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 [6]:

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

#import gensim
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```python
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 [7]:

```python
# 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 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""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

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

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

Number of data points in our data (100000, 10)

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240(
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600(

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

In [8]:

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

In [9]:

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

```
(80668, 7)
```

Out[9]:

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

In [10]:

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

Out[10]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [11]:

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

Out[11]:

```
393063
```

[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:

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(

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 [16]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [17]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

Out[17]:

```
(87775, 10)
```

In [18]:

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

Out[18]:

```
87.775
```

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 [19]:

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

display.head()
```

Out[19]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883

In [20]:

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

In [21]:

```
#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()
```

```
(87773, 10)
```

Out[21]:

```
1    73592
0    14181
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 [22]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [23]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [24]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an
-element
from bs4 import BeautifulSoup

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
```

In [25]:

```
# 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 [26]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [27]:

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [28]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [29]:

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

In [30]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
```

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

```
100%|████████████████████████████████████████████████████████████████████████████| 87773/87773
[01:00<00:00, 1457.82it/s]
```

In [31]:

```
final['Cleaned Text'] = preprocessed_reviews
```

In [32]:

```
sample1 = pd.DataFrame()
```

In [33]:

```
sample1['Cleaned Text'] =preprocessed_reviews
```

In [34]:

```
sample1.tail(3)
```

Out[34]:

	Cleaned Text
87770	trader joe product good quality buy straight t...
87771	coffee supposedly premium tastes watery thin n...
87772	purchased product local store ny kids love qui...

In [35]:

```
k1 = []
```

In [36]:

```
sample1.shape
```

Out[36]:

```
(87773, 1)
```

In [37]:

```
for i in range(0,87773):
    k1.append(len(preprocessed_reviews[i]))
```

In [38]:

```
sample1['Length'] = k1
```

In [39]:

```
sample1.head(3)
```

Out[39]:

	Cleaned Text	Length

	Cleaned Text	Length
0	dogs loves chicken product china wont buying a...	162
1	dogs love saw pet store tag attached regarding...	72
2	infestation fruitflies literally everywhere fl...	406

[3.2] Preprocessing Review Summary

In [40]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

Splitting the Data with feature engineering

In [41]:

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(sample1,final['Score'].values,test_size=0.3
,shuffle=False)
```

In [42]:

```
y_train1.shape
```

Out[42]:

```
(61441,)
```

In [43]:

```
X_train1.shape
```

Out[43]:

```
(61441, 2)
```

In [44]:

```
X_test1.shape
```

Out[44]:

```
(26332, 2)
```

In [45]:

```
type(y_test1)
```

Out[45]:

```
numpy.ndarray
```

In [46]:

```
type(X_test1)
```

Out[46]:

```
pandas.core.frame.DataFrame
```

In [47]:

```
X_train1.head(3)
```

Out[47]:

	Cleaned Text	Length
0	dogs loves chicken product china wont buying a...	162
1	dogs love saw pet store tag attached regarding...	72
2	infestation fruitflies literally everywhere fl...	406

In [48]:

```
X_test1.head(3)
```

Out[48]:

	Cleaned Text	Length
61441	used treat training reward dog loves easy brea...	66
61442	much fun watching puppies asking chicken treat...	134
61443	little shih tzu absolutely loves cesar softies...	181

In [49]:

```
X_trainbow = pd.DataFrame()
```

In [50]:

```
X_trainbow['Cleaned Text'] = X_train1['Cleaned Text']
```

In [51]:

```
X_trainbow.head(3)
```

Out[51]:

	Cleaned Text
0	dogs loves chicken product china wont buying a...
1	dogs love saw pet store tag attached regarding...
2	infestation fruitflies literally everywhere fl...

In [52]:

```
X_testbow = pd.DataFrame()
```

In [53]:

```
X_testbow['Cleaned Text'] = X_test1['Cleaned Text']
```

In [54]:

```
X_testbow.head(3)
```

Out[54]:

	Cleaned Text
61441	used treat training reward dog loves easy brea...
61442	much fun watching puppies asking chicken treat...
61443	little shih tzu absolutely loves cesar softies...

61443	little shih tzu absolutely loves cesar softies...
	Cleaned Text

BAG OF WORDS WITH FEATURE ENGINEERING

In [55]:

```
X_trainbow.shape
```

Out[55]:

```
(61441, 1)
```

In [56]:

```
X_testbow.shape
```

Out[56]:

```
(26332, 1)
```

In [57]:

```
count_vect = CountVectorizer()
a1 = count_vect.fit_transform(X_trainbow['Cleaned Text'].values)
b1 = count_vect.transform(X_testbow['Cleaned Text'])
```

In [58]:

```
print("the type of count vectorizer :",type(a1))
print("the shape of out text BOW vectorizer : ",a1.get_shape())
print("the number of unique words :", a1.get_shape()[1])
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer :  (61441, 46008)
the number of unique words : 46008
```

ADDING LENGTH OF REVIEWS AS ONE FEATURE

In [59]:

```
a1 = preprocessing.normalize(a1)
```

In [60]:

```
from scipy import sparse
```

In [61]:

```
from scipy.sparse import csr_matrix
```

In [62]:

```
a2 = sparse.csr_matrix(X_train1['Length'].values)
```

In [63]:

```
a2 = preprocessing.normalize(a2)
```

In [64]:

```
a1
```

Out[64]:

```
<61441x46008 sparse matrix of type '<class 'numpy.float64'>'
 with 2002037 stored elements in Compressed Sparse Row format>
```

In [65]:

```
a2.T
```

Out[65]:

```
<61441x1 sparse matrix of type '<class 'numpy.float64'>'
 with 61271 stored elements in Compressed Sparse Column format>
```

In [66]:

```
a3 = sparse.hstack([a1, a2.T])
```

In [67]:

```
a3.shape
```

Out[67]:

```
(61441, 46009)
```

In [68]:

```
b1 = preprocessing.normalize(b1)
```

In [69]:

```
b2 = sparse.csr_matrix(X_test1['Length'].values)
```

In [70]:

```
b2 = preprocessing.normalize(b2)
```

In [71]:

```
b1
```

Out[71]:

```
<26332x46008 sparse matrix of type '<class 'numpy.float64'>'
 with 888781 stored elements in Compressed Sparse Row format>
```

In [72]:

```
b2.T
```

Out[72]:

```
<26332x1 sparse matrix of type '<class 'numpy.float64'>'
 with 26286 stored elements in Compressed Sparse Column format>
```

In [73]:

```
b3 = sparse.hstack([b1, b2.T])
```

In [74]:

```
a3.shape
```

Out[74]:

```
(61441, 46009)
```

In [75]:

```
b3.shape
```

Out[75]:

```
(26332, 46009)
```

In [76]:

```
y_test1.shape
```

Out[76]:

```
(26332,)
```

In [77]:

```
y_train1.shape
```

Out[77]:

```
(61441,)
```

Decision Tree for BOW with Feature Engineering

Logistic Regression for BOW with Feature Engineering

In [78]:

```
from sklearn.model_selection import train_test_split
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression
```

In [79]:

```
from sklearn.model_selection  import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection  import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

In [76]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model_bow = GridSearchCV(DecisionTreeClassifier(max_features="log2",class_weight = 'balanced'), tr
ee_para, scoring = 'roc_auc', cv=5, return_train_score= True)
model_bow.fit(a3, y_train1)
print(model_bow.best_estimator_)
print(model_bow.score(b3, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=1000, max_features='log2', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
```

```
            splitter='best')
0.6838225922011263
```

In [77]:

```python
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model_bow1 = GridSearchCV(DecisionTreeClassifier(class_weight = 'balanced'), tree_para, scoring =
'roc_auc', cv=5, return_train_score= True)
model_bow1.fit(a3, y_train1)
print(model_bow1.best_estimator_)
print(model_bow1.score(b3, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=50, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.8006141680733525
```

Observations for Decision Tree Classifier (BOW)

1) When all features are taken in to consideration, we observe that the AUC is more. However it takes lot of time to compute

2) When log(n) features were takeen the score is lower. But the computation time is faster.

3) There is almost 17% increase in AUC value when i chose all the features.

In [124]:

```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,recall_score
```

Running the Model with Optimal max_depth and splits

In [5]:

```python
from sklearn.metrics import roc_auc_score
```

In [92]:

```python
alph_depth = [1,1,1,1,5,5,5,5,10,10,10,10,50,50,50,50,100,100,100,100,500,500,500,500,1000,1000,1000
,1000]
alph_split = [5,10,100,500]*7
```

In [83]:

```python
model_bow1.cv_results_
```

Out[83]:

```
{'mean_fit_time': array([  1.01787515,   1.11951375,   1.02847285,   1.038726  ,
          1.73348079,   1.73733039,   1.6189199 ,   1.56021628,
          3.12114091,   3.06207037,   2.75605822,   2.57234359,
         14.86219864,  14.33729925,  11.15002289,   8.21201043,
         48.90694871,  46.04256434,  38.4334065 ,  27.82593322,
         71.29822412,  69.97879577,  60.19951959,  46.83349395,
         57.44790826,  56.36955051,  58.01324306, 147.42662067]),
 'std_fit_time': array([1.86974580e-02, 4.46724717e-02, 5.16100208e-02, 3.69513760e-02,
        6.27996366e-02, 6.13830474e-02, 3.14645684e-02, 2.08351998e-02,
        7.28540668e-02, 6.73345018e-02, 4.12170461e-02, 4.99461708e-02,
        2.25300235e-01, 2.72376790e-01, 4.07473522e-01, 1.74778685e-01,
        1.79941406e+00, 2.36646278e+00, 9.96946429e-01, 5.57440178e-01,
        1.89855306e+00, 2.28714628e+00, 2.35629371e+00, 3.63952011e+00,
        1.59679876e+01, 1.72424733e+01, 1.45248307e+00, 2.54985619e+02]),
 'mean_score_time': array([0.01042366, 0.01170645, 0.01470037, 0.00799179, 0.00931377,
        0.00638962, 0.00937257, 0.01249471, 0.01249595, 0.01562071,
```

