Assignment 9 - RandomForest, GBDT

April 7, 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",category=DeprecationWarning)


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 import metrics
from sklearn.metrics import roc_curve, auc, f1_score
from nltk.stem.porter import PorterStemmer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix,roc_curve, auc,f1_score
from sklearn.ensemble import RandomForestClassifier

from wordcloud import WordCloud, STOPWORDS
from PIL import Image
from xgboost import XGBClassifier

import re
from bs4 import BeautifulSoup
# 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
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('./Dataset/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 500
        # 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
        def partition(x):
            if x < 3:
                return 0
            return 1

        def findMinorClassPoints(df):
            posCount = int(df[df['Score']==1].shape[0]);
            negCount = int(df[df['Score']==0].shape[0]);
            if negCount < posCount:
                return negCount
            return posCount

        #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

        #Performing Downsampling
        samplingCount = findMinorClassPoints(filtered_data)
        postive_df = filtered_data[filtered_data['Score'] == 1].sample(n=samplingCount)
        negative_df = filtered_data[filtered_data['Score'] == 0].sample(n=samplingCount)

        filtered_data = pd.concat([postive_df, negative_df])

        print("Number of data points in our data", filtered_data.shape)
        filtered_data.head(3)
```

```
Number of data points in our data (164074, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	\
34861	37923	B000F6SNPS	A10Q55Z18SU2PL	Cheryl C Nims	
123518	133965	B0046HC0VA	A2ZY88G1ZNCU18	Zap	
176041	190910	B001NY01C4	A2JR240CG4YOZS	Avid Cook "Budding chef"	

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\

```
34861                      3                       3      1  1331251200
123518                     1                       1      1  1331856000
176041                     1                       1      1  1264982400

                                        Summary  \
34861       The artificial flavor controversy
123518  What can I say great to the last one.
176041                          my dogs love'm

                                                    Text
34861   (3/12)  I have purchased this product in the p...
123518  It's the craziest thing I'm a diabetic but I j...
176041  my dogs love the taste of these and they do he...
```

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

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

```
(80668, 7)
```

Out[4]:

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

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	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	HelpfulnessDenominator	Score	Time	Summary	Text
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	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=False
```

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

```
Out[9]: (128340, 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]: 78.22080280848886
```

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()
```

```
(128340, 10)
```

```
Out[13]: 1    71228
         0    57112
         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]: # 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)
```

```
Great book, perfect condition arrived in a short amount of time, long before the expected delive
==================================================
I started using Wilton's dyes when I was thirteen, and a set lasted me until I was twenty-nine b
==================================================
Was disappointed to find out that this tree that I purchased as a gift for someone has died afte
==================================================
My Rottweiler Kirin loves playing with it.  She runs around the house scooting it along like a h
==================================================
```

```
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)
```

```
Great book, perfect condition arrived in a short amount of time, long before the expected delive
```

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

         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)
```

```
Great book, perfect condition arrived in a short amount of time, long before the expected delive
==================================================
I started using Wilton's dyes when I was thirteen, and a set lasted me until I was twenty-nine b
==================================================
Was disappointed to find out that this tree that I purchased as a gift for someone has died afte
==================================================
```

```
My Rottweiler Kirin loves playing with it.  She runs around the house scooting it along like a h
```

```
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)
```

```
Was disappointed to find out that this tree that I purchased as a gift for someone has died afte
==================================================
```

```
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)
```

```
Great book, perfect condition arrived in a short amount of time, long before the expected delive
```

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

```
Was disappointed to find out that this tree that I purchased as a gift for someone has died afte
```

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

```
        stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                    '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', 'ov
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                    'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
        # 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 stopwor
            preprocessed_reviews.append(sentance.strip())
```

```
100%|| 128340/128340 [00:52<00:00, 2428.34it/s]
```

In [23]:
```
        preprocessed_reviews[1500]
```

Out[23]:
```
'disappointed find tree purchased gift someone died continuous care months'
```

[3.2] Preprocessing Review Summary

In [24]:
```
        ## Similartly you can do preprocessing for review summary also.
        def concatenateSummaryWithText(str1, str2):
            return str1 + ' ' + str2

        preprocessed_summary = []
        # tqdm is for printing the status bar
        for sentence in tqdm(final['Summary'].values):
            sentence = re.sub(r"http\S+", "", sentence)
            #sentence = BeautifulSoup(sentence, 'lxml').get_text()
```

```
            sentence = decontracted(sentence)
            sentence = re.sub("\S*\d\S*", "", sentence).strip()
            sentence = re.sub('[^A-Za-z]+', ' ', sentence)
            # https://gist.github.com/sebleier/554280
            sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwor
            preprocessed_summary.append(sentence.strip())

        preprocessed_reviews = list(map(concatenateSummaryWithText, preprocessed_reviews, prepr
        final['CleanedText'] = preprocessed_reviews
        final['CleanedText'] = final['CleanedText'].astype('str')
```

```
100%|| 128340/128340 [00:02<00:00, 46574.41it/s]
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [25]: # #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 [26]: # #bi-gram, tri-gram and n-gram

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

         # # 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
```

5.3 [4.3] TF-IDF

```
In [27]: # 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_name
# print('='*50)

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

5.4 [4.4] Word2Vec

In [28]:
```
# # 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 [29]:
```
# # 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.b
#         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 [30]:
```
# 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 [31]:
```
# # 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 t
#     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 [32]:
```
# # 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 [33]:
```
# # 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 = []; # the tfidf-w2v for each sentence/review is stored in this l
# 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

```
<li><strong>Apply Random Forests & GBDT on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)</
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicour
<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 tas
    </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 enginee
        <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 e
<img src='3d_plot.JPG' width=500px> with X-axis as <strong>n_estimators</strong>, Y-axis as <str
        <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 e
<img src='heat_map.JPG' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.heatm
<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 fin
<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.co
<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 form
    <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

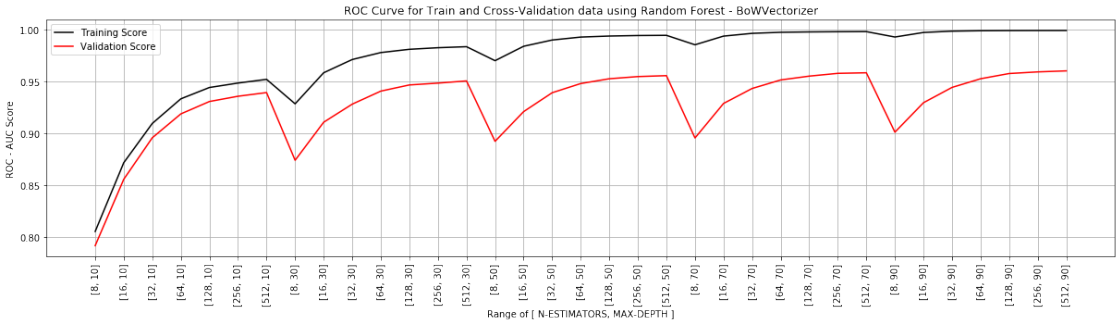
```
In [34]: global result_report
         result_report = pd.DataFrame(columns=['ALGORITHM', 'VECTORIZER', 'DATASET-SIZE', 'N_EST
                                               'MAX_DEPTH (HYPERPARAMETER)', 'F1_SCORE', 'AUC'])

In [35]: #Sorting according to the time for time-based splitting
         final['Time'] = pd.to_datetime(final['Time'], unit='s')
         final = final.sort_values(by='Time', ascending=True)

