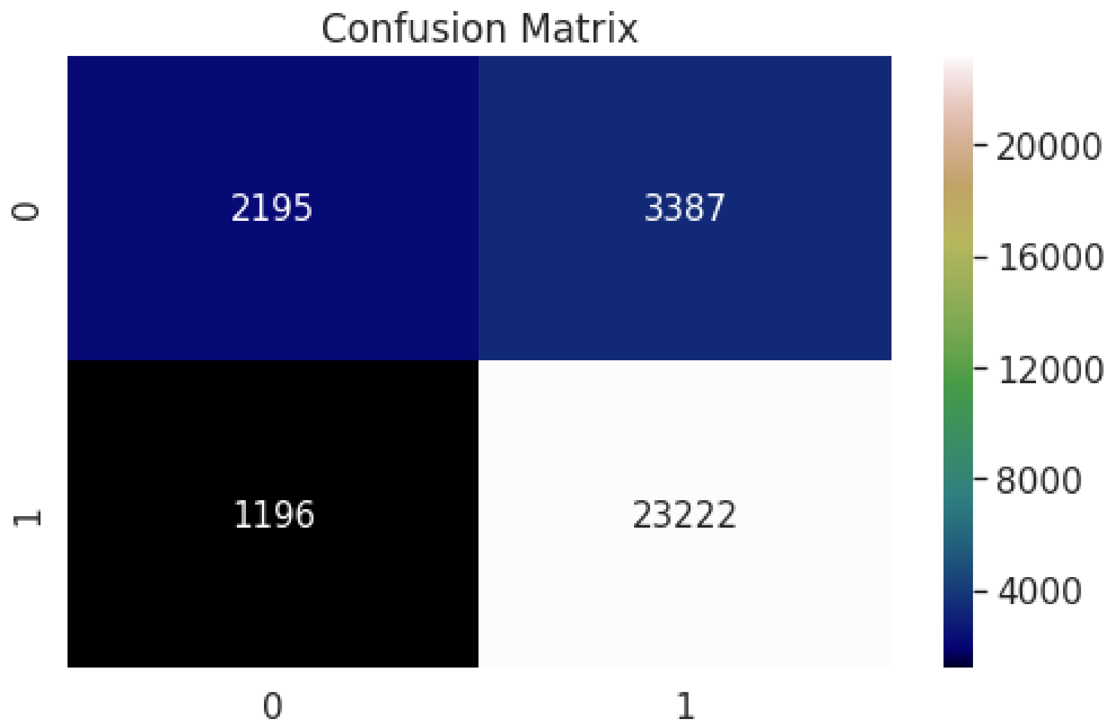



```
In [116]: xgb_optimal = rf_xgb_optimal('XGBoost', avgw2v_max_depth_optimal2, avgw2v_n_estimato

          cm_fig(xgb_optimal, y_train, train_vect)
```