```
       0.0124959 , 0.0124969 , 0.00937185, 0.01249909, 0.01562033,
       0.0236743 , 0.03074226, 0.02981906, 0.03147559, 0.02962437,
       0.03113232, 0.03468828, 0.03214574, 0.03203936, 0.01840134,
       0.02863951, 0.02811975, 0.01541147]),
 'std_score_time': array([5.71254499e-03, 2.09455166e-03, 1.84986748e-03, 4.04703886e-03,
       5.10399277e-03, 5.23008766e-03, 7.65266991e-03, 6.24735441e-03,
       6.24797379e-03, 3.82362751e-06, 6.24795017e-03, 6.24845032e-03,
       7.65208625e-03, 6.24954711e-03, 1.17383324e-06, 1.61118274e-02,
       1.06633146e-03, 2.85031865e-03, 4.67053961e-04, 3.24244659e-03,
       1.35684816e-04, 6.11959562e-03, 1.88127170e-03, 1.03326088e-03,
       1.13302196e-02, 6.53426862e-03, 6.24833125e-03, 1.87865701e-03]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50,
                    100, 100, 100, 100, 500, 500, 500, 500, 1000, 1000,
                    1000, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500,
5,
                    10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10,
                    100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 500},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 500},
  {'max_depth': 1000, 'min_samples_split': 5},
  {'max_depth': 1000, 'min_samples_split': 10},
  {'max_depth': 1000, 'min_samples_split': 100},
  {'max_depth': 1000, 'min_samples_split': 500}],
 'split0_test_score': array([0.63426819, 0.63426819, 0.63426819, 0.63426819, 0.74384757,
       0.74394404, 0.74417213, 0.74424257, 0.78268758, 0.78422195,
       0.7885172 , 0.78841443, 0.72764493, 0.728061  , 0.76989206,
       0.80918118, 0.72728482, 0.73135216, 0.76858947, 0.79356155,
       0.70314064, 0.71283961, 0.74008774, 0.77060906, 0.69754528,
       0.7111527 , 0.74432909, 0.76769488]),
 'split1_test_score': array([0.61859718, 0.61859718, 0.61859718, 0.61859718, 0.72082364,
       0.72082364, 0.72055826, 0.7226477 , 0.76679079, 0.76791141,
       0.77490393, 0.77802516, 0.71293251, 0.71648403, 0.75726628,
       0.79498613, 0.70418527, 0.71375358, 0.75081507, 0.78721446,
       0.69245798, 0.69908263, 0.72390494, 0.7591948 , 0.69811922,
       0.68785331, 0.7349716 , 0.76715856]),
 'split2_test_score': array([0.6317278 , 0.6317278 , 0.6317278 , 0.6317278 , 0.72973054,
       0.72903261, 0.72992813, 0.73035505, 0.76683781, 0.7653872 ,
       0.77182906, 0.77710689, 0.71088041, 0.71745596, 0.7546174 ,
       0.79506343, 0.71157147, 0.72002097, 0.75773672, 0.78650421,
       0.68790433, 0.70207834, 0.72748574, 0.75010472, 0.69183582,
       0.70060441, 0.72522778, 0.7550027 ]),
 'split3_test_score': array([0.62775583, 0.62775583, 0.62775583, 0.62775583, 0.73158621,
```