In [36]: #Using only 100k points
         min_final = final.sample(n=100000)
         x_train, x_test, y_train, y_test = train_test_split(min_final['CleanedText'], min_final

         n_estimator_range =  [2**i for i in range(3, 10)]
         max_depth_range = [i for i in range(10, 100, 20)]
         tot_hyp_length = np.arange(0, len(n_estimator_range) * len(max_depth_range))
```

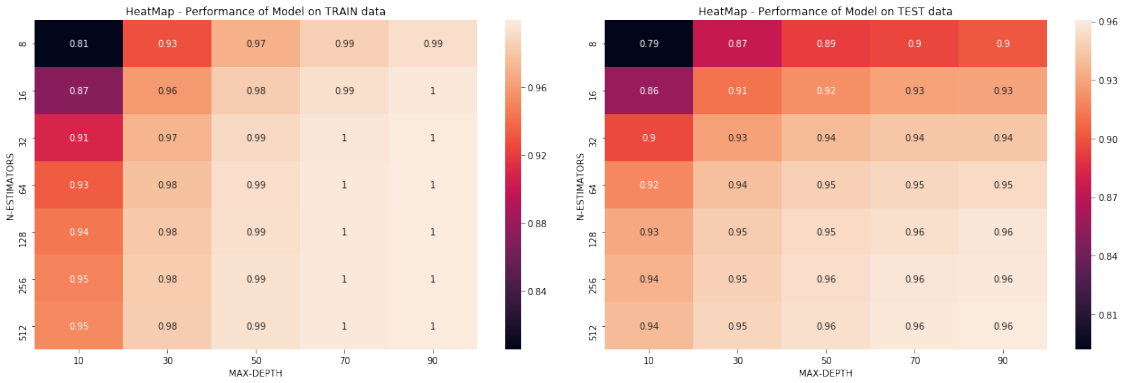
6.1.1 [5.1.1] Applying Random Forests on BOW, SET 1

In [37]:
```
# Applying BOW Vectorizer
bow_model = CountVectorizer()
bow_model.fit(x_train)

x_train_bow = bow_model.transform(x_train)
x_test_bow = bow_model.transform(x_test)
```

In [38]:
```
# Applying RandomForestClassfier using GridSearch CV=10
rfc = RandomForestClassifier(n_jobs=-1)
parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
clf = GridSearchCV(rfc, parameters, cv=10, scoring = 'roc_auc', return_train_score=True
clf.fit(x_train_bow, y_train)

mean_train_score = clf.cv_results_['mean_train_score']
mean_test_score = clf.cv_results_['mean_test_score']

param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using Random Forest - BoWVectc
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
```

```
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```



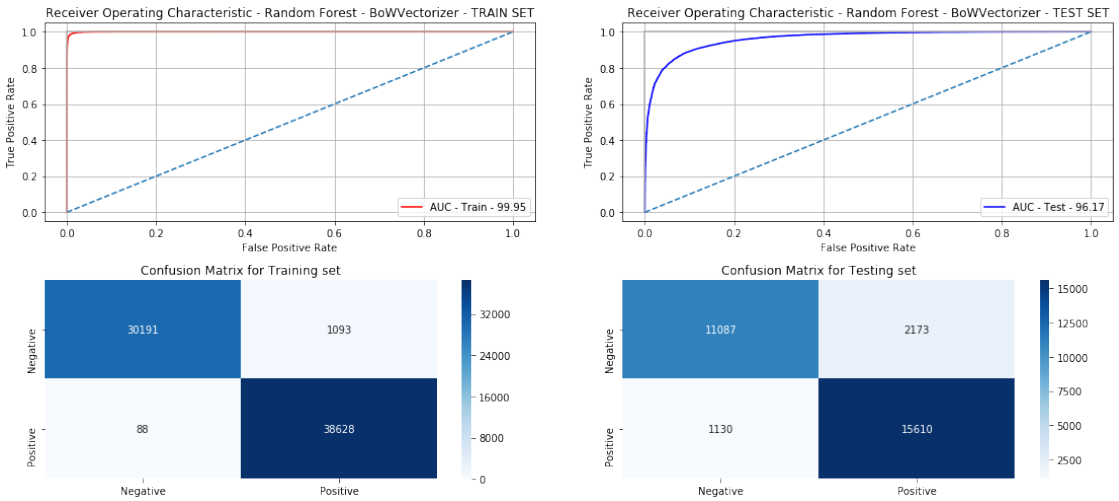


In [39]:
```
optimal_depth = clf.best_params_['max_depth']
optimal_estimator = clf.best_params_['n_estimators']

clf = RandomForestClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n
clf.fit(x_train_bow, y_train)

# Get predicted values for test data
pred_train = clf.predict(x_train_bow)
pred_test = clf.predict(x_test_bow)
pred_proba_train = clf.predict_proba(x_train_bow)[:,1]
pred_proba_test = clf.predict_proba(x_test_bow)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
```

```
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
                                                                              optimal_
#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'Random-Forest',
                                      'VECTORIZER': 'Bag of Words(BoW)',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)

plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - Random Forest - BoWVectorizer - TRAIN SE
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - Random Forest - BoWVectorizer - TEST SET
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
```

```
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

```
Optimal Depth: 90 with Optimal Estimators: 512 with AUC: 96.17%
```



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

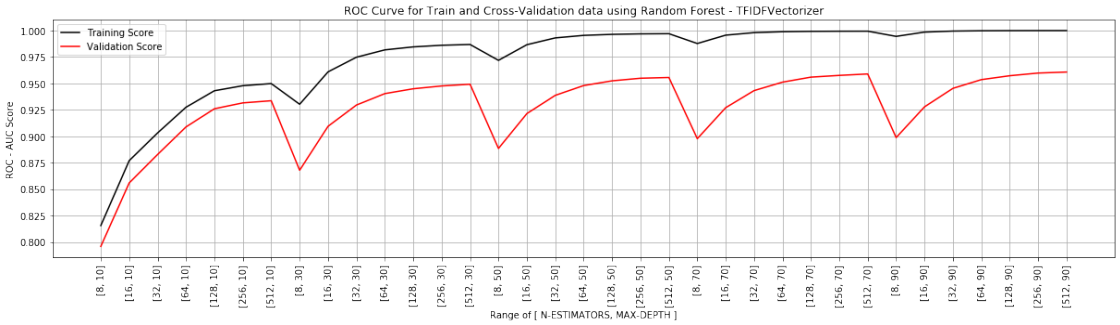
```
In [40]: #Reference - https://blog.goodaudience.com/how-to-generate-a-word-cloud-of-any-shape-in
         def generate_wordcloud(words):
             word_cloud = WordCloud(width = 512, height = 512,
                                    background_color='white', stopwords=STOPWORDS).generate(word
             plt.figure(figsize=(10,8),facecolor = 'white', edgecolor='blue')
             plt.title("WORDCLOUD FOR TOP 20 IMPORTANT FEATURES - BoW VECTORIZER")
             plt.imshow(word_cloud)
             plt.axis('off')
             plt.tight_layout(pad=0)
             plt.show()

         feature_imp_values = clf.feature_importances_
         bow_features_names = bow_model.get_feature_names()
         indices = np.argsort(feature_imp_values)[::-1]
         names = [bow_features_names[i] for i in indices]
         generate_wordcloud(' '.join(names[:20]))
```

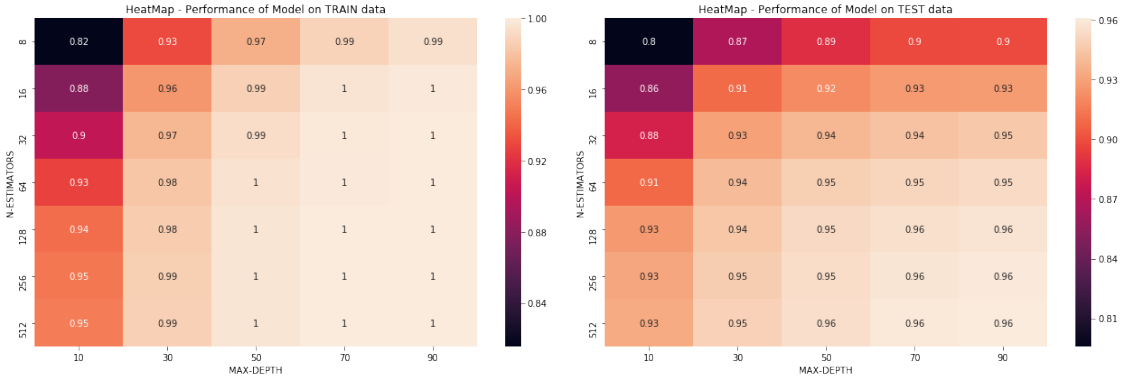
6.1.3 [5.1.3] Applying Random Forests on TFIDF, SET 2

In [41]:
```
# Applying TFIDF Vectorizer
tfidf_model = TfidfVectorizer()
tfidf_model.fit(x_train)

x_train_tfidf = tfidf_model.transform(x_train)
x_test_tfidf = tfidf_model.transform(x_test)
```

In [42]:
```
# Applying RandomForestClassfier using GridSearch CV=10
rfc = RandomForestClassifier(n_jobs=-1)
parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
clf = GridSearchCV(rfc, parameters, cv=10, scoring = 'roc_auc', return_train_score=True
```

