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.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


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

from sklearn.metrics import accuracy_score
import seaborn as sn
from sklearn.metrics import classification_report

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

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import validation_curve

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
```

```
C:\Users\Adarsh\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
#loading train and test and validation dataset

file=open("x_train.pkl","rb")
x_train=pickle.load(file) # loading 'train' dataset
```

```python
file=open("x_cv.pkl",'rb')
x_cv=pickle.load(file) # loading 'validation' dataset

file=open("x_test.pkl",'rb')
x_test=pickle.load(file) # loading 'test' dataset

file=open("y_train.pkl","rb")
y_train=pickle.load(file) # loading 'train' dataset

file=open("y_cv.pkl",'rb')
y_cv=pickle.load(file) # loading 'validation' dataset

file=open("y_test.pkl",'rb')
y_test=pickle.load(file) # loading 'test' dataset

#loading train_bow and test _bow
file=open('x_train_bow.pkl','rb')
x_train_bow=pickle.load(file)

file=open('x_test_bow.pkl','rb')
x_test_bow=pickle.load(file)

file=open('x_cv_bow.pkl','rb')
x_cv_bow=pickle.load(file)

#loading train_tf_idf and test_tf_idf
file=open('train_tf_idf.pkl','rb')
train_tf_idf=pickle.load(file)

file=open('cv_tf_idf.pkl','rb')
cv_tf_idf=pickle.load(file)

file=open('test_tf_idf.pkl','rb')
test_tf_idf=pickle.load(file)

#loading train_w2v and test_w2v
file=open('train_w2v.pkl','rb')
train_w2v=pickle.load(file)

file=open('cv_w2v.pkl','rb')
cv_w2v=pickle.load(file)

file=open('test_w2v.pkl','rb')
test_w2v=pickle.load(file)

#loading train_tf_idf_w2v and test_tf_idf_w2v
file=open('train_tf_idf_w2v.pkl','rb')
train_tf_idf_w2v=pickle.load(file)

file=open('cv_tf_idf_w2v.pkl','rb')
cv_tf_idf_w2v=pickle.load(file)

file=open('test_tf_idf_w2v.pkl','rb')
test_tf_idf_w2v=pickle.load(file)
```

In [7]:

```python
# using csv Table to read data.

dataset=pd.read_csv("Reviews.csv")

print(dataset.shape)
dataset.head(3)
```

(568454, 10)

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	

In [8]:

```
# 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 da
ta points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3
LIMIT 500000""", con)

# taking reviews whose score is not equal to 3
filtered_dataset=dataset[dataset['Score']!=3]
filtered_dataset.shape

#creating a function to filter the reviews (if score>3 --> positive , if score
<3 --> negative)
def partition(x):
    if x>3:
        return 1
    return 0

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

Number of data points in our data (525814, 10)

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	

1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

[2] Exploratory Data Analysis

[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:

observation:

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 [9]:
```
# sorting the value
sorted_data=filtered_dataset.sort_values(by='Id',inplace=True )
#finding the dublicate values using 'df.dublicated'
filtered_dataset[filtered_dataset.duplicated(subset={'ProfileName','Helpfulnes
sNumerator','HelpfulnessDenominator','Score','Time'})].shape
#alternate way to drop dublicate values
dataset_no_dup=filtered_dataset.drop_duplicates(subset={'ProfileName','Score',
'Time','Summary'},keep='first')
print(f"before {dataset.shape}")
print(f"after removing duplicate values-->shape = {dataset_no_dup.shape}")
# %age of no. of review reamin in data set
print('percentage of data reamin after removing duplicate values and removing
```

```
reviews with neutral scores % .2f'
      %((dataset_no_dup.size/dataset.size)*100))
```

```
before (568454, 10)
after removing duplicate values-->shape = (363255, 10)
percentage of data reamin after removing duplicate values and removing revie
ws with neutral scores  63.90
```

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 [10]:
```
# removing reviews where "HelpfulnessNumerator>HelpfulnessDenominator"
final=dataset_no_dup[dataset_no_dup['HelpfulnessNumerator']<=dataset_no_dup['H
elpfulnessDenominator']]
```

In [11]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

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

```
(363253, 10)
```

Out[11]:
```
1    306222
0     57031
Name: Score, dtype: int64
```

[3] Preprocessing

[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 [8]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
ove-all-tags-from-an-element
from bs4 import BeautifulSoup

# 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


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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he'
, 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becau
se', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [9]:

```
# 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 i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████
█| 363253/363253 [02:56<00:00, 2063.41it/s]
```

In [10]:
```
preprocessed_reviews[:5]
```

Out[10]:
```
['bought several vitality canned dog food products found good quality produc
t looks like stew processed meat smells better labrador finicky appreciates
product better',
 'product arrived labeled jumbo salted peanuts peanuts actually small sized
unsalted not sure error vendor intended represent product jumbo',
 'confection around centuries light pillowy citrus gelatin nuts case filbert
s cut tiny squares liberally coated powdered sugar tiny mouthful heaven not
chewy flavorful highly recommend yummy treat familiar story c lewis lion wit
ch wardrobe treat seduces edmund selling brother sisters witch',
 'looking secret ingredient robitussin believe found got addition root beer
extract ordered good made cherry soda flavor medicinal',
 'great taffy great price wide assortment yummy taffy delivery quick taffy l
over deal']
```

In [11]:
```
final['preprocessed_reviews']=preprocessed_reviews
```

In [12]:
```
# splitting the dataset in train , cv and test

n=final.shape[0] #size of final dataset

train=final.iloc[:round(0.60*n),:]
cv=final.iloc[round(0.60*n):round(0.80*n),:]
test=final.iloc[round(0.80*n):round(1.0*n),:]
```

In [13]:
```
from sklearn.model_selection import train_test_split
x_training,x_test,y_training,y_test=train_test_split(preprocessed_reviews,fina
l["Score"] ,test_size=0.20, random_state=42)
x_train,x_cv,y_train,y_cv=train_test_split(x_training,y_training, test_size=0.
25, random_state=42)
```

In [48]:
```
len(x_train),len(y_train),len(x_test),len(y_test),len(x_cv),len(y_cv)
```

Out[48]:
```
(217951, 217951, 72651, 72651, 72651, 72651)
```

In [14]:
```
# saving train and test dataset using pickle for fututre use

'''file=open("x_train.pkl","wb")
pickle.dump(x_train,file)
file.close()

file=open('x_cv.pkl','wb')
pickle.dump(x_cv,file)
file.close

file=open("x_test.pkl",'wb')
```

```
pickle.dump(x_test,file)
file.close()


file=open("y_train.pkl","wb")
pickle.dump(y_train,file)
file.close()

file=open('y_cv.pkl','wb')
pickle.dump(y_cv,file)
file.close

file=open("y_test.pkl",'wb')
pickle.dump(y_test,file)
file.close()

'''
```

In [2]:

```
#loading train and test and validation dataset

file=open("x_train.pkl","rb")
x_train=pickle.load(file) # loading 'train' dataset

file=open("x_cv.pkl",'rb')
x_cv=pickle.load(file) # loading 'validation' dataset

file=open("x_test.pkl",'rb')
x_test=pickle.load(file) # loading 'test' dataset

file=open("y_train.pkl","rb")
y_train=pickle.load(file) # loading 'train' dataset

file=open("y_cv.pkl",'rb')
y_cv=pickle.load(file) # loading 'validation' dataset

file=open("y_test.pkl",'rb')
y_test=pickle.load(file) # loading 'test' dataset
```

[4] Featurization

[4.1] BAG OF WORDS

In [3]:

```
#BoW

count_vect = CountVectorizer()#in scikit-learn

x_train_bow=count_vect.fit_transform(x_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*100)

# transform cv and test dataset
x_cv_bow=count_vect.transform(x_cv)
x_test_bow=count_vect.transform(x_test)
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa', 'aaaaaaaaaaaa',
'aaaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaaaaaaaaaa', 'aaaaaaaaaaaaaaaaaaaaaa

aaaaaaaaaaaaaaaaaaaaaaaa']
```

