09 Amazon Fine Food Reviews Analysis_RF

February 11, 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 [2]: %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
```

```
In [2]: # using SQLite Table to read data.
        con = sqlite3.connect('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 5000

        # 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]: # 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 Victor
```

```
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 oth
==================================================
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]: # 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%|| 4986/4986 [00:01<00:00, 3137.37it/s]
```

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

Out[23]: 'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey
```

[3.2] Preprocessing Review Summary

```
In [6]: ## Similartly you can do preprocessing for review summary also.
```

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])

some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

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_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

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_names
         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
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

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 [42]: # 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.bi
                 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, t
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321664810
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.999275088310
```

In [36]:
```
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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

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

[4.4.1.1] Avg W2v

In [38]:
```
# 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]))
```

```
100%|| 4986/4986 [00:03<00:00, 1330.47it/s]


4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [39]: # 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 [41]: # 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
```

```
100%|| 4986/4986 [00:20<00:00, 245.63it/s]
```

6 [5] Assignment 9: Random Forests

Apply Random Forests & GBDT on these feature sets
SET 1:Review text, preprocessed one converted into vectors using (BOW)
SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)
The hyper paramter tuning (Consider any two hyper parameters)
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 engin
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='3d_plot.JPG' width=500px> with X-axis as <strong>n_estimators</strong>, Y-axis as <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 [6]: # after preprocessing
        df = pd.read_pickle("files/preprocessed.pkl")
        df.shape
```

```
Out[6]: (364171, 12)
```

```
In [7]: df.head()
```

```
Out[7]:             Id   ProductId          UserId                 ProfileName  \
        138706  150524  0006641040   ACITT7DI6IDDL             shari zychinski
        138688  150506  0006641040  A2IW4PEEKO2R0U                       Tracy
        138689  150507  0006641040  A1S4A3IQ2MU7V4       sally sue "sally sue"
        138690  150508  0006641040     AZGXZ2UUK6X  Catherine Hallberg "(Kate)"
        138691  150509  0006641040  A3CMRKGE0P909G                      Teresa

                HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        138706                     0                       0      1   939340800
        138688                     1                       1      1  1194739200
        138689                     1                       1      1  1191456000
        138690                     1                       1      1  1076025600
        138691                     3                       4      1  1018396800

                                                  Summary  \
        138706                  EVERY book is educational
        138688  Love the book, miss the hard cover version
        138689                 chicken soup with rice months
        138690        a good swingy rhythm for reading aloud
        138691               A great way to learn the months
```

```
                                                     Text  \
138706  this witty little book makes my son laugh at l...
138688  I grew up reading these Sendak books, and watc...
138689  This is a fun way for children to learn their ...
138690  This is a great little book to read aloud- it ...
138691  This is a book of poetry about the months of t...

                                              CleanedText  \
138706  witty little book makes son laugh loud recite ...
138688  grew reading sendak books watching really rosi...
138689  fun way children learn months year learn poems...
138690  great little book read aloud nice rhythm well ...
138691  book poetry months year goes month cute little...

                           CleanedSummary
138706             every book educational
138688  love book miss hard cover version
138689           chicken soup rice months
138690   good swingy rhythm reading aloud
138691             great way learn months
```

```
In [8]: from sklearn.model_selection import train_test_split
        from sklearn.grid_search import GridSearchCV
        from sklearn.datasets import *
        from sklearn.metrics import accuracy_score , f1_score , confusion_matrix
        from collections import Counter
        from sklearn.metrics import accuracy_score, roc_auc_score , roc_curve
        from sklearn.model_selection import train_test_split
```

```
In [9]: # take 50k sample data randomly
        sample_data = df.sample(100000)
        sample_data.shape

Out[9]: (100000, 12)
```

```
In [12]: # sorted the data using time based
         sorted_data = sample_data.sort_values('Time', axis=0, inplace=False)
         sorted_data.shape

Out[12]: (100000, 12)
```

```
In [13]: sorted_data['Score'].value_counts()

Out[13]: 1    84258
         0    15742
         Name: Score, dtype: int64
```

```
In [14]: X = np.array(sorted_data['CleanedText'])
         y = np.array(sorted_data['Score'])
         print(X.shape)
         print(y.shape)
```

```
(100000,)
(100000,)
```

```
In [15]: # Simple cross validation
         # split the data sent into train and test
         train , test , train_y , test_y = train_test_split(X, y, test_size = 0.3, random_stat

         # split the train data set into cross validation train and cross validation test
         train, cv , train_y, cv_y = train_test_split(train, train_y, test_size=0.3, random_sta

         print("train data = ", train.shape)
         print("cros validation = ", cv.shape)
         print("test data = ", test.shape)
```

```
train data =  (49000,)
cros validation =  (21000,)
test data =  (30000,)
```

```
In [ ]:
```

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

```
In [11]: # Please write all the code with proper documentation
         #BoW
         count_vect = CountVectorizer(min_df=20) #in scikit-learn
         count_vect.fit(train)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         bow_train = count_vect.fit_transform(train)
         bow_cv = count_vect.transform(cv)
         bow_test = count_vect.transform(test)
         print("============Train Data=========")
         print("the type of count vectorizer ",type(bow_train))
         print("the shape of out text BOW vectorizer ",bow_train.get_shape())
         print("the number of unique words ", bow_train.get_shape()[1])
         print("============Cross validation Data=========")
         print("the type of count vectorizer ",type(bow_cv))
         print("the shape of out text BOW vectorizer ",bow_cv.get_shape())
         print("the number of unique words ", bow_cv.get_shape()[1])
         print("============Test Data=========")
         print("the type of count vectorizer ",type(bow_test))
         print("the shape of out text BOW vectorizer ",bow_test.get_shape())
         print("the number of unique words ", bow_test.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'absorbed', 'absor
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 5927)
the number of unique words  5927
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 5927)
the number of unique words  5927
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 5927)
the number of unique words  5927
```

```
In [16]: from sklearn.ensemble import RandomForestClassifier
```

```
In [15]: n_estimators = [20,40,60,80,100,120]
         max_depth = [1,5,10,100,500,1000]
         bow_train_auc = []
         bow_cv_auc = []
         for i in n_estimators:
             for j in max_depth:
                 RFDTC = RandomForestClassifier(n_estimators=i,criterion='gini', max_depth=j)
                 RFDTC.fit(bow_train, train_y)
                 # train data
                 y_prob_train = RFDTC.predict_proba(bow_train)[:,1]
                 y_pred = np.where(y_prob_train > 0.5, 1, 0)
                 auc_roc_train = roc_auc_score(train_y , y_prob_train)
                 print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (st
                 bow_train_auc.append(auc_roc_train)
                 # CV
                 y_prob_cv = RFDTC.predict_proba(bow_cv)[:,1]
                 y_pred = np.where(y_prob_cv > 0.5, 1, 0)
                 auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
                 print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j
                 bow_cv_auc.append(auc_roc_cv)
                 print("="*50)
```

```

Train AUC for max_depth = 1 and n_estimators = 20 is 75.46%

CV AUC for max_depth = 1 and n_estimators = 20 is 74.33%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 82.62%
```

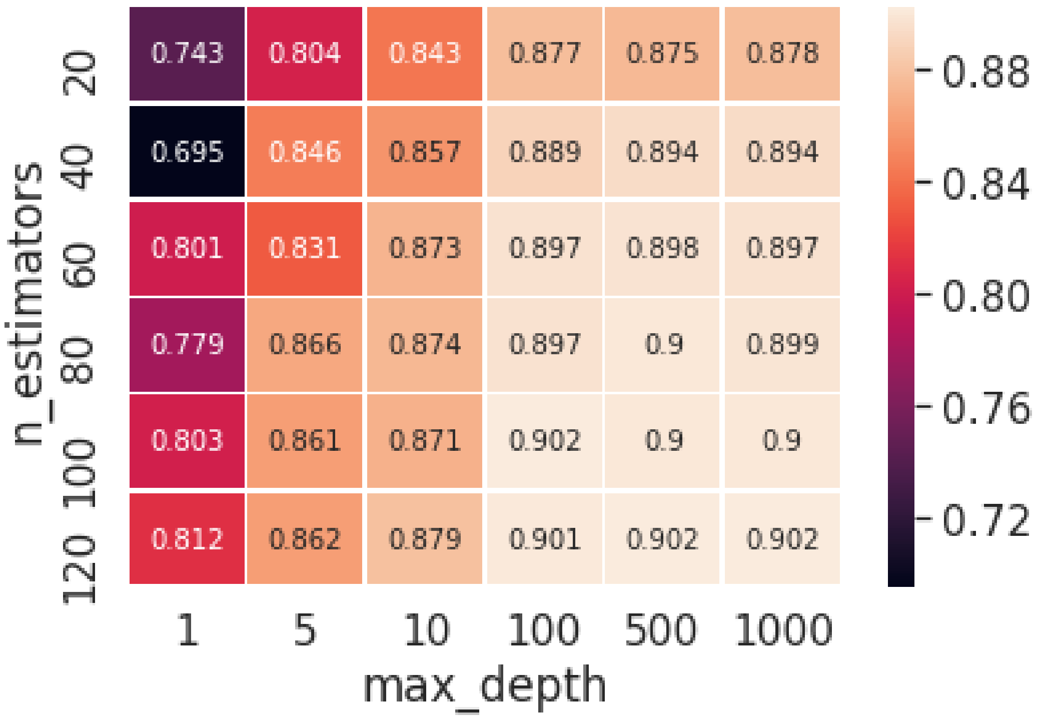
```
CV AUC for max_depth = 5 and n_estimators = 20 is 80.42%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 86.79%

CV AUC for max_depth = 10 and n_estimators = 20 is 84.33%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.77%

CV AUC for max_depth = 100 and n_estimators = 20 is 87.74%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 500 and n_estimators = 20 is 87.50%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 1000 and n_estimators = 20 is 87.76%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 70.86%

CV AUC for max_depth = 1 and n_estimators = 40 is 69.53%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 86.12%

CV AUC for max_depth = 5 and n_estimators = 40 is 84.57%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 88.74%

CV AUC for max_depth = 10 and n_estimators = 40 is 85.68%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.87%

CV AUC for max_depth = 100 and n_estimators = 40 is 88.92%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 40 is 89.35%
==================================================
Train AUC for max_depth = 1000 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 40 is 89.35%
==================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 80.86%

CV AUC for max_depth = 1 and n_estimators = 60 is 80.08%
==================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 85.07%

CV AUC for max_depth = 5 and n_estimators = 60 is 83.09%
==================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 90.12%

CV AUC for max_depth = 10 and n_estimators = 60 is 87.25%
==================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 99.91%

CV AUC for max_depth = 100 and n_estimators = 60 is 89.70%
==================================================

Train AUC for max_depth = 500 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 60 is 89.78%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 60 is 89.72%
==================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 78.81%

CV AUC for max_depth = 1 and n_estimators = 80 is 77.92%
==================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 88.71%

CV AUC for max_depth = 5 and n_estimators = 80 is 86.62%
==================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 90.31%

CV AUC for max_depth = 10 and n_estimators = 80 is 87.44%
```

```
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 99.91%

CV AUC for max_depth = 100 and n_estimators = 80 is 89.71%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 80 is 90.03%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 80 is 89.88%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 81.72%

CV AUC for max_depth = 1 and n_estimators = 100 is 80.26%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 88.07%

CV AUC for max_depth = 5 and n_estimators = 100 is 86.13%
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 90.19%

CV AUC for max_depth = 10 and n_estimators = 100 is 87.12%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 99.91%

CV AUC for max_depth = 100 and n_estimators = 100 is 90.21%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 100 is 90.00%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 100 is 90.00%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 82.28%
```

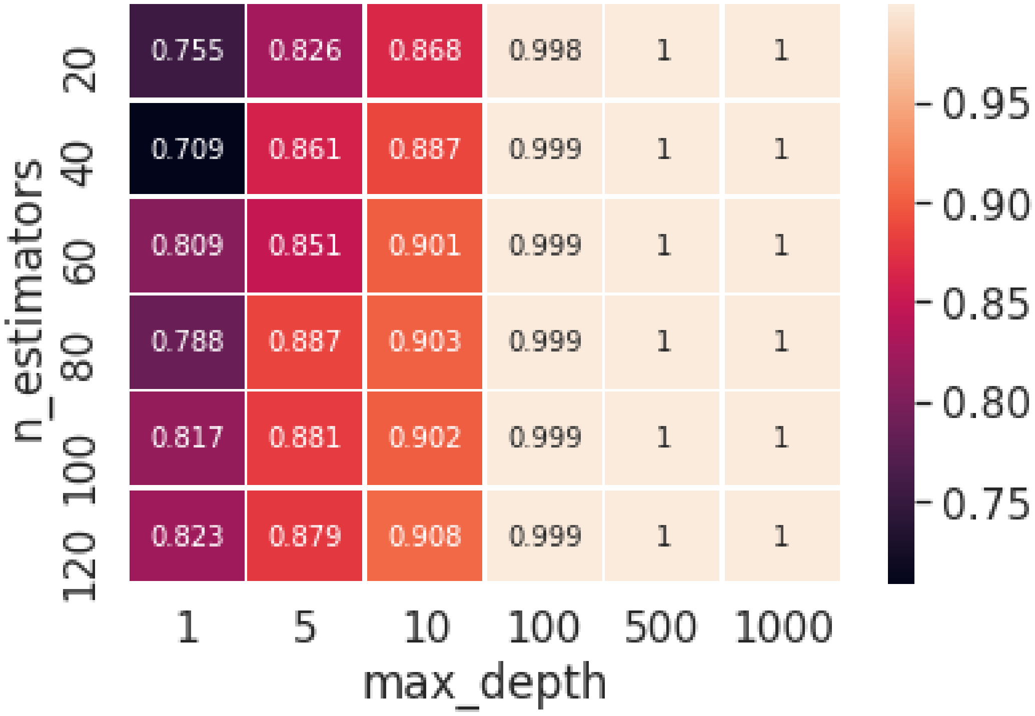
```
CV AUC for max_depth = 1 and n_estimators = 120 is 81.19%
===================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 87.88%

CV AUC for max_depth = 5 and n_estimators = 120 is 86.15%
===================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 90.83%

CV AUC for max_depth = 10 and n_estimators = 120 is 87.93%
===================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 99.92%

CV AUC for max_depth = 100 and n_estimators = 120 is 90.10%
===================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 120 is 90.23%
===================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 120 is 90.20%
===================================================
```

```
In [62]: fig, ax = plt.subplots()
         # auc on cv
         print("===================CV Data===================")
         cv_scores = np.array(bow_cv_auc).reshape(len(n_estimators),len(max_depth))
         df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
         sns.set(font_scale=1.4)
         ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=.
         ax.set_xlabel("max_depth")
         ax.set_ylabel("n_estimators")
         plt.show()
         print("===================Train Data================")
         train_scores = np.array(bow_train_auc).reshape(len(n_estimators),len(max_depth))
         df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
         sns.set(font_scale=1.4)
         ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths
         ax.set_xlabel("max_depth")
         ax.set_ylabel("n_estimators")
         plt.show()
```

```
===================CV Data===================
```

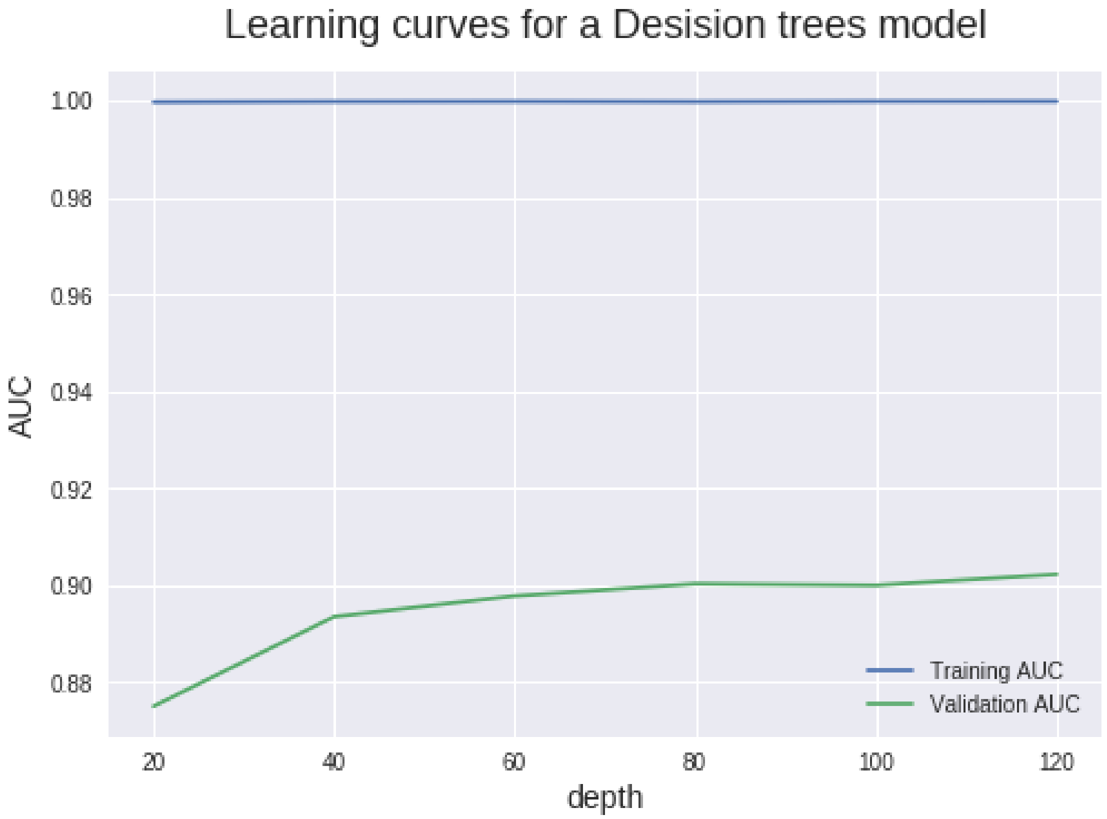


```
===================Train Data================
```



In [65]:
```
# depth is 500
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(n_estimators,train_scores[:,-2],label = 'Training AUC')
plt.plot(n_estimators,cv_scores[:,-2], label = 'Validation AUC')

plt.ylabel('AUC', fontsize = 14)
plt.xlabel('depth', fontsize = 14)
plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
plt.legend()
```

Out[65]:
```
<matplotlib.legend.Legend at 0x7f9570918668>
```



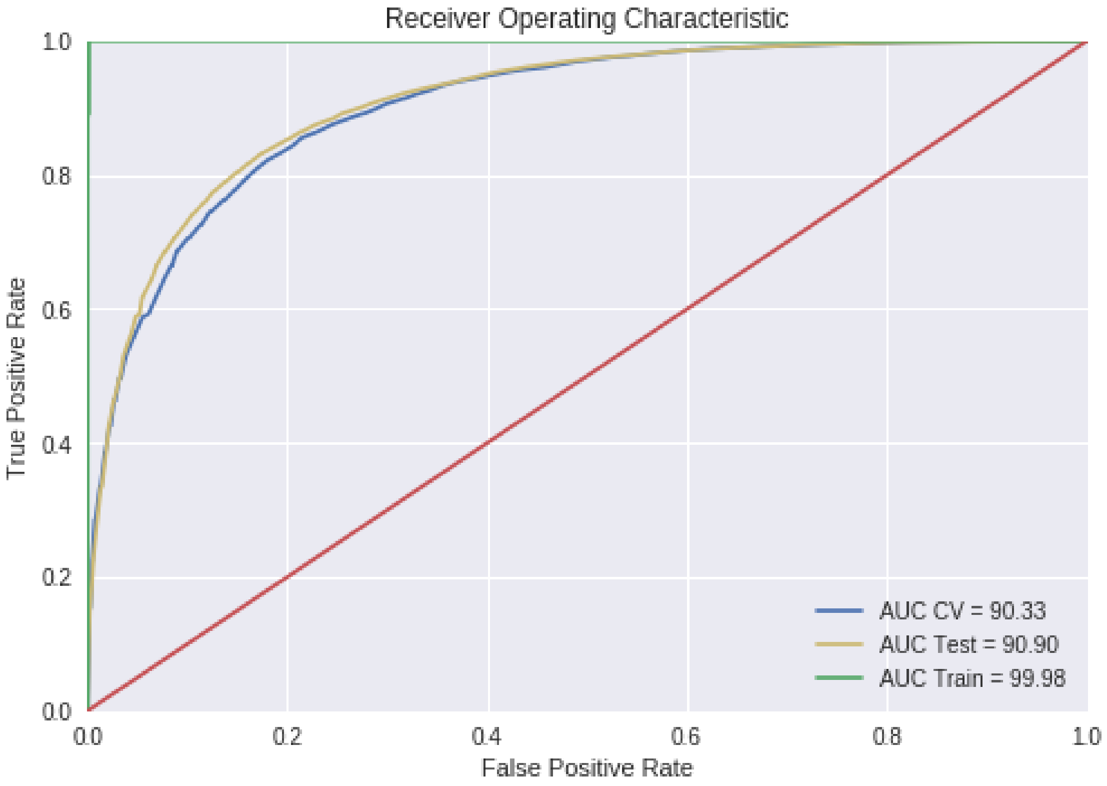
In [66]:

```
i = 120
j = 500

RFDTC = RandomForestClassifier(criterion='gini' , max_depth=j, n_estimators=i)
RFDTC.fit(bow_train, train_y)
# train data
y_prob_train = RFDTC.predict_proba(bow_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
# CV
y_prob_cv = RFDTC.predict_proba(bow_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i)
# Test
y_prob_test = RFDTC.predict_proba(bow_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
```

```
        auc_roc_test = roc_auc_score(test_y , y_prob_test)
        print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(
```

```
Train AUC for max depth = 500 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 120 is 90.33%

Test AUC for max_depth = 500 and n_estimators = 120 is 90.90%
```

```
In [67]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

         import matplotlib.pyplot as plt
         plt.clf()
         plt.title('Receiver Operating Characteristic')
         plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
         plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
         plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1],'r')
         plt.xlim([0, 1])
         plt.ylim([0, 1])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```