```
clf.fit(x_train_tfidf, y_train)

mean_train_score = clf.cv_results_['mean_train_score']
mean_test_score = clf.cv_results_['mean_test_score']

param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using Random Forest - TFIDFVec
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```



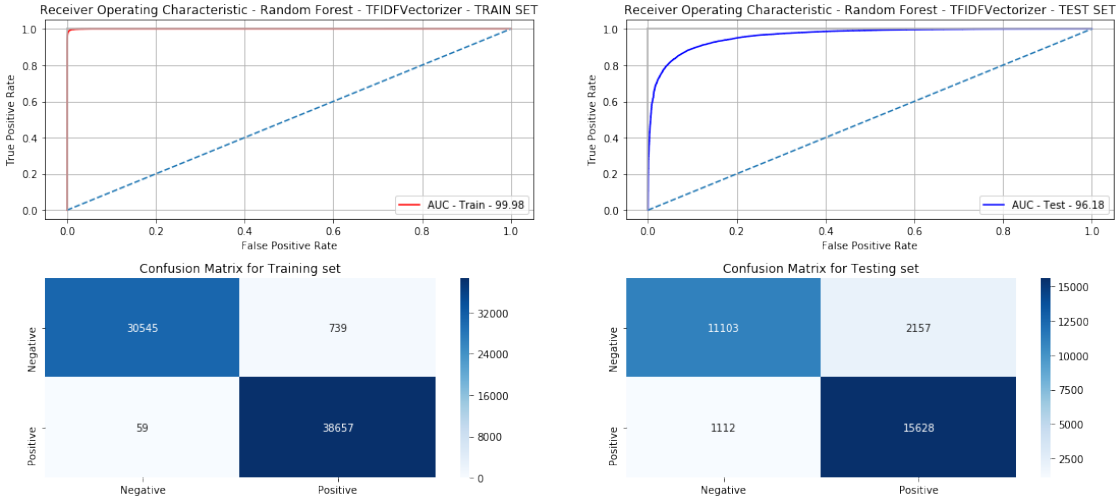



```
In [43]: optimal_depth = clf.best_params_['max_depth']
         optimal_estimator = clf.best_params_['n_estimators']

         clf = RandomForestClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n
         clf.fit(x_train_tfidf, y_train)

         # Get predicted values for test data
         pred_train = clf.predict(x_train_tfidf)
         pred_test = clf.predict(x_test_tfidf)
         pred_proba_train = clf.predict_proba(x_train_tfidf)[:,1]
         pred_proba_test = clf.predict_proba(x_test_tfidf)[:,1]

         fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
         fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
         conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
         conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
         f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
         auc_sc_train = auc(fpr_train, tpr_train)
         auc_sc = auc(fpr_test, tpr_test)

         print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
```

```
                                                                                optimal_
#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'Random-Forest',
                                      'VECTORIZER': 'TF-IDF',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)

plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - Random Forest - TFIDFVectorizer - TRAIN
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - Random Forest - TFIDFVectorizer - TEST S
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')
```

```
        plt.tight_layout()
        plt.show()
```

```
Optimal Depth: 90 with Optimal Estimators: 512 with AUC: 96.18%
```



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

```
In [44]: #Reference - https://blog.goodaudience.com/how-to-generate-a-word-cloud-of-any-shape-in
        def generate_wordcloud(words):
            word_cloud = WordCloud(width = 512, height = 512,
                                   background_color='white', stopwords=STOPWORDS).generate(word
            plt.figure(figsize=(10,8),facecolor = 'white', edgecolor='blue')
            plt.title("WORDCLOUD FOR TOP 20 IMPORTANT FEATURES - TF-IDF VECTORIZER")
            plt.imshow(word_cloud)
            plt.axis('off')
            plt.tight_layout(pad=0)
            plt.show()

        feature_imp_values = clf.feature_importances_
        tfidf_features_names = tfidf_model.get_feature_names()
        indices = np.argsort(feature_imp_values)[::-1]
        names = [tfidf_features_names[i] for i in indices]
        generate_wordcloud(' '.join(names[:20]))
```

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

```
In [45]: list_of_sent_train = []
         list_of_sent_test = []

         for sent in x_train:
             list_of_sent_train.append(sent.split())
         for sent in x_test:
             list_of_sent_test.append(sent.split())

         w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50)
         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])
```

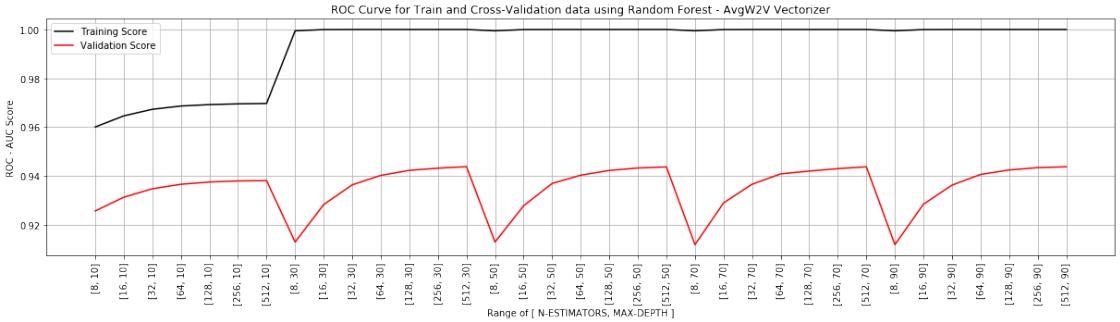
```
number of words that occured minimum 5 times  16937
sample words  ['wiping', 'stopping', 'liken', 'bacillus', 'oiliness', 'clinical', 'prefect', 've
```

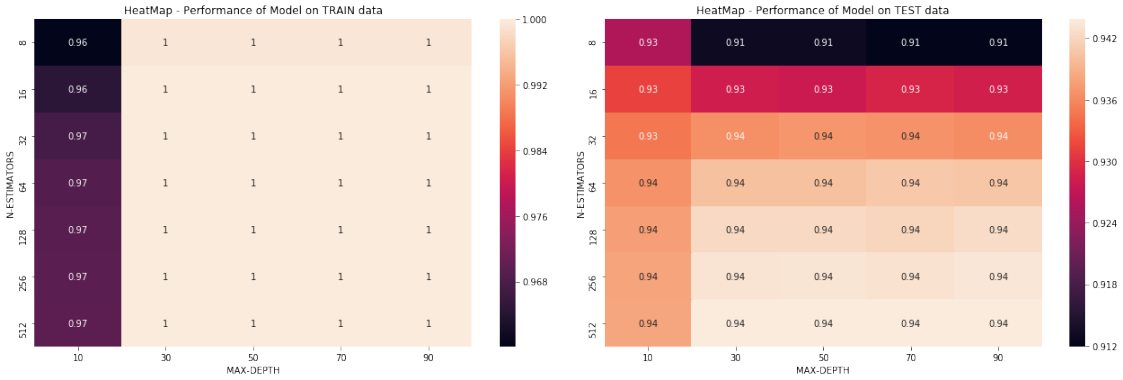
In [46]:
```
        # compute average word2vec for each review for train data
        avgw2v_train = [] # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sent_train, ascii=True, desc="Training Set W2V"): # for each r
            sent_vec = np.zeros(50)
            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)

        # 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, ascii=True, desc="Testing Set W2V"): # for each rev
            sent_vec = np.zeros(50)
            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)
```

```
Training Set W2V: 100%|##########| 70000/70000 [17:36<00:00, 63.94it/s]
Testing Set W2V: 100%|##########| 30000/30000 [07:26<00:00, 67.20it/s]
```