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

In [15]:
```
x_train_bow.shape,x_test_bow.shape,x_cv_bow.shape
```

Out[15]:
```
((217951, 90000), (72651, 90000), (72651, 90000))
```

In [17]:
```
# saving train_bow and test_bow dataset using pickle for future use

'''file=open("x_train_bow.pkl","wb")
pickle.dump(x_train_bow,file)
file.close()

file=open("x_test_bow.pkl",'wb')
pickle.dump(x_test_bow,file)
file.close()

file=open("x_cv_bow.pkl",'wb')
pickle.dump(x_cv_bow,file)
file.close()
'''
```

In [15]:
```
#loading train_bow and test _bow
file=open('x_train_bow.pkl','rb')
x_train_bow=pickle.load(file)

file=open('x_test_bow.pkl','rb')
x_test_bow=pickle.load(file)

file=open('x_cv_bow.pkl','rb')
x_cv_bow=pickle.load(file)
```

[4.3] TF-IDF

In [8]:
```
# tf-idf "from sklearn.feature_extraction.text.TfidfVectorizer"
tf_idf=TfidfVectorizer()

train_tf_idf=tf_idf.fit_transform(x_train)
cv_tf_idf=tf_idf.transform(x_cv)
test_tf_idf=tf_idf.transform(x_test)
```

In [9]:
```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler(with_mean=False)

train_tf_idf=sc.fit_transform(train_tf_idf)
cv_tf_idf=sc.transform(cv_tf_idf)
test_tf_idf=sc.transform(test_tf_idf)
```

In [21]:
```
# saving train_tf_idf and test_tf_idf dataset using pickle for fututre use

'''file=open("train_tf_idf.pkl","wb")
pickle.dump(train_tf_idf,file)
file.close()

file=open("cv_tf_idf.pkl",'wb')
pickle.dump(cv_tf_idf,file)
```

```
file.close()

file=open("test_tf_idf.pkl",'wb')
pickle.dump(test_tf_idf,file)
file.close()

'''
```

In [4]:

```
#loading train_tf_idf and test_tf_idf
file=open('train_tf_idf.pkl','rb')
train_tf_idf=pickle.load(file)

file=open('cv_tf_idf.pkl','rb')
cv_tf_idf=pickle.load(file)

file=open('test_tf_idf.pkl','rb')
test_tf_idf=pickle.load(file)
```

[4.4] Word2Vec

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

[4.4.1.1] Avg W2v

In [ ]:

```
# converting our text-->vector using w2v with 50-dim
# more the dimension of each word = better the semantic of word
# using lib from "gensim.models.Word2Vec"
# to run w2v we need list of list of the words as w2v covert each world into n
umber of dim


# for train_w2v
list_of_sent_train=[]
for sent in x_train:
    list_of_sent_train.append((str(sent)).split())
w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50)

# vocablary of w2v model of amazon dataset
vocab=w2v_model.wv.vocab
len(vocab)

#-----------------------------------------------------------------------------
--------------------

# for test_w2v
list_of_sent_cv=[]
for sent in x_cv:
    list_of_sent_cv.append((str(sent)).split())

#-----------------------------------------------------------------------------
--------------------

# for test_w2v
list_of_sent_test=[]
```

```
for sent in x_test:
    list_of_sent_test.append((str(sent)).split())
```

```
C:\Users\Adarsh\Anaconda3\lib\site-packages\gensim\models\base_any2vec.py:74
3: UserWarning: C extension not loaded, training will be slow. Install a C c
ompiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
```

In [7]:

```
'''
    -->procedure to make avg w2v of each reviews
    1. find the w2v of each word
    2. sum-up w2v of each word in a sentence
    3. divide the total w2v of sentence by total no. of words in the sentence
'''

# average Word2Vec
# compute average word2vec for each review.
train_w2v = []; # the avg-w2v for each sentence/review in train dataset is sto
red in this list

for sent in list_of_sent_train: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in vocab:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_w2v.append(sent_vec)

print(len(train_w2v))

#-----------------------------------------------------------------------------
------

cv_w2v = []; # the avg-w2v for each sentence/review in test dataset is stored
 in this list

for sent in list_of_sent_cv: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in vocab:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_w2v.append(sent_vec)

print(len(cv_w2v))


#-----------------------------------------------------------------------------
------

test_w2v = []; # the avg-w2v for each sentence/review in test dataset is store
d in this list
```

```
for sent in list_of_sent_test: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in vocab:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_w2v.append(sent_vec)

print(len(test_w2v))
```

```
217951
72651
72651
```

In [9]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler(with_mean=True)

train_w2v=sc.fit_transform(train_w2v)
cv_w2v=sc.transform(cv_w2v)
test_w2v=sc.transform(test_w2v)
```

In [10]:

```
# saving train_w2v and test_w2v dataset using pickle for fututre use

'''file=open("train_w2v.pkl","wb")
pickle.dump(train_w2v,file)
file.close()

file=open("cv_w2v.pkl",'wb')
pickle.dump(cv_w2v,file)
file.close()

file=open("test_w2v.pkl",'wb')
pickle.dump(test_w2v,file)
file.close()
'''
```

In [5]:

```
#loading train_w2v and test_w2v
file=open('train_w2v.pkl','rb')
train_w2v=pickle.load(file)

file=open('cv_w2v.pkl','rb')
cv_w2v=pickle.load(file)

file=open('test_w2v.pkl','rb')
test_w2v=pickle.load(file)
```

In [21]:

```
train_w2v[0]
```

Out[21]:

```
array([-0.0080363 , -0.55650226, -2.24174777,  1.02574886,  0.12327979,
       -0.19916425, -1.06522974,  0.89144715, -1.13231167,  2.54008377,
        0.8032532 ,  0.3404576 ,  1.6792167 , -0.98081078,  1.08851643,
       -0.72007858, -0.65714762, -0.56007184,  0.01985994,  2.12137305,
       -0.09203752, -0.23671867, -1.63326771,  1.04496922,  0.45004579,
        0.3219116 ,  0.78335079,  0.54301334, -2.4968575 ,  0.35478244,
        1.46397278, -0.01982212, -0.1817636 ,  1.35729521, -0.61338792,
       -1.68822842, -0.84256537, -0.59978494,  0.40587478, -0.49775708,
```

```
        0.31289323,  0.34938107, -0.18756661, -2.25982333,  0.01440547,
       -0.97699964, -0.10107761,  0.28043456,  1.88480264, -0.81507891])
```

In [12]:
```
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  26749
sample words  ['really', 'nice', 'seasoning', 'bought', 'product', 'sam', 'c
lub', 'happy', 'able', 'purchase', 'cannot', 'get', 'anymore', 'use', 'meat
s', 'spaghetti', 'coffee', 'not', 'bad', 'defiantly', 'anything', 'special',
'price', 'could', 'ethical', 'fair', 'trade', 'organic', 'shade', 'grown',
'etc', 'taste', 'ok', 'stick', 'dean', 'beans', 'smooth', 'flavorful', 'medi
um', 'roast', 'pleasantly', 'surprised', 'k', 'cup', 'would', 'deal', 'dre
w', 'glad', 'dogs', 'love']
```

[4.4.1.2] TFIDF weighted W2v

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

In [ ]:
```
# TF-IDF weighted Word2Vec Train
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

train_tf_idf_w2v = []; # the tfidf-w2v for each sentence/review is stored in t
his list
row=0;

#list_of_sentence_train=list_of_sent_train[:30000] # reducing the size of trai
n list due to computational constrain

for sent in tqdm(list_of_sent_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_tf_idf_w2v.append(sent_vec)
    row += 1

len(train_tf_idf_w2v)
```

In [15]:
```
# TF-IDF weighted Word2Vec cv
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

cv_tf_idf_w2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;

#list_of_sentence_train=list_of_sent_train[:30000] # reducing the size of train list due to computational constrain

for sent in tqdm(list_of_sent_cv[:20000]): # 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
    cv_tf_idf_w2v.append(sent_vec)
    row += 1

len(cv_tf_idf_w2v)
```

```
100%|██████████| 20000/20000 [17:19<00:00, 19.23it/s]
```

Out[15]: 20000

In [16]:

```
# TF-IDF weighted Word2Vec Test
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

test_tf_idf_w2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;