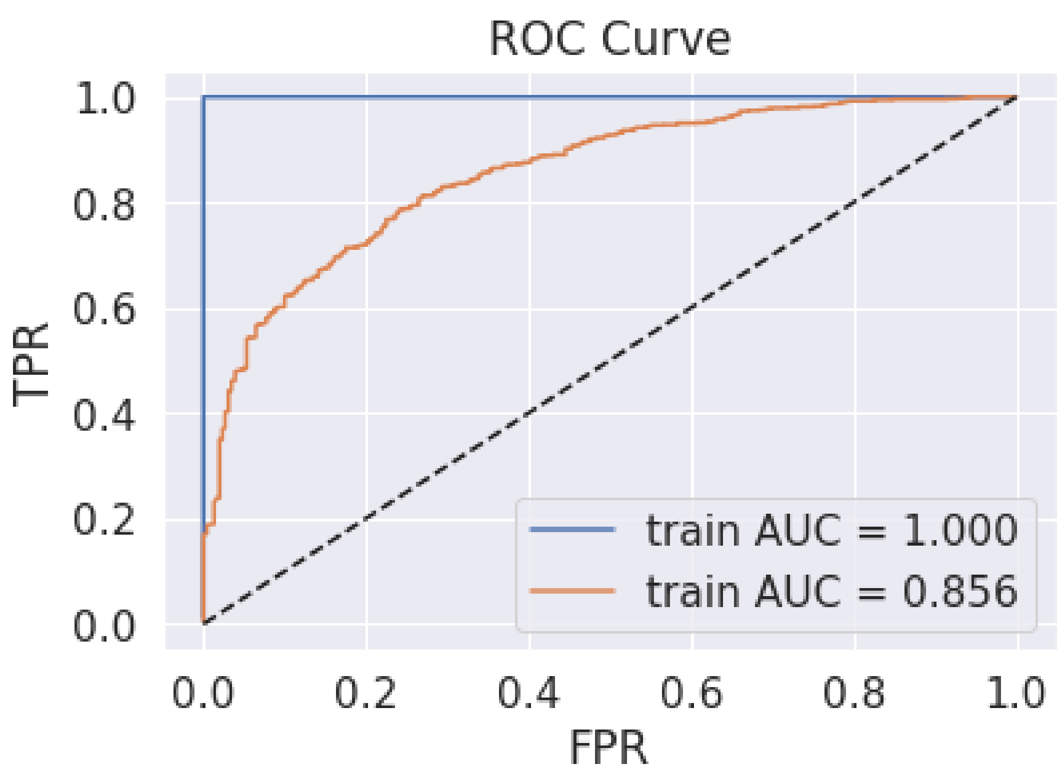
```
/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```




```
In [117]: cm_fig(xgb_optimal, y_test, test_vect)

/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```




```
In [118]: avgw2v_auc2 = error_plot(xgb_optimal, np.array(train_vect), y_train, np.array(test_ve
```



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

```
In [119]: # Please write all the code with proper documentation
```

```
In [120]: X = np.array(final['Text_Summary'])
          y = np.array(final['Score'])
          data_split(X,y)
          X_train = frompicklefile('X_train')
          X_test = frompicklefile('X_test')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
          count_vect = apply_vectorizers_train_test('TF-IDF W2V', X_train, X_test)
```

```
100%|| 70000/70000 [22:54<00:00, 50.94it/s]
100%|| 30000/30000 [09:18<00:00, 53.67it/s]
```