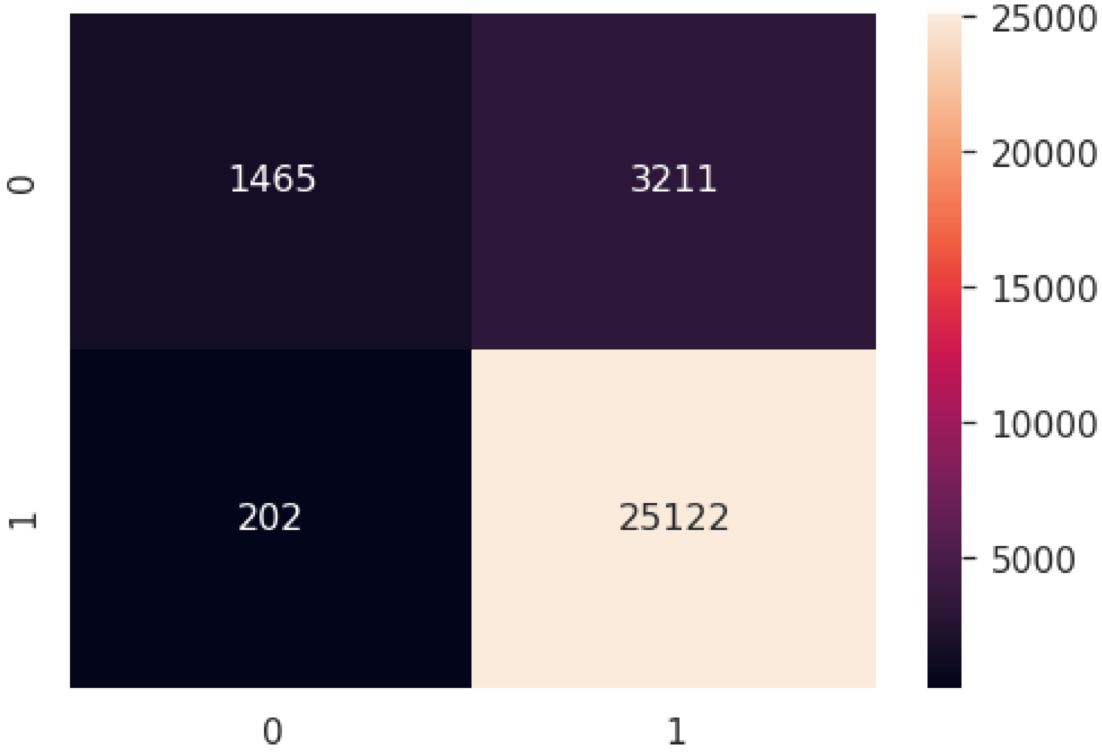


```
In [68]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.94
```

```
In [69]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
         sns.set(font_scale=1.4)
         sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x7f95705407b8>
```



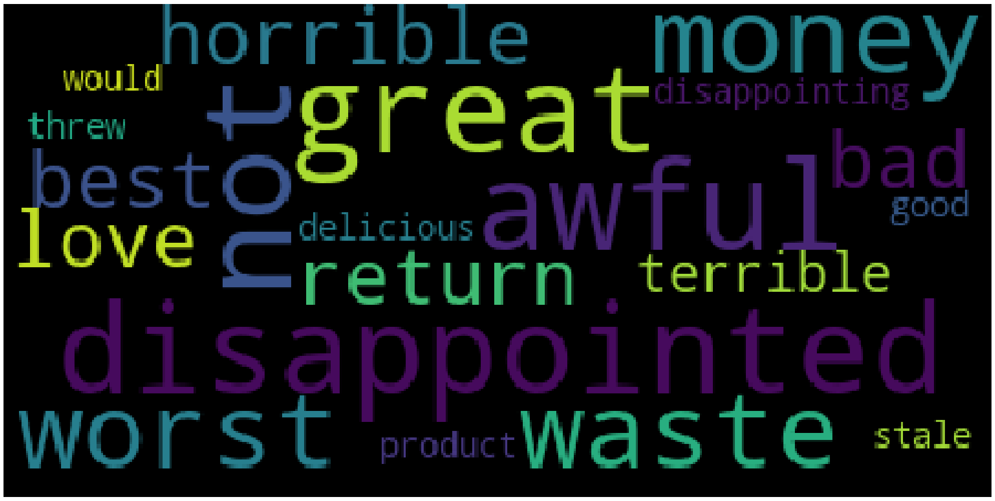
6.1.2 [5.1.2] Wordcloud of top 20 important features from SET 1

```
In [73]: # Please write all the code with proper documentation
         feture_imp = RFDTC.feature_importances_
         feature_names = count_vect.get_feature_names()
         features = dict(zip(feture_imp,feature_names))
         features_df = pd.DataFrame.from_dict(features, orient='index')
         features_df.columns = ["Words"]
         sorted_features = features_df.sort_index(axis=0,ascending=False )
```

```
In [76]: from wordcloud import WordCloud, STOPWORDS
```

```
In [102]: # https://www.kaggle.com/adiljadoon/word-cloud-with-python
          # top 20 important features
          top_20 = sorted_features.head(20)
          stopwords = set(STOPWORDS) - set(top_20["Words"])
          wordcloud = WordCloud(background_color='black',stopwords=stopwords,max_words=20,max_
          print(wordcloud)
          fig = plt.figure()
          plt.imshow(wordcloud)
          plt.axis('off')
          plt.show()
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f95851a59b0>
```



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

```
In [104]: # Please write all the code with proper documentation

          model = TfidfVectorizer(min_df=20, ngram_range=(1,2))
          #tf_idf_matrix = model.fit_transform(train)

          print("================Train Data===========")
          tf_idf_train = model.fit_transform(train)
          print("the type of count vectorizer ",type(tf_idf_train))
          print("the shape of out text TFIDF vectorizer ",tf_idf_train.get_shape())
          print("the number of unique words including both unigrams and bigrams ",tf_idf_train
          print("================CV Data===========")
          tf_idf_cv = model.transform(cv)
          print("the type of count vectorizer ",type(tf_idf_cv))
          print("the shape of out text TFIDF vectorizer ",tf_idf_cv.get_shape())
          print("the number of unique words including both unigrams and bigrams ",tf_idf_cv.ge
          print("================Test Data===========")
          tf_idf_test = model.transform(test)
          print("the type of count vectorizer ",type(tf_idf_test))
          print("the shape of out text TFIDF vectorizer ",tf_idf_test.get_shape())
          print("the number of unique words including both unigrams and bigrams ", tf_idf_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_)))
```

```
================Train Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (49000, 13653)
the number of unique words including both unigrams and bigrams  13653
================CV Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (21000, 13653)
the number of unique words including both unigrams and bigrams  13653
================Test Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (30000, 13653)
the number of unique words including both unigrams and bigrams  13653
```

```
In [105]: n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          tfidf_train_auc = []
          tfidf_cv_auc = []
          for i in n_estimators:
              for j in max_depth:
                  RFDTC = RandomForestClassifier(n_estimators=i,criterion='gini', max_depth=j)
                  RFDTC.fit(tf_idf_train, train_y)
                  # train data
                  y_prob_train = RFDTC.predict_proba(tf_idf_train)[:,1]
                  y_pred = np.where(y_prob_train > 0.5, 1, 0)
                  auc_roc_train = roc_auc_score(train_y , y_prob_train)
                  print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
                  tfidf_train_auc.append(auc_roc_train)
                  # CV
                  y_prob_cv = RFDTC.predict_proba(tf_idf_cv)[:,1]
                  y_pred = np.where(y_prob_cv > 0.5, 1, 0)
                  auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
                  print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
                  tfidf_cv_auc.append(auc_roc_cv)
                  print("="*50)
```

```
Train AUC for max_depth = 1 and n_estimators = 20 is 66.60%

CV AUC for max_depth = 1 and n_estimators = 20 is 65.91%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 83.63%

CV AUC for max_depth = 5 and n_estimators = 20 is 81.05%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 87.89%

CV AUC for max_depth = 10 and n_estimators = 20 is 85.10%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.79%

CV AUC for max_depth = 100 and n_estimators = 20 is 89.63%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.94%

CV AUC for max_depth = 500 and n_estimators = 20 is 89.85%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 20 is 89.95%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 74.33%

CV AUC for max_depth = 1 and n_estimators = 40 is 73.54%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 87.63%

CV AUC for max_depth = 5 and n_estimators = 40 is 85.07%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 90.58%

CV AUC for max_depth = 10 and n_estimators = 40 is 87.25%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.86%
```

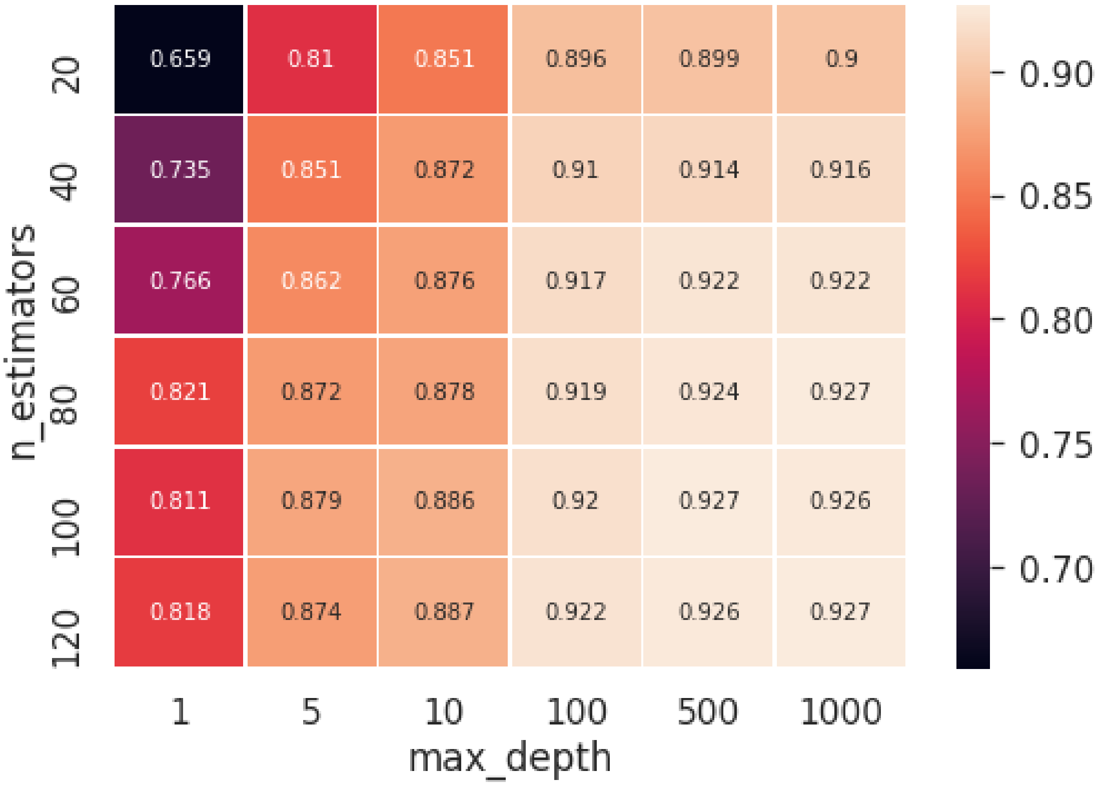
```
CV AUC for max_depth = 100 and n_estimators = 40 is 91.02%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.95%

CV AUC for max_depth = 500 and n_estimators = 40 is 91.36%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 40 is 91.60%
==================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 78.62%

CV AUC for max_depth = 1 and n_estimators = 60 is 76.60%
==================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 88.85%

CV AUC for max_depth = 5 and n_estimators = 60 is 86.23%
==================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 91.26%

CV AUC for max_depth = 10 and n_estimators = 60 is 87.62%
==================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 99.90%

CV AUC for max_depth = 100 and n_estimators = 60 is 91.67%
==================================================

Train AUC for max_depth = 500 and n_estimators = 60 is 99.97%

CV AUC for max_depth = 500 and n_estimators = 60 is 92.21%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 60 is 92.23%
==================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 83.59%

CV AUC for max_depth = 1 and n_estimators = 80 is 82.11%
==================================================
Train AUC for max_depth = 5 and n_estimators = 80 is 89.71%

CV AUC for max_depth = 5 and n_estimators = 80 is 87.24%
==================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 91.42%

CV AUC for max_depth = 10 and n_estimators = 80 is 87.81%
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 99.91%

CV AUC for max_depth = 100 and n_estimators = 80 is 91.89%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 99.96%

CV AUC for max_depth = 500 and n_estimators = 80 is 92.37%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 80 is 92.70%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 82.75%

CV AUC for max_depth = 1 and n_estimators = 100 is 81.13%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 89.91%

CV AUC for max_depth = 5 and n_estimators = 100 is 87.92%
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 91.92%

CV AUC for max_depth = 10 and n_estimators = 100 is 88.64%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 99.91%

CV AUC for max_depth = 100 and n_estimators = 100 is 91.96%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 99.97%

CV AUC for max_depth = 500 and n_estimators = 100 is 92.69%
```

```
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 100 is 92.59%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 83.56%

CV AUC for max_depth = 1 and n_estimators = 120 is 81.83%
==================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 89.63%

CV AUC for max_depth = 5 and n_estimators = 120 is 87.45%
==================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 92.08%

CV AUC for max_depth = 10 and n_estimators = 120 is 88.71%
==================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 99.91%

CV AUC for max_depth = 100 and n_estimators = 120 is 92.17%
==================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 99.96%

CV AUC for max_depth = 500 and n_estimators = 120 is 92.58%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 120 is 92.73%
==================================================
```

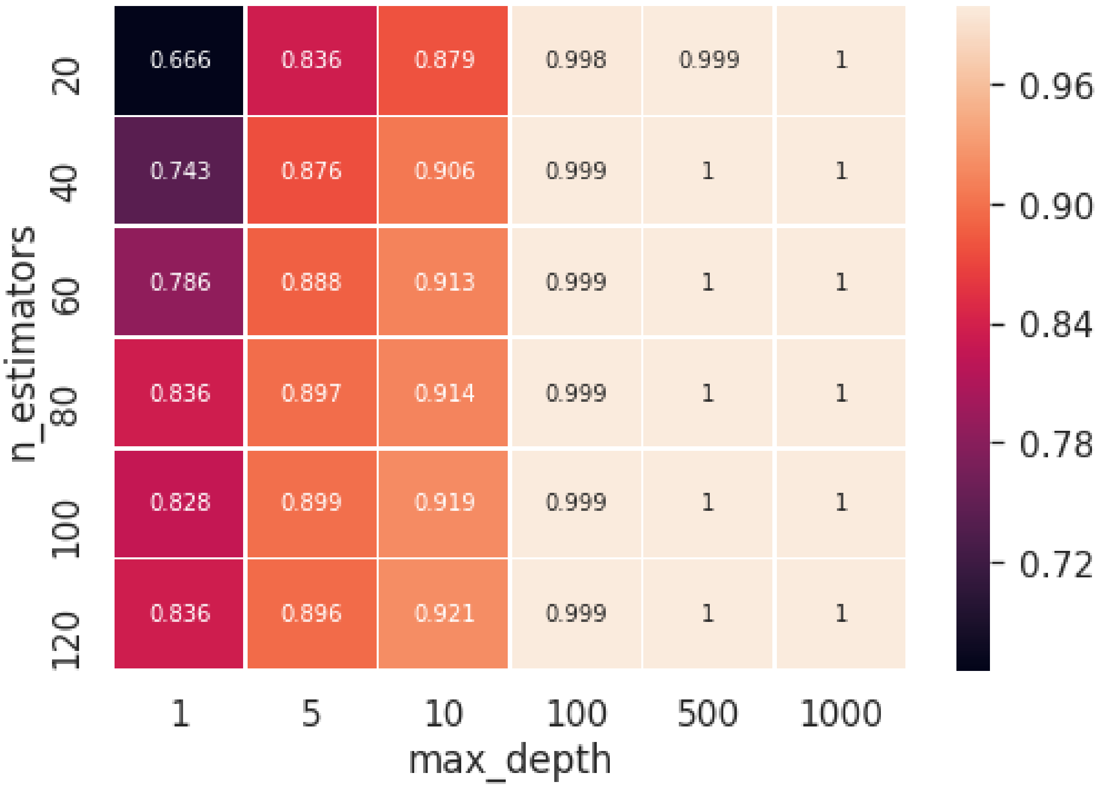
```
In [106]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(tfidf_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
```

```
print("==================Train Data================")
train_scores = np.array(tfidf_train_auc).reshape(len(n_estimators),len(max_depth))
df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
sns.set(font_scale=1.4)
ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidt
ax.set_xlabel("max_depth")
ax.set_ylabel("n_estimators")
plt.show()
```

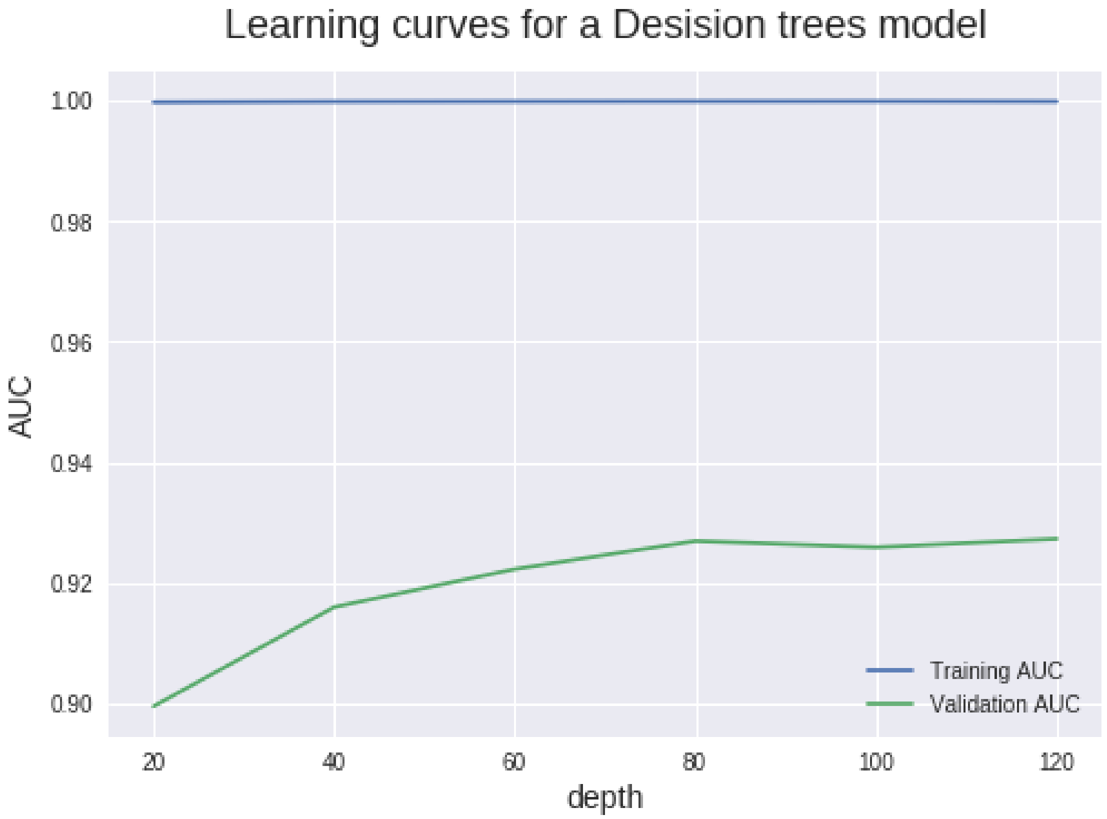
```
===================CV Data===================
```



```
==================Train Data================
```



```
In [107]: # depth is 1000
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-1],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-1], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[107]: <matplotlib.legend.Legend at 0x7f956f7525c0>
```

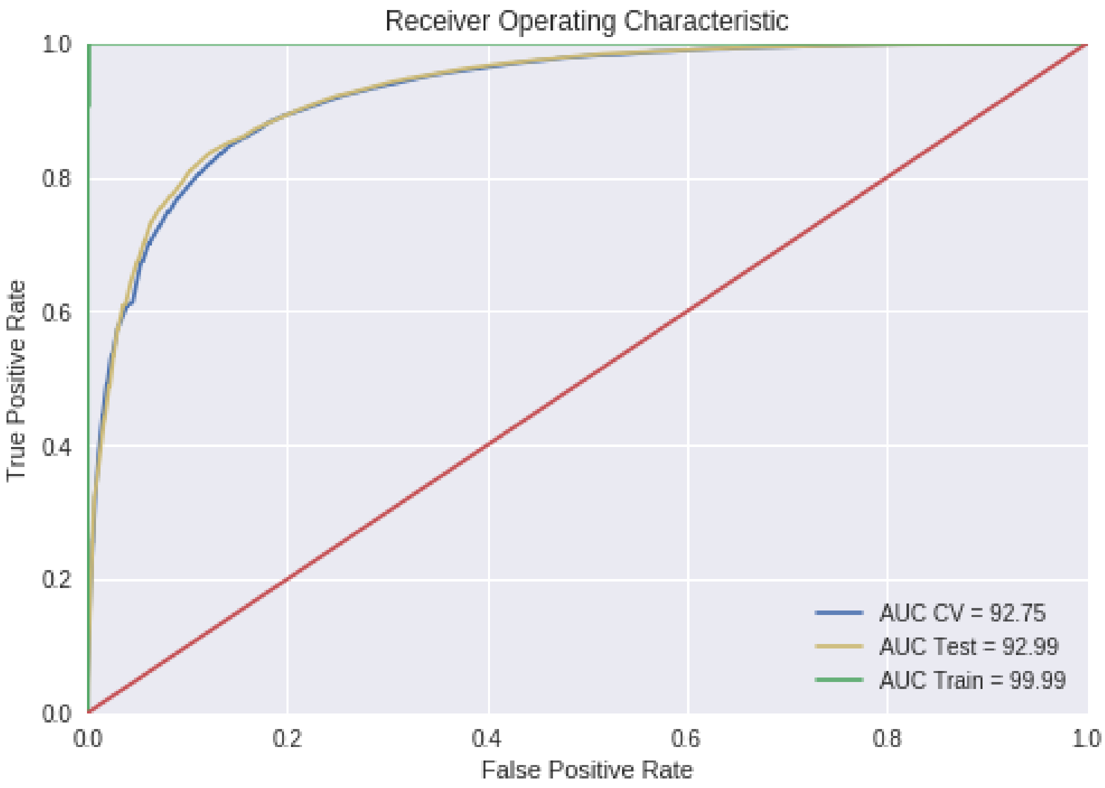


```
In [108]: i = 120
          j = 1000

          RFDTC = RandomForestClassifier(criterion='gini' , max_depth=j, n_estimators=i)
          RFDTC.fit(tf_idf_train, train_y)
          # train data
          y_prob_train = RFDTC.predict_proba(tf_idf_train)[:,1]
          fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
          y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
          auc_roc_train = roc_auc_score(train_y , y_prob_train)
          print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
          # CV
          y_prob_cv = RFDTC.predict_proba(tf_idf_cv)[:,1]
          fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
          y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
          auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
          print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
          # Test
          y_prob_test = RFDTC.predict_proba(tf_idf_test)[:,1]
          fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
          y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
```

```
        auc_roc_test = roc_auc_score(test_y , y_prob_test)
        print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

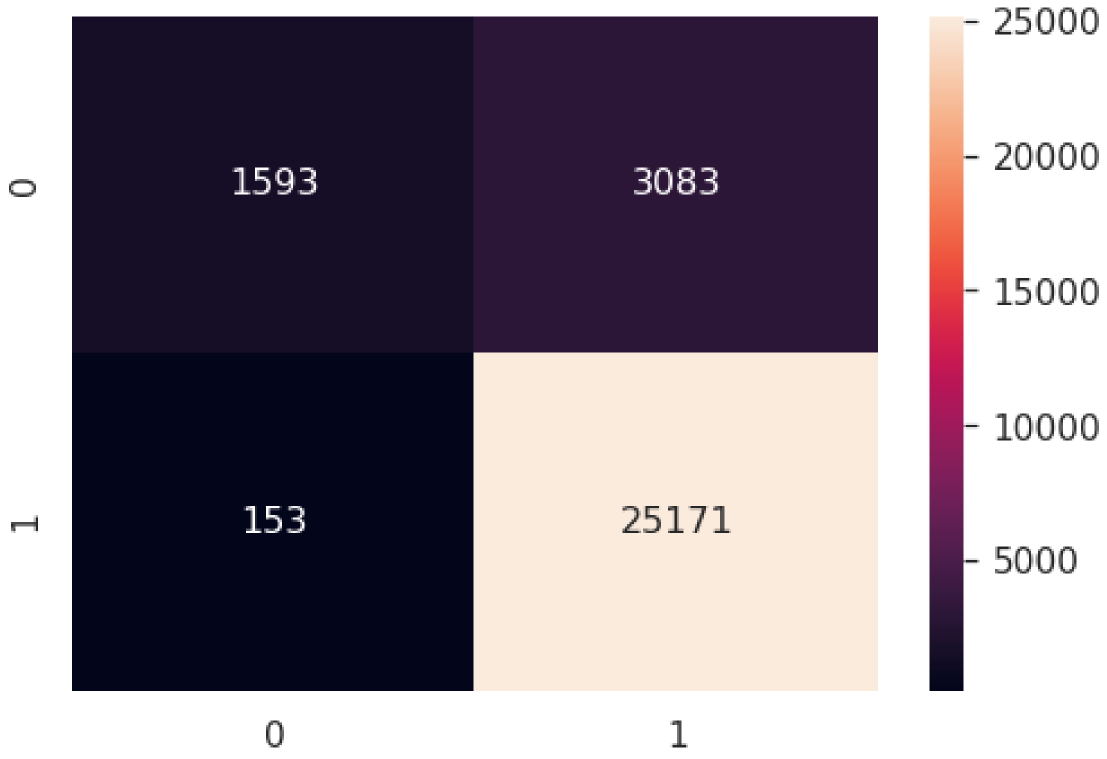
```
Train AUC for max depth = 1000 and n_estimators = 120 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 120 is 92.75%