#list_of_sentence_train=list_of_sent_train[:30000] # reducing the size of train list due to computational constrain

for sent in tqdm(list_of_sent_test[:20000]): # 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
    test_tf_idf_w2v.append(sent_vec)
    row += 1
```

```
len(test_tf_idf_w2v)
```

```
100%|██████████| 20000/20000 [17:31<00:00, 19.03it/s]
```

Out[16]: 20000

In [15]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler(with_mean=True)

train_tf_idf_w2v=sc.fit_transform(train_tf_idf_w2v)
cv_tf_idf_w2v=sc.transform(cv_tf_idf_w2v)
test_tf_idf_w2v=sc.transform(test_tf_idf_w2v)
```

In [16]:
```
# saving train_tf_idf_w2v and test_tf_idf_w2v dataset using pickle for fututre
use

'''file=open("train_tf_idf_w2v.pkl","wb")
pickle.dump(train_tf_idf_w2v,file)
file.close()

file=open("cv_tf_idf_w2v.pkl",'wb')
pickle.dump(cv_tf_idf_w2v,file)
file.close()

file=open("test_tf_idf_w2v.pkl",'wb')
pickle.dump(test_tf_idf_w2v,file)
file.close()

'''
```

In [6]:
```
#loading train_tf_idf_w2v and test_tf_idf_w2v
file=open('train_tf_idf_w2v.pkl','rb')
train_tf_idf_w2v=pickle.load(file)

file=open('cv_tf_idf_w2v.pkl','rb')
cv_tf_idf_w2v=pickle.load(file)

file=open('test_tf_idf_w2v.pkl','rb')
test_tf_idf_w2v=pickle.load(file)
```

[5] Assignment 8: Decision Trees

1. **Apply Decision Trees 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)

2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in range [5, 10, 100, 500])**

    - 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

3. **Graphviz**

    - Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
    - Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
    - Make sure to print the words in each node of the decision tree instead of printing its index.
    - Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.

4. **Feature importance**

    - Find the top 20 important features from both feature sets Set 1 and Set 2 using `feature_importances_` method of [Decision Tree Classifier]() and print their corresponding feature names

5. **Feature engineering**

    - To increase the performance of your model, you can also experiment with with feature engineering like :
        - Taking length of reviews as another feature.
        - Considering some features from review summary as well.

6. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
      Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
      Along with plotting ROC curve, you need to print the [confusion matrix]() with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps.]()

7. [**Conclusion**]()

    - [You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link]()

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.]()

Applying Decision Trees

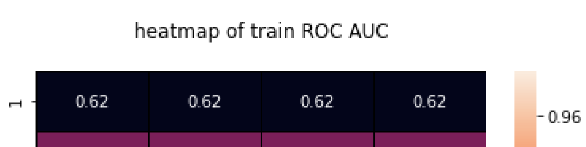
[5.1] Applying Decision Trees on BOW, SET 1

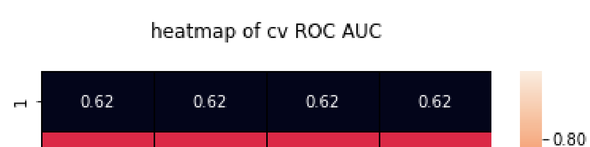
In [55]:

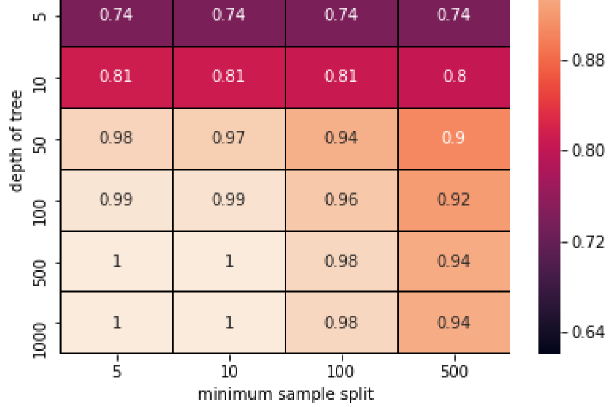
```python
# By using grid search
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param={'max_depth':[1, 5, 10, 50, 100, 500, 1000],'min_samples_split':[5, 10, 100, 500]}

# taking decision tree as estimator for grid search
dt_clf=DecisionTreeClassifier(criterion='gini',class_weight='balanced')

gsCV=GridSearchCV(estimator=dt_clf, param_grid=param, scoring='roc_auc',return_train_score=True,n_jobs=-2)
gsCV.fit(x_train_bow,y_train)

# storing results
result=gsCV.cv_results_

roc_auc_cv=result['mean_test_score'].reshape(7,4) #reshaping for heatmap visualisation
roc_auc_train=result['mean_train_score'].reshape(7,4) #reshaping for heatmap visualisation
```

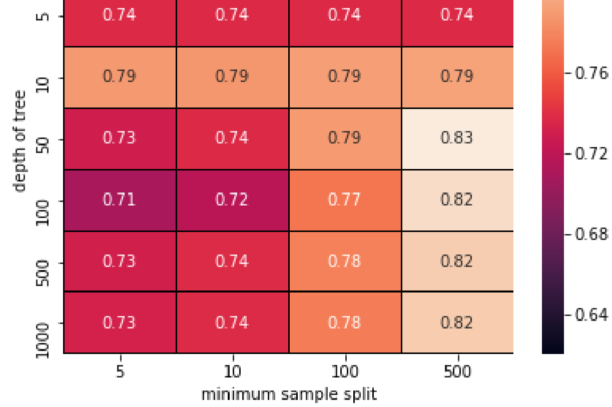
In [56]:

```python
# visualising train_ROC_AUC and cv_ROC_AUC using heatmap

from matplotlib import gridspec

fig = plt.figure(figsize=(14,5))
gs  = gridspec.GridSpec(1, 2)

ax0 = plt.subplot(gs[0])

# train heatmap
sn.heatmap(data=roc_auc_train,linewidths=0.01,annot=True,xticklabels=param['min_samples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax0)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of train ROC AUC\n')

#*****************************************************************************************************

ax1 = plt.subplot(gs[1])

# cv heatmap
sn.heatmap(data=roc_auc_cv,linewidths=0.01,annot=True,xticklabels=param['min_samples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax1)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of cv ROC AUC\n')
plt.show()
```









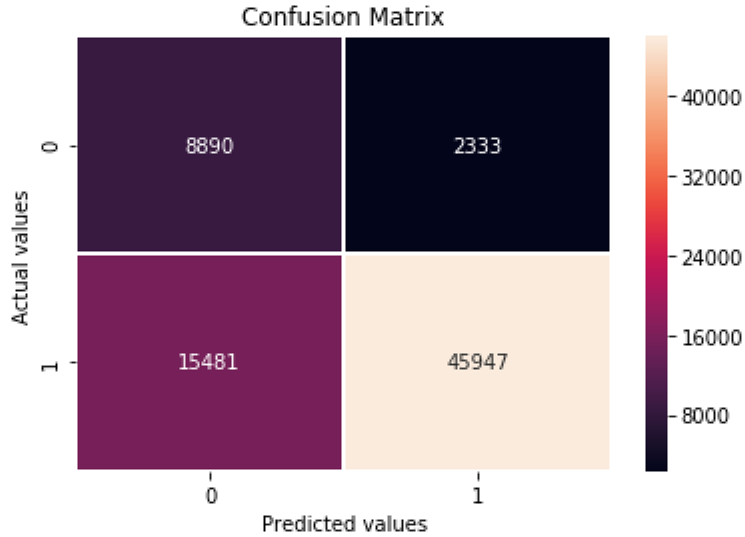
best hyperparamter

In [58]:
```
# best estimators
print(f"best hyperparameters are: {gsCV.best_params_} ")
```