In [47]:
```
        # Applying RandomForestClassfier using GridSearch CV=10
        rfc = RandomForestClassifier(n_jobs=-1)
        parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
        clf = GridSearchCV(rfc, parameters, cv=10, scoring = 'roc_auc', return_train_score=True
        clf.fit(avgw2v_train, y_train)

        mean_train_score = clf.cv_results_['mean_train_score']
        mean_test_score = clf.cv_results_['mean_test_score']

        param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_
```

```
plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using Random Forest - AvgW2V V
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```

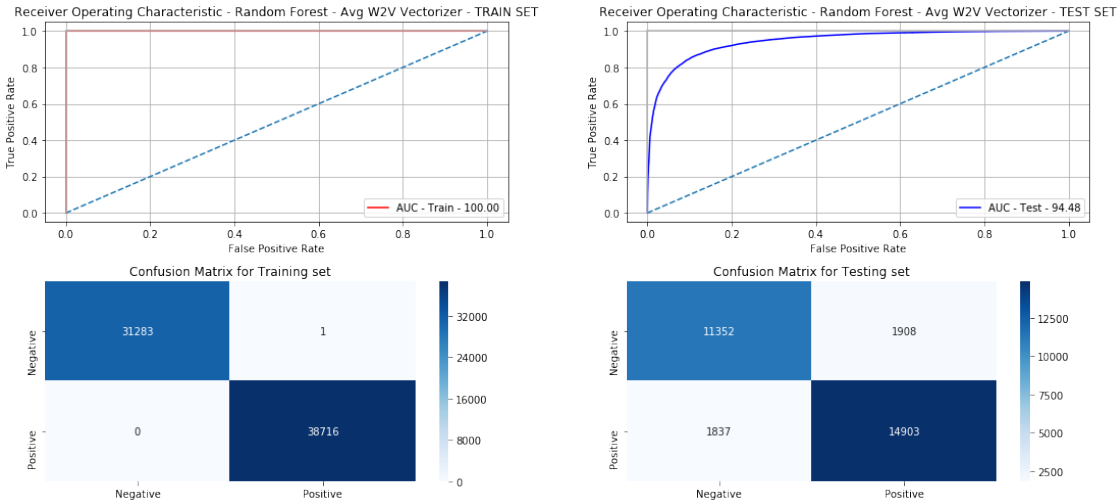




In [48]:
```
optimal_depth = clf.best_params_['max_depth']
optimal_estimator = clf.best_params_['n_estimators']

clf = RandomForestClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n
clf.fit(avgw2v_train, y_train)

# Get predicted values for test data
pred_train = clf.predict(avgw2v_train)
pred_test = clf.predict(avgw2v_test)
pred_proba_train = clf.predict_proba(avgw2v_train)[:,1]
pred_proba_test = clf.predict_proba(avgw2v_test)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
                                                                              optimal_

#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'Random-Forest',
                                      'VECTORIZER': 'Avg-W2V',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)

plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
```

```
        plt.subplot(2, 2, 1)
        plt.title('Receiver Operating Characteristic - Random Forest - Avg W2V Vectorizer - TRA
        plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        # Plot ROC curve for test set
        plt.subplot(2, 2, 2)
        plt.title('Receiver Operating Characteristic - Random Forest - Avg W2V Vectorizer - TES
        plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        #Plotting the confusion matrix for train
        plt.subplot(2, 2, 3)
        plt.title('Confusion Matrix for Training set')
        df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        #Plotting the confusion matrix for test
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

```
Optimal Depth: 30 with Optimal Estimators: 512 with AUC: 94.48%
```



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

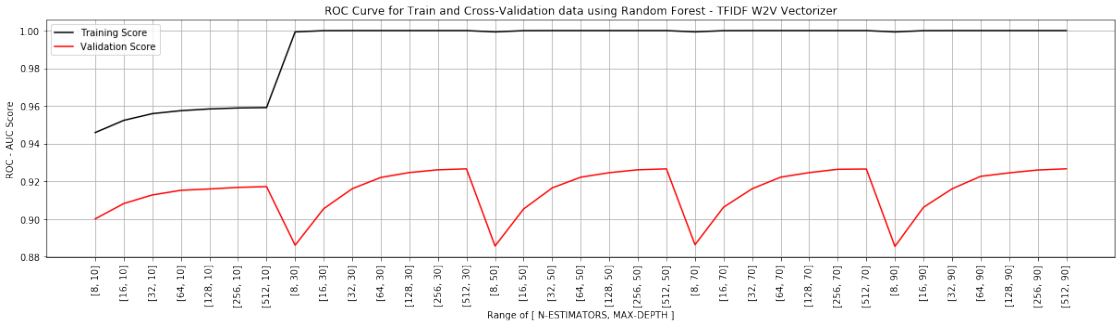
```
In [49]: model = TfidfVectorizer()
         model.fit(x_train)

         #Creating the TFIDF W2V Training Set
         # 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

         tfidfw2v_train = [];
         tfidfw2v_test = [];

         for sent in tqdm(list_of_sent_train, ascii=True, desc='Training Set for TFIDF W2V'): #
             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 = 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)

         for sent in tqdm(list_of_sent_test, ascii=True, desc='Testing Set for TFIDF W2V'): # fo
```

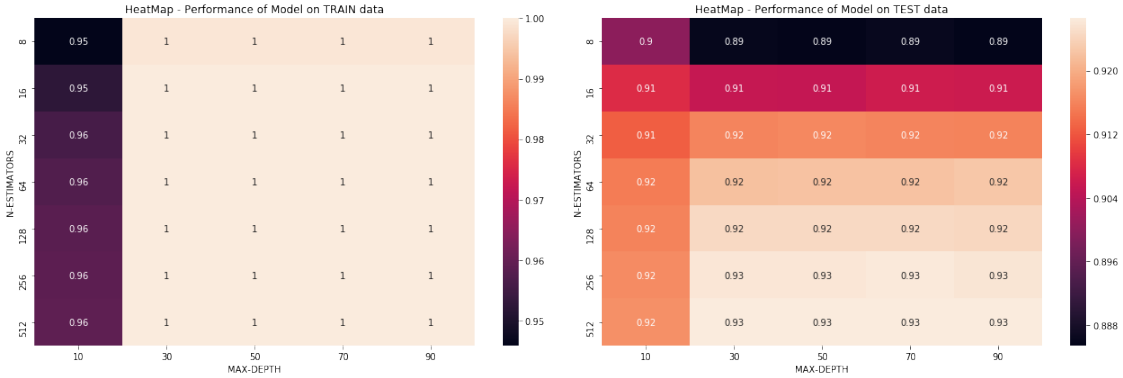
```
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
                    tf_idf = 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)
```

```
Training Set for TFIDF W2V: 100%|##########| 70000/70000 [55:05<00:00, 21.17it/s]
Testing Set for TFIDF W2V: 100%|##########| 30000/30000 [24:37<00:00, 22.22it/s]
```