Test AUC for max_depth = 1000 and n_estimators = 120 is 92.99%
```

```
In [109]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```



```
In [110]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))

F1-Score on test set: 0.94
```

```
In [112]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')

Out[112]: <matplotlib.axes._subplots.AxesSubplot at 0x7f95708e57b8>
```

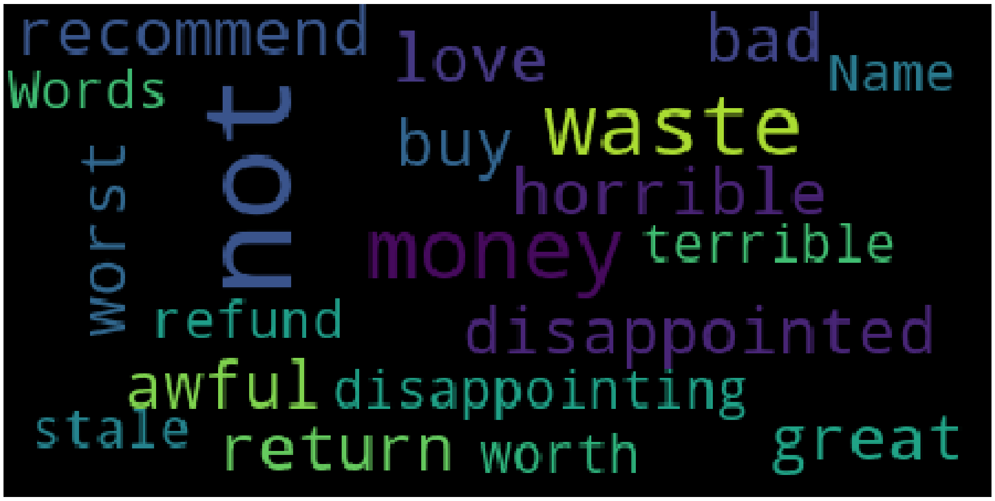


6.1.4 [5.1.4] Wordcloud of top 20 important features from SET 2

```
In [113]: # Please write all the code with proper documentation
          feture_imp = RFDTC.feature_importances_
          feature_names = model.get_feature_names()
          features = dict(zip(feture_imp,feature_names))
          features_df = pd.DataFrame.from_dict(features, orient='index')
          features_df.columns = ["Words"]
          sorted_features = features_df.sort_index(axis=0,ascending=False )
```

```
In [114]: # Please write all the code with proper documentation
          # https://www.kaggle.com/adiljadoon/word-cloud-with-python
          # top 20 important features
          top_20 = sorted_features.head(20)
          stopwords = set(STOPWORDS) - set(top_20["Words"])
          wordcloud = WordCloud(background_color='black',stopwords=stopwords,max_words=20,max_
          print(wordcloud)
          fig = plt.figure()
          plt.imshow(wordcloud)
          plt.axis('off')
          plt.show()
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f956a203b38>
```



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

```
In [115]: # Please write all the code with proper documentation
          # Train your own Word2Vec model using your own text corpus
          ######## Train Set #########
          i=0
          list_of_train_sentance=[]
          for sentance in train:
              list_of_train_sentance.append(sentance.split())
          ######## CV Set ###########
          i=0
          list_of_cv_sentance=[]
          for sentance in cv:
              list_of_cv_sentance.append(sentance.split())
          ######## Test Set #########
          i=0
          list_of_test_sentance=[]
          for sentance in test:
              list_of_test_sentance.append(sentance.split())
          print("Length of Train = ", len(list_of_train_sentance))
          print("Length of CV = ", len(list_of_cv_sentance))
          print("Length of Test = ", len(list_of_test_sentance))
```

```
Length of Train =  49000
Length of CV =  21000
Length of Test =  30000
```

```
In [116]: w2v_model=Word2Vec(list_of_train_sentance,min_count=15,size=100, workers=4)
          print(w2v_model.wv.most_similar('great'))
          print('='*50)
          print(w2v_model.wv.most_similar('worst'))
```

```
[('fantastic', 0.7984999418258667), ('wonderful', 0.7739488482475281), ('awesome', 0.7482377290
==================================================
[('greatest', 0.7162206172943115), ('tastiest', 0.7049369812011719), ('best', 0.700813770294189
```

```
In [117]: 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  7524
sample words  ['coffee', 'smells', 'great', 'first', 'thing', 'morning', 'occasionally', 'enjoy
```

```
In [118]: ######## Train data ########
          # average Word2Vec
          # compute average word2vec for each review.
          sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this lis
          for sent in tqdm(list_of_train_sentance): # for each review/sentence
              sent_vec = np.zeros(100) # 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_train.append(sent_vec)
          print(len(sent_vectors_train))
          print(len(sent_vectors_train[0]))
```

```
100%|| 49000/49000 [02:19<00:00, 351.43it/s]

49000
100
```

```
In [119]: ######## CV data ########
          # average Word2Vec
          # compute average word2vec for each review.
          sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
          for sent in tqdm(list_of_cv_sentance): # for each review/sentence
              sent_vec = np.zeros(100) # 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_cv.append(sent_vec)
          print(len(sent_vectors_cv))
          print(len(sent_vectors_cv[0]))
```

```
100%|| 21000/21000 [00:58<00:00, 357.99it/s]

21000
100
```

```
In [120]: ######## Test data ########
          # average Word2Vec
          # compute average word2vec for each review.
          sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this lis
          for sent in tqdm(list_of_test_sentance): # for each review/sentence
              sent_vec = np.zeros(100) # 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_test.append(sent_vec)
          print(len(sent_vectors_test))
          print(len(sent_vectors_test[0]))
```

```
100%|| 30000/30000 [01:24<00:00, 353.24it/s]

30000
100
```

```
In [121]: # save the datasets as numpy array
          w2v_train = np.array(sent_vectors_train)
          w2v_cv = np.array(sent_vectors_cv)
          w2v_test = np.array(sent_vectors_test)
```

```
In [122]: n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          w2v_train_auc = []
          w2v_cv_auc = []
          for i in n_estimators:
              for j in max_depth:
                  RFDTC = RandomForestClassifier(n_estimators=i,criterion='gini', max_depth=j)
                  RFDTC.fit(w2v_train, train_y)
                  # train data
                  y_prob_train = RFDTC.predict_proba(w2v_train)[:,1]
                  y_pred = np.where(y_prob_train > 0.5, 1, 0)
                  auc_roc_train = roc_auc_score(train_y , y_prob_train)
                  print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
                  w2v_train_auc.append(auc_roc_train)
                  # CV
                  y_prob_cv = RFDTC.predict_proba(w2v_cv)[:,1]
                  y_pred = np.where(y_prob_cv > 0.5, 1, 0)
                  auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
                  print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
                  w2v_cv_auc.append(auc_roc_cv)
                  print("="*50)
```

```
Train AUC for max_depth = 1 and n_estimators = 20 is 82.38%

CV AUC for max_depth = 1 and n_estimators = 20 is 81.93%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 88.58%
```

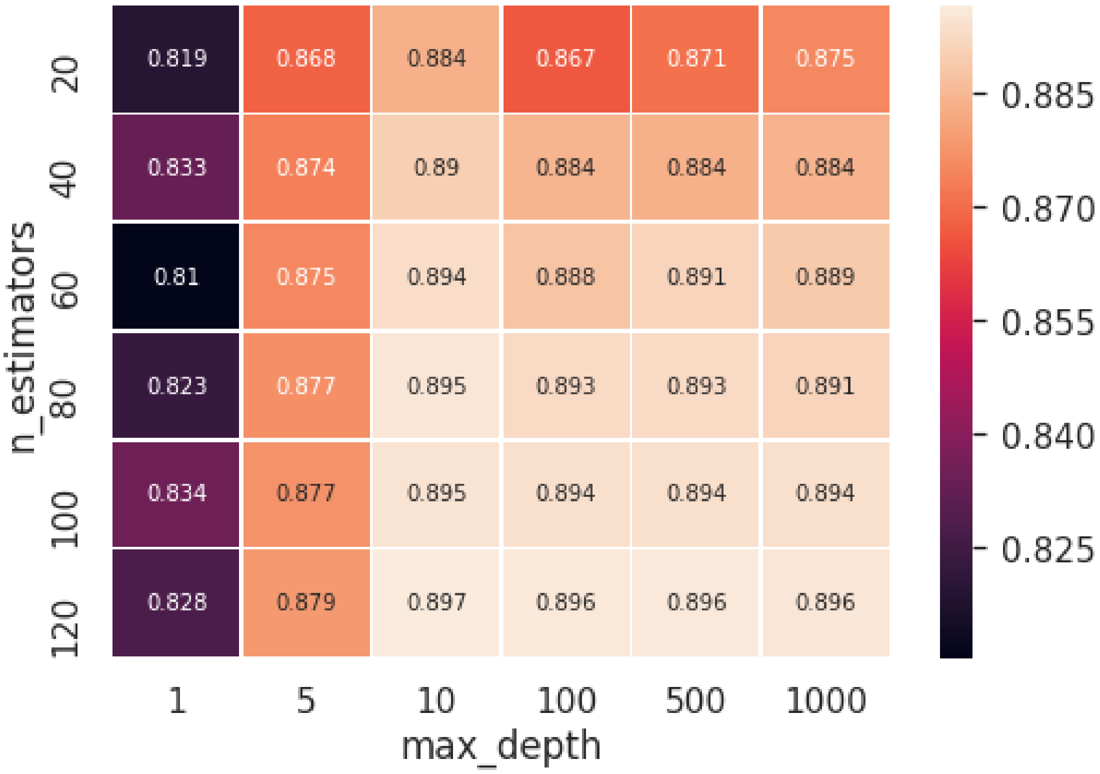
```
CV AUC for max_depth = 5 and n_estimators = 20 is 86.83%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 96.68%

CV AUC for max_depth = 10 and n_estimators = 20 is 88.37%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 100 and n_estimators = 20 is 86.66%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 500 and n_estimators = 20 is 87.10%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 1000 and n_estimators = 20 is 87.54%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 83.84%

CV AUC for max_depth = 1 and n_estimators = 40 is 83.33%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 88.93%

CV AUC for max_depth = 5 and n_estimators = 40 is 87.36%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 96.97%

CV AUC for max_depth = 10 and n_estimators = 40 is 89.00%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 40 is 88.43%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 40 is 88.37%
==================================================
Train AUC for max_depth = 1000 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 40 is 88.42%
===================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 81.69%

CV AUC for max_depth = 1 and n_estimators = 60 is 81.05%
===================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 89.12%

CV AUC for max_depth = 5 and n_estimators = 60 is 87.53%
===================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 97.15%

CV AUC for max_depth = 10 and n_estimators = 60 is 89.36%
===================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 60 is 88.79%
===================================================

Train AUC for max_depth = 500 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 60 is 89.14%
===================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 60 is 88.91%
===================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 82.84%

CV AUC for max_depth = 1 and n_estimators = 80 is 82.31%
===================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 89.38%

CV AUC for max_depth = 5 and n_estimators = 80 is 87.67%
===================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 97.24%

CV AUC for max_depth = 10 and n_estimators = 80 is 89.54%
```

```
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 80 is 89.30%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 80 is 89.28%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 99.97%

CV AUC for max_depth = 1000 and n_estimators = 80 is 89.14%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 84.06%

CV AUC for max_depth = 1 and n_estimators = 100 is 83.45%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 89.33%

CV AUC for max_depth = 5 and n_estimators = 100 is 87.75%
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 97.23%

CV AUC for max_depth = 10 and n_estimators = 100 is 89.48%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 100 is 89.43%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 100 is 89.42%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 100 is 89.42%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 83.27%
```

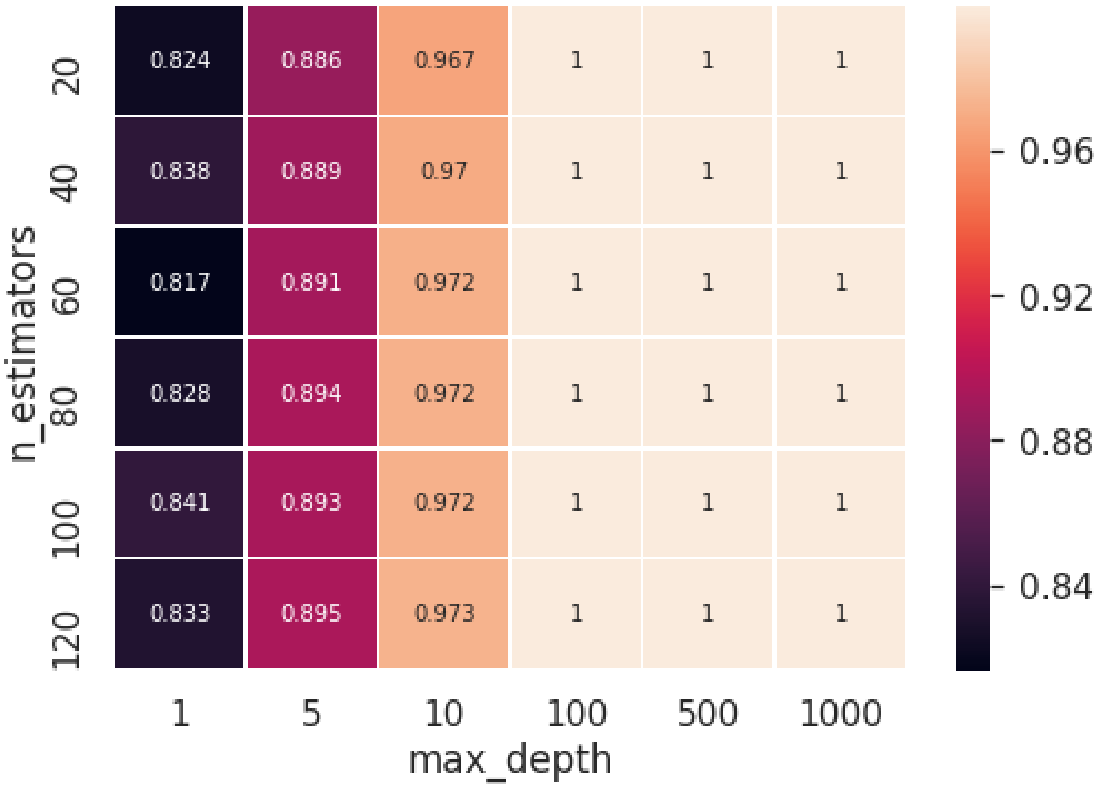
```
CV AUC for max_depth = 1 and n_estimators = 120 is 82.76%
==================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 89.48%

CV AUC for max_depth = 5 and n_estimators = 120 is 87.88%
==================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 97.32%

CV AUC for max_depth = 10 and n_estimators = 120 is 89.67%
==================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 120 is 89.57%
==================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 120 is 89.62%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 120 is 89.55%
==================================================
```

```
In [123]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(w2v_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
          print("===================Train Data================")
          train_scores = np.array(w2v_train_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidt
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
```

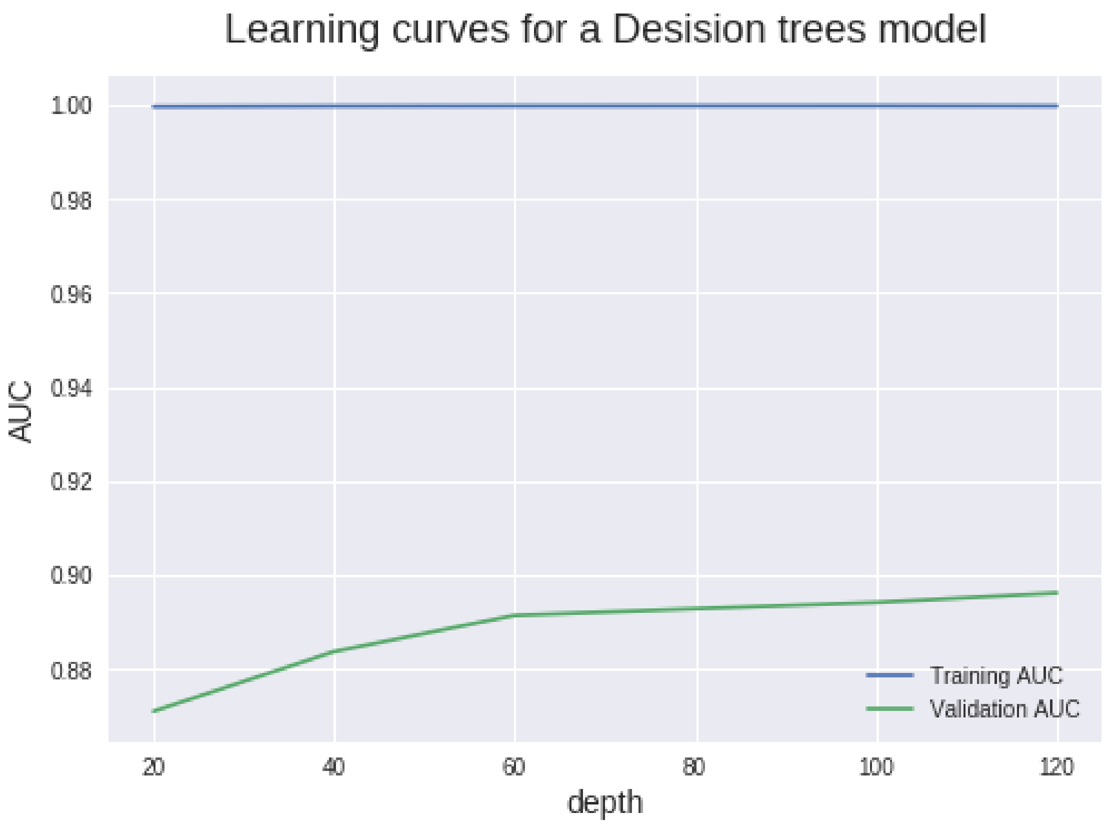
```
==================CV Data==================
```



```
==================Train Data================
```



```
In [124]: # depth is 500
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-2],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-2], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[124]: <matplotlib.legend.Legend at 0x7f955020e588>
```

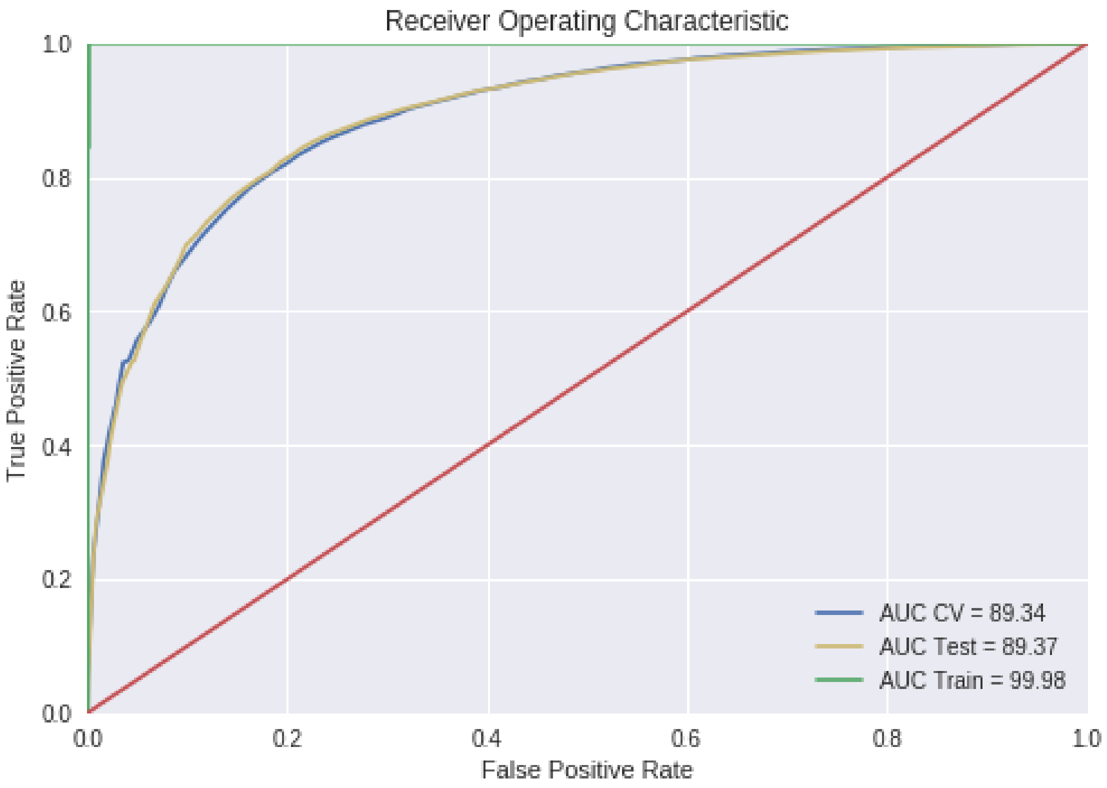


In [125]:
```
i = 120
j = 500

RFDTC = RandomForestClassifier(criterion='gini' , max_depth=j, n_estimators=i)
RFDTC.fit(w2v_train, train_y)
# train data
y_prob_train = RFDTC.predict_proba(w2v_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
# CV
y_prob_cv = RFDTC.predict_proba(w2v_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
# Test
y_prob_test = RFDTC.predict_proba(w2v_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
```

```
        auc_roc_test = roc_auc_score(test_y , y_prob_test)
        print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