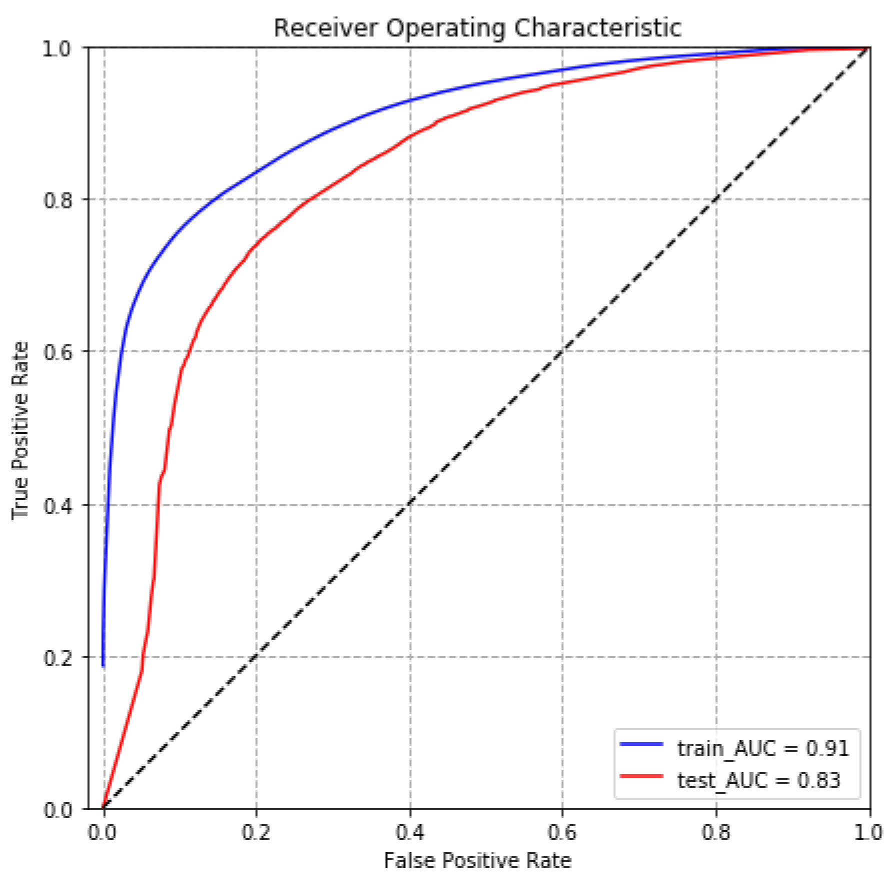
```
best hyperparameters are: {'max_depth': 50, 'min_samples_split': 500}
```

In [57]:
```
#Testing
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import roc_curve

#probabilty score for ROC_AUC score
y_train_pred_proba=gsCV.predict_proba(x_train_bow)[:,1]
y_test_pred_proba=gsCV.predict_proba(x_test_bow)[:,1]

train_roc_score=roc_auc_score(y_train,y_train_pred_proba)
test_roc_score=roc_auc_score(y_test,y_test_pred_proba)


#ploting confusion matrix
y_pred=gsCV.predict(x_test_bow)
sn.heatmap(confusion_matrix(y_test,y_pred),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test,y_pred))


# ROC Curve (reference:stack overflow with little modification)
train_fpr, train_tpr, train_threshold =roc_curve(y_train, y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test, y_test_pred_proba)

#ploting ROC curve
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % train_roc_sc
ore)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % test_roc_score)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
plt.grid(linestyle='--', linewidth=1)
plt.show()
```




```
classification report:
             precision    recall  f1-score   support

          0       0.36      0.79      0.50     11223
          1       0.95      0.75      0.84     61428

avg / total       0.86      0.75      0.79     72651
```



[5.1.1] Top 20 important features from SET 1

In [61]:
```
feature_importance_scores=(gsCV.best_estimator_.feature_importances_) # featur
e importance scores from decision tree classifier

# storing the index of top 20 features
top_features_index=feature_importance_scores.argsort()[-21:-1]

#storing the name of features from BOW vectorizer
bow_feat=count_vect.get_feature_names()
feature_names=pd.DataFrame(bow_feat)

top_features=pd.DataFrame({'feature importance scores':feature_importance_scor
es[top_features_index],
                           'feature names':feature_names.iloc[top_features_ind
ex][0]})
```

In [62]:
```
# top 20 important features
top_features
```

Out[62]:

	feature importance scores	feature names
78335	0.009242	tasty
24534	0.009809	easy
5533	0.010182	awful
88178	0.010621	wonderful
50646	0.010942	money
37571	0.011154	horrible
52848	0.011728	nice
88439	0.011950	worst
5791	0.014831	bad
27089	0.014842	excellent
28439	0.015853	favorite
80070	0.016073	thought
46321	0.019932	loves
33533	0.027168	good
58437	0.028075	perfect
22377	0.032052	disappointed

46287	0.033785	love
20754	0.037634	delicious
7482	0.046579	best
34245	0.097637	great

[5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

note: install pydotplus and graphviz libraries if not installed.

In [71]:

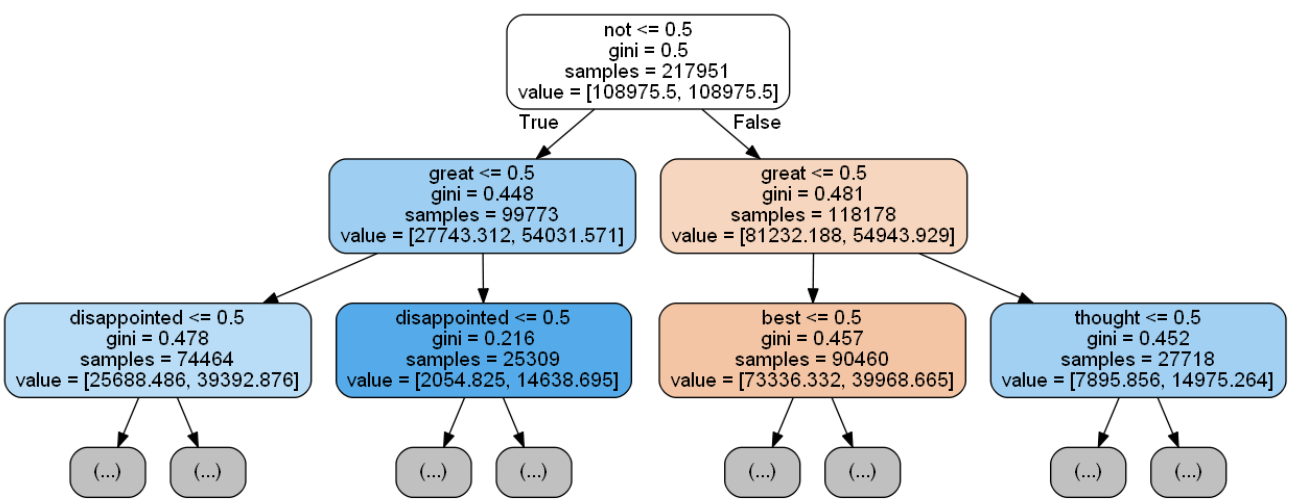
```
from sklearn.tree import export_graphviz
import pydotplus

# Visualize data
dot_data = tree.export_graphviz(gsCV.best_estimator_,
                                max_depth=2,
                                feature_names=bow_feat,
                                out_file=None,
                                filled=True,
                                rounded=True)  #feature names are taken from BOW count vectoriser

graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png('tree_bow.png')
```

Out[71]: True

In [36]:

```
from IPython.display import Image
Image('tree_bow.png')
```

Out[36]:



[5.2] Applying Decision Trees on TFIDF, SET 2

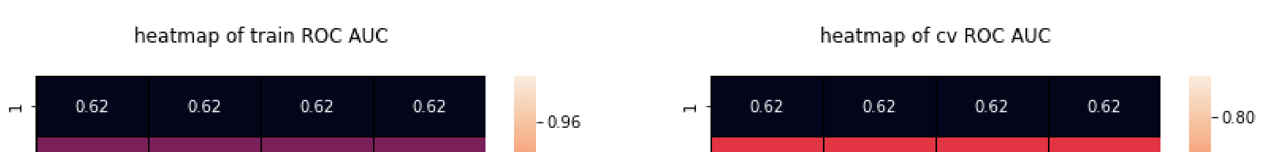
In [3]:

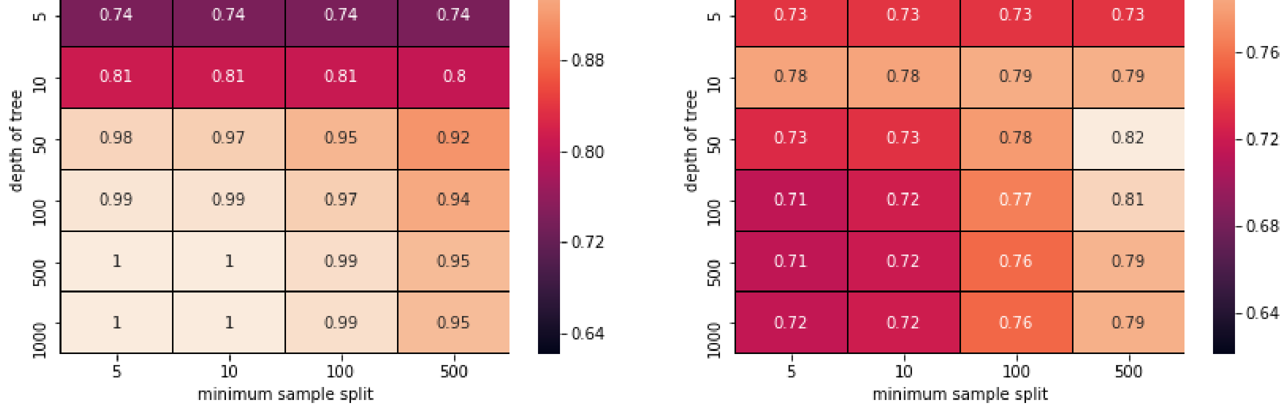
```
# By using grid search
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param={'max_depth':[1, 5, 10, 50, 100, 500, 1000],'min_samples_split':[5, 10, 100, 500]}

# taking decision tree as estimator for grid search
```

```
dt_clf=DecisionTreeClassifier(criterion='gini',class_weight='balanced')

gsCV=GridSearchCV(estimator=dt_clf, param_grid=param, scoring='roc_auc',return
_train_score=True,n_jobs=-2)
gsCV.fit(train_tf_idf,y_train)

# storing results
result=gsCV.cv_results_

roc_auc_cv=result['mean_test_score'].reshape(7,4) #reshaping for heatmap visua
lisation
roc_auc_train=result['mean_train_score'].reshape(7,4) #reshaping for heatmap v
isualisation
```

In [34]:

```
# visualising train_ROC_AUC and cv_ROC_AUC using heatmap

from matplotlib import gridspec

fig = plt.figure(figsize=(14,5))
gs  = gridspec.GridSpec(1, 2)

ax0 = plt.subplot(gs[0])

# train heatmap
sn.heatmap(data=roc_auc_train,linewidths=0.01,annot=True,xticklabels=param['mi
n_samples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax0)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of train ROC AUC\n')

#*****************************************************************************
******************************

ax1 = plt.subplot(gs[1])

# cv heatmap
sn.heatmap(data=roc_auc_cv,linewidths=0.01,annot=True,xticklabels=param['min_s
amples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax1)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of cv ROC AUC\n')
plt.show()
```





best hyperparamter

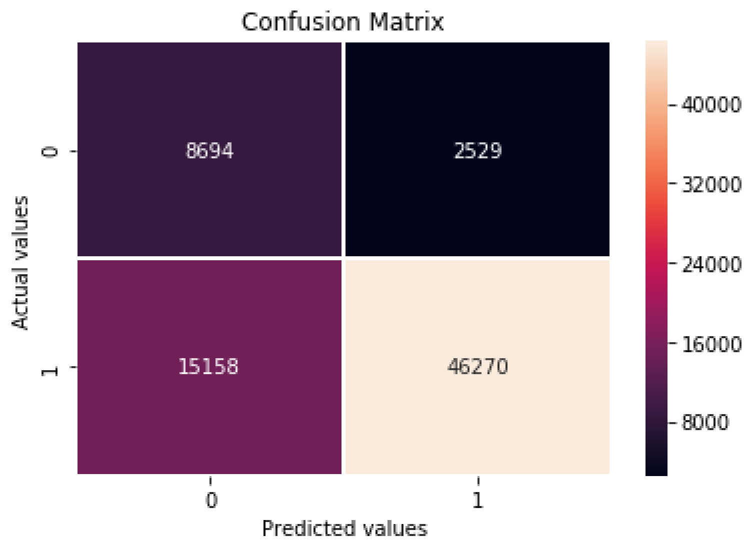
In [30]:
```
# best estimators
print(f"best hyperparameters are: {gsCV.best_params_} ")
```

```
best hyperparameters are: {'max_depth': 50, 'min_samples_split': 500}
```

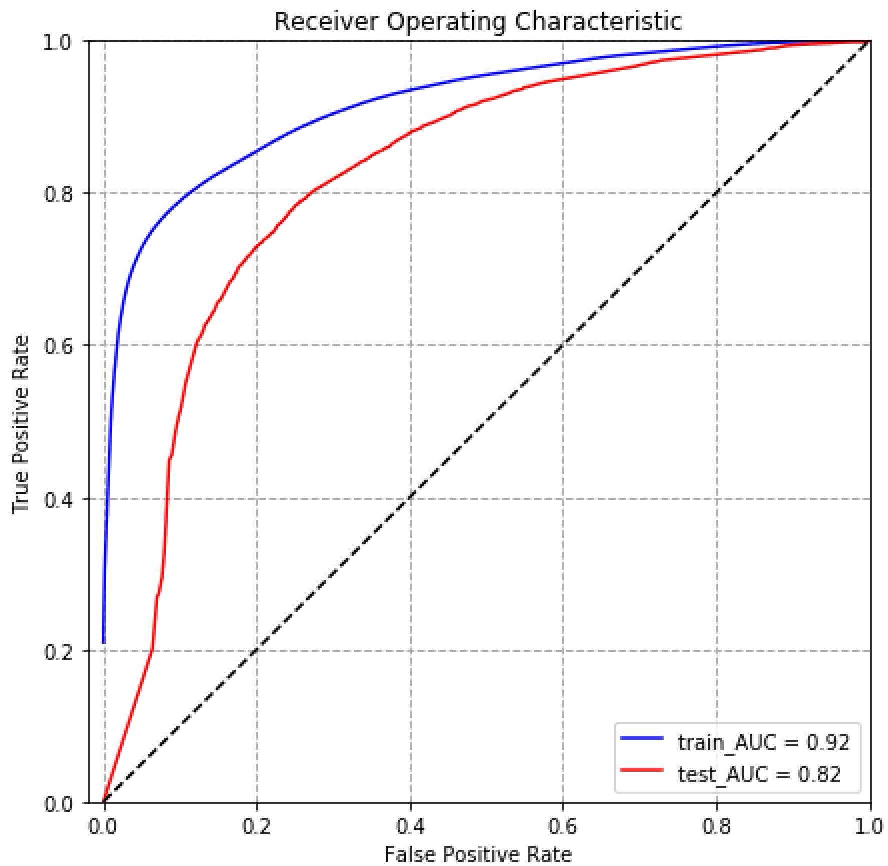
In [15]:
```
#Testing
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import roc_curve

optimum_depth=gsCV.best_estimator_.max_depth
optimum_split=gsCV.best_estimator_.min_samples_split

print(f"succesfully trained using optimum_depth: {optimum_depth} and optimim_s
plit: {optimum_split}" )

#probabilty score for ROC_AUC score
y_train_pred_proba=gsCV.predict_proba(train_tf_idf)[:,1]
y_test_pred_proba=gsCV.predict_proba(test_tf_idf)[:,1]

train_roc_score=roc_auc_score(y_train,y_train_pred_proba)
test_roc_score=roc_auc_score(y_test,y_test_pred_proba)


#ploting confusion matrix
y_pred=gsCV.predict(test_tf_idf)
sn.heatmap(confusion_matrix(y_test,y_pred),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test,y_pred))


# ROC Curve (reference:stack overflow with little modification)
train_fpr, train_tpr, train_threshold =roc_curve(y_train, y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test, y_test_pred_proba)

#ploting ROC curve
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % train_roc_sc
ore)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % test_roc_score)
```

```
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
succesfully trained using optimum_depth: 50 and optimim_split: 500
```




```
classification report:
             precision    recall  f1-score   support

          0       0.36      0.77      0.50     11223
          1       0.95      0.75      0.84     61428

avg / total       0.86      0.76      0.79     72651
```