In [50]:
```
# Applying RandomForestClassfier using GridSearch CV=10
rfc = RandomForestClassifier(n_jobs=-1)
parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
clf = GridSearchCV(rfc, parameters, cv=10, scoring = 'roc_auc', return_train_score=True
clf.fit(tfidfw2v_train, y_train)

mean_train_score = clf.cv_results_['mean_train_score']
mean_test_score = clf.cv_results_['mean_test_score']

param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using Random Forest - TFIDF W2
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
```

```
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```



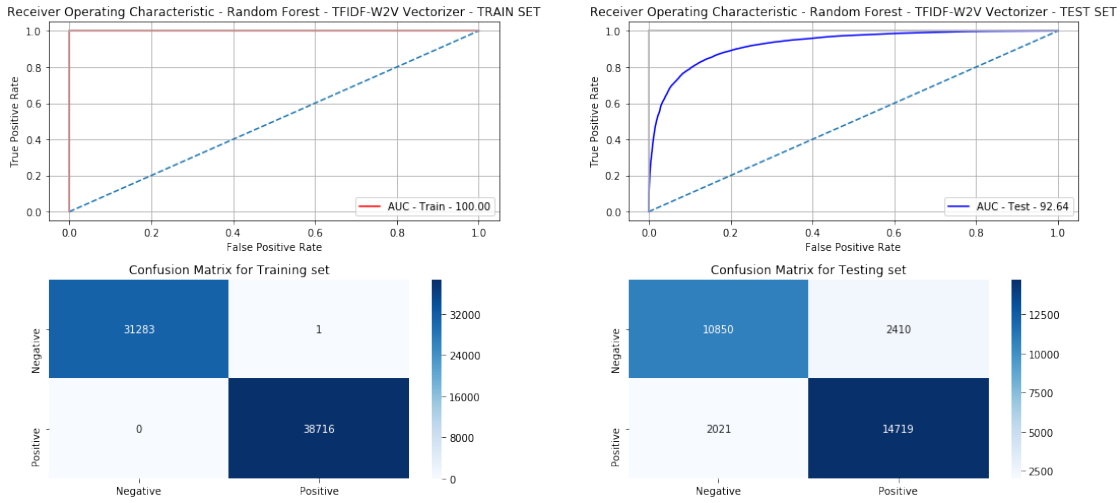


In [51]:
```
optimal_depth = clf.best_params_['max_depth']
optimal_estimator = clf.best_params_['n_estimators']

clf = RandomForestClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n
clf.fit(tfidfw2v_train, y_train)

# Get predicted values for test data
```

```
pred_train = clf.predict(tfidfw2v_train)
pred_test = clf.predict(tfidfw2v_test)
pred_proba_train = clf.predict_proba(tfidfw2v_train)[:,1]
pred_proba_test = clf.predict_proba(tfidfw2v_test)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
                                                                              optimal_

#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'Random-Forest',
                                      'VECTORIZER': 'TFIDF-W2V',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)


plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - Random Forest - TFIDF-W2V Vectorizer - T
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - Random Forest - TFIDF-W2V Vectorizer - T
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')
```

```
        #Plotting the confusion matrix for train
        plt.subplot(2, 2, 3)
        plt.title('Confusion Matrix for Training set')
        df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        #Plotting the confusion matrix for test
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

```
Optimal Depth: 90 with Optimal Estimators: 512 with AUC: 92.64%
```



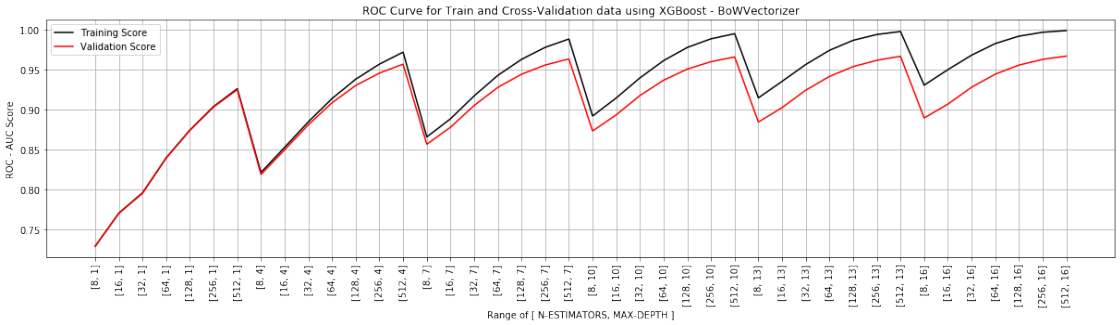
6.2 [5.2] Applying GBDT using XGBOOST

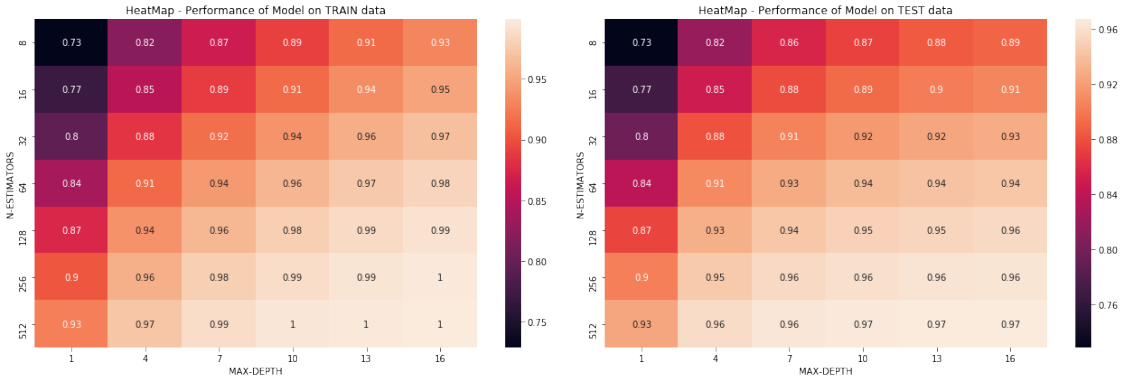
```
In [55]: n_estimator_range =  [2**i for i in range(3, 10)]
         max_depth_range = [i for i in range(1, 17, 3)]
         tot_hyp_length = np.arange(0, len(n_estimator_range) * len(max_depth_range))
```

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

```
In [56]: # Applying XGBClassifier using GridSearch CV=10
         xgbC = XGBClassifier(n_jobs=-1)
         parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
```

```
clf = GridSearchCV(xgbC, parameters, cv=10, scoring = 'roc_auc', return_train_score=Tru
clf.fit(x_train_bow, y_train)

mean_train_score = clf.cv_results_['mean_train_score']
mean_test_score = clf.cv_results_['mean_test_score']

param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using XGBoost - BoWVectorizer"
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```



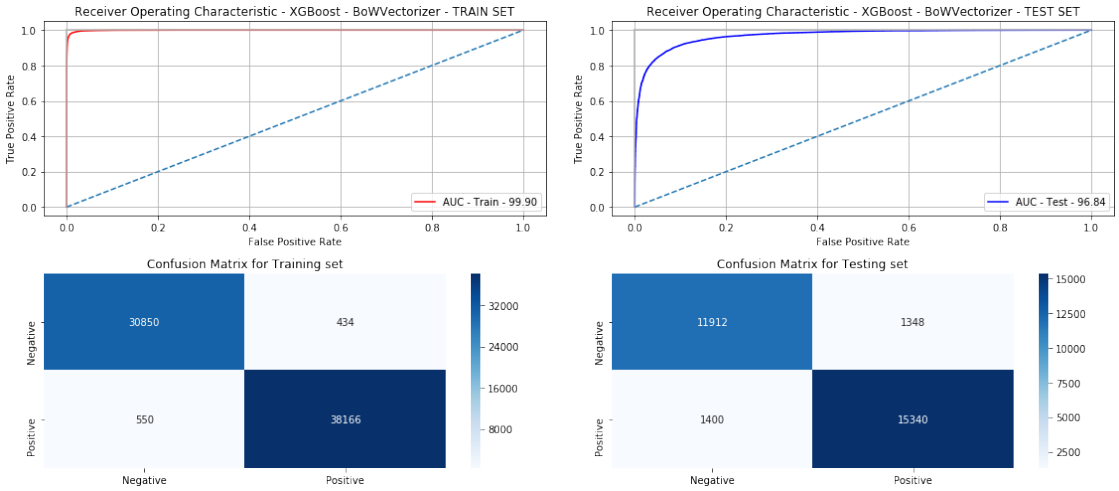


In [57]:
```
optimal_depth = clf.best_params_['max_depth']
optimal_estimator = clf.best_params_['n_estimators']

clf = XGBClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n_jobs=-1)
clf.fit(x_train_bow, y_train)