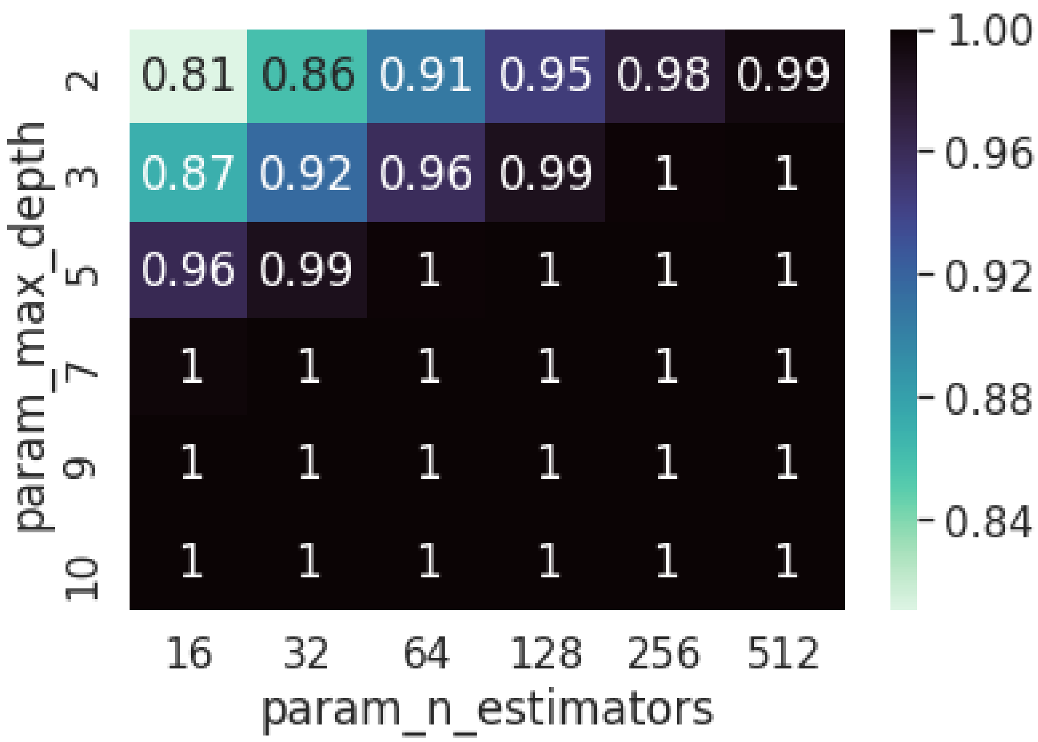
```
'train_vect' and 'test_vect' are the pickle files.
```

```
In [121]: train_vect = frompicklefile('train_vect')
          test_vect = frompicklefile('test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

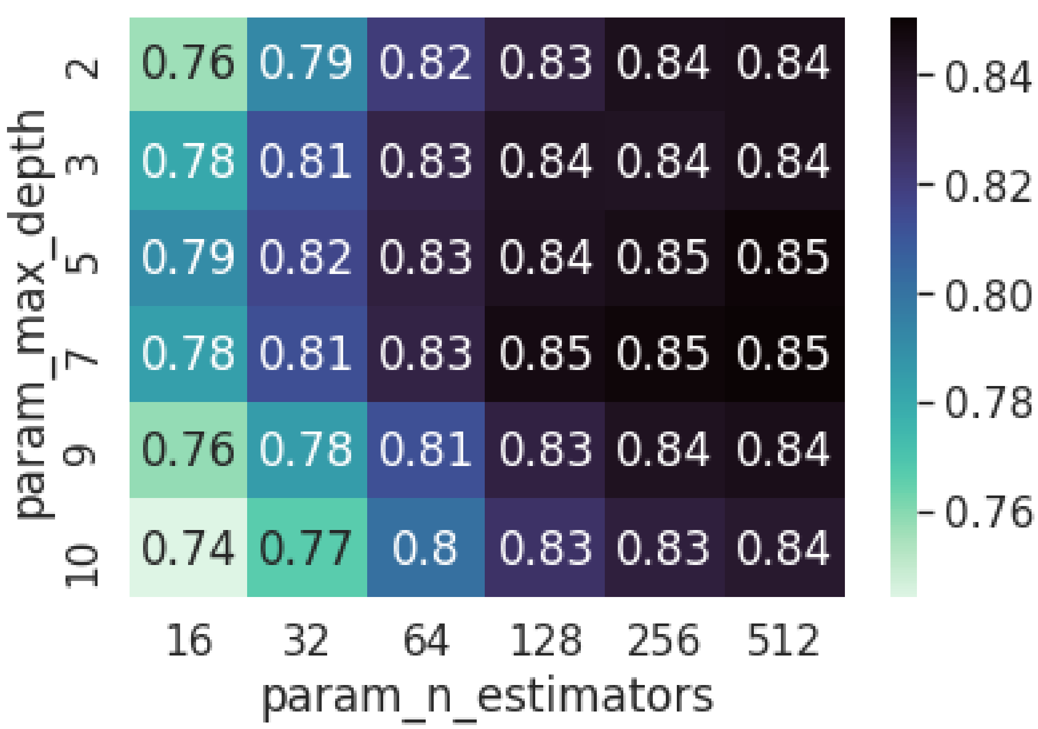
```
In [122]: # `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` i

          tree_max_depth = [2, 3, 5, 7, 9, 10]
          estimators = [16, 32, 64, 128, 256, 512]

          parameters = {'max_depth':tree_max_depth, 'n_estimators':estimators}

          cv_results, tfidfw2v_max_depth_optimal2, tfidfw2v_n_estimators_optimal2 = applying_r

          print('tfidfw2v_max_depth_optimal2, tfidfw2v_n_estimators_optimal2 :',tfidfw2v_max_de
```