```
        0.73158621, 0.73091551, 0.73311295, 0.77072356, 0.7686117 ,
        0.77308506, 0.77742287, 0.70356951, 0.71408724, 0.76053364,
        0.78840921, 0.70195577, 0.72259432, 0.75889241, 0.77685636,
        0.69188949, 0.6999582 , 0.73177869, 0.7443097 , 0.69429406,
        0.70095908, 0.73243802, 0.74541238]),
 'split4_test_score': array([0.61944271, 0.61944271, 0.61944271, 0.61944271, 0.71668772,
        0.71668772, 0.71639849, 0.71661225, 0.75395227, 0.75588368,
        0.76135514, 0.76694846, 0.70349364, 0.71368342, 0.76124693,
        0.78548973, 0.7099664 , 0.70952471, 0.75558404, 0.78152807,
        0.68421035, 0.68766942, 0.72954266, 0.75482678, 0.68244623,
        0.68970394, 0.72649491, 0.74946776]),
 'mean_test_score': array([0.62635852, 0.62635852, 0.62635852, 0.62635852, 0.72853542,
        0.72841512, 0.72839481, 0.7293944 , 0.76819878, 0.76840359,
        0.77393852, 0.77758391, 0.71170473, 0.71795459, 0.76071125,
        0.79462643, 0.71099308, 0.71944937, 0.75832361, 0.78513332,
        0.69192081, 0.70032606, 0.73055995, 0.75580942, 0.69284841,
        0.69805486, 0.73269249, 0.75694787]),
 'std_test_score': array([0.00634691, 0.00634691, 0.00634691, 0.00634691, 0.00943442,
        0.00945253, 0.00962245, 0.00942465, 0.00919881, 0.00912529,
        0.00868318, 0.00679496, 0.00883045, 0.00524888, 0.00516947,
        0.00817675, 0.0088864 , 0.00752111, 0.00583008, 0.00563469,
        0.00635234, 0.00801905, 0.00542306, 0.00890222, 0.00567724,
        0.00848891, 0.00685289, 0.00908357]),
 'rank_test_score': array([25, 25, 25, 25, 14, 15, 16, 13,  6,  5,  4,  3, 19, 18,  7,  1, 20,
        17,  8,  2, 24, 21, 12, 10, 23, 22, 11,  9]),
 'split0_train_score': array([0.62586411, 0.62586411, 0.62586411, 0.62586411, 0.73891562,
        0.73891562, 0.73834099, 0.73771804, 0.82232896, 0.82103495,
        0.81147832, 0.80657683, 0.97870282, 0.97138292, 0.94592221,
        0.91111256, 0.99203909, 0.98805652, 0.96697669, 0.92875767,
        0.99989033, 0.99881839, 0.98540013, 0.95032076, 0.9998731 ,
        0.99881768, 0.98474159, 0.95082477]),
 'split1_train_score': array([0.62894217, 0.62894217, 0.62894217, 0.62894217, 0.7415565 ,
        0.7415565 , 0.74094658, 0.73942864, 0.8251841 , 0.82460395,
        0.81616405, 0.80987708, 0.98171278, 0.9757535 , 0.94492072,
        0.91296666, 0.99238129, 0.98862151, 0.96454059, 0.92954101,
        0.99989911, 0.9989245 , 0.98235343, 0.94806029, 0.99988732,
        0.99893861, 0.98306177, 0.9471386 ]),
 'split2_train_score': array([0.6262038 , 0.6262038 , 0.6262038 , 0.6262038 , 0.73953683,
        0.73953683, 0.73891029, 0.7384078 , 0.82097841, 0.82011725,
        0.81241086, 0.80635333, 0.97906464, 0.9748786 , 0.94837458,
        0.90827226, 0.99226066, 0.98858116, 0.96634743, 0.92594379,
        0.99989406, 0.99909296, 0.98578378, 0.94869233, 0.9998875 ,
        0.99907947, 0.985568  , 0.94735276]),
 'split3_train_score': array([0.6271887 , 0.6271887 , 0.6271887 , 0.6271887 , 0.74097088,
        0.74097088, 0.74034326, 0.73868999, 0.82198563, 0.82122799,
        0.81318001, 0.80635119, 0.98085728, 0.97518217, 0.94291995,
        0.91313148, 0.99078408, 0.98706166, 0.96087093, 0.93015351,
        0.99988255, 0.99889105, 0.983271  , 0.9556959 , 0.99988817,
        0.99894714, 0.98160596, 0.95462947]),
 'split4_train_score': array([0.62873009, 0.62873009, 0.62873009, 0.62873009, 0.74121056,
        0.74111331, 0.74066663, 0.73984905, 0.82252697, 0.82144618,
        0.8129402 , 0.80786249, 0.97656906, 0.97070443, 0.94091739,
        0.91250043, 0.98886059, 0.98479107, 0.95965869, 0.93048146,
        0.99986283, 0.99902111, 0.98390258, 0.9569715 , 0.99987223,
        0.99902652, 0.98369365, 0.95753698]),
 'mean_train_score': array([0.62738578, 0.62738578, 0.62738578, 0.62738578, 0.74043808,
        0.74041863, 0.73984155, 0.7388187 , 0.82260081, 0.82168606,
        0.81323469, 0.80740418, 0.97938132, 0.97358033, 0.94461097,
        0.91159668, 0.99126514, 0.98742239, 0.96367886, 0.92897549,
        0.99988578, 0.9989496 , 0.98414218, 0.95194815, 0.99988166,
        0.99896188, 0.98373419, 0.95149651]),
 'std_train_score': array([1.26340120e-03, 1.26340120e-03, 1.26340120e-03, 1.26340120e-03,
        1.02582516e-03, 1.01182076e-03, 1.02688611e-03, 7.52396874e-04,
        1.39738021e-03, 1.52751913e-03, 1.57698349e-03, 1.35814924e-03,
        1.79352312e-03, 2.10111727e-03, 2.54880828e-03, 1.80750530e-03,
        1.33034907e-03, 1.43085377e-03, 2.92526273e-03, 1.62567634e-03,
        1.26807039e-05, 9.68922178e-05, 1.28788812e-03, 3.67814857e-03,
        7.35759657e-06, 8.89461972e-05, 1.36827216e-03, 4.07224726e-03])}
```

In [78]:

```
#train_auc= model.cv_results_['mean_train_score']
#cv_auc= model.cv_results_['mean_test_score']
```

In [84]:

```python
train_auc1= model_bow1.cv_results_['mean_train_score']
cv_auc1= model_bow1.cv_results_['mean_test_score']
```

In [85]:

```python
train_auc1
```

Out[85]:

```
array([0.62738578, 0.62738578, 0.62738578, 0.62738578, 0.74043808,
       0.74041863, 0.73984155, 0.7388187 , 0.82260081, 0.82168606,
       0.81323469, 0.80740418, 0.97938132, 0.97358033, 0.94461097,
       0.91159668, 0.99126514, 0.98742239, 0.96367886, 0.92897549,
       0.99988578, 0.9989496 , 0.98414218, 0.95194815, 0.99988166,
       0.99896188, 0.98373419, 0.95149651])
```

In [86]:

```python
cv_auc1
```

Out[86]:

```
array([0.62635852, 0.62635852, 0.62635852, 0.62635852, 0.72853542,
       0.72841512, 0.72839481, 0.7293944 , 0.76819878, 0.76840359,
       0.77393852, 0.77758391, 0.71170473, 0.71795459, 0.76071125,
       0.79462643, 0.71099308, 0.71944937, 0.75832361, 0.78513332,
       0.69192081, 0.70032606, 0.73055995, 0.75580942, 0.69284841,
       0.69805486, 0.73269249, 0.75694787])
```

In [82]:

```python
#train_auc
```

Out[82]:

```
array([0.5       , 0.5       , 0.76796783, 0.96421071, 0.99985648,
       0.99999725, 0.99999725])
```

In [83]:

```python
#cv_auc
```

Out[83]:

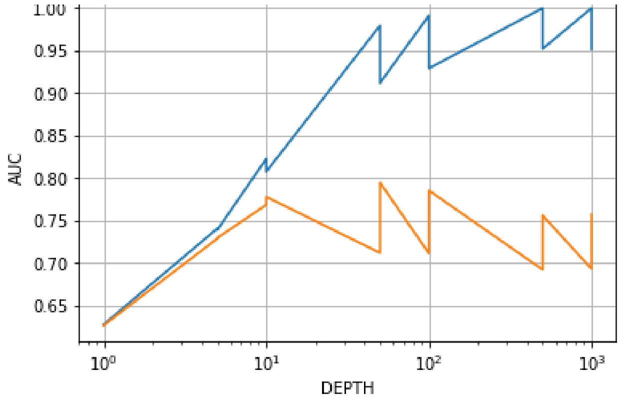
```
array([0.5       , 0.5       , 0.76602852, 0.94476225, 0.90027524,
       0.8802149 , 0.85237726])
```

In [87]:

```python
import math
from math import log
```

In [101]:

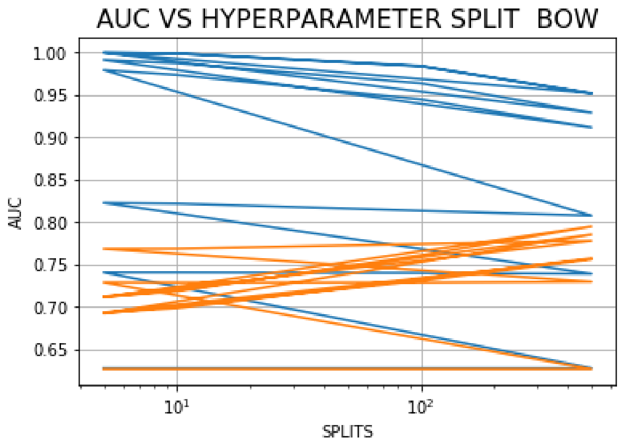
```python
# Firstly I am plotting depth vs AUC and then split vs AUC
plt.plot(alph_depth,train_auc1)
plt.plot(alph_depth,cv_auc1)
plt.xlabel('DEPTH',size=10)
plt.ylabel('AUC',size=10)
plt.title('AUC VS HYPERPARAMETER DEPTH  BOW',size=16)
plt.xscale('log')
plt.grid()
plt.show()
print("\n\n Depth Values :\n", alph_depth)
print("\n Train AUC for each alpha value is :\n ", np.round(train_auc1,5))
print("\n CV AUC for each alpha value is :\n ", np.round(cv_auc1,5))
```




```
 Depth Values :
 [1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50, 100, 100, 100, 100, 500, 500, 500, 500,
1000, 1000, 1000, 1000]

 Train AUC for each alpha value is :
  [0.62739 0.62739 0.62739 0.62739 0.74044 0.74042 0.73984 0.73882 0.8226
 0.82169 0.81323 0.8074  0.97938 0.97358 0.94461 0.9116  0.99127 0.98742
 0.96368 0.92898 0.99989 0.99895 0.98414 0.95195 0.99988 0.99896 0.98373
 0.9515 ]

 CV AUC for each alpha value is :
  [0.62636 0.62636 0.62636 0.62636 0.72854 0.72842 0.72839 0.72939 0.7682
 0.7684  0.77394 0.77758 0.7117  0.71795 0.76071 0.79463 0.71099 0.71945
 0.75832 0.78513 0.69192 0.70033 0.73056 0.75581 0.69285 0.69805 0.73269
 0.75695]
```

In [100]:

```
plt.plot(alph_split,train_auc1)
plt.plot(alph_split,cv_auc1)
plt.xlabel('SPLITS',size=10)
plt.ylabel('AUC',size=10)
plt.title('AUC VS HYPERPARAMETER SPLIT  BOW',size=16)
plt.xscale('log')
plt.grid()
plt.show()
print("\n\n Split Values :\n", alph_split)
print("\n Train AUC for each alpha value is :\n ", np.round(train_auc1,5))
print("\n CV AUC for each alpha value is :\n ", np.round(cv_auc1,5))
```