# Get predicted values for test data
pred_train = clf.predict(x_train_bow)
pred_test = clf.predict(x_test_bow)
pred_proba_train = clf.predict_proba(x_train_bow)[:,1]
pred_proba_test = clf.predict_proba(x_test_bow)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
```

```
                                                                                optimal_
#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'XGBoost',
                                      'VECTORIZER': 'Bag of Words(BoW)',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)

plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - XGBoost - BoWVectorizer - TRAIN SET')
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - XGBoost - BoWVectorizer - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')
```

```
        plt.tight_layout()
        plt.show()
```

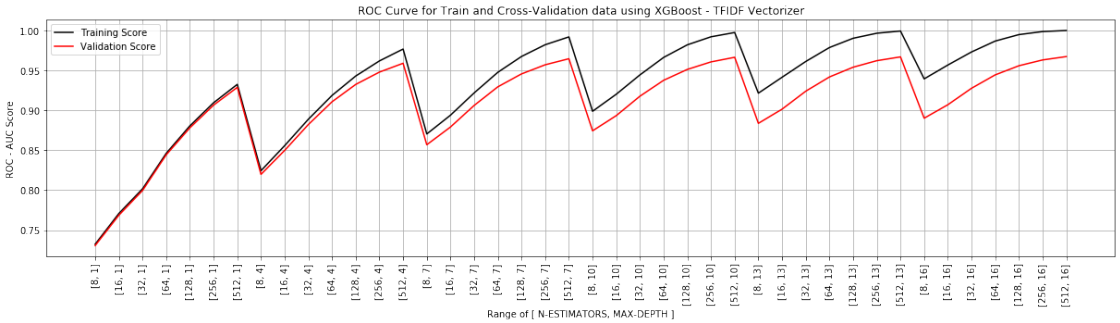
```
Optimal Depth: 16 with Optimal Estimators: 512 with AUC: 96.84%
```

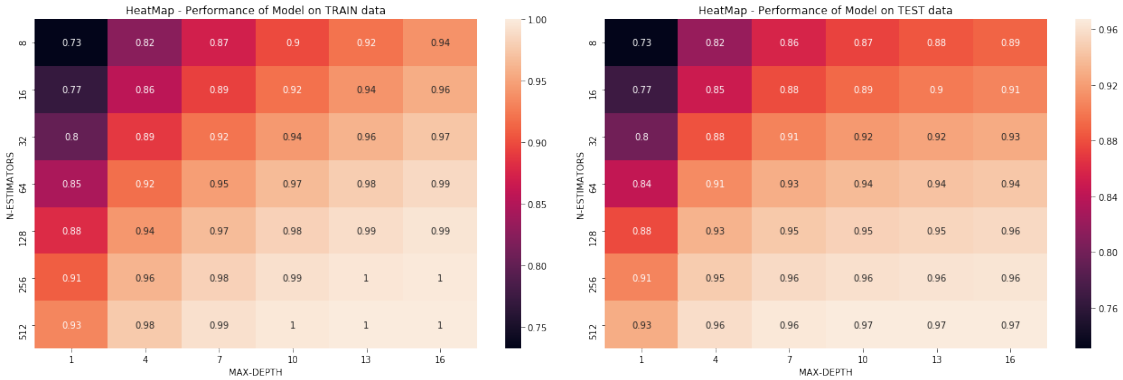


6.2.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

```
In [58]: # Applying XGBClassifier using GridSearch CV=10
         xgbC = XGBClassifier(n_jobs=-1)
         parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
         clf = GridSearchCV(xgbC, parameters, cv=10, scoring = 'roc_auc', return_train_score=Tru
         clf.fit(x_train_tfidf, y_train)

         mean_train_score = clf.cv_results_['mean_train_score']
         mean_test_score = clf.cv_results_['mean_test_score']

         param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

         plt.close()
         plt.figure(figsize=(17, 5))
         #Plot mean accuracy for train and cv set scores
         plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
         plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
         plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

         # Create plot
         plt.title("ROC Curve for Train and Cross-Validation data using XGBoost - TFIDF Vectoriz
         plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
         plt.ylabel("ROC - AUC Score")
         plt.tight_layout()
```