[5.2.1] Top 20 important features from SET 2

In [19]:

```
feature_importance_scores=(gsCV.best_estimator_.feature_importances_) # featur
e importance scores from decision tree classifier

# storing the index of top 20 features
top_features_index=feature_importance_scores.argsort()[-21:-1]

#storing the name of features from BOW vectorizer
tfidf_feat=tf_idf.get_feature_names()
feature_names=pd.DataFrame(tfidf_feat)

top_features=pd.DataFrame({'feature importance scores':feature_importance_scor
es[top_features_index],
                           'feature names':feature_names.iloc[top_features_ind
ex][0]})

# top 20 important features
top_features
```

Out[19]:

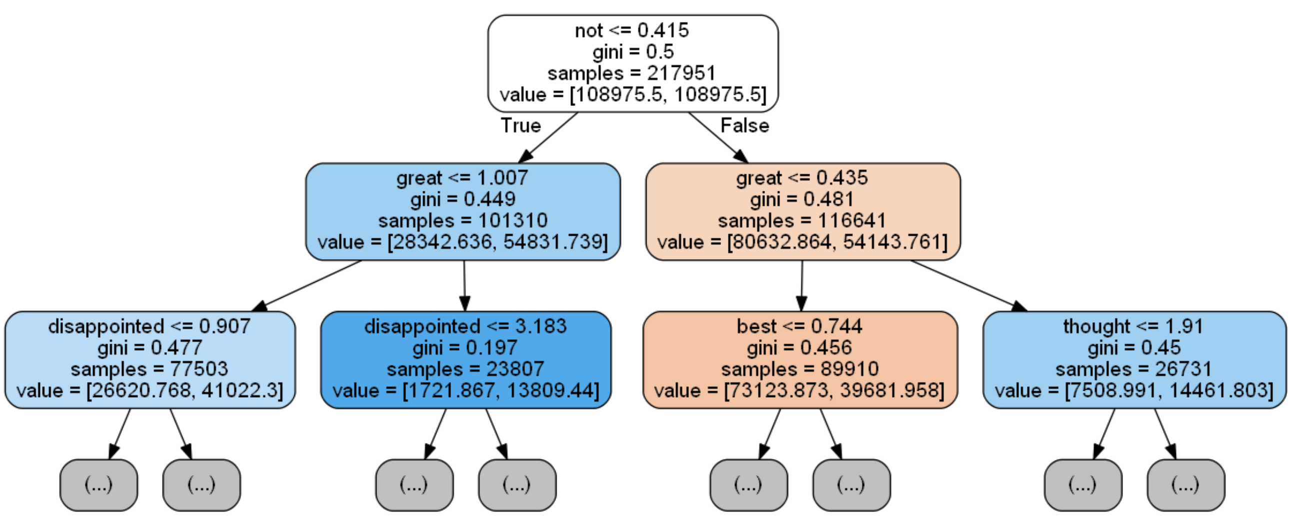
	feature importance scores	feature names
78335	0.009322	tasty
36836	0.009568	highly
24534	0.009762	easy
5533	0.010068	awful
88178	0.011002	wonderful
88439	0.011354	worst
50646	0.011728	money
52848	0.012564	nice

52848	0.012564	nice
80070	0.012902	thought
27089	0.015061	excellent
28439	0.016532	favorite
5791	0.017061	bad
46321	0.020176	loves
58437	0.026924	perfect
33533	0.028194	good
22377	0.031310	disappointed
46287	0.033811	love
20754	0.036823	delicious
7482	0.044036	best
34245	0.095048	great

[5.2.2] Graphviz visualization of Decision Tree on TFIDF, SET 2

In [23]:
```
from sklearn.tree import export_graphviz
import pydotplus

# Visualize data
dot_data = export_graphviz(gsCV.best_estimator_,
                           max_depth=2,
                           feature_names=tf_idf.get_feature_names(),
                           out_file=None,
                           filled=True,
                           rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png('tree_tfidf.png')

from IPython.display import Image
Image('tree_tfidf.png')
```

Out[23]:



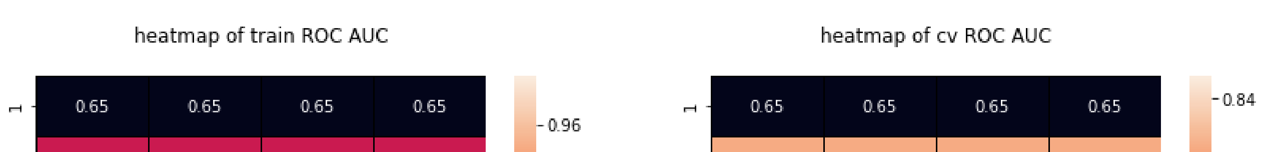
[5.3] Applying Decision Trees on AVG W2V, SET 3

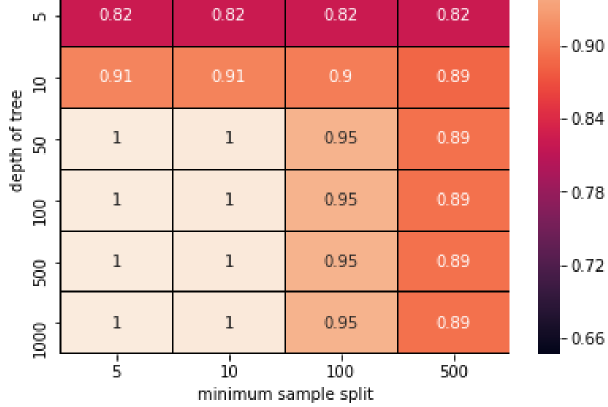
In [37]:
```
# By using grid search
```

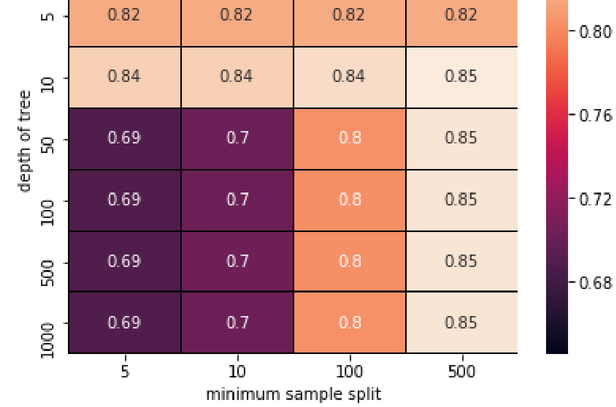
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param={'max_depth':[1, 5, 10, 50, 100, 500, 1000],'min_samples_split':[5, 10, 100, 500]}

# taking decision tree as estimator for grid search
dt_clf=DecisionTreeClassifier(criterion='gini',class_weight='balanced')

gsCV=GridSearchCV(estimator=dt_clf, param_grid=param, scoring='roc_auc',return_train_score=True,n_jobs=-2)
gsCV.fit(train_w2v,y_train)

# storing results
result=gsCV.cv_results_

roc_auc_cv=result['mean_test_score'].reshape(7,4) #reshaping for heatmap visualisation
roc_auc_train=result['mean_train_score'].reshape(7,4) #reshaping for heatmap visualisation
```

In [38]:

```python
# visualising train_ROC_AUC and cv_ROC_AUC using heatmap

from matplotlib import gridspec

fig = plt.figure(figsize=(14,5))
gs  = gridspec.GridSpec(1, 2)

ax0 = plt.subplot(gs[0])

# train heatmap
sn.heatmap(data=roc_auc_train,linewidths=0.01,annot=True,xticklabels=param['min_samples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax0)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of train ROC AUC\n')

#*****************************************************************************************************************

ax1 = plt.subplot(gs[1])