```
 Split Values :
 [5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100,
500, 5, 10, 100, 500]

 Train AUC for each alpha value is :
  [0.62739 0.62739 0.62739 0.62739 0.74044 0.74042 0.73984 0.73882 0.8226
 0.82169 0.81323 0.8074  0.97938 0.97358 0.94461 0.9116  0.99127 0.98742
 0.96368 0.92898 0.99989 0.99895 0.98414 0.95195 0.99988 0.99896 0.98373
 0.9515 ]
```

```
CV AUC for each alpha value is :
 [0.62636 0.62636 0.62636 0.62636 0.72854 0.72842 0.72839 0.72939 0.7682
0.7684  0.77394 0.77758 0.7117  0.71795 0.76071 0.79463 0.71099 0.71945
0.75832 0.78513 0.69192 0.70033 0.73056 0.75581 0.69285 0.69805 0.73269
0.75695]
```

In [102]:

```
max(cv_auc1)
```

Out[102]:

```
0.7946264342396073
```

Observations

1) We have found that the hyperparmeters max_depth should be 50 and min_splits should be 500 for having maximum AUC for CV

In [87]:

```
optimalalpha2_bow = 1
auc_bow = max(cv_auc)
auc_bow1 = max(cv_auc1)
```

In [88]:

```
from IPython.display import HTML, display
import tabulate
table = [["Vectorizer","Model","Regularisation","Hyperparameter","AUC"],
         ["BOW", "Logistic Regression ", "L1 " ,optimalalpha2_bow, np.round(auc_bow,4) ],
         ["BOW", "Logistic Regression ", "L2 " ,optimalalpha2_bow, np.round(auc_bow1,4)]]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

Vectorizer	Model	Regularisation	Hyperparameter	AUC
BOW	Logistic Regression	L1	1	0.9448
BOW	Logistic Regression	L2	1	0.9439

In [90]:

```
# 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 h
eatmaps.
```

Training the model with the best hyper parameter

In [80]:

```
om_bow = DecisionTreeClassifier(class_weight = 'balanced', max_depth = 50 , min_samples_split = 500
)
```

In [81]:

```
#om_bow = MultinomialNB(alpha = optimalalpha2_bow)
# fitting the model and predicting the responses
om_bow.fit(a3, y_train1)
ompredictions_bow  = om_bow.predict(b3)
```

In [82]:

```
len(ompredictions_bow)
```

Out[82]:

26332

In [83]:

```
len(y_test1)
```

Out[83]:

26332

In [84]:

```
probs = om_bow.predict_proba(b3)
```

In [85]:

```
probs1 = om_bow.predict_proba(a3)
```

In [86]:

```
len(probs1)
```

Out[86]:

61441

In [87]:

```
len(probs)
```

Out[87]:

26332

In [88]:

```
probs = probs[:, 1]
```

In [89]:

```
probs1 = probs1[:, -1]
```

Graphviz

In [122]:

```
from sklearn.datasets import load_iris
from sklearn import tree
```