```
Train AUC for max depth = 500 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 120 is 89.34%

Test AUC for max_depth = 500 and n_estimators = 120 is 89.37%
```

```
In [126]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```

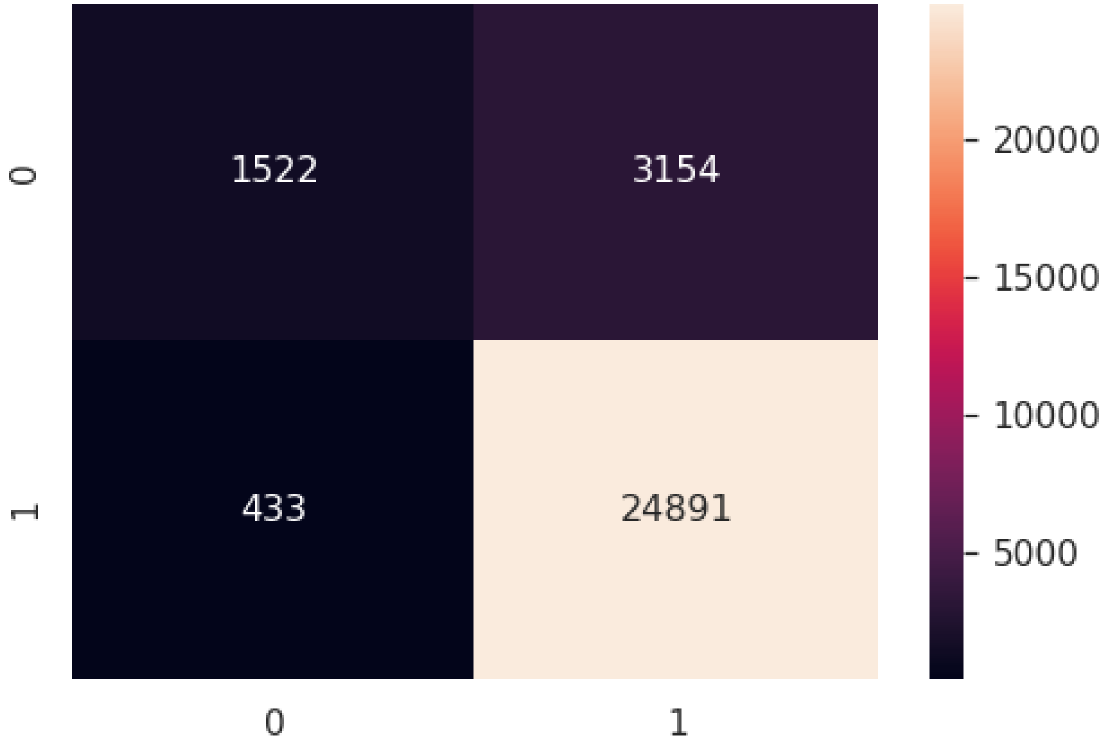


```
In [127]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.93
```

```
In [128]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[128]: <matplotlib.axes._subplots.AxesSubplot at 0x7f95501eff60>
```



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

```
In [129]: # Please write all the code with proper documentation

          model = TfidfVectorizer()
          #tf_idf_matrix = model.fit_transform(train)

          print("================Train Data===========")
          final_tf_idf_train = model.fit_transform(train)
```

```
        print("the type of count vectorizer ",type(final_tf_idf_train))
        print("the shape of out text TFIDF vectorizer ",final_tf_idf_train.get_shape())
        print("the number of unique words including both unigrams and bigrams ", final_tf_id
        print("================CV Data===========")
        final_tf_idf_cv = model.transform(cv)
        print("the type of count vectorizer ",type(final_tf_idf_cv))
        print("the shape of out text TFIDF vectorizer ",final_tf_idf_cv.get_shape())
        print("the number of unique words including both unigrams and bigrams ", final_tf_id
        print("================Test Data===========")
        final_tf_idf_test = model.transform(test)
        print("the type of count vectorizer ",type(final_tf_idf_test))
        print("the shape of out text TFIDF vectorizer ",final_tf_idf_test.get_shape())
        print("the number of unique words including both unigrams and bigrams ", final_tf_id

        # 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_)))
```

```
================Train Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (49000, 42656)
the number of unique words including both unigrams and bigrams  42656
================CV Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (21000, 42656)
the number of unique words including both unigrams and bigrams  42656
================Test Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (30000, 42656)
the number of unique words including both unigrams and bigrams  42656
```

```
In [130]: ######## Train #######
          # 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 = tfid

          train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
          row=0;
          for sent in tqdm(list_of_train_sentance): # for each review/sentence
              sent_vec = np.zeros(100) # 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
    train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|| 49000/49000 [38:08<00:00, 21.41it/s]
```

```
In [131]: ######## CV #######
          # 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 = tfid

          cv_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
          row=0;
          for sent in tqdm(list_of_cv_sentance): # for each review/sentence
              sent_vec = np.zeros(100) # 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
              cv_tfidf_sent_vectors.append(sent_vec)
              row += 1
```

```
100%|| 21000/21000 [2:55:38<00:00,  1.99it/s]
```

```
In [132]: ######## Train #######
          # 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 = tfid

          test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
          row=0;
          for sent in tqdm(list_of_test_sentance): # for each review/sentence
              sent_vec = np.zeros(100) # 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
    test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|| 30000/30000 [23:37<00:00, 21.17it/s]
```

```
In [133]: # save the datasets as numpy array
          tfidf_w2v_train = np.array(train_tfidf_sent_vectors)
          tfidf_w2v_cv = np.array(cv_tfidf_sent_vectors)
          tfidf_w2v_test = np.array(test_tfidf_sent_vectors)
```

```
In [134]: n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          tfidf_w2v_train_auc = []
          tfidf_w2v_cv_auc = []
          for i in n_estimators:
              for j in max_depth:
                  RFDTC = RandomForestClassifier(n_estimators=i,criterion='gini', max_depth=j)
                  RFDTC.fit(tfidf_w2v_train, train_y)
                  # train data
                  y_prob_train = RFDTC.predict_proba(tfidf_w2v_train)[:,1]
                  y_pred = np.where(y_prob_train > 0.5, 1, 0)
                  auc_roc_train = roc_auc_score(train_y , y_prob_train)
                  print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
                  tfidf_w2v_train_auc.append(auc_roc_train)
                  # CV
                  y_prob_cv = RFDTC.predict_proba(tfidf_w2v_cv)[:,1]
                  y_pred = np.where(y_prob_cv > 0.5, 1, 0)
                  auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
                  print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
                  tfidf_w2v_cv_auc.append(auc_roc_cv)
                  print("="*50)
```

```

Train AUC for max_depth = 1 and n_estimators = 20 is 78.71%

CV AUC for max_depth = 1 and n_estimators = 20 is 78.54%
```

```
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 85.61%

CV AUC for max_depth = 5 and n_estimators = 20 is 84.12%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 95.73%

CV AUC for max_depth = 10 and n_estimators = 20 is 86.15%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.96%

CV AUC for max_depth = 100 and n_estimators = 20 is 84.52%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.96%

CV AUC for max_depth = 500 and n_estimators = 20 is 84.66%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.96%

CV AUC for max_depth = 1000 and n_estimators = 20 is 84.36%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 80.89%

CV AUC for max_depth = 1 and n_estimators = 40 is 80.80%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 86.48%

CV AUC for max_depth = 5 and n_estimators = 40 is 85.04%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 96.23%

CV AUC for max_depth = 10 and n_estimators = 40 is 86.73%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.97%

CV AUC for max_depth = 100 and n_estimators = 40 is 86.10%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.97%
```

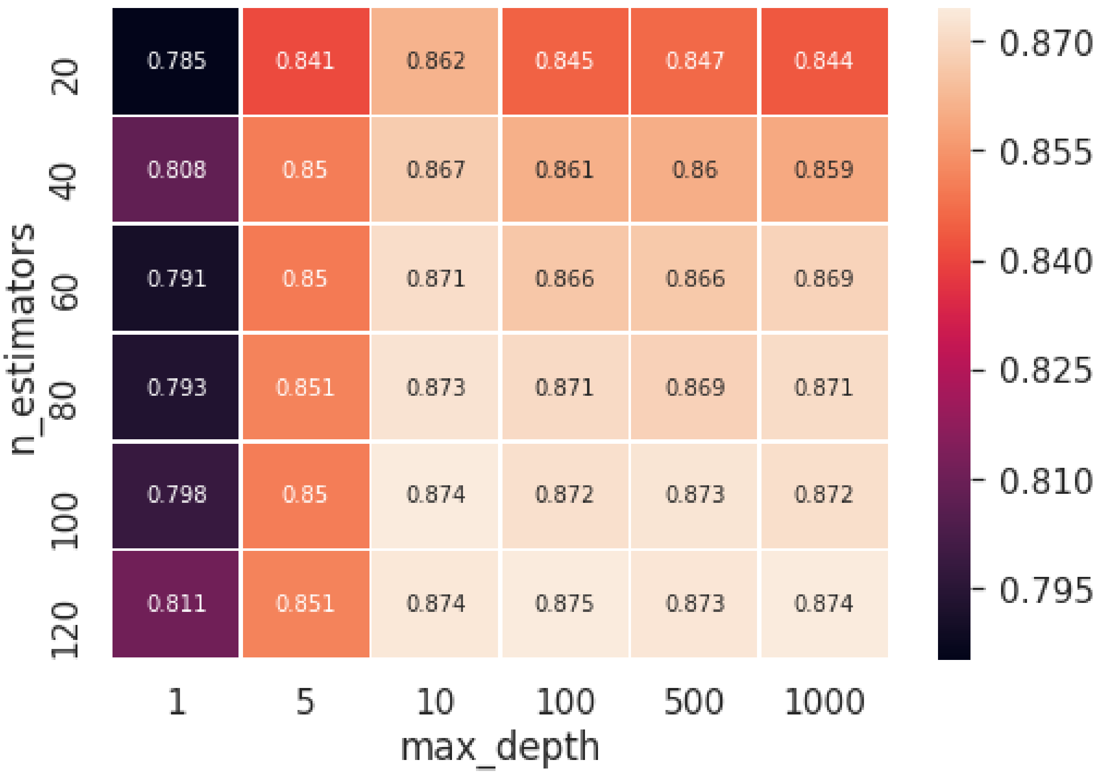
```
CV AUC for max_depth = 500 and n_estimators = 40 is 86.04%
===================================================

Train AUC for max_depth = 1000 and n_estimators = 40 is 99.97%

CV AUC for max_depth = 1000 and n_estimators = 40 is 85.94%
===================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 78.85%

CV AUC for max_depth = 1 and n_estimators = 60 is 79.08%
===================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 86.46%

CV AUC for max_depth = 5 and n_estimators = 60 is 84.97%
===================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 96.36%

CV AUC for max_depth = 10 and n_estimators = 60 is 87.13%
===================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 60 is 86.60%
===================================================

Train AUC for max_depth = 500 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 60 is 86.63%
===================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 60 is 86.87%
===================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 79.45%

CV AUC for max_depth = 1 and n_estimators = 80 is 79.33%
===================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 86.51%

CV AUC for max_depth = 5 and n_estimators = 80 is 85.13%
===================================================
Train AUC for max_depth = 10 and n_estimators = 80 is 96.42%

CV AUC for max_depth = 10 and n_estimators = 80 is 87.27%
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 99.97%

CV AUC for max_depth = 100 and n_estimators = 80 is 87.07%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 99.97%

CV AUC for max_depth = 500 and n_estimators = 80 is 86.85%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 99.97%

CV AUC for max_depth = 1000 and n_estimators = 80 is 87.11%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 79.81%

CV AUC for max_depth = 1 and n_estimators = 100 is 79.84%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 86.57%

CV AUC for max_depth = 5 and n_estimators = 100 is 85.03%
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 96.42%

CV AUC for max_depth = 10 and n_estimators = 100 is 87.44%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 100 is 87.22%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 100 is 87.30%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 99.98%
```

```
CV AUC for max_depth = 1000 and n_estimators = 100 is 87.16%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 81.13%

CV AUC for max_depth = 1 and n_estimators = 120 is 81.09%
==================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 86.73%

CV AUC for max_depth = 5 and n_estimators = 120 is 85.14%
==================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 96.56%

CV AUC for max_depth = 10 and n_estimators = 120 is 87.39%
==================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 120 is 87.47%
==================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 500 and n_estimators = 120 is 87.34%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 1000 and n_estimators = 120 is 87.44%
==================================================
```

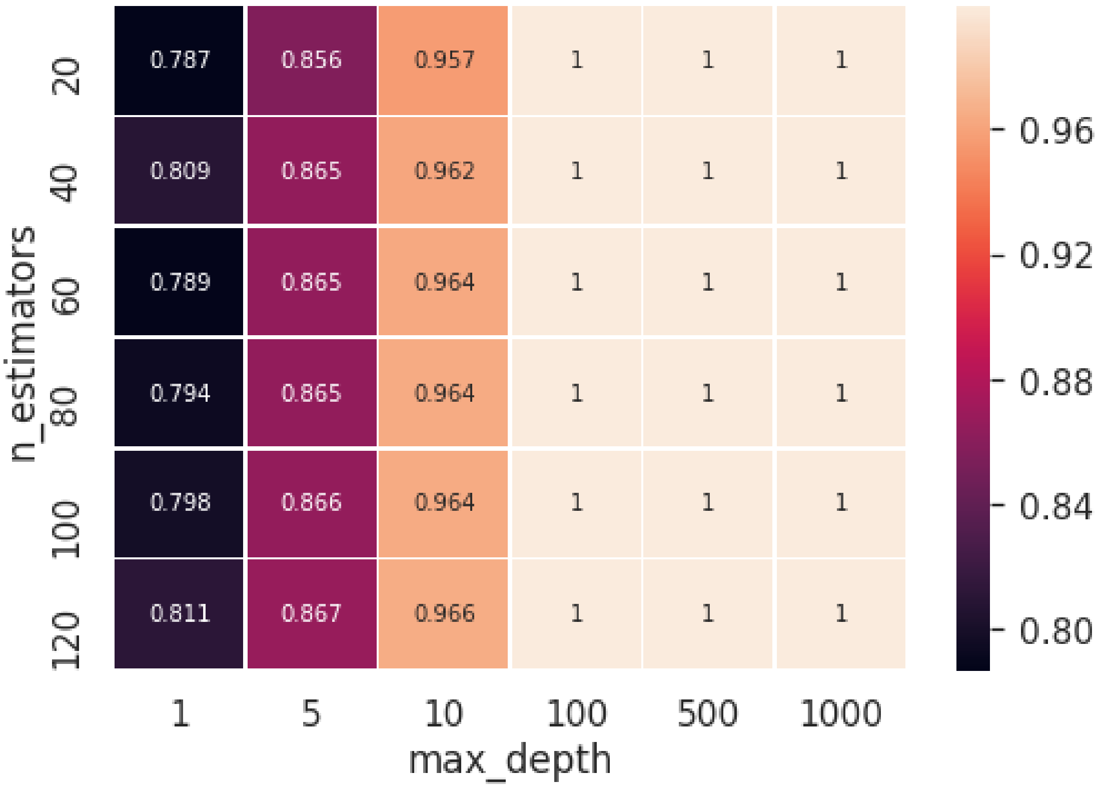
```
In [135]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(tfidf_w2v_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
          print("===================Train Data================")
          train_scores = np.array(tfidf_w2v_train_auc).reshape(len(n_estimators),len(max_depth
          df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
```

```
        ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidt
        ax.set_xlabel("max_depth")
        ax.set_ylabel("n_estimators")
        plt.show()
```

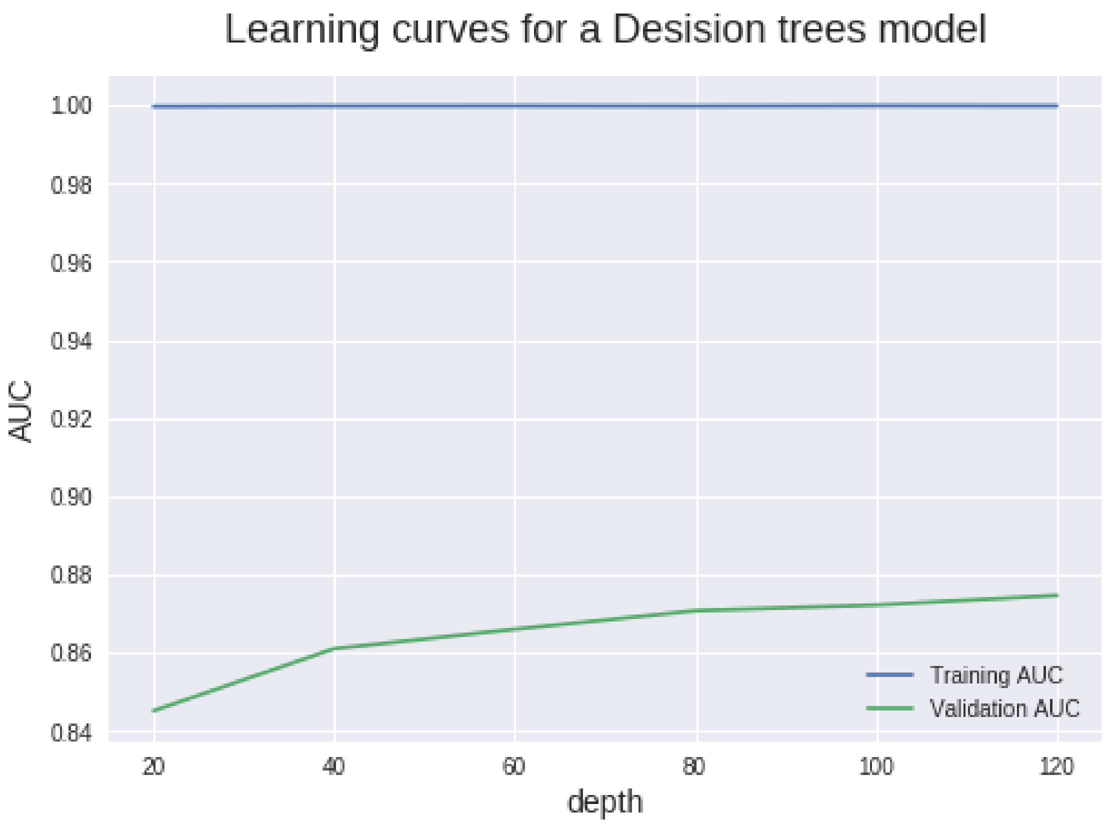
```
===================CV Data===================
```



```
===================Train Data================
```



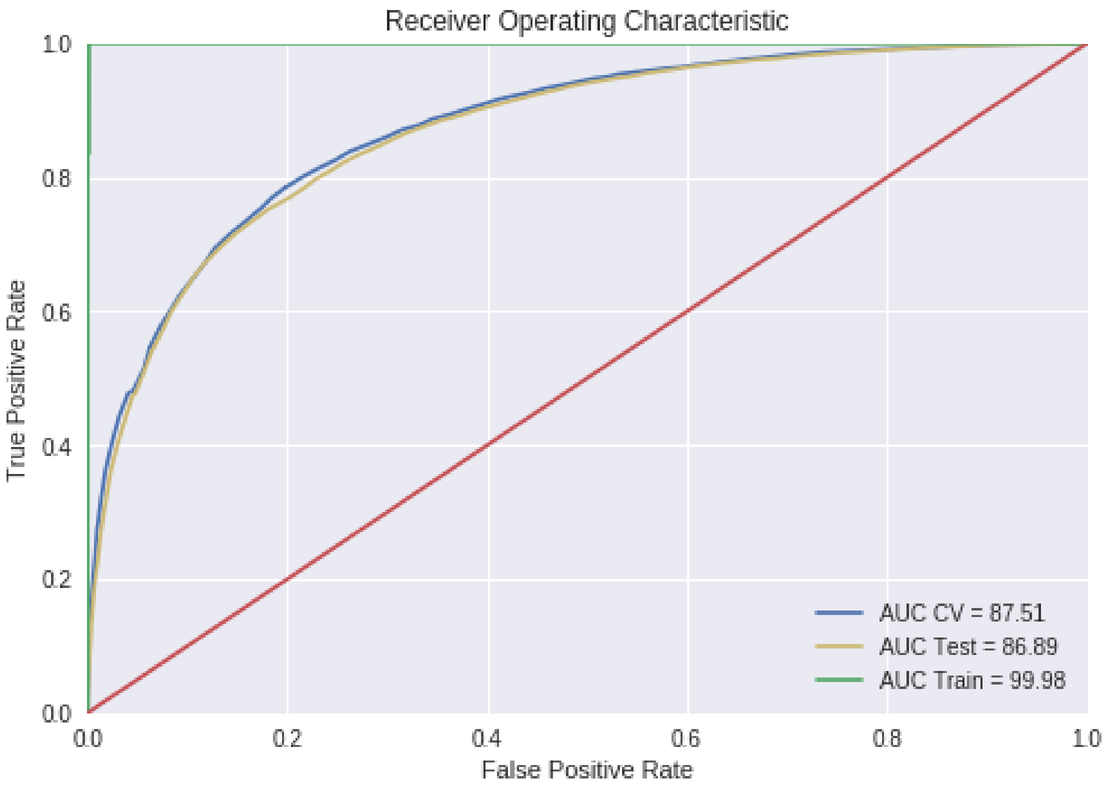
```
In [136]: # depth is 100
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-3],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-3], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[136]: <matplotlib.legend.Legend at 0x7f95506dd080>
```