```
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```

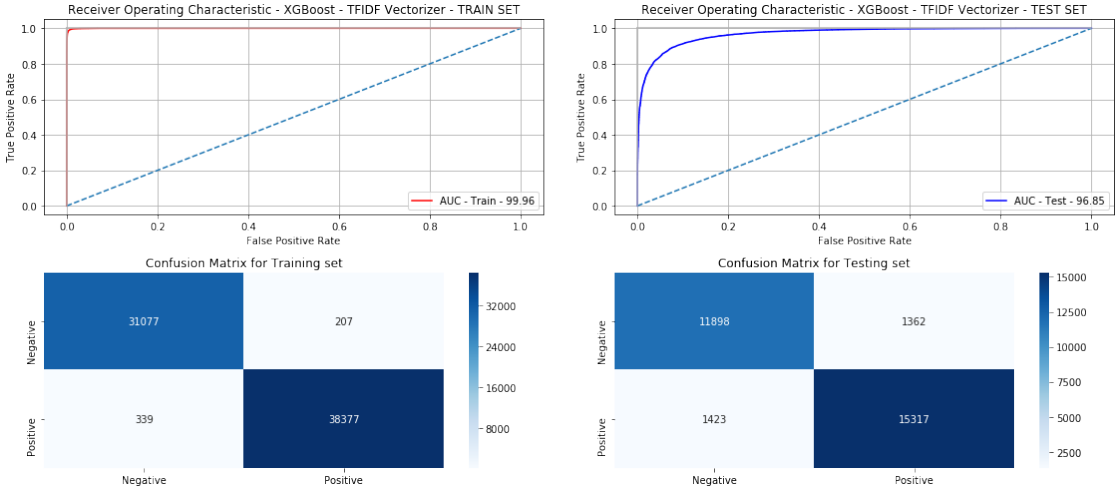




In [59]:
```
optimal_depth = clf.best_params_['max_depth']
optimal_estimator = clf.best_params_['n_estimators']

clf = XGBClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n_jobs=-1)
clf.fit(x_train_tfidf, y_train)

# Get predicted values for test data
pred_train = clf.predict(x_train_tfidf)
pred_test = clf.predict(x_test_tfidf)
pred_proba_train = clf.predict_proba(x_train_tfidf)[:,1]
pred_proba_test = clf.predict_proba(x_test_tfidf)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'XGBoost',
                                      'VECTORIZER': 'TF-IDF',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)


plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - XGBoost - TFIDF Vectorizer - TRAIN SET')
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - XGBoost - TFIDF Vectorizer - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
```

```
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        #Plotting the confusion matrix for train
        plt.subplot(2, 2, 3)
        plt.title('Confusion Matrix for Training set')
        df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        #Plotting the confusion matrix for test
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

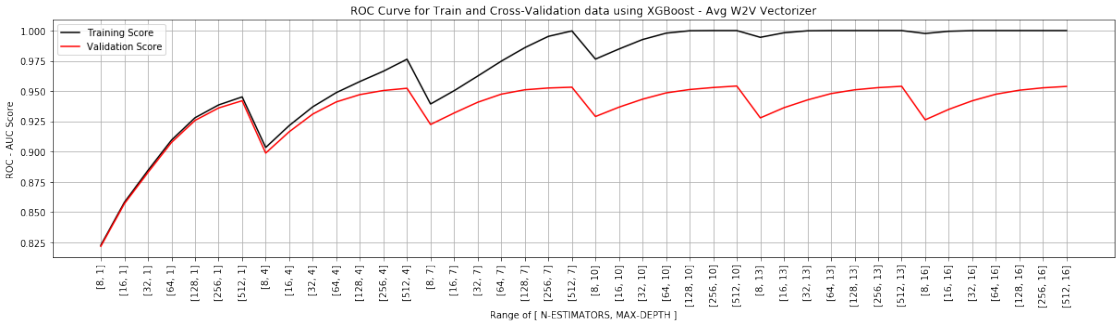
```
Optimal Depth: 16 with Optimal Estimators: 512 with AUC: 96.85%
```

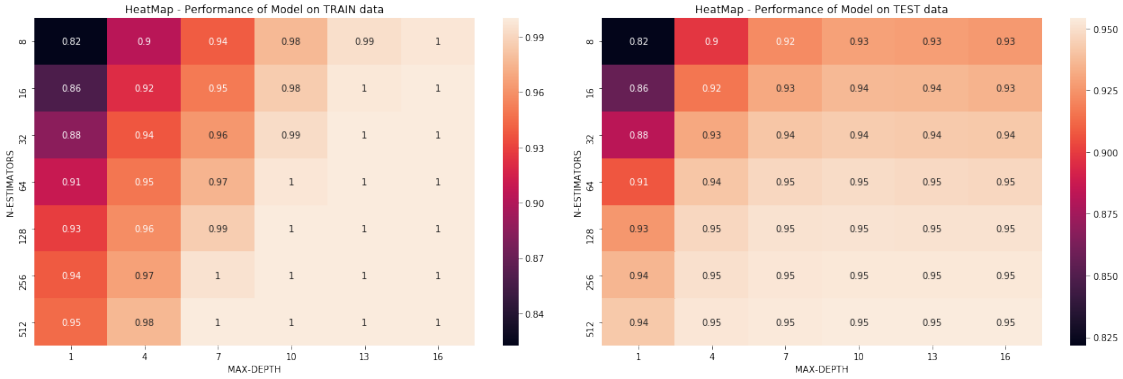


6.2.3 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

```
In [60]: avgw2v_train = np.array(avgw2v_train)
         avgw2v_test = np.array(avgw2v_test)
```

In [61]:
```
# Applying XGBClassifier using GridSearch CV=10
xgbC = XGBClassifier(n_jobs=-1)
parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
clf = GridSearchCV(xgbC, parameters, cv=10, scoring = 'roc_auc', return_train_score=Tru
clf.fit(avgw2v_train, y_train)

mean_train_score = clf.cv_results_['mean_train_score']
mean_test_score = clf.cv_results_['mean_test_score']

param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using XGBoost - Avg W2V Vector
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```



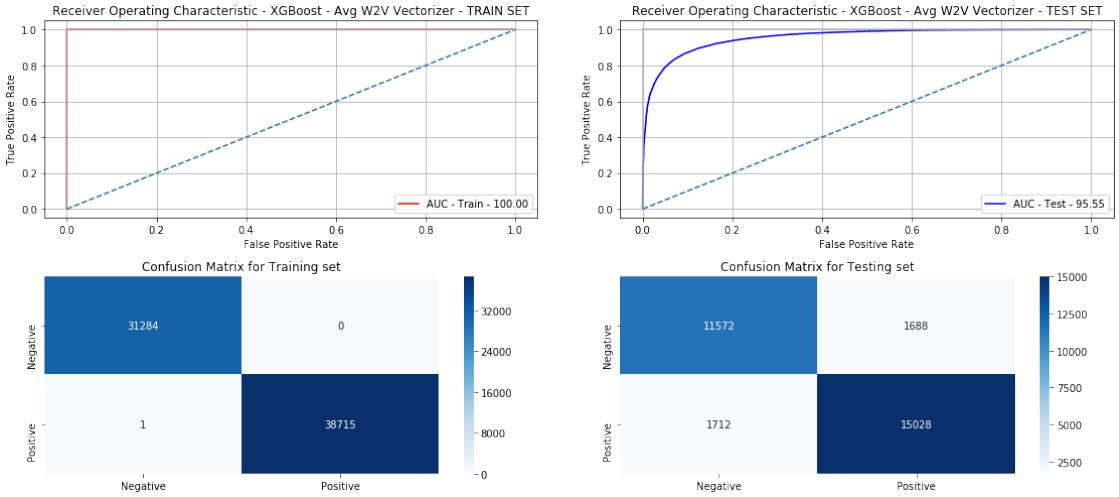



```
In [62]: optimal_depth = clf.best_params_['max_depth']
         optimal_estimator = clf.best_params_['n_estimators']

         clf = XGBClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n_jobs=-1)
         clf.fit(avgw2v_train, y_train)

         # Get predicted values for test data
         pred_train = clf.predict(avgw2v_train)
         pred_test = clf.predict(avgw2v_test)
         pred_proba_train = clf.predict_proba(avgw2v_train)[:,1]
         pred_proba_test = clf.predict_proba(avgw2v_test)[:,1]

         fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
         fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
         conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
         conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
         f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
         auc_sc_train = auc(fpr_train, tpr_train)
         auc_sc = auc(fpr_test, tpr_test)

         print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
```

```
                                                                                optimal_
#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'XGBoost',
                                      'VECTORIZER': 'Avg W2V',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)

plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - XGBoost - Avg W2V Vectorizer - TRAIN SET
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - XGBoost - Avg W2V Vectorizer - TEST SET'
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                     columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')
```

```
        plt.tight_layout()
        plt.show()
```

```
Optimal Depth: 10 with Optimal Estimators: 512 with AUC: 95.55%
```

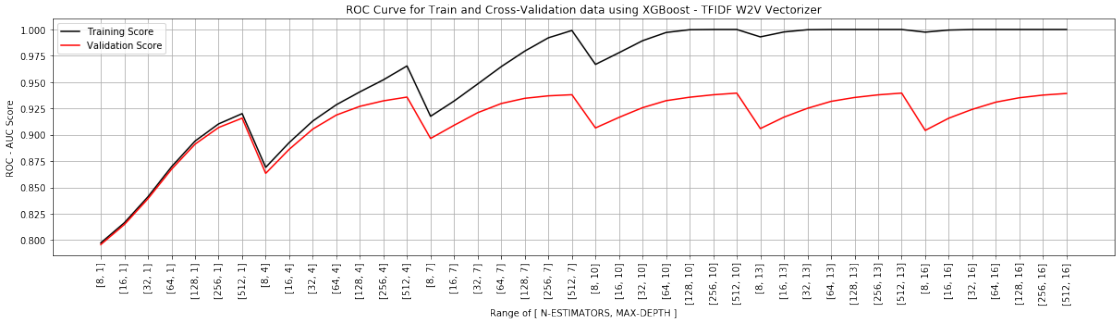


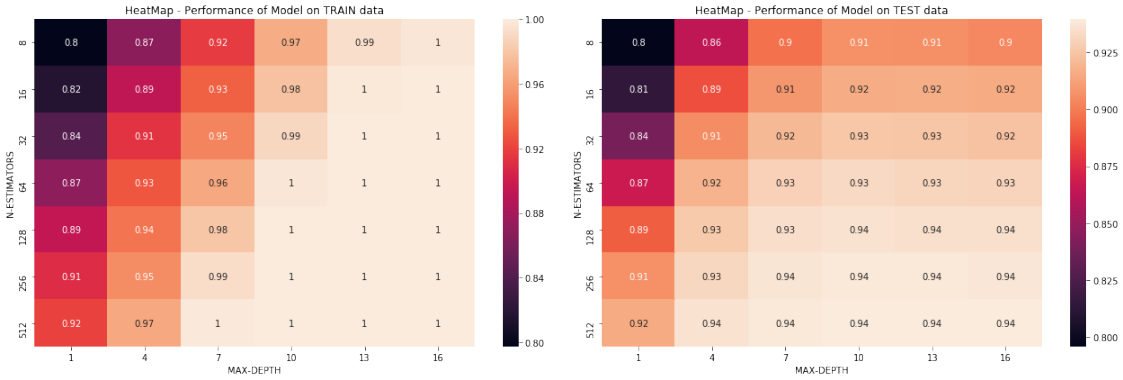
6.2.4 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [63]:
```
tfidfw2v_train = np.array(tfidfw2v_train)
tfidfw2v_test = np.array(tfidfw2v_test)
```