```
tfidfw2v_max_depth_optimal2, tfidfw2v_n_estimators_optimal2 : 7 512
```

```
In [123]: # print(cv_results[['mean_train_score','param_max_depth','param_min_samples_split']]
          train_cv_error_plot(cv_results, 'mean_train_score')
```

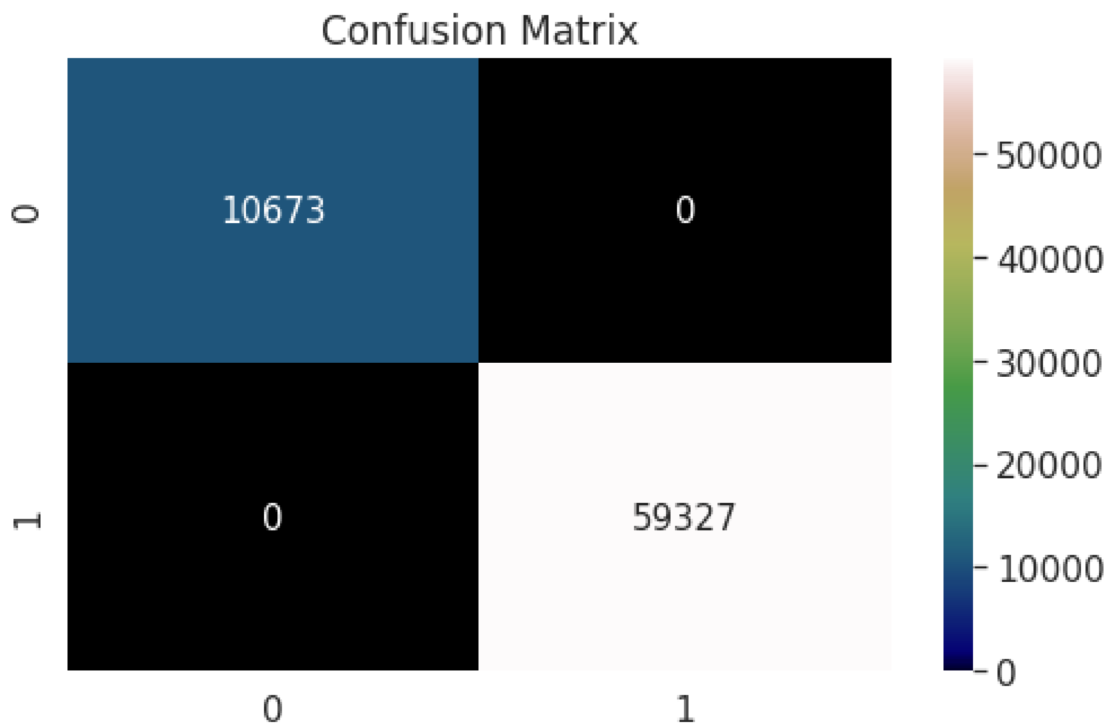


In [124]:
```
# print(cv_results[['mean_test_score','param_max_depth','param_min_samples_split']])
train_cv_error_plot(cv_results, 'mean_test_score')
```



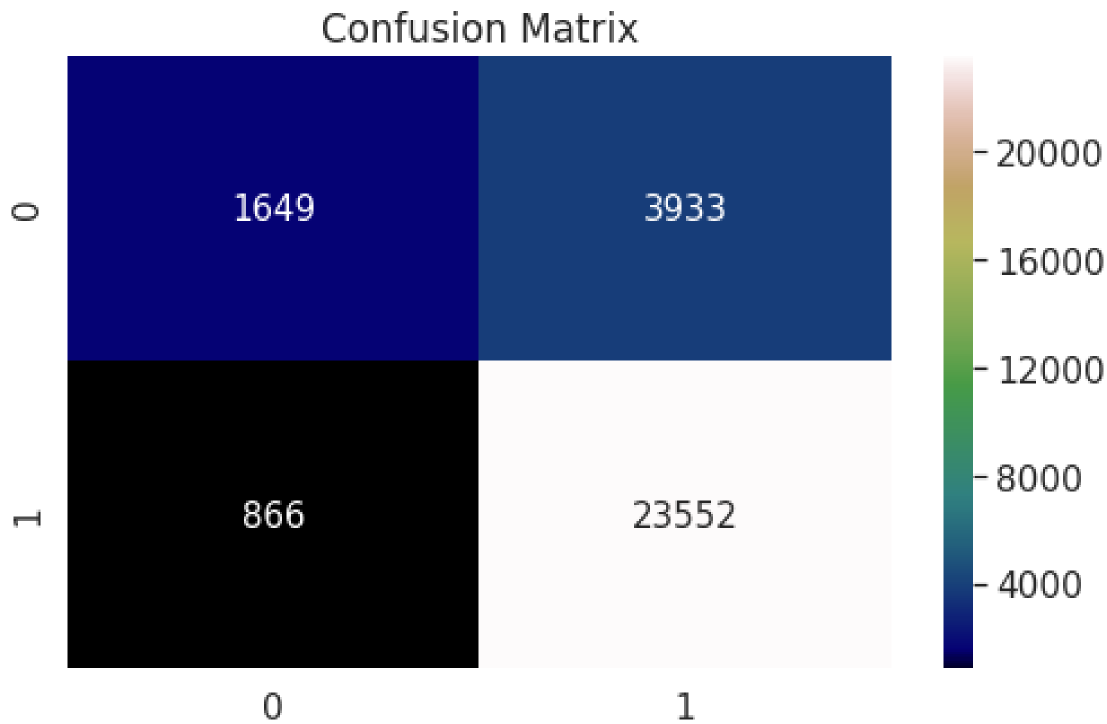
In [125]:
```
xgb_optimal = rf_xgb_optimal('XGBoost', tfidfw2v_max_depth_optimal2, tfidfw2v_n_esti