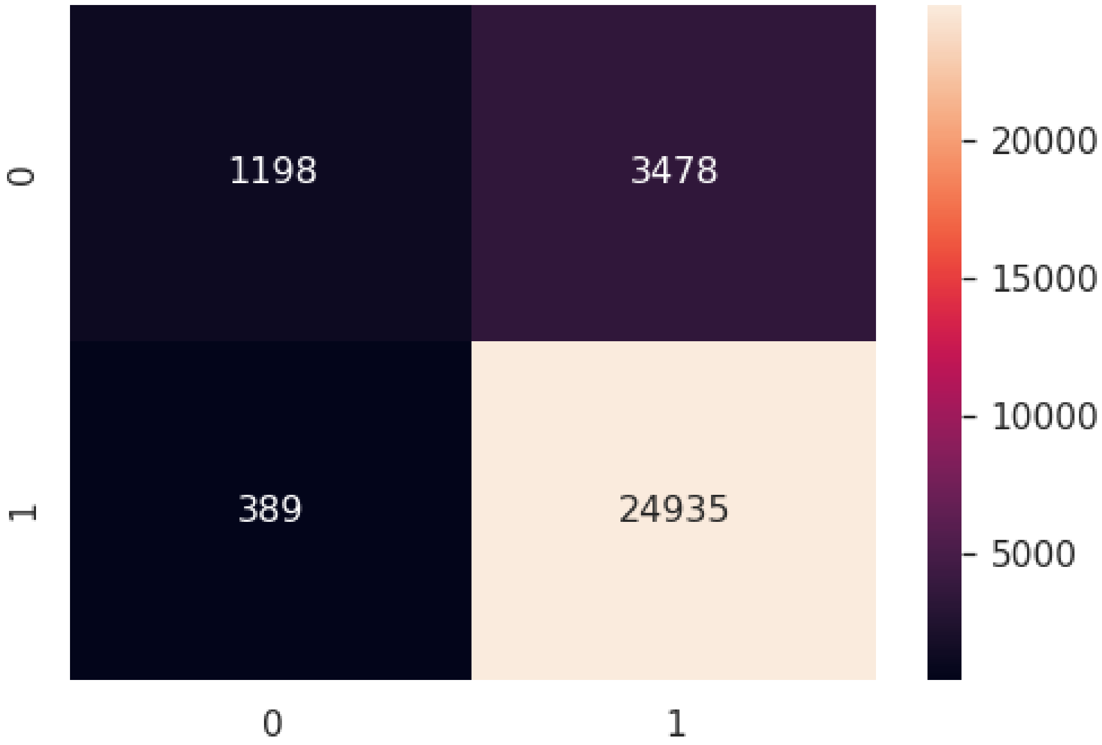
```
In [138]: i = 120
          j = 100

          RFDTC = RandomForestClassifier(criterion='gini' , max_depth=j, n_estimators=i)
          RFDTC.fit(tfidf_w2v_train, train_y)
          # train data
          y_prob_train = RFDTC.predict_proba(tfidf_w2v_train)[:,1]
          fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
          y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
          auc_roc_train = roc_auc_score(train_y , y_prob_train)
          print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
          # CV
          y_prob_cv = RFDTC.predict_proba(tfidf_w2v_cv)[:,1]
          fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
          y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
          auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
          print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
          # Test
          y_prob_test = RFDTC.predict_proba(tfidf_w2v_test)[:,1]
          fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
          y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
          auc_roc_test = roc_auc_score(test_y , y_prob_test)
          print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

```
Train AUC for max depth = 100 and n_estimators = 120 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 120 is 87.51%

Test AUC for max_depth = 100 and n_estimators = 120 is 86.89%
```

```
In [139]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```



```
In [140]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))

F1-Score on test set: 0.93


In [141]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')

Out[141]: <matplotlib.axes._subplots.AxesSubplot at 0x7f956f14fb00>
```



6.2 [5.2] Applying GBDT using XGBOOST

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

```
In [142]: # Please write all the code with proper documentation
          from xgboost import XGBClassifier

In [143]: n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          bow_train_auc = []
```

```
bow_cv_auc = []
for i in n_estimators:
    for j in max_depth:
        XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1)
        XGBC.fit(bow_train, train_y)
        # train data
        y_prob_train = XGBC.predict_proba(bow_train)[:,1]
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
        bow_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = XGBC.predict_proba(bow_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
        bow_cv_auc.append(auc_roc_cv)
        print("="*50)
```

```
Train AUC for max_depth = 1 and n_estimators = 20 is 73.37%

CV AUC for max_depth = 1 and n_estimators = 20 is 71.95%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 83.21%

CV AUC for max_depth = 5 and n_estimators = 20 is 80.78%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 89.53%

CV AUC for max_depth = 10 and n_estimators = 20 is 84.23%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.66%

CV AUC for max_depth = 100 and n_estimators = 20 is 87.45%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.69%

CV AUC for max_depth = 500 and n_estimators = 20 is 87.24%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.69%

CV AUC for max_depth = 1000 and n_estimators = 20 is 87.24%
```

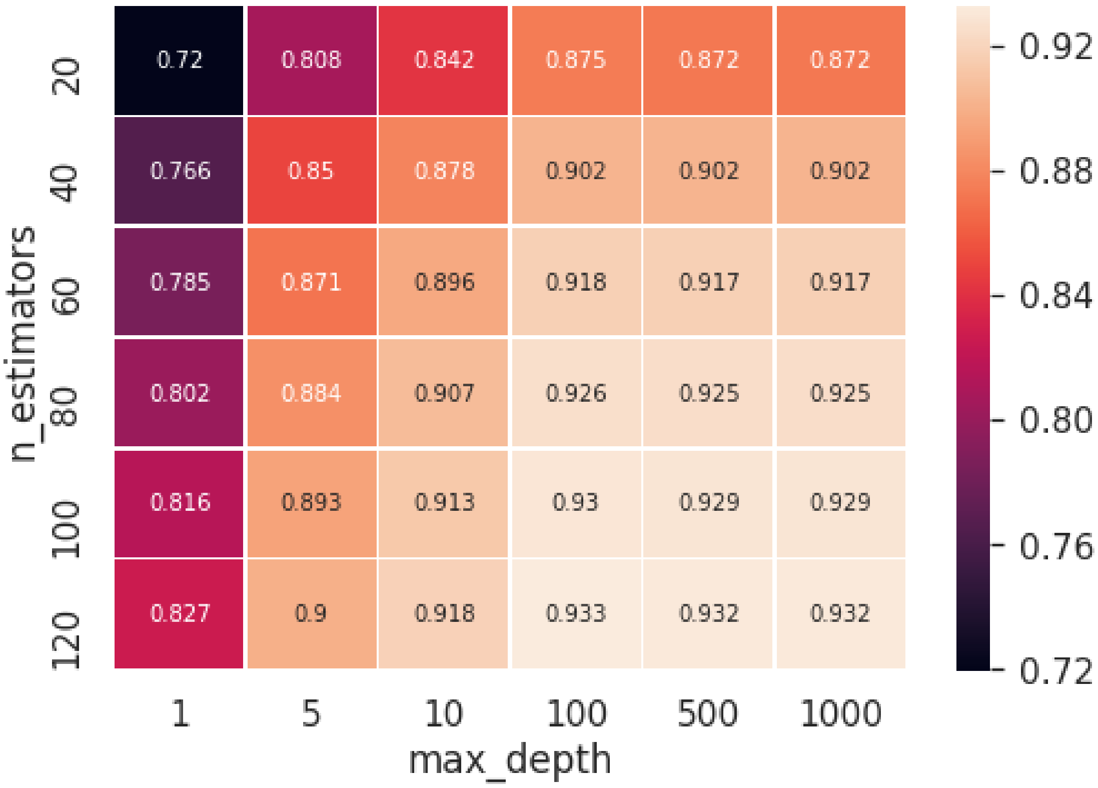
```
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 77.93%

CV AUC for max_depth = 1 and n_estimators = 40 is 76.61%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 87.68%

CV AUC for max_depth = 5 and n_estimators = 40 is 85.03%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 93.35%

CV AUC for max_depth = 10 and n_estimators = 40 is 87.85%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.94%

CV AUC for max_depth = 100 and n_estimators = 40 is 90.23%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.95%

CV AUC for max_depth = 500 and n_estimators = 40 is 90.22%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 40 is 99.95%

CV AUC for max_depth = 1000 and n_estimators = 40 is 90.22%
==================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 79.92%

CV AUC for max_depth = 1 and n_estimators = 60 is 78.48%
==================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 89.89%

CV AUC for max_depth = 5 and n_estimators = 60 is 87.05%
==================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 95.20%

CV AUC for max_depth = 10 and n_estimators = 60 is 89.64%
==================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 99.98%
```

```
CV AUC for max_depth = 100 and n_estimators = 60 is 91.78%
==================================================

Train AUC for max_depth = 500 and n_estimators = 60 is 99.99%

CV AUC for max_depth = 500 and n_estimators = 60 is 91.72%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 60 is 91.72%
==================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 81.63%

CV AUC for max_depth = 1 and n_estimators = 80 is 80.20%
==================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 91.38%

CV AUC for max_depth = 5 and n_estimators = 80 is 88.39%
==================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 96.26%

CV AUC for max_depth = 10 and n_estimators = 80 is 90.66%
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 100 and n_estimators = 80 is 92.59%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 500 and n_estimators = 80 is 92.50%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 80 is 92.50%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 83.04%

CV AUC for max_depth = 1 and n_estimators = 100 is 81.55%
==================================================
Train AUC for max_depth = 5 and n_estimators = 100 is 92.43%

CV AUC for max_depth = 5 and n_estimators = 100 is 89.30%
===================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 96.96%

CV AUC for max_depth = 10 and n_estimators = 100 is 91.33%
===================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 100 is 93.02%
===================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 100 is 92.91%
===================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 100 is 92.91%
===================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 83.97%

CV AUC for max_depth = 1 and n_estimators = 120 is 82.70%
===================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 93.22%

CV AUC for max_depth = 5 and n_estimators = 120 is 90.02%
===================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 97.44%

CV AUC for max_depth = 10 and n_estimators = 120 is 91.81%
===================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 120 is 93.29%
===================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 100.00%
```

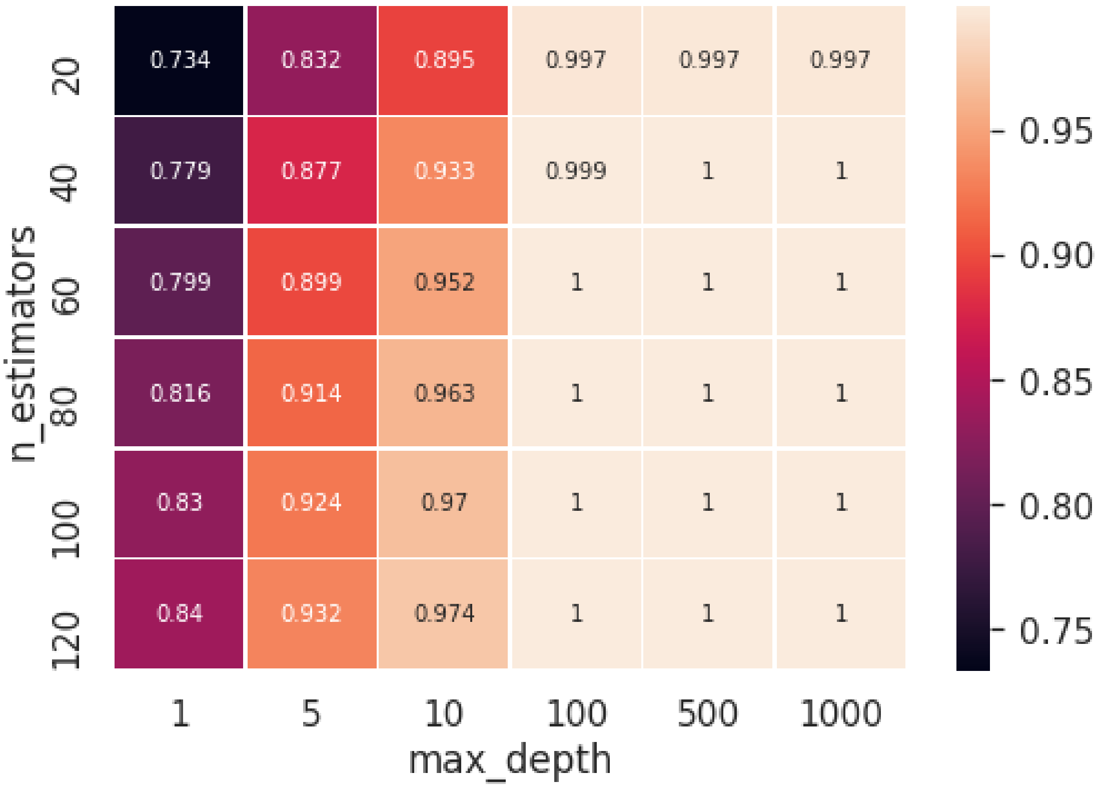
```
CV AUC for max_depth = 500 and n_estimators = 120 is 93.17%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 120 is 93.17%
==================================================
```

```
In [144]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(bow_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
          print("===================Train Data================")
          train_scores = np.array(bow_train_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidth
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
```

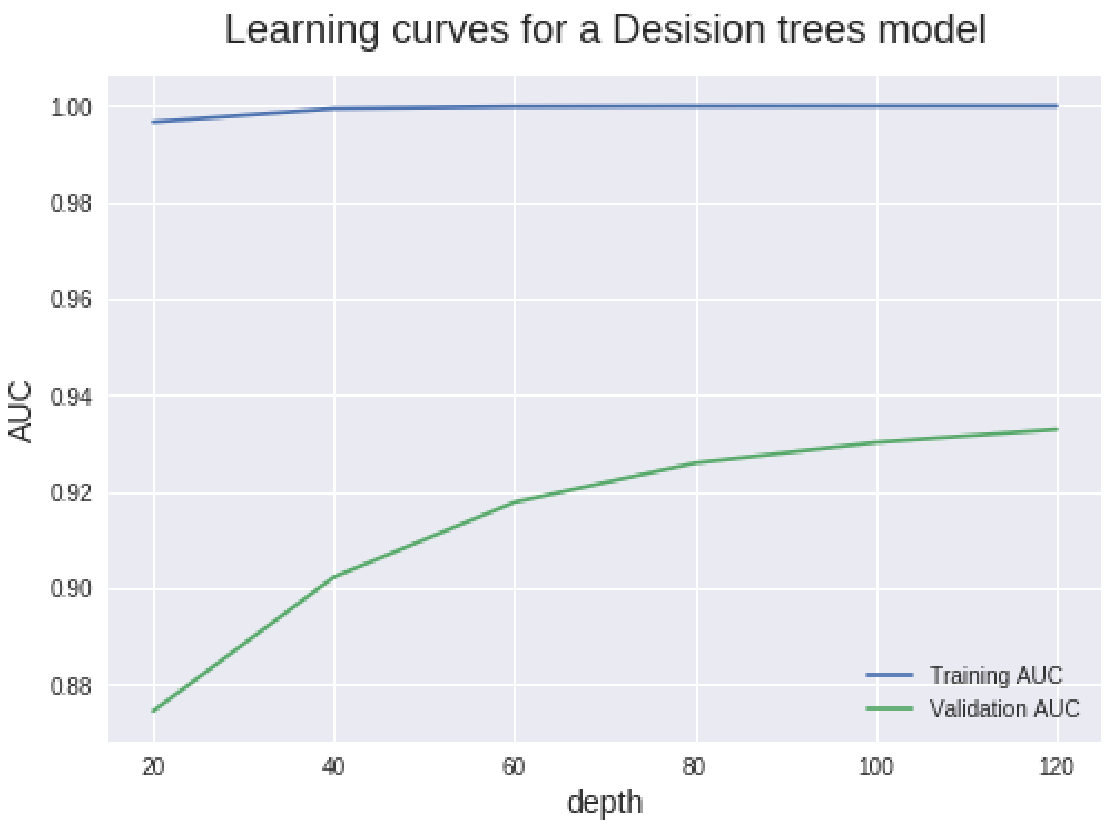
```
===================CV Data===================
```


n_estimators
20
40
60
80
100
120
max_depth
1
5
10
100
500
1000
0.72
0.808
0.842
0.875
0.872
0.872
0.766
0.85
0.878
0.902
0.902
0.902
0.785
0.871
0.896
0.918
0.917
0.917
0.802
0.884
0.907
0.926
0.925
0.925
0.816
0.893
0.913
0.93
0.929
0.929
0.827
0.9
0.918
0.933
0.932
0.932
0.92
0.88
0.84
0.80
0.76
0.72

```
===================Train Data================
```



```
In [145]: # depth is 100
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-3],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-3], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[145]: <matplotlib.legend.Legend at 0x7f9567a308d0>
```

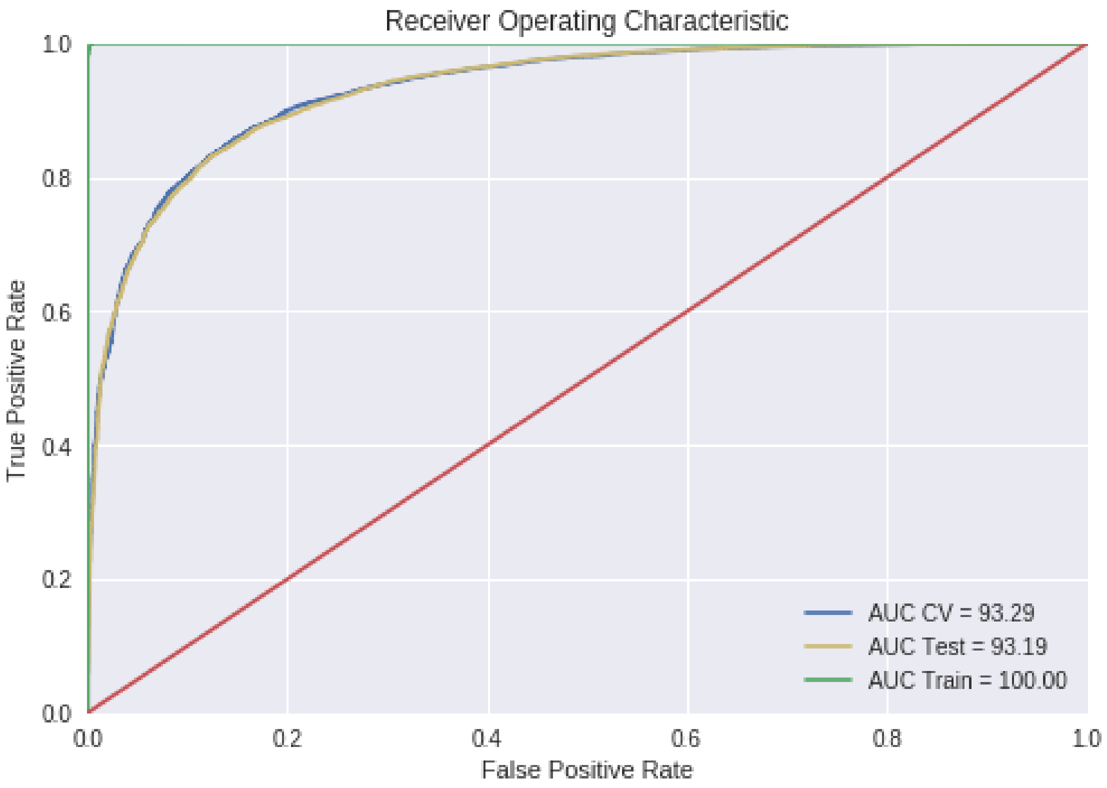


```
In [146]: i = 120
          j = 100

          XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1)
          XGBC.fit(bow_train, train_y)
          # train data
          y_prob_train = XGBC.predict_proba(bow_train)[:,1]
          fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
          y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
          auc_roc_train = roc_auc_score(train_y , y_prob_train)
          print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
          # CV
          y_prob_cv = XGBC.predict_proba(bow_cv)[:,1]
          fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
          y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
          auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
          print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
          # Test
          y_prob_test = XGBC.predict_proba(bow_test)[:,1]
          fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
          y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
```

```
            auc_roc_test = roc_auc_score(test_y , y_prob_test)
            print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

```
Train AUC for max depth = 100 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 120 is 93.29%

Test AUC for max_depth = 100 and n_estimators = 120 is 93.19%
```

```
In [147]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```

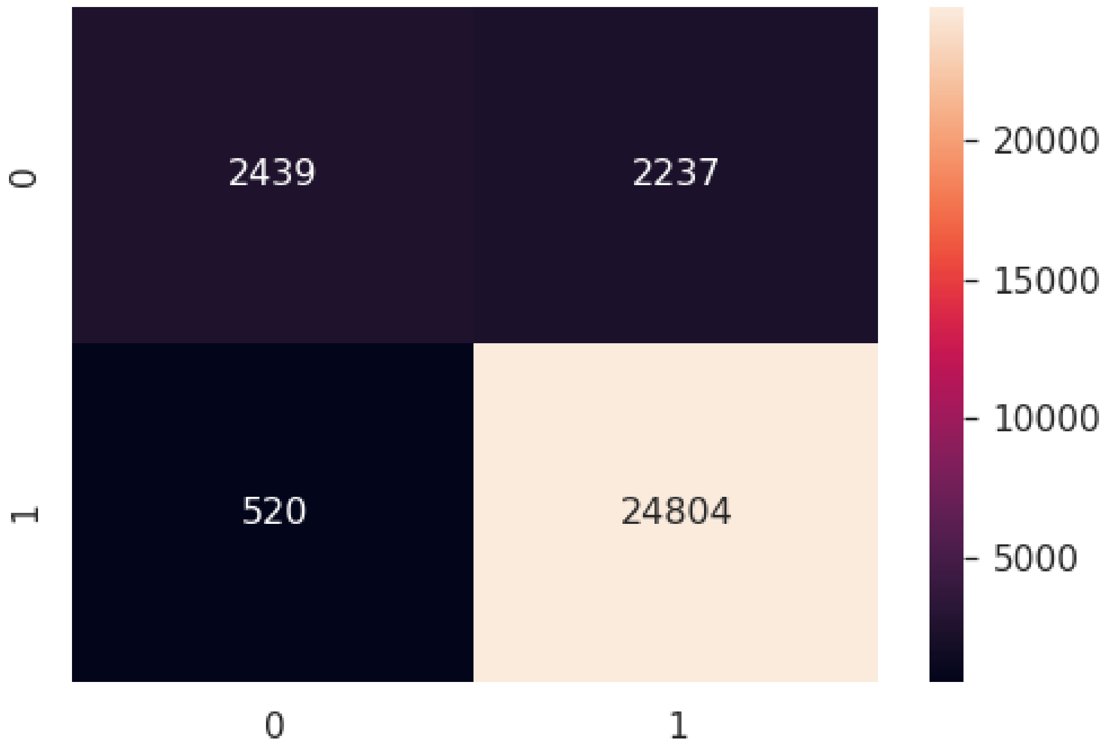


```
In [148]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.95
```

```
In [149]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[149]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9567a35eb8>
```