In [91]:

```
tree.plot_tree(om_bow.fit(a3, y_train1))
```

Out[91]:

```
[Text(249.493,222.24,'X[27139] <= 0.079\nentropy = 0.5\nsamples = 61441\nvalue = [30720.5, 
30720.5]'),
 Text(185.169,217.839,'X[17569] <= 0.059\nentropy = 0.446\nsamples = 29196\nvalue = [8019.855, 158
24.13]'),
 Text(173.329,213.438,'X[3744] <= 0.043\nentropy = 0.476\nsamples = 21797\nvalue = [7386.541, 1155
3.11]'),
 Text(161.517,209.037,'X[23595] <= 0.148\nentropy = 0.487\nsamples = 18970\nvalue = [7147.855, 992
0.631]'),
```

```
 Text(149.76,204.636,'X[17183] <= 0.188\nentropy = 0.495\nsamples = 16033\nvalue = [6762.775, 8250
.188]'),
 Text(138.114,200.236,'X[23609] <= 0.099\nentropy = 0.499\nsamples = 13480\nvalue = [6336.322, 681
5.243]'),
 Text(126.691,195.835,'X[10560] <= 0.035\nentropy = 0.5\nsamples = 12416\nvalue = [6256.76,
6198.911]'),
 Text(115.714,191.434,'X[11440] <= 0.06\nentropy = 0.499\nsamples = 11357\nvalue = [6170.833,
5586.732]'),
 Text(113.34,187.033,'X[13900] <= 0.092\nentropy = 0.5\nsamples = 11174\nvalue = [5750.745,
5556.479]'),
 Text(103.252,182.633,'X[14617] <= 0.05\nentropy = 0.499\nsamples = 10594\nvalue = [5715.738,
5218.95]'),
 Text(94.9444,178.232,'X[29622] <= 0.114\nentropy = 0.497\nsamples = 9831\nvalue = [5604.351, 4787
.102]'),
 Text(92.5708,173.831,'X[18895] <= 0.07\nentropy = 0.495\nsamples = 9383\nvalue = [5569.344,
4527.875]'),
 Text(90.1972,169.43,'X[26852] <= 0.137\nentropy = 0.492\nsamples = 9030\nvalue = [5550.249, 4322.
035]'),
 Text(87.8236,165.029,'X[12625] <= 0.008\nentropy = 0.489\nsamples = 8634\nvalue = [5512.059, 4094
.248]'),
 Text(85.45,160.629,'X[18272] <= 0.175\nentropy = 0.486\nsamples = 8279\nvalue = [5477.052,
3890.188]'),
 Text(83.0764,156.228,'X[45177] <= 0.11\nentropy = 0.483\nsamples = 8037\nvalue = [5467.504,
3748.414]'),
 Text(80.7028,151.827,'X[2903] <= 0.091\nentropy = 0.478\nsamples = 7701\nvalue = [5426.132, 3556.
811]'),
 Text(78.3292,147.426,'X[44564] <= 0.112\nentropy = 0.483\nsamples = 7535\nvalue = [5085.606, 3521
.812]'),
 Text(73.5819,143.025,'X[23597] <= 0.152\nentropy = 0.478\nsamples = 6938\nvalue = [4907.387, 3200
.892]'),
 Text(71.2083,138.625,'X[14943] <= 0.167\nentropy = 0.474\nsamples = 6731\nvalue = [4888.293, 3081
.66]'),
 Text(68.8347,134.224,'X[25745] <= 0.083\nentropy = 0.469\nsamples = 6398\nvalue = [4811.913, 2898
.362]'),
 Text(66.4611,129.823,'X[2750] <= 0.145\nentropy = 0.475\nsamples = 6244\nvalue = [4522.307, 2860.
99]'),
 Text(64.0875,125.422,'X[10449] <= 0.012\nentropy = 0.479\nsamples = 6120\nvalue = [4280.438, 2832
.517]'),
 Text(61.7139,121.022,'X[32887] <= 0.069\nentropy = 0.476\nsamples = 5944\nvalue = [4258.161, 2732
.267]'),
 Text(59.3403,116.621,'X[45296] <= 0.038\nentropy = 0.482\nsamples = 5749\nvalue = [3946.278, 2674
.726]'),
 Text(56.9667,112.22,'X[40533] <= 0.025\nentropy = 0.484\nsamples = 5704\nvalue = [3815.796, 2672.
354]'),
 Text(54.5931,107.819,'X[30415] <= 0.127\nentropy = 0.487\nsamples = 5652\nvalue = [3685.314, 2665
.829]'),
 Text(52.2194,103.418,'X[1285] <= 0.061\nentropy = 0.486\nsamples = 5563\nvalue = [3685.314, 2613.
034]'),
 Text(49.8458,99.0176,'X[40199] <= 0.233\nentropy = 0.484\nsamples = 5468\nvalue = [3682.132, 2557
.273]'),
 Text(47.4722,94.6169,'X[41942] <= 0.039\nentropy = 0.482\nsamples = 5357\nvalue = [3672.584, 2493
.208]'),
 Text(45.0986,90.2161,'X[25886] <= 0.066\nentropy = 0.479\nsamples = 5186\nvalue = [3640.76, 2397.
703]'),
 Text(42.725,85.8153,'X[2768] <= 0.057\nentropy = 0.476\nsamples = 5069\nvalue = [3628.03,
2330.672]'),
 Text(40.3514,81.4145,'X[402] <= 0.019\nentropy = 0.479\nsamples = 5033\nvalue = [3529.373,
2327.706]'),
 Text(37.9778,77.0137,'X[2762] <= 0.164\nentropy = 0.475\nsamples = 4817\nvalue = [3475.27, 2209.6
6]'),
 Text(35.6042,72.6129,'X[40640] <= 0.013\nentropy = 0.474\nsamples = 4760\nvalue = [3475.27, 2175.
848]'),
 Text(33.2306,68.2122,'X[21842] <= 0.183\nentropy = 0.471\nsamples = 4683\nvalue = [3468.906, 2131