# cv heatmap
sn.heatmap(data=roc_auc_cv,linewidths=0.01,annot=True,xticklabels=param['min_samples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax1)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of cv ROC AUC\n')
plt.show()
```







best hyperparameter

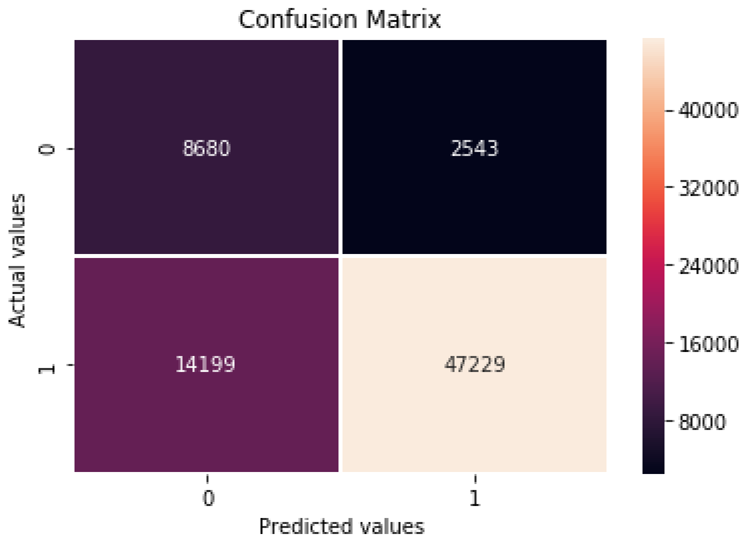
In [39]:
```
# best estimators
print(f"best hyperparameters are: {gsCV.best_params_} ")
```

```
best hyperparameters are: {'max_depth': 10, 'min_samples_split': 500}
```

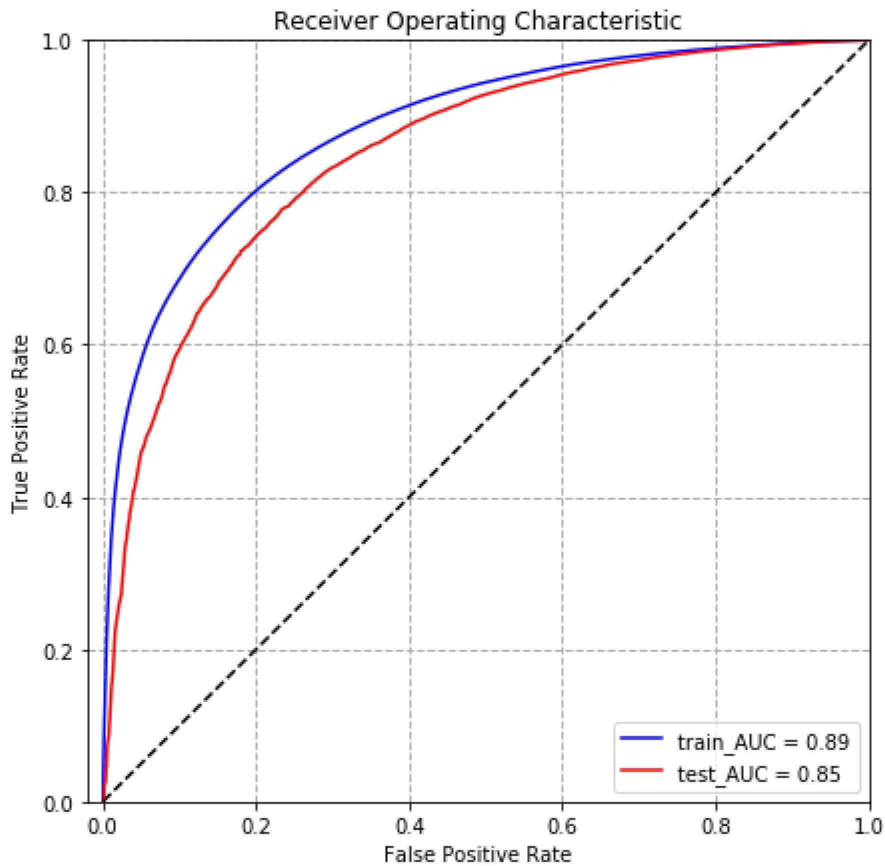
In [40]:
```
#Testing
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import roc_curve

optimum_depth=gsCV.best_estimator_.max_depth
optimum_split=gsCV.best_estimator_.min_samples_split

print(f"succesfully trained using optimum_depth: {optimum_depth} and optimim_s
plit: {optimum_split}" )

#probabilty score for ROC_AUC score
y_train_pred_proba=gsCV.predict_proba(train_w2v)[:,1]
y_test_pred_proba=gsCV.predict_proba(test_w2v)[:,1]

train_roc_score=roc_auc_score(y_train,y_train_pred_proba)
test_roc_score=roc_auc_score(y_test,y_test_pred_proba)


#ploting confusion matrix
y_pred=gsCV.predict(test_w2v)
sn.heatmap(confusion_matrix(y_test,y_pred),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test,y_pred))


# ROC Curve (reference:stack overflow with little modification)
train_fpr, train_tpr, train_threshold =roc_curve(y_train, y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test, y_test_pred_proba)

#ploting ROC curve
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % train_roc_sc
ore)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % test_roc_score)
```

```
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
succesfully trained using optimum_depth: 10 and optimim_split: 500
```




```
classification report:
             precision    recall  f1-score   support

          0       0.38      0.77      0.51     11223
          1       0.95      0.77      0.85     61428

avg / total       0.86      0.77      0.80     72651
```



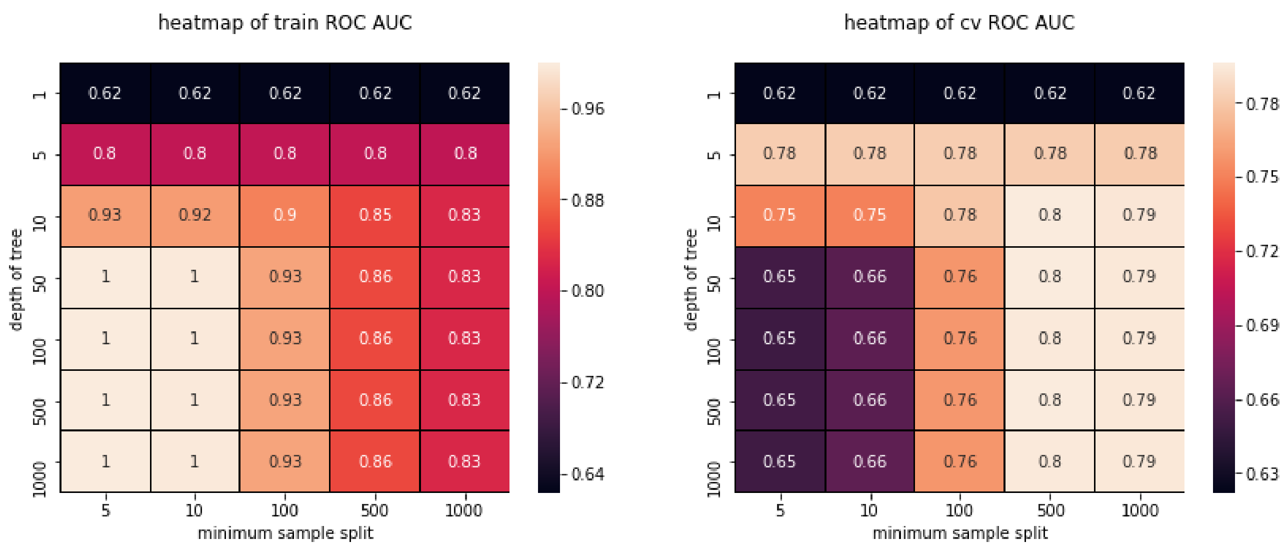
[5.4] Applying Decision Trees on TFIDF W2V, SET 4

Note: only 60k datapoints are used to train model and 20k datapoints for testing

In [54]:
```
# By using grid search
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param={'max_depth':[1, 5, 10, 50, 100, 500, 1000],'min_samples_split':[5, 10,
100, 500,1000]}

# taking decision tree as estimator for grid search
dt_clf=DecisionTreeClassifier(criterion='gini',class_weight='balanced')

gsCV=GridSearchCV(estimator=dt_clf, param_grid=param, scoring='roc_auc',return
_train_score=True,n_jobs=-2)
gsCV.fit(train_tf_idf_w2v[:60000],y_train[:60000])