6.2.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

```
In [150]: n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          tfidf_train_auc = []
          tfidf_cv_auc = []
          for i in n_estimators:
              for j in max_depth:
                  XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1, colsampl
                  XGBC.fit(tf_idf_train, train_y)
                  # train data
                  y_prob_train = XGBC.predict_proba(tf_idf_train)[:,1]
                  y_pred = np.where(y_prob_train > 0.5, 1, 0)
                  auc_roc_train = roc_auc_score(train_y , y_prob_train)
                  print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
                  tfidf_train_auc.append(auc_roc_train)
                  # CV
                  y_prob_cv = XGBC.predict_proba(tf_idf_cv)[:,1]
                  y_pred = np.where(y_prob_cv > 0.5, 1, 0)
                  auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
                  print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
                  tfidf_cv_auc.append(auc_roc_cv)
                  print("="*50)
```

```

Train AUC for max_depth = 1 and n_estimators = 20 is 76.56%

CV AUC for max_depth = 1 and n_estimators = 20 is 74.98%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 84.57%

CV AUC for max_depth = 5 and n_estimators = 20 is 82.51%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 90.83%

CV AUC for max_depth = 10 and n_estimators = 20 is 86.70%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.86%

CV AUC for max_depth = 100 and n_estimators = 20 is 91.55%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.86%

CV AUC for max_depth = 500 and n_estimators = 20 is 91.13%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.86%

CV AUC for max_depth = 1000 and n_estimators = 20 is 91.35%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 78.74%
```

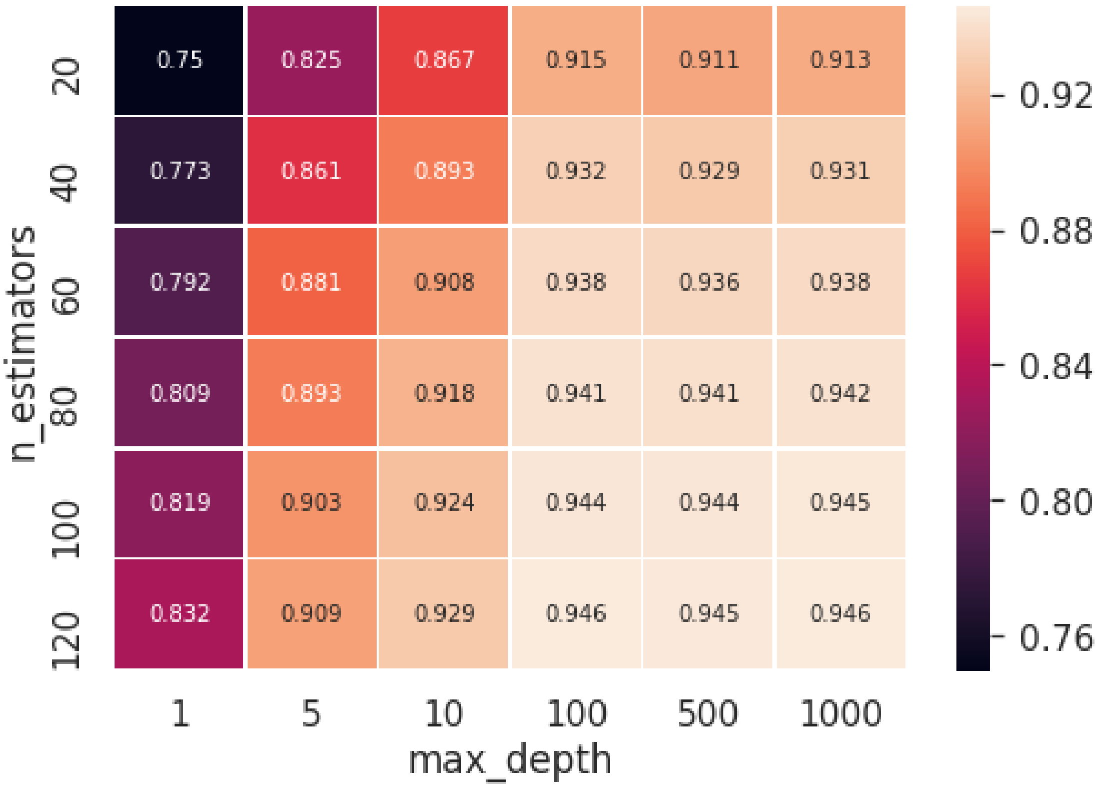
```
CV AUC for max_depth = 1 and n_estimators = 40 is 77.29%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 88.74%

CV AUC for max_depth = 5 and n_estimators = 40 is 86.12%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 94.33%

CV AUC for max_depth = 10 and n_estimators = 40 is 89.30%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.98%

CV AUC for max_depth = 100 and n_estimators = 40 is 93.23%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.96%

CV AUC for max_depth = 500 and n_estimators = 40 is 92.90%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 40 is 99.96%

CV AUC for max_depth = 1000 and n_estimators = 40 is 93.12%
==================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 80.78%

CV AUC for max_depth = 1 and n_estimators = 60 is 79.19%
==================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 90.97%

CV AUC for max_depth = 5 and n_estimators = 60 is 88.14%
==================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 95.81%

CV AUC for max_depth = 10 and n_estimators = 60 is 90.80%
==================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 99.99%

CV AUC for max_depth = 100 and n_estimators = 60 is 93.76%
==================================================
Train AUC for max_depth = 500 and n_estimators = 60 is 99.99%

CV AUC for max_depth = 500 and n_estimators = 60 is 93.62%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 60 is 93.79%
==================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 82.39%

CV AUC for max_depth = 1 and n_estimators = 80 is 80.85%
==================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 92.24%

CV AUC for max_depth = 5 and n_estimators = 80 is 89.31%
==================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 96.81%

CV AUC for max_depth = 10 and n_estimators = 80 is 91.77%
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 100 and n_estimators = 80 is 94.12%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 500 and n_estimators = 80 is 94.06%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 80 is 94.16%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 83.54%

CV AUC for max_depth = 1 and n_estimators = 100 is 81.92%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 93.29%

CV AUC for max_depth = 5 and n_estimators = 100 is 90.27%
```

```
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 97.41%

CV AUC for max_depth = 10 and n_estimators = 100 is 92.36%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 100 is 94.42%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 100 is 94.36%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 100 is 94.47%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 84.78%

CV AUC for max_depth = 1 and n_estimators = 120 is 83.22%
==================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 94.03%

CV AUC for max_depth = 5 and n_estimators = 120 is 90.91%
==================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 97.88%

CV AUC for max_depth = 10 and n_estimators = 120 is 92.92%
==================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 120 is 94.59%
==================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 120 is 94.52%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 100.00%
```

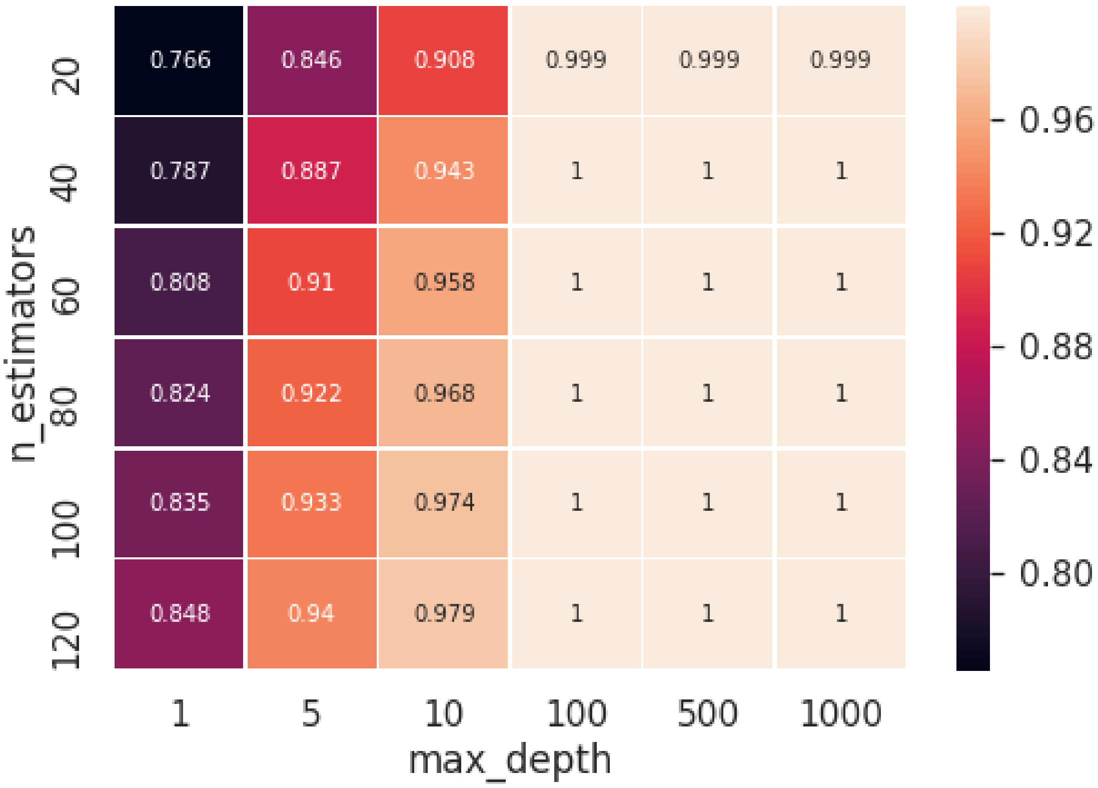
```
CV AUC for max_depth = 1000 and n_estimators = 120 is 94.65%
==================================================
```

```
In [151]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(tfidf_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
          print("===================Train Data================")
          train_scores = np.array(tfidf_train_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidth
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
```

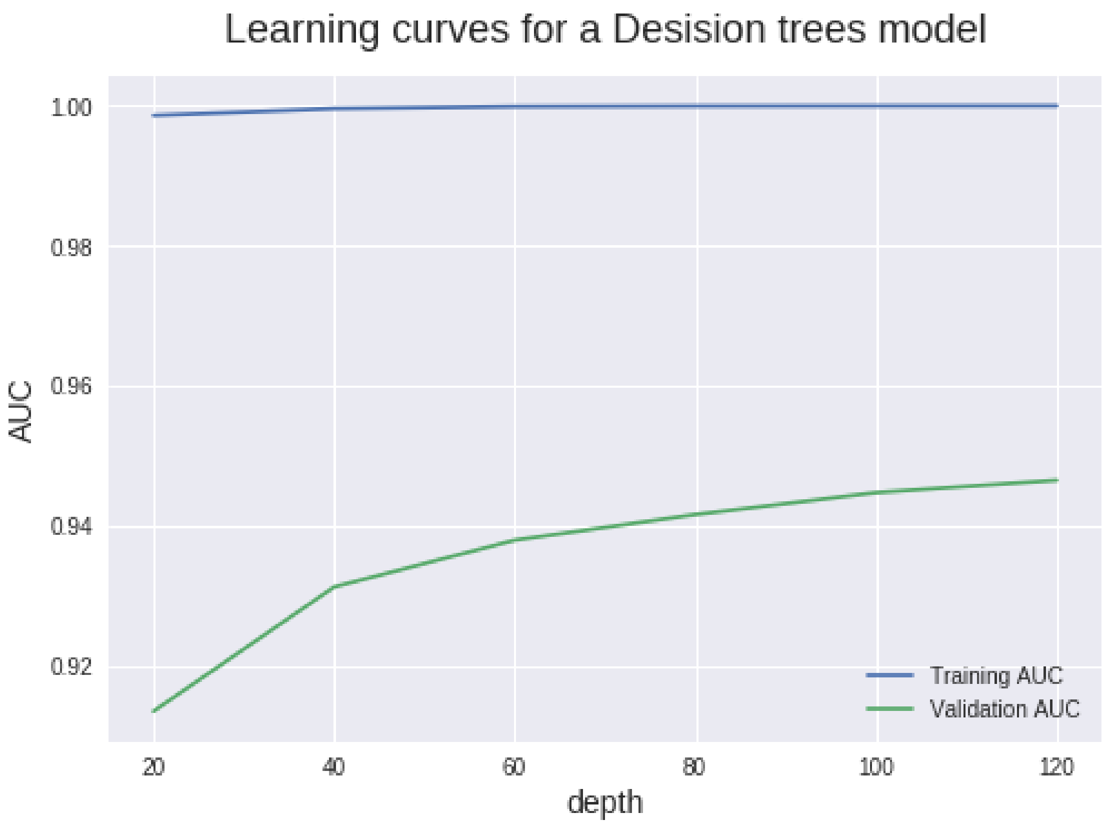
```
===================CV Data===================
```


n_estimators
20
40
60
80
100
120
max_depth
1
5
10
100
500
1000
0.75
0.825
0.867
0.915
0.911
0.913
0.773
0.861
0.893
0.932
0.929
0.931
0.792
0.881
0.908
0.938
0.936
0.938
0.809
0.893
0.918
0.941
0.941
0.942
0.819
0.903
0.924
0.944
0.944
0.945
0.832
0.909
0.929
0.946
0.945
0.946
0.92
0.88
0.84
0.80
0.76

```
===================Train Data================
```



```
In [152]: # depth is 1000
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-1],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-1], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[152]: <matplotlib.legend.Legend at 0x7f956c070ac8>
```

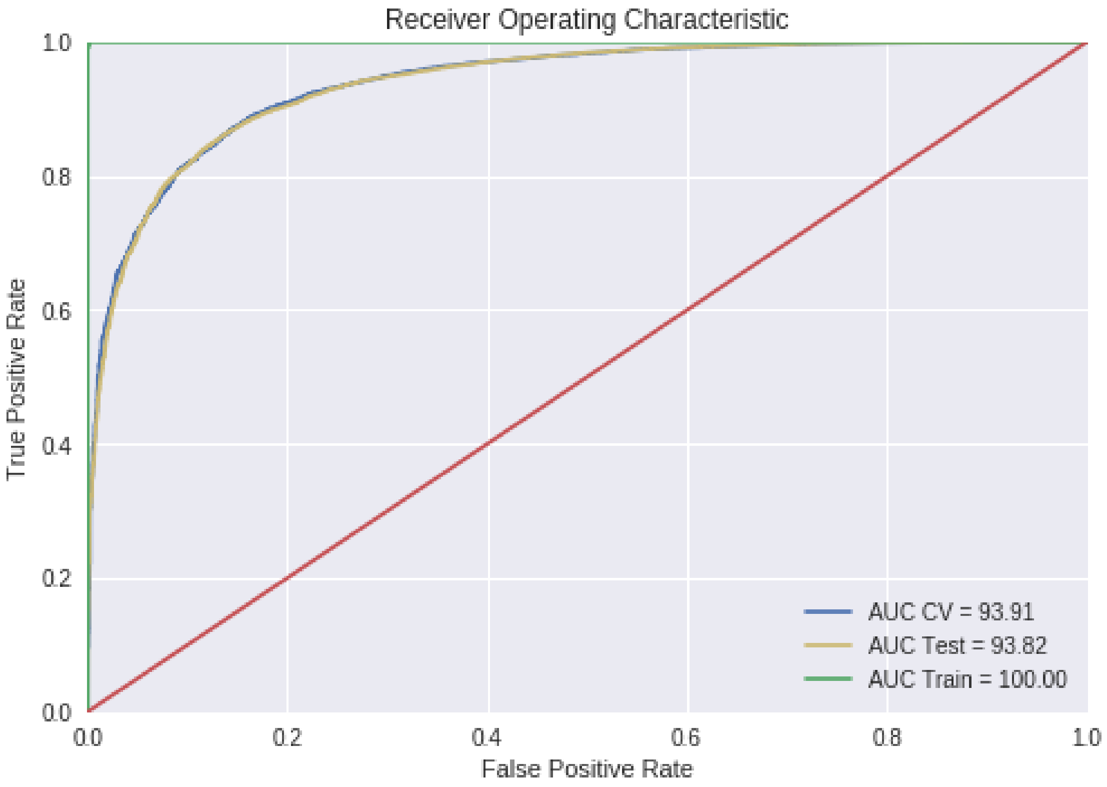


```
In [153]: i = 120
          j = 100

          XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1)
          XGBC.fit(tf_idf_train, train_y)
          # train data
          y_prob_train = XGBC.predict_proba(tf_idf_train)[:,1]
          fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
          y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
          auc_roc_train = roc_auc_score(train_y , y_prob_train)
          print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
          # CV
          y_prob_cv = XGBC.predict_proba(tf_idf_cv)[:,1]
          fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
          y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
          auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
          print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
          # Test
          y_prob_test = XGBC.predict_proba(tf_idf_test)[:,1]
          fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
          y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
```

```
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

```
Train AUC for max depth = 100 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 120 is 93.91%

Test AUC for max_depth = 100 and n_estimators = 120 is 93.82%
```

```
In [154]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```

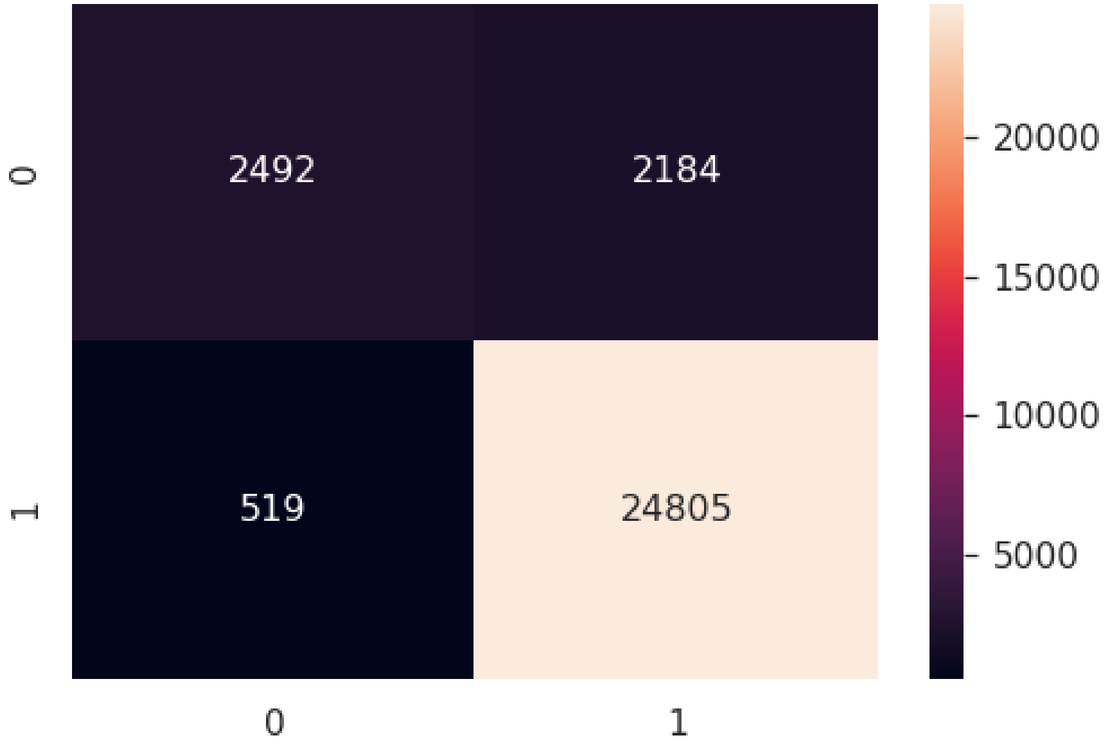


```
In [155]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.95
```

```
In [156]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[156]: <matplotlib.axes._subplots.AxesSubplot at 0x7f956a73fcf8>
```



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

```
In [159]: n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          w2v_train_auc = []
          w2v_cv_auc = []
          for i in n_estimators:
              for j in max_depth:
                  XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1, colsampl
                  XGBC.fit(w2v_train, train_y)
                  # train data
                  y_prob_train = XGBC.predict_proba(w2v_train)[:,1]
                  y_pred = np.where(y_prob_train > 0.5, 1, 0)
                  auc_roc_train = roc_auc_score(train_y , y_prob_train)
                  print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
                  w2v_train_auc.append(auc_roc_train)
                  # CV
                  y_prob_cv = XGBC.predict_proba(w2v_cv)[:,1]
                  y_pred = np.where(y_prob_cv > 0.5, 1, 0)
                  auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
                  print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
                  w2v_cv_auc.append(auc_roc_cv)
                  print("="*50)
```

```

Train AUC for max_depth = 1 and n_estimators = 20 is 82.14%

CV AUC for max_depth = 1 and n_estimators = 20 is 81.69%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 90.69%

CV AUC for max_depth = 5 and n_estimators = 20 is 88.42%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 98.83%

CV AUC for max_depth = 10 and n_estimators = 20 is 89.55%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 100 and n_estimators = 20 is 89.05%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 500 and n_estimators = 20 is 89.05%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.97%

CV AUC for max_depth = 1000 and n_estimators = 20 is 89.05%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 85.36%
```

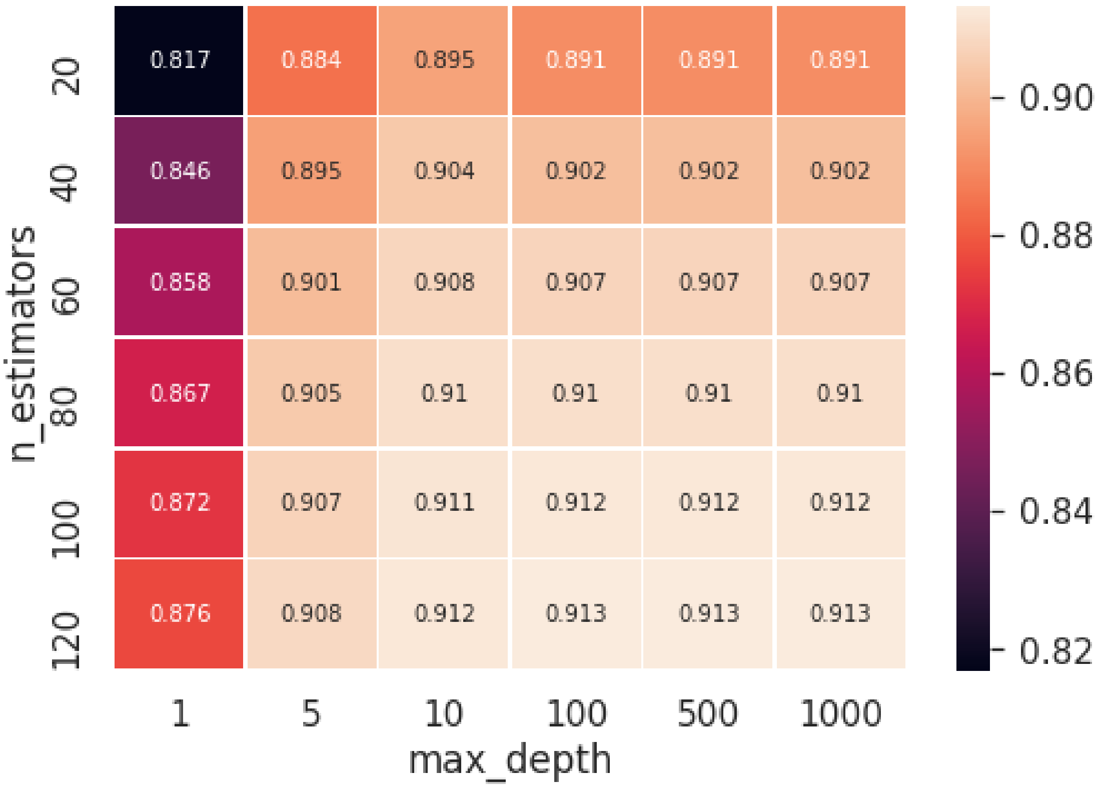
```
CV AUC for max_depth = 1 and n_estimators = 40 is 84.63%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 92.40%

CV AUC for max_depth = 5 and n_estimators = 40 is 89.47%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 99.62%

CV AUC for max_depth = 10 and n_estimators = 40 is 90.45%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.99%

CV AUC for max_depth = 100 and n_estimators = 40 is 90.19%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.99%

CV AUC for max_depth = 500 and n_estimators = 40 is 90.19%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 40 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 40 is 90.19%
==================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 86.57%

CV AUC for max_depth = 1 and n_estimators = 60 is 85.77%
==================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 93.46%

CV AUC for max_depth = 5 and n_estimators = 60 is 90.12%
==================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 99.85%

CV AUC for max_depth = 10 and n_estimators = 60 is 90.77%
==================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 60 is 90.72%
==================================================
Train AUC for max_depth = 500 and n_estimators = 60 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 60 is 90.72%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 60 is 90.72%
==================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 87.49%

CV AUC for max_depth = 1 and n_estimators = 80 is 86.66%
==================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 94.21%

CV AUC for max_depth = 5 and n_estimators = 80 is 90.48%
==================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 99.95%

CV AUC for max_depth = 10 and n_estimators = 80 is 91.00%
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 80 is 91.01%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 80 is 91.01%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 80 is 91.01%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 88.16%

CV AUC for max_depth = 1 and n_estimators = 100 is 87.24%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 94.80%

CV AUC for max_depth = 5 and n_estimators = 100 is 90.72%
```

```
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 10 and n_estimators = 100 is 91.13%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 100 is 91.21%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 100 is 91.21%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 100 is 91.21%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 88.62%

CV AUC for max_depth = 1 and n_estimators = 120 is 87.64%
==================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 95.26%

CV AUC for max_depth = 5 and n_estimators = 120 is 90.85%
==================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 99.99%

CV AUC for max_depth = 10 and n_estimators = 120 is 91.22%
==================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 120 is 91.32%
==================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 120 is 91.32%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 120 is 100.00%
```