cm_fig(xgb_optimal, y_train, train_vect)
```

```
/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```

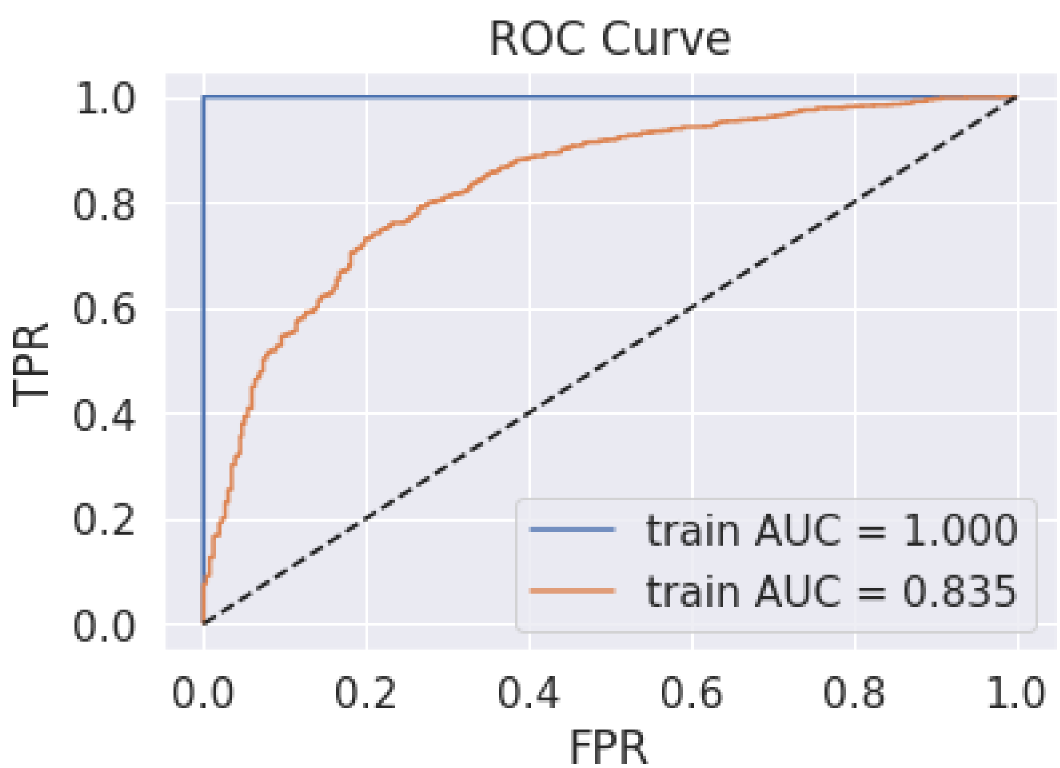


In [126]:
```
cm_fig(xgb_optimal, y_test, np.array(test_vect))
```

```
/usr/local/lib/python3.5/site-packages/sklearn/preprocessing/label.py:151: DeprecationWarning:
  if diff:
```




```
In [127]: tfidfw2v_auc2 = error_plot(xgb_optimal,np.array(train_vect), y_train, np.array(test_v
```



7 [6] Conclusions

```
In [128]: # Please compare all your models using Prettytable library
```

```
In [129]: from prettytable import PrettyTable

          model_metric = PrettyTable()

          model_metric = PrettyTable(["Vectorizer", "Model Name", "Hyperparameter- Max Depth",

          model_metric.add_row(["Bag of Words", "Random Forest", bow_max_depth_optimal1, bow_n_
          model_metric.add_row(["TF-IDF", "Random Forest", tfidf_max_depth_optimal1, tfidf_n_es
          model_metric.add_row(["Avg W2V", "Random Forest", avgw2v_max_depth_optimal1, avgw2v_n
          model_metric.add_row(["TF-IDF W2V", "Random Forest", tfidfw2v_max_depth_optimal1, tf
          model_metric.add_row(["Bag of Words", "XGBoost", bow_max_depth_optimal2, bow_n_estima
          model_metric.add_row(["TF-IDF", "XGBoost", tfidf_max_depth_optimal2, tfidf_n_estimato
          model_metric.add_row(["Avg W2V","XGBoost", avgw2v_max_depth_optimal2, avgw2v_n_estima
          model_metric.add_row(["TF-IDF W2V", "XGBoost", tfidfw2v_max_depth_optimal2, tfidfw2v_

          print(model_metric.get_string(start=0, end=8))
```

```
+--------------+---------------+---------------------------+-----------------------------+----