# storing results
result=gsCV.cv_results_

roc_auc_cv=result['mean_test_score'].reshape(7,5) #reshaping for heatmap visua
lisation
roc_auc_train=result['mean_train_score'].reshape(7,5) #reshaping for heatmap v
isualisation
```

In [52]:
```
# visualising train_ROC_AUC and cv_ROC_AUC using heatmap

from matplotlib import gridspec

fig = plt.figure(figsize=(14,5))
```

```
gs   = gridspec.GridSpec(1, 2)

ax0 = plt.subplot(gs[0])

# train heatmap
sn.heatmap(data=roc_auc_train,linewidths=0.01,annot=True,xticklabels=param['mi
n_samples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax0)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of train ROC AUC\n')

#**********************************************************************
******************************

ax1 = plt.subplot(gs[1])

# cv heatmap
sn.heatmap(data=roc_auc_cv,linewidths=0.01,annot=True,xticklabels=param['min_s
amples_split'],
           yticklabels=param['max_depth'],linecolor='black',fmt='.2g',ax=ax1)

plt.xlabel('minimum sample split')
plt.ylabel('depth of tree')
plt.title('heatmap of cv ROC AUC\n')
plt.show()
```



best hyperparameter

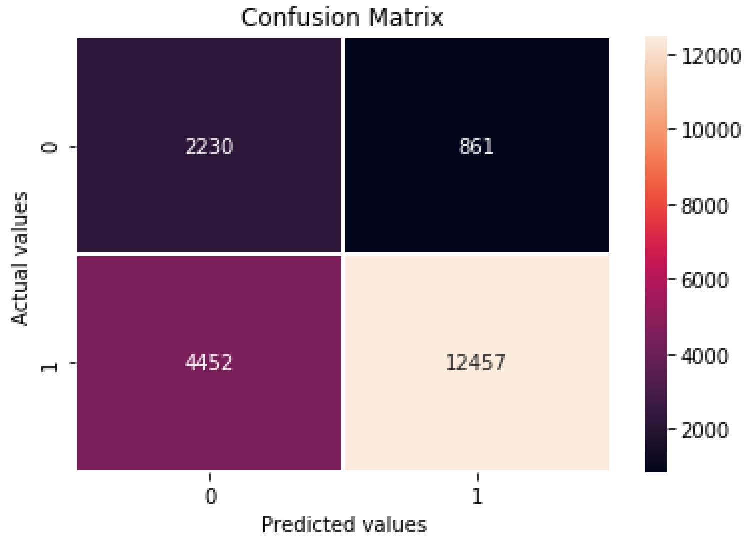
In [53]:
```
# best estimators
print(f"best hyperparameters are: {gsCV.best_params_} ")
```

```
best hyperparameters are: {'max_depth': 10, 'min_samples_split': 500}
```

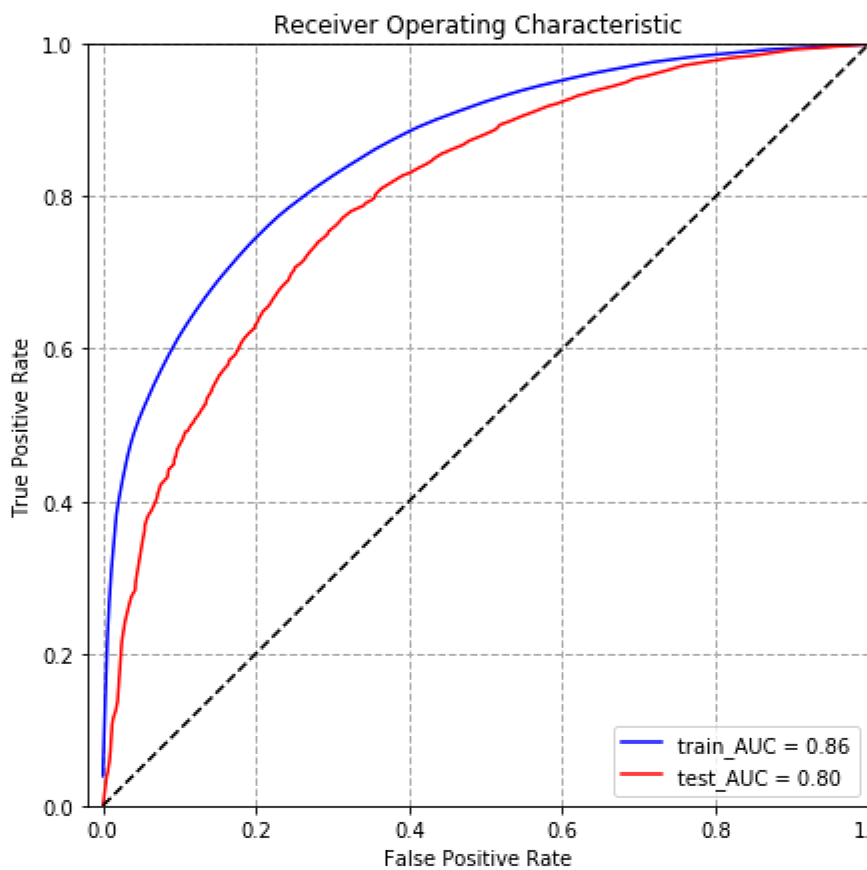
In [48]:
```
#Testing
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import roc_curve

optimum_depth=gsCV.best_estimator_.max_depth
optimum_split=gsCV.best_estimator_.min_samples_split
```

```python
print(f"succesfully trained using optimum_depth: {optimum_depth} and optimim_split: {optimum_split}" )

#probabilty score for ROC_AUC score
y_train_pred_proba=gsCV.predict_proba(train_tf_idf_w2v[:60000])[:,1]
y_test_pred_proba=gsCV.predict_proba(test_tf_idf_w2v[:20000])[:,1]

train_roc_score=roc_auc_score(y_train[:60000],y_train_pred_proba)
test_roc_score=roc_auc_score(y_test[:20000],y_test_pred_proba)


#ploting confusion matrix
y_pred=gsCV.predict(test_tf_idf_w2v[:20000])
sn.heatmap(confusion_matrix(y_test[:20000],y_pred),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pred))


# ROC Curve (reference:stack overflow with little modification)
train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_test_pred_proba)

#ploting ROC curve
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % train_roc_score)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % test_roc_score)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
succesfully trained using optimum_depth: 10 and optimim_split: 500
```




```
classification report:
             precision    recall  f1-score   support

          0       0.33      0.72      0.46      3091
          1       0.94      0.74      0.82     16909

avg / total       0.84      0.73      0.77     20000
```



[6] Conclusions

In [59]:
```
# compare all your models using Prettytable library
from prettytable import PrettyTable

x=PrettyTable(field_names=['text_featurization','train_AUC','test_AUC',{'hyper
```

```
parameter':['max_depth','max_sample,split']}])

x.add_row(['BOW', 91, 83, [50, 500]])
x.add_row(['TFIDF', 92, 82, [50, 500]])
x.add_row(['W2V', 89, 85, [10 ,500]])
x.add_row(['TFIDF_W2V', 86, 80, [10 ,500]])

print(35*"*","prettytable",35*"*","\n")
print(x)
```

```
*********************************** prettytable ***************************
*******

+--------------------+-----------+----------+---------------------------------
------------------------+
| text_featurization | train_AUC | test_AUC | {'hyperparameter': ['max_dept
h', 'max_sample,split']} |
+--------------------+-----------+----------+---------------------------------
------------------------+
|        BOW         |     91    |    83    |                        [50, 50
0]                      |
|       TFIDF        |     92    |    82    |                        [50, 50
0]                      |
|        W2V         |     89    |    85    |                        [10, 50
0]                      |
|     TFIDF_W2V      |     86    |    80    |                        [10, 50
0]                      |
+--------------------+-----------+----------+---------------------------------
------------------------+
```

Observation

1. Runtime complexity increases dramitically with increase of dimentions
2. Best AUC is obtained using W2V vectoriser using optimum_max_depth: 10 and optimum_max_sample_split: 500