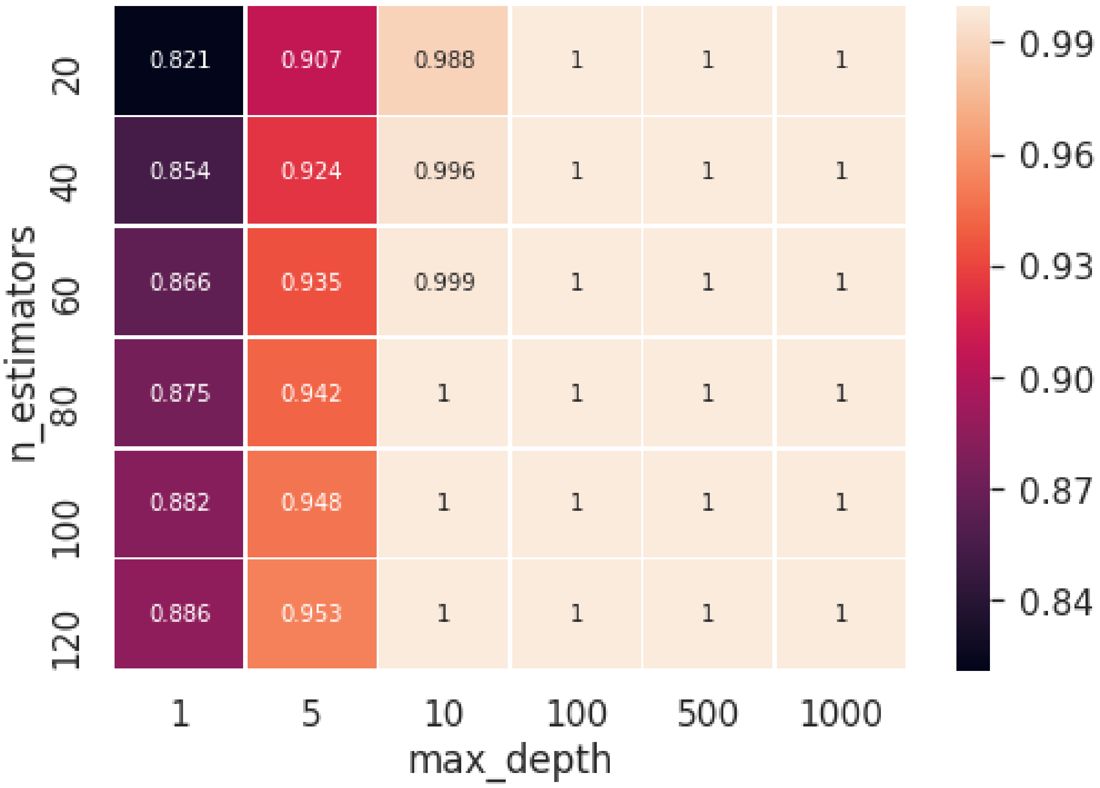
```
CV AUC for max_depth = 1000 and n_estimators = 120 is 91.32%
==================================================
```

```
In [160]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(w2v_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
          print("===================Train Data================")
          train_scores = np.array(w2v_train_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidt
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
```

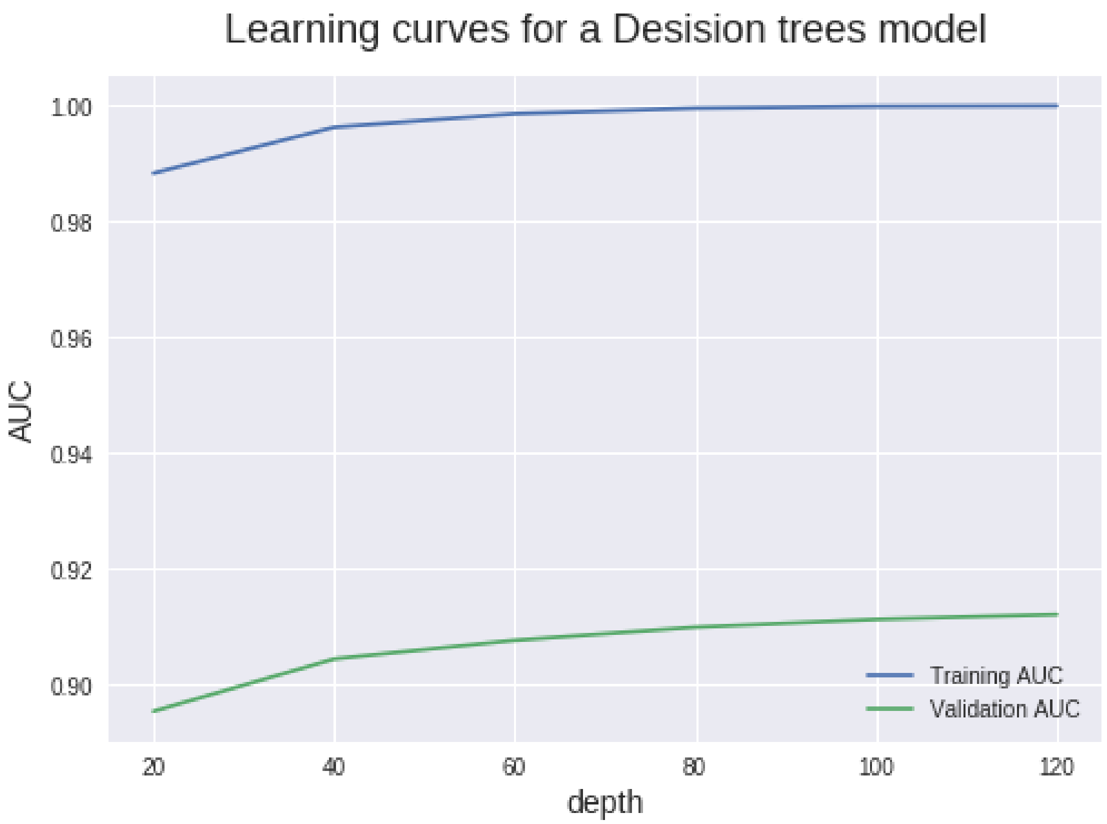
```
===================CV Data===================
```


0.817
0.884
0.895
0.891
0.891
0.891
0.846
0.895
0.904
0.902
0.902
0.902
0.858
0.901
0.908
0.907
0.907
0.907
0.867
0.905
0.91
0.91
0.91
0.91
0.872
0.907
0.911
0.912
0.912
0.912
0.876
0.908
0.912
0.913
0.913
0.913
n_estimators
20
40
60
80
100
120
1
5
10
100
500
1000
max_depth
0.90
0.88
0.86
0.84
0.82

```
===================Train Data================
```



```
In [161]: # depth is 10
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-4],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-4], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[161]: <matplotlib.legend.Legend at 0x7f956b083668>
```

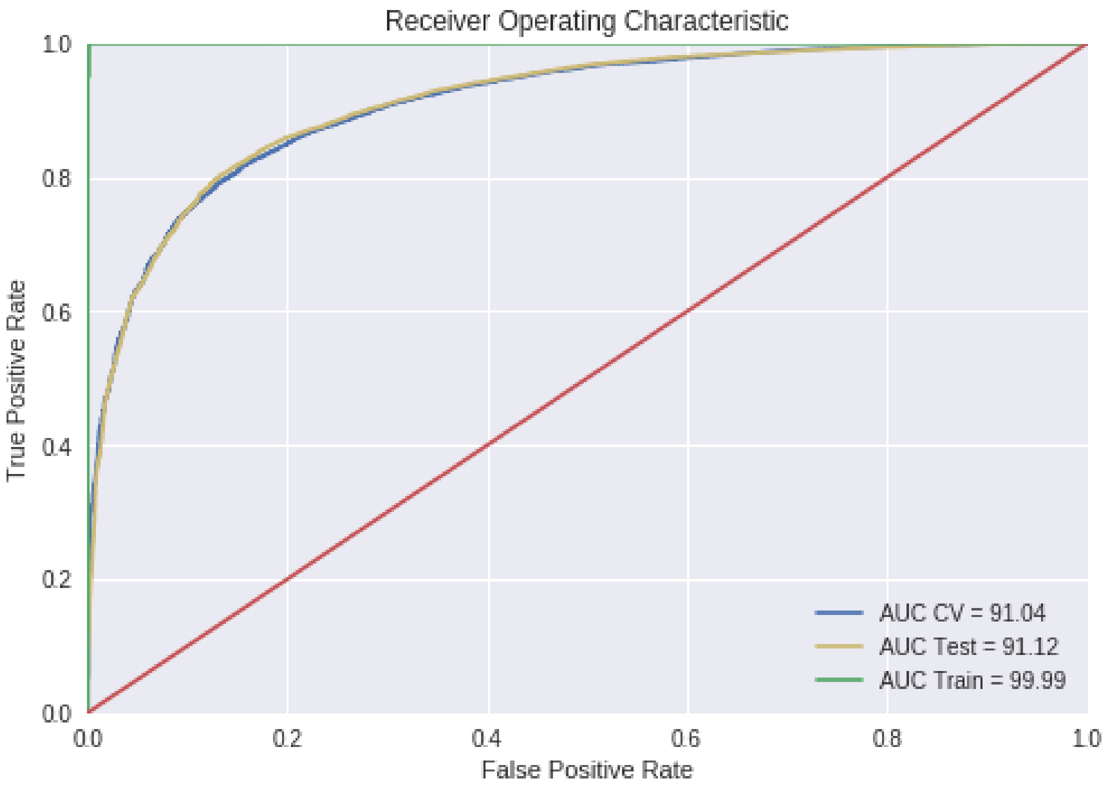


```
In [163]: i = 120
          j = 10

          XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1)
          XGBC.fit(w2v_train, train_y)
          # train data
          y_prob_train = XGBC.predict_proba(w2v_train)[:,1]
          fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
          y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
          auc_roc_train = roc_auc_score(train_y , y_prob_train)
          print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
          # CV
          y_prob_cv = XGBC.predict_proba(w2v_cv)[:,1]
          fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
          y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
          auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
          print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
          # Test
          y_prob_test = XGBC.predict_proba(w2v_test)[:,1]
          fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
          y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
```

```
        auc_roc_test = roc_auc_score(test_y , y_prob_test)
        print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

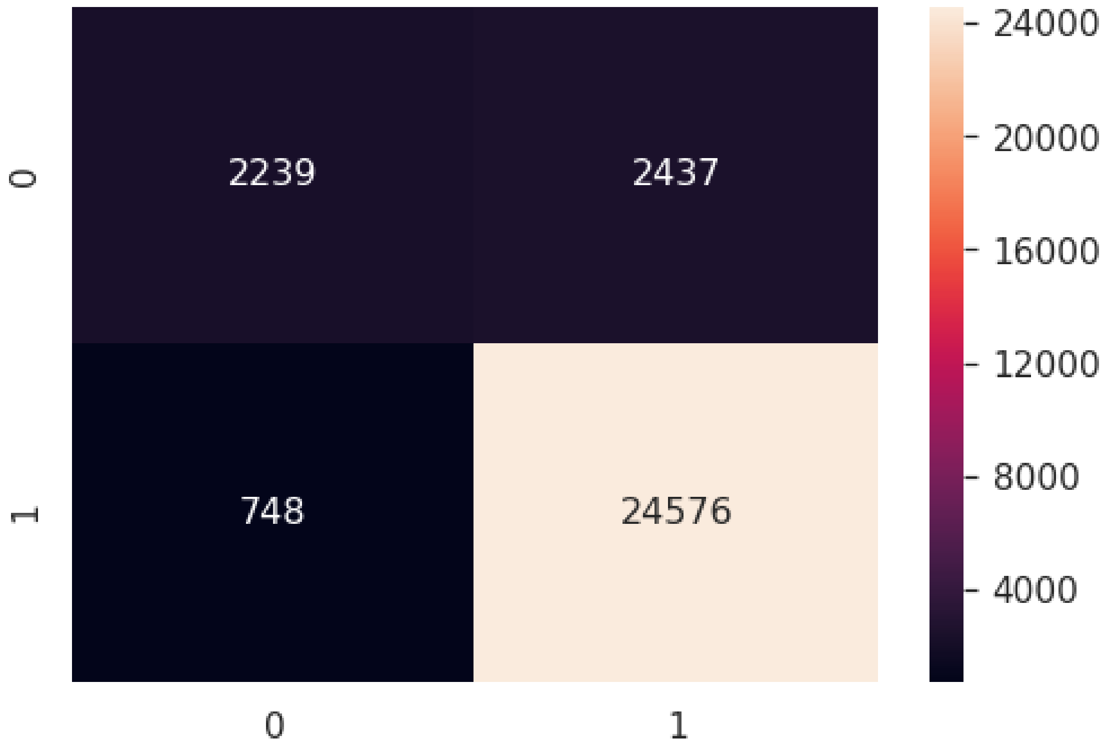
```
Train AUC for max depth = 10 and n_estimators = 120 is 99.99%

CV AUC for max_depth = 10 and n_estimators = 120 is 91.04%

Test AUC for max_depth = 10 and n_estimators = 120 is 91.12%
```

```
In [164]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```



```
In [165]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))

F1-Score on test set: 0.94
```

```
In [166]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')

Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x7f956b0a4dd8>
```



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

```
In [167]: # Please write all the code with proper documentation
          n_estimators = [20,40,60,80,100,120]
          max_depth = [1,5,10,100,500,1000]
          tfidf_w2v_train_auc = []
          tfidf_w2v_cv_auc = []
          for i in n_estimators:
              for j in max_depth:
```

```
        XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1, colsampl
        XGBC.fit(tfidf_w2v_train, train_y)
        # train data
        y_prob_train = XGBC.predict_proba(tfidf_w2v_train)[:,1]
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (s
        tfidf_w2v_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = XGBC.predict_proba(tfidf_w2v_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(
        tfidf_w2v_cv_auc.append(auc_roc_cv)
        print("="*50)
```

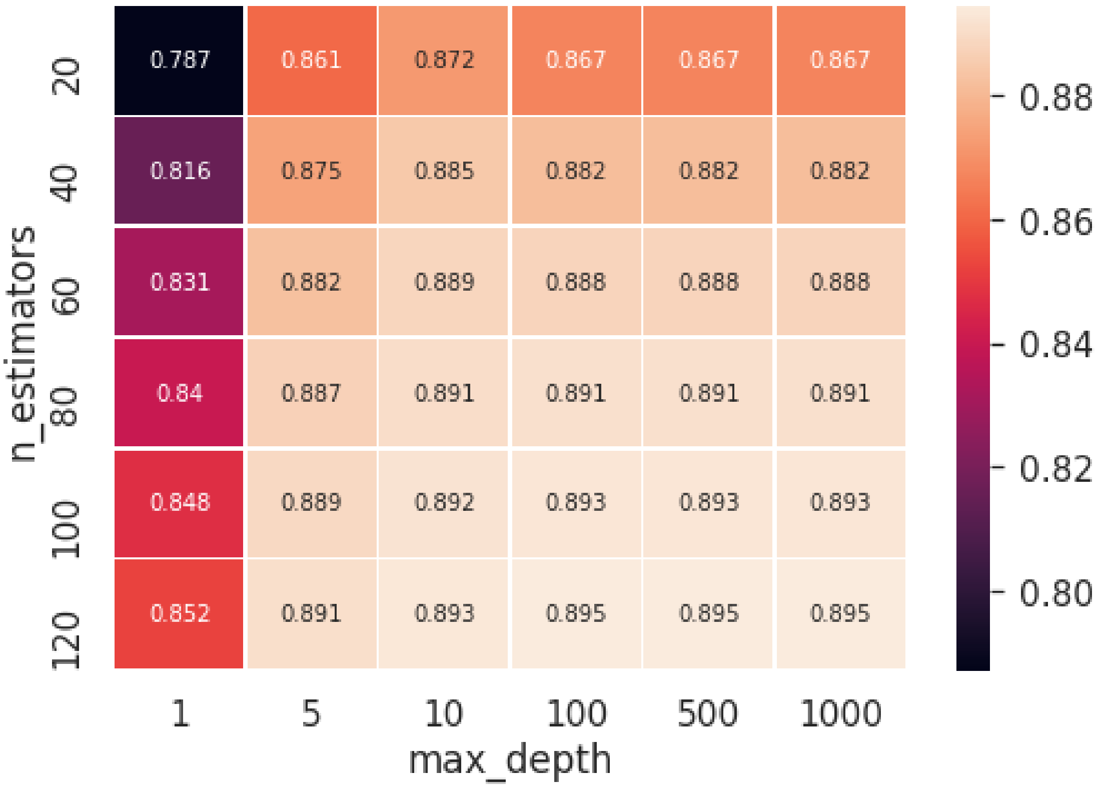
```

Train AUC for max_depth = 1 and n_estimators = 20 is 78.69%

CV AUC for max_depth = 1 and n_estimators = 20 is 78.72%
==================================================

Train AUC for max_depth = 5 and n_estimators = 20 is 88.37%

CV AUC for max_depth = 5 and n_estimators = 20 is 86.07%
==================================================

Train AUC for max_depth = 10 and n_estimators = 20 is 98.53%

CV AUC for max_depth = 10 and n_estimators = 20 is 87.20%
==================================================

Train AUC for max_depth = 100 and n_estimators = 20 is 99.96%

CV AUC for max_depth = 100 and n_estimators = 20 is 86.70%
==================================================

Train AUC for max_depth = 500 and n_estimators = 20 is 99.96%

CV AUC for max_depth = 500 and n_estimators = 20 is 86.70%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 20 is 99.96%

CV AUC for max_depth = 1000 and n_estimators = 20 is 86.70%
==================================================

Train AUC for max_depth = 1 and n_estimators = 40 is 81.84%
```

```
CV AUC for max_depth = 1 and n_estimators = 40 is 81.60%
==================================================

Train AUC for max_depth = 5 and n_estimators = 40 is 90.60%

CV AUC for max_depth = 5 and n_estimators = 40 is 87.46%
==================================================

Train AUC for max_depth = 10 and n_estimators = 40 is 99.58%

CV AUC for max_depth = 10 and n_estimators = 40 is 88.48%
==================================================

Train AUC for max_depth = 100 and n_estimators = 40 is 99.99%

CV AUC for max_depth = 100 and n_estimators = 40 is 88.18%
==================================================

Train AUC for max_depth = 500 and n_estimators = 40 is 99.99%

CV AUC for max_depth = 500 and n_estimators = 40 is 88.18%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 40 is 99.99%

CV AUC for max_depth = 1000 and n_estimators = 40 is 88.18%
==================================================

Train AUC for max_depth = 1 and n_estimators = 60 is 83.49%

CV AUC for max_depth = 1 and n_estimators = 60 is 83.14%
==================================================

Train AUC for max_depth = 5 and n_estimators = 60 is 91.95%

CV AUC for max_depth = 5 and n_estimators = 60 is 88.25%
==================================================

Train AUC for max_depth = 10 and n_estimators = 60 is 99.84%

CV AUC for max_depth = 10 and n_estimators = 60 is 88.86%
==================================================

Train AUC for max_depth = 100 and n_estimators = 60 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 60 is 88.81%
==================================================
Train AUC for max_depth = 500 and n_estimators = 60 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 60 is 88.81%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 60 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 60 is 88.81%
==================================================

Train AUC for max_depth = 1 and n_estimators = 80 is 84.47%

CV AUC for max_depth = 1 and n_estimators = 80 is 84.03%
==================================================

Train AUC for max_depth = 5 and n_estimators = 80 is 92.83%

CV AUC for max_depth = 5 and n_estimators = 80 is 88.71%
==================================================

Train AUC for max_depth = 10 and n_estimators = 80 is 99.94%

CV AUC for max_depth = 10 and n_estimators = 80 is 89.06%
==================================================

Train AUC for max_depth = 100 and n_estimators = 80 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 80 is 89.15%
==================================================

Train AUC for max_depth = 500 and n_estimators = 80 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 80 is 89.15%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 80 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 80 is 89.15%
==================================================

Train AUC for max_depth = 1 and n_estimators = 100 is 85.33%

CV AUC for max_depth = 1 and n_estimators = 100 is 84.75%
==================================================

Train AUC for max_depth = 5 and n_estimators = 100 is 93.53%
```