In [64]:
```
# Applying XGBClassifier using GridSearch CV=10
xgbC = XGBClassifier(n_jobs=-1)
parameters = {'n_estimators': n_estimator_range, 'max_depth': max_depth_range}
clf = GridSearchCV(xgbC, parameters, cv=10, scoring = 'roc_auc', return_train_score=Tru
clf.fit(tfidfw2v_train, y_train)

mean_train_score = clf.cv_results_['mean_train_score']
mean_test_score = clf.cv_results_['mean_test_score']

param_arr = list(map(lambda obj: list([obj['n_estimators'], obj['max_depth']]), clf.cv_

plt.close()
plt.figure(figsize=(17, 5))
#Plot mean accuracy for train and cv set scores
plt.plot(tot_hyp_length, mean_train_score, label='Training Score', color='black')
plt.plot(tot_hyp_length, mean_test_score, label='Validation Score', color='red')
plt.xticks(tot_hyp_length, param_arr, rotation='vertical')

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using XGBoost - TFIDF W2V Vect
```

```
plt.xlabel("Range of [ N-ESTIMATORS, MAX-DEPTH ]")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.grid()
plt.legend(loc="best")


m_train_score = clf.cv_results_['mean_train_score'].reshape((len(max_depth_range),len(n
m_test_score = clf.cv_results_['mean_test_score'].reshape((len(max_depth_range),len(n_e

plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
plt.title('HeatMap - Performance of Model on TRAIN data')
ax1 = sns.heatmap(m_train_score, annot=True, xticklabels=max_depth_range, yticklabels=n
ax1.set_xlabel('MAX-DEPTH')
ax1.set_ylabel('N-ESTIMATORS')

plt.subplot(1, 2, 2)
plt.title('HeatMap - Performance of Model on TEST data')
ax2 = sns.heatmap(m_test_score, annot=True, xticklabels=max_depth_range, yticklabels=n_
ax2.set_xlabel('MAX-DEPTH')
ax2.set_ylabel('N-ESTIMATORS')

plt.tight_layout()
plt.show()
```

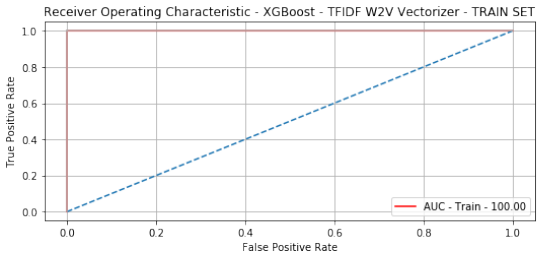


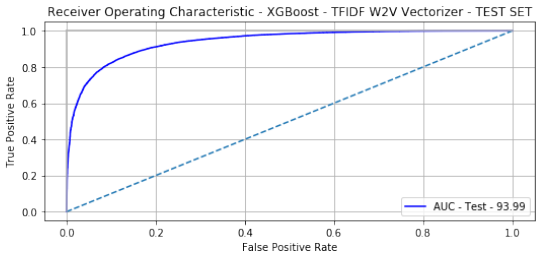


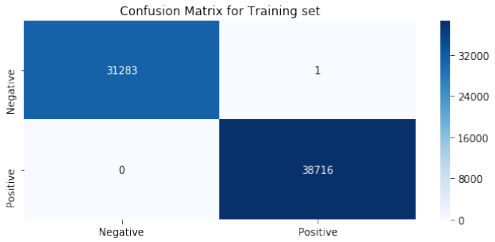
In [65]:
```
optimal_depth = clf.best_params_['max_depth']
optimal_estimator = clf.best_params_['n_estimators']

clf = XGBClassifier(n_estimators=optimal_estimator, max_depth=optimal_depth, n_jobs=-1)
clf.fit(tfidfw2v_train, y_train)

# Get predicted values for test data
pred_train = clf.predict(tfidfw2v_train)
pred_test = clf.predict(tfidfw2v_test)
pred_proba_train = clf.predict_proba(tfidfw2v_train)[:,1]
pred_proba_test = clf.predict_proba(tfidfw2v_test)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal Depth: {} with Optimal Estimators: {} with AUC: {:.2f}%".format(optimal_
                                                                              optimal_

#Saving the report in a global variable
result_report = result_report.append({'ALGORITHM': 'XGBoost',
                                      'VECTORIZER': 'TFIDF W2V',
                                      'DATASET-SIZE': '{0:,.0f}'.format(int(min_final
                                      'N_ESTIMATOR (HYPERPARAMETER)':optimal_estimato
                                      'MAX_DEPTH (HYPERPARAMETER)':optimal_depth,
                                      'F1_SCORE':f1_sc,
                                      'AUC':auc_sc}, ignore_index=True)

plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
```

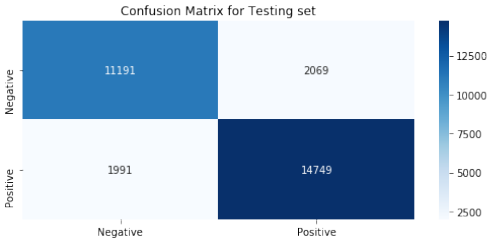
```
        plt.subplot(2, 2, 1)
        plt.title('Receiver Operating Characteristic - XGBoost - TFIDF W2V Vectorizer - TRAIN S
        plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        # Plot ROC curve for test set
        plt.subplot(2, 2, 2)
        plt.title('Receiver Operating Characteristic - XGBoost - TFIDF W2V Vectorizer - TEST SE
        plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        #Plotting the confusion matrix for train
        plt.subplot(2, 2, 3)
        plt.title('Confusion Matrix for Training set')
        df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        #Plotting the confusion matrix for test
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                             columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

```
Optimal Depth: 13 with Optimal Estimators: 512 with AUC: 93.99%
```









7 [6] Conclusions

```
In [66]: result_report
```

Out[66]:

	ALGORITHM	VECTORIZER	DATASET-SIZE	N_ESTIMATOR (HYPERPARAMETER)	MAX_DEPTH (HYPERPARAMETER)	F1_SCORE	AUC
0	Random-Forest	Bag of Words(BoW)	100,000	512	90	0.904325	0.961678
1	Random-Forest	TF-IDF	100,000	512	90	0.905315	0.961804
2	Random-Forest	Avg-W2V	100,000	512	30	0.888379	0.944773
3	Random-Forest	TFIDF-W2V	100,000	512	90	0.869172	0.926442
4	XGBoost	Bag of Words(BoW)	100,000	512	16	0.917793	0.968366
5	XGBoost	TF-IDF	100,000	512	16	0.916664	0.968493
6	XGBoost	Avg W2V	100,000	512	10	0.898374	0.955471
7	XGBoost	TFIDF W2V	100,000	512	13	0.879015	0.939878