|  Vectorizer  |   Model Name  | Hyperparameter- Max Depth | Hyperparameter- n_estimators |
+--------------+---------------+---------------------------+-----------------------------+----
| Bag of Words | Random Forest |             10            |             512             | 0.9
|    TF-IDF    | Random Forest |             9             |             512             | 0.9
|   Avg W2V    | Random Forest |             10            |             128             | 0.8
|  TF-IDF W2V  | Random Forest |             9             |             512             | 0.8
| Bag of Words |    XGBoost    |             3             |             512             | 0.9
|    TF-IDF    |    XGBoost    |             3             |             512             | 0.9
|   Avg W2V    |    XGBoost    |             7             |             512             | 0.8
|  TF-IDF W2V  |    XGBoost    |             7             |             512             | 0.8
+--------------+---------------+---------------------------+-----------------------------+----
```

7.1 [6.1] Observations

1) Train Time: It is slighlty higher for both the models 'Random Forest' and 'XGBoost' because of the higher number of base learners.

2) Hyperparameters: Models for all the vectorizers considered the highest value for the Hyperparameters from the GridSearchCV and this tend to slightly overfit the on Train data.

3) Confusion Matrix: For Train data the models slightly overfit as seen in the matrix and they did a very great job on the Test data.

4) AUC Scores: Random Forest and XGBoost are one of the best performing models so far on the data and AUC scores are above 0.85 for all the models.