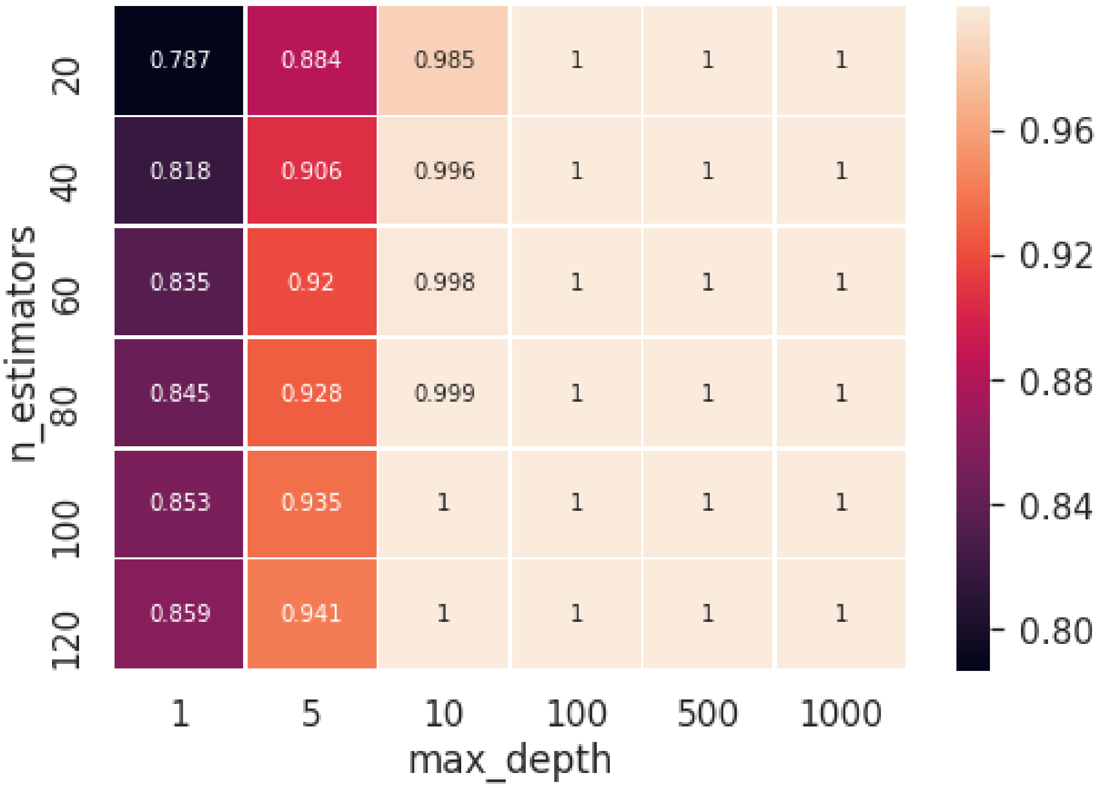
```
CV AUC for max_depth = 5 and n_estimators = 100 is 88.94%
==================================================

Train AUC for max_depth = 10 and n_estimators = 100 is 99.98%

CV AUC for max_depth = 10 and n_estimators = 100 is 89.24%
==================================================

Train AUC for max_depth = 100 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 100 is 89.32%
==================================================

Train AUC for max_depth = 500 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 100 is 89.32%
==================================================

Train AUC for max_depth = 1000 and n_estimators = 100 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 100 is 89.32%
==================================================

Train AUC for max_depth = 1 and n_estimators = 120 is 85.87%

CV AUC for max_depth = 1 and n_estimators = 120 is 85.24%
==================================================

Train AUC for max_depth = 5 and n_estimators = 120 is 94.15%

CV AUC for max_depth = 5 and n_estimators = 120 is 89.10%
==================================================

Train AUC for max_depth = 10 and n_estimators = 120 is 99.99%

CV AUC for max_depth = 10 and n_estimators = 120 is 89.33%
==================================================

Train AUC for max_depth = 100 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 100 and n_estimators = 120 is 89.46%
==================================================

Train AUC for max_depth = 500 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 500 and n_estimators = 120 is 89.46%
==================================================
Train AUC for max_depth = 1000 and n_estimators = 120 is 100.00%

CV AUC for max_depth = 1000 and n_estimators = 120 is 89.46%
==================================================
```

```
In [168]: fig, ax = plt.subplots()
          # auc on cv
          print("===================CV Data===================")
          cv_scores = np.array(tfidf_w2v_cv_auc).reshape(len(n_estimators),len(max_depth))
          df_cm_cv = pd.DataFrame(cv_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_cv, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
          print("===================Train Data================")
          train_scores = np.array(tfidf_w2v_train_auc).reshape(len(n_estimators),len(max_depth
          df_cm_train = pd.DataFrame(train_scores, n_estimators, max_depth)
          sns.set(font_scale=1.4)
          ax = sns.heatmap(df_cm_train, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidt
          ax.set_xlabel("max_depth")
          ax.set_ylabel("n_estimators")
          plt.show()
```

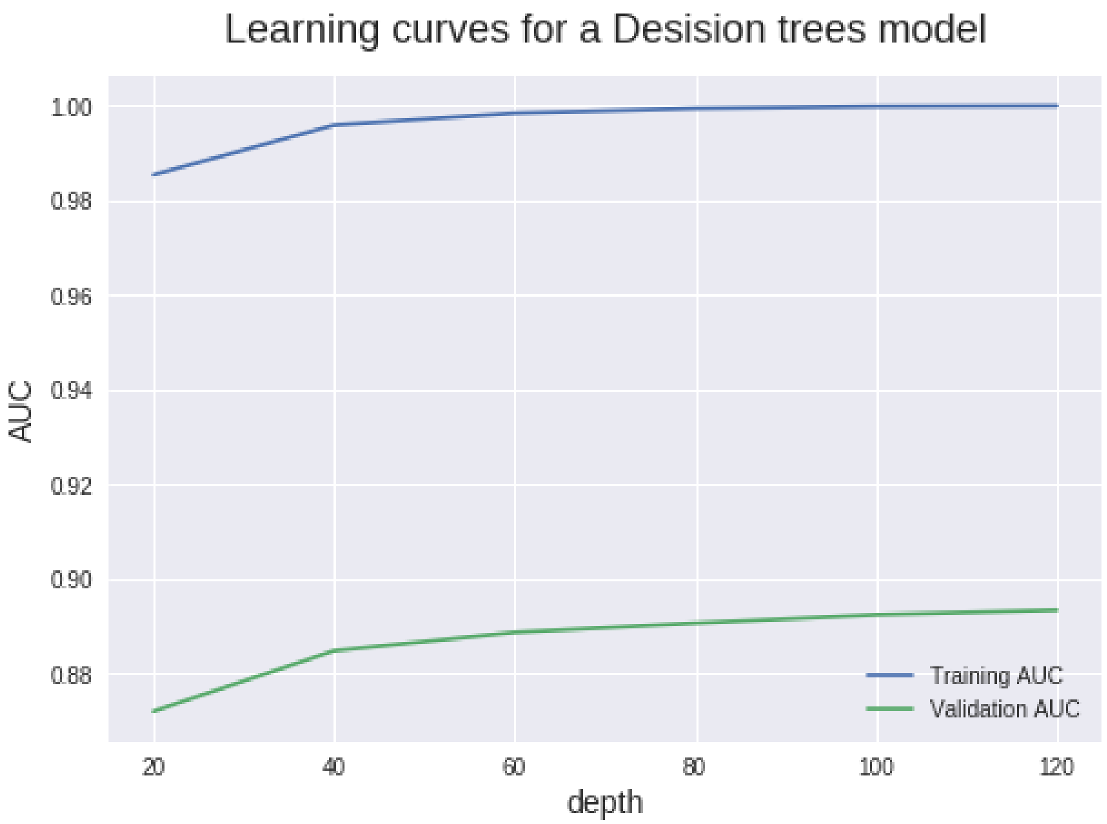
```
===================CV Data===================
```


0.787
0.861
0.872
0.867
0.867
0.867
0.816
0.875
0.885
0.882
0.882
0.882
0.831
0.882
0.889
0.888
0.888
0.888
0.84
0.887
0.891
0.891
0.891
0.891
0.848
0.889
0.892
0.893
0.893
0.893
0.852
0.891
0.893
0.895
0.895
0.895
n_estimators
20
40
60
80
100
120
1
5
10
100
500
1000
max_depth
0.88
0.86
0.84
0.82
0.80

```
===================Train Data================
```



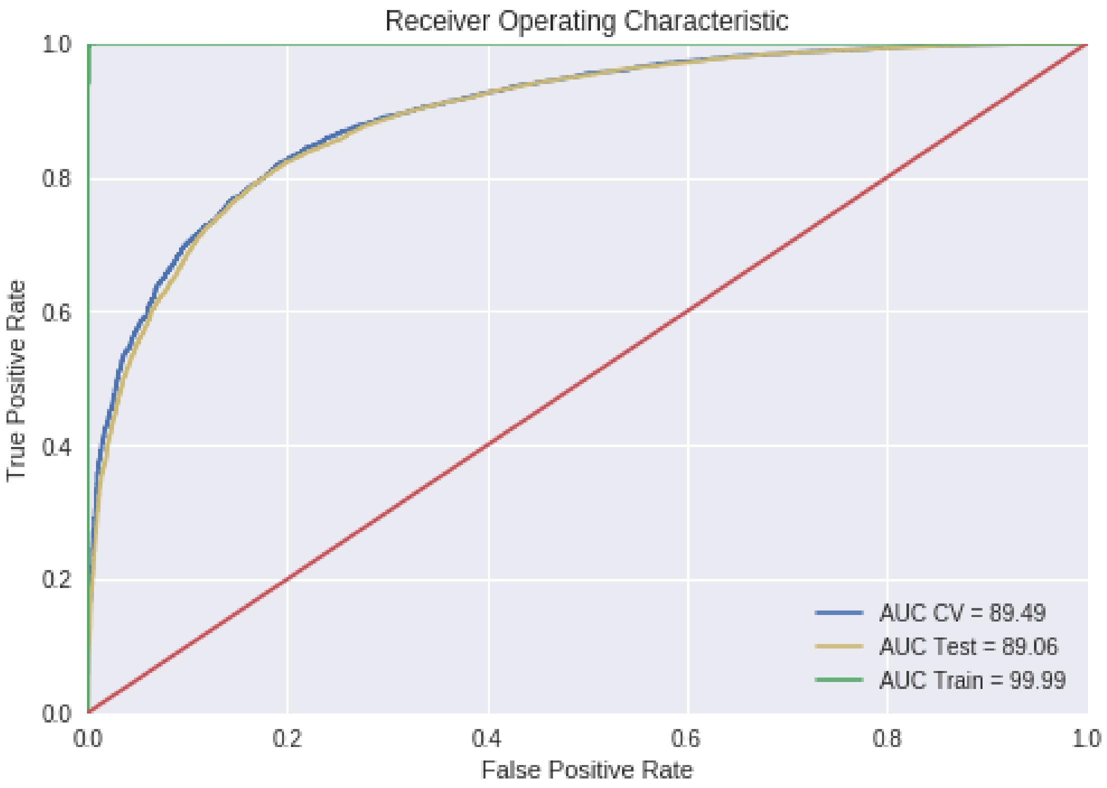
```
In [169]: # depth is 10
          # https://www.dataquest.io/blog/learning-curves-machine-learning/

          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.style.use('seaborn')

          plt.plot(n_estimators,train_scores[:,-4],label = 'Training AUC')
          plt.plot(n_estimators,cv_scores[:,-4], label = 'Validation AUC')

          plt.ylabel('AUC', fontsize = 14)
          plt.xlabel('depth', fontsize = 14)
          plt.title('Learning curves for a Desision trees model', fontsize = 18, y = 1.03)
          plt.legend()

Out[169]: <matplotlib.legend.Legend at 0x7f956a2614a8>
```



```
In [170]: i = 120
          j = 10

          XGBC = XGBClassifier(n_estimators=i, max_depth=j, learning_rate=0.1)
          XGBC.fit(tfidf_w2v_train, train_y)
          # train data
          y_prob_train = XGBC.predict_proba(tfidf_w2v_train)[:,1]
          fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
          y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
          auc_roc_train = roc_auc_score(train_y , y_prob_train)
          print('\nTrain AUC for max depth = %s and n_estimators = %s is %0.2f%%' % (str(j),st
          # CV
          y_prob_cv = XGBC.predict_proba(tfidf_w2v_cv)[:,1]
          fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
          y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
          auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
          print('\nCV AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str(i
          # Test
          y_prob_test = XGBC.predict_proba(tfidf_w2v_test)[:,1]
          fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
          y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
          auc_roc_test = roc_auc_score(test_y , y_prob_test)
          print('\nTest AUC for max_depth = %s and n_estimators = %s is %0.2f%%' % (str(j),str
```

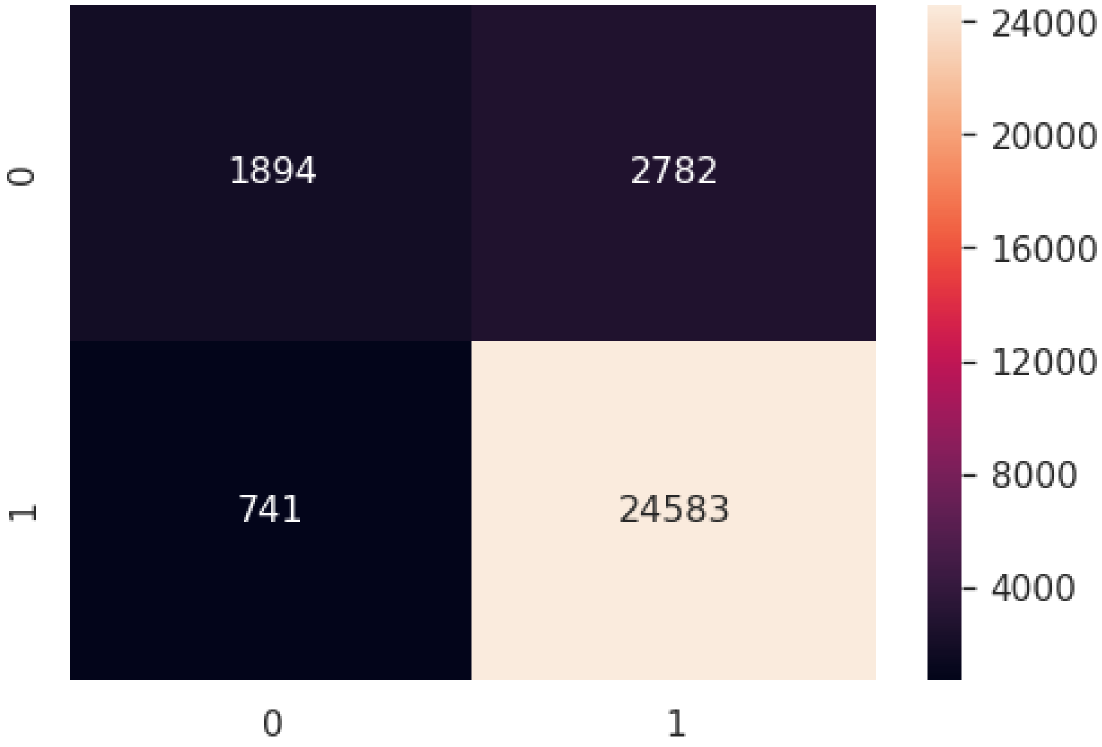
```
Train AUC for max depth = 10 and n_estimators = 120 is 99.99%

CV AUC for max_depth = 10 and n_estimators = 120 is 89.49%

Test AUC for max_depth = 10 and n_estimators = 120 is 89.06%
```

```
In [171]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

          import matplotlib.pyplot as plt
          plt.clf()
          plt.title('Receiver Operating Characteristic')
          plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
          plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100))
          plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1],'r')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```



```
In [172]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))

F1-Score on test set: 0.93
```

```
In [173]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')

Out[173]: <matplotlib.axes._subplots.AxesSubplot at 0x7f956d3d05f8>
```



7 Feature engineering

Taking length of reviews as another feature

```
In [18]: # Please compare all your models using Prettytable library
         X_review = np.array(sorted_data['CleanedText'])
         X_summary = np.array(sorted_data['Summary'])
         y = np.array(sorted_data['Score'])
         print(X_review.shape)
         print(X_summary.shape)
         print(y.shape)
```

```
(100000,)
(100000,)
(100000,)
```

```
In [19]: # Simple cross validation
         # split the data sent into train and test
         train_review , test_review, train_summary, test_summary , train_y , test_y = train_tes

         # split the train data set into cross validation train and cross validation test
         train_review, cv_review, train_summary, cv_summary , train_y, cv_y = train_test_split

         print("train review", train_review.shape)
         print("cv review", cv_review.shape)
         print("test review", test_review.shape)
         print("train summary", train_summary.shape)
         print("cv summary", cv_summary.shape)
         print("test summary", test_summary.shape)
```

```
train review (49000,)
cv review (21000,)
test review (30000,)
train summary (49000,)
cv summary (21000,)
test summary (30000,)
```

```
In [20]: # bow for summary
         # Please write all the code with proper documentation
         #BoW
         count_vect = CountVectorizer(min_df=15, ngram_range=(1,2)) #in scikit-learn
         count_vect.fit(train_summary)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         bow_train_summary = count_vect.fit_transform(train_summary)
         bow_cv_summary = count_vect.transform(cv_summary)
         bow_test_summary = count_vect.transform(test_summary)
         print("============Train Data=========")
         print("the type of count vectorizer ",type(bow_train_summary))
         print("the shape of out text BOW vectorizer ",bow_train_summary.get_shape())
         print("the number of unique words ", bow_train_summary.get_shape()[1])
         print("============Cross validation Data=========")
         print("the type of count vectorizer ",type(bow_cv_summary))
         print("the shape of out text BOW vectorizer ",bow_cv_summary.get_shape())
         print("the number of unique words ", bow_cv_summary.get_shape()[1])
         print("============Test Data=========")
         print("the type of count vectorizer ",type(bow_test_summary))
         print("the shape of out text BOW vectorizer ",bow_test_summary.get_shape())
         print("the number of unique words ", bow_test_summary.get_shape()[1])
```

```
some feature names  ['10', '100', '11', '12', '16', '20', '24', '40', '50', 'about']
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 2274)
the number of unique words  2274
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 2274)
the number of unique words  2274
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 2274)
the number of unique words  2274
```

```
In [21]: # bow for review
         # Please write all the code with proper documentation
         #BoW
         count_vect = CountVectorizer(min_df=15, ngram_range=(1,2)) #in scikit-learn
         count_vect.fit(train_review)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         bow_train_review = count_vect.fit_transform(train_review)
         bow_cv_review = count_vect.transform(cv_review)
         bow_test_review = count_vect.transform(test_review)
         print("============Train Data=========")
         print("the type of count vectorizer ",type(bow_train_review))
         print("the shape of out text BOW vectorizer ",bow_train_review.get_shape())
         print("the number of unique words ", bow_train_review.get_shape()[1])
         print("============Cross validation Data=========")
         print("the type of count vectorizer ",type(bow_cv_review))
         print("the shape of out text BOW vectorizer ",bow_cv_review.get_shape())
         print("the number of unique words ", bow_cv_review.get_shape()[1])
         print("============Test Data=========")
         print("the type of count vectorizer ",type(bow_test_review))
         print("the shape of out text BOW vectorizer ",bow_test_review.get_shape())
         print("the number of unique words ", bow_test_review.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'able buy', 'able drink', 'able eat', 'able enjoy', 'al
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 18450)
the number of unique words  18450
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 18450)
the number of unique words  18450
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 18450)
the number of unique words  18450
```

7.0.1 StackingClassifier

```
In [22]: from sklearn import model_selection
         from sklearn.linear_model import LogisticRegression
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.svm import SVC
         from sklearn.ensemble import RandomForestClassifier
         from mlxtend.classifier import StackingClassifier
```

```
In [24]: clf1 = LogisticRegression(penalty='l2', C=0.1)
         clf2 = MultinomialNB(alpha=1)
         clf3 = RandomForestClassifier(n_estimators=120,criterion='gini', max_depth=50)
         lr = LogisticRegression()
         sclf = StackingClassifier(classifiers=[clf1, clf2, clf3], meta_classifier=lr , use_pr

         print("3-flod cross validation:\n")

         for clf, label in zip([clf1, clf2, clf3, sclf],
                               ['Logistic rgression',
                                'MultinomialNB',
                                'RF Classifire',
                                'Staking Classifier']):
             scores_summary = model_selection.cross_val_score(clf, bow_train_summary, train_y,
                                                       cv=3, scoring='roc_auc')

             scores_review = model_selection.cross_val_score(clf, bow_train_review, train_y,
                                                       cv=3, scoring='roc_auc')

             new_scores = (scores_summary + scores_review) / 2
             print("AUC: %0.2f (+/- %0.2f) [%s]"
                   % (new_scores.mean(), new_scores.std(), label))
```

```
3-flod cross validation:

AUC: 0.94 (+/- 0.00) [Logistic rgression]
AUC: 0.93 (+/- 0.00) [MultinomialNB]
AUC: 0.91 (+/- 0.00) [RF Classifire]
```

```
AUC: 0.94 (+/- 0.00) [Staking Classifier]
```

8 [6] Conclusions

```
In [27]: # Please compare all your models using Prettytable library
         from prettytable import PrettyTable

         x = PrettyTable(["Vectorizer" , "max_depth", "n_estimaters","AUC", "F1 Score"])

         x.add_row(["BOW", 500, 120,"90.90%", 0.94])
         x.add_row(["TFIDF",1000, 120,"92.99%", 0.94])
         x.add_row(["AVG-W2V", 500, 120,"89.37%", 0.93])
         x.add_row(["TFIDF-w2v",100, 120,"86.89%", 0.93])
         print(x.get_string(title="RF Model"))
```

```
+------------+-----------+--------------+--------+----------+
| Vectorizer | max_depth | n_estimaters |  AUC   | F1 Score |
+------------+-----------+--------------+--------+----------+
|    BOW     |    500    |     120      | 90.90% |   0.94   |
|   TFIDF    |    1000   |     120      | 92.99% |   0.94   |
|  AVG-W2V   |    500    |     120      | 89.37% |   0.93   |
| TFIDF-w2v  |    100    |     120      | 86.89% |   0.93   |
+------------+-----------+--------------+--------+----------+
```

```
In [28]: x = PrettyTable(["Vectorizer" , "max_depth", "n_estimaters","AUC", "F1 Score"])

         x.add_row(["BOW", 100, 120,"93.19%", 0.95])
         x.add_row(["TFIDF",100, 120,"93.83%", 0.95])
         x.add_row(["AVG-W2V", 10, 120,"91.12%", 0.94])
         x.add_row(["TFIDF-w2v",10, 120,"89.06%", 0.93])
         print(x.get_string(title="GDBT Model"))
```

```
+------------+-----------+--------------+--------+----------+
| Vectorizer | max_depth | n_estimaters |  AUC   | F1 Score |
+------------+-----------+--------------+--------+----------+
|    BOW     |    100    |     120      | 93.19% |   0.95   |
|   TFIDF    |    100    |     120      | 93.83% |   0.95   |
|  AVG-W2V   |     10    |     120      | 91.12% |   0.94   |
| TFIDF-w2v  |     10    |     120      | 89.06% |   0.93   |
+------------+-----------+--------------+--------+----------+
```

```
In [ ]:
```