.358]'),
 Text(30.8569,63.8114,'X[1233] <= 0.194\nentropy = 0.469\nsamples = 4602\nvalue = [3459.358, 2085.
088]'),
 Text(28.4833,59.4106,'X[16898] <= 0.137\nentropy = 0.467\nsamples = 4527\nvalue = [3452.993, 2041
.785]'),
 Text(26.1097,55.0098,'X[46008] <= 0.001\nentropy = 0.465\nsamples = 4469\nvalue = [3449.811, 2007
.973]'),
 Text(23.7361,50.609,'entropy = 0.495\nsamples = 331\nvalue = [140.029, 170.248]'),
 Text(28.4833,50.609,'X[15200] <= 0.065\nentropy = 0.459\nsamples = 4138\nvalue = [3309.781, 1837.
725]'),
 Text(26.1097,46.2082,'X[45586] <= 0.146\nentropy = 0.457\nsamples = 4095\nvalue = [3309.781, 1812
.218]'),
 Text(23.7361,41.8075,'X[35114] <= 0.032\nentropy = 0.454\nsamples = 3999\nvalue = [3290.687, 1758
.83]'),
```

```
 Text(21.3625,37.4067,'X[45829] <= 0.166\nentropy = 0.452\nsamples = 3961\nvalue = [3290.687, 1736
.288]'),
 Text(18.9889,33.0059,'X[18726] <= 0.018\nentropy = 0.45\nsamples = 3923\nvalue = [3290.687,
1713.747]'),
 Text(16.6153,28.6051,'X[17183] <= 0.125\nentropy = 0.448\nsamples = 3867\nvalue = [3284.322, 1681
.714]'),
 Text(14.2417,24.2043,'X[32979] <= 0.206\nentropy = 0.441\nsamples = 3578\nvalue = [3147.475, 1535
.788]'),
 Text(11.8681,19.8035,'X[14481] <= 0.153\nentropy = 0.439\nsamples = 3545\nvalue = [3147.475, 1516
.212]'),
 Text(9.49444,15.4027,'X[32351] <= 0.136\nentropy = 0.437\nsamples = 3513\nvalue = [3147.475, 1497
.23]'),
 Text(7.12083,11.002,'X[50] <= 0.114\nentropy = 0.435\nsamples = 3483\nvalue = [3147.475,
1479.434]'),
 Text(4.74722,6.60118,'X[13285] <= 0.114\nentropy = 0.432\nsamples = 3436\nvalue = [3141.11, 1452.
74]'),
 Text(2.37361,2.20039,'entropy = 0.427\nsamples = 3307\nvalue = [3090.19, 1385.709]'),
 Text(7.12083,2.20039,'entropy = 0.491\nsamples = 129\nvalue = [50.92, 67.031]'),
 Text(9.49444,6.60118,'entropy = 0.311\nsamples = 47\nvalue = [6.365, 26.694]'),
 Text(11.8681,11.002,'entropy = -0.0\nsamples = 30\nvalue = [0.0, 17.796]'),
 Text(14.2417,15.4027,'entropy = -0.0\nsamples = 32\nvalue = [0.0, 18.982]'),
 Text(16.6153,19.8035,'entropy = -0.0\nsamples = 33\nvalue = [0.0, 19.576]'),
 Text(18.9889,24.2043,'entropy = 0.499\nsamples = 289\nvalue = [136.847, 145.927]'),
 Text(21.3625,28.6051,'entropy = 0.277\nsamples = 56\nvalue = [6.365, 32.033]'),
 Text(23.7361,33.0059,'entropy = -0.0\nsamples = 38\nvalue = [0.0, 22.541]'),
 Text(26.1097,37.4067,'entropy = -0.0\nsamples = 38\nvalue = [0.0, 22.541]'),
 Text(28.4833,41.8075,'entropy = 0.388\nsamples = 96\nvalue = [19.095, 53.388]'),
 Text(30.8569,46.2082,'entropy = -0.0\nsamples = 43\nvalue = [0.0, 25.507]'),
 Text(30.8569,55.0098,'entropy = 0.157\nsamples = 58\nvalue = [3.182, 33.812]'),
 Text(33.2306,59.4106,'entropy = 0.223\nsamples = 75\nvalue = [6.365, 43.303]'),
 Text(35.6042,63.8114,'entropy = 0.284\nsamples = 81\nvalue = [9.547, 46.269]'),
 Text(37.9778,68.2122,'entropy = 0.219\nsamples = 77\nvalue = [6.365, 44.49]'),
 Text(40.3514,72.6129,'entropy = -0.0\nsamples = 57\nvalue = [0.0, 33.812]'),
 Text(42.725,77.0137,'entropy = 0.431\nsamples = 216\nvalue = [54.102, 118.046]'),
 Text(45.0986,81.4145,'entropy = 0.057\nsamples = 36\nvalue = [98.657, 2.966]'),
 Text(47.4722,85.8153,'entropy = 0.268\nsamples = 117\nvalue = [12.73, 67.031]'),
 Text(49.8458,90.2161,'entropy = 0.375\nsamples = 171\nvalue = [31.825, 95.505]'),
 Text(52.2194,94.6169,'entropy = 0.226\nsamples = 111\nvalue = [9.547, 64.065]'),
 Text(54.5931,99.0176,'entropy = 0.102\nsamples = 95\nvalue = [3.182, 55.761]'),
 Text(56.9667,103.418,'entropy = -0.0\nsamples = 89\nvalue = [0.0, 52.795]'),
 Text(59.3403,107.819,'entropy = 0.091\nsamples = 52\nvalue = [130.482, 6.525]'),
 Text(61.7139,112.22,'entropy = 0.035\nsamples = 45\nvalue = [130.482, 2.373]'),
 Text(64.0875,116.621,'entropy = 0.263\nsamples = 195\nvalue = [311.883, 57.54]'),
 Text(66.4611,121.022,'entropy = 0.298\nsamples = 176\nvalue = [22.277, 100.25]'),
 Text(68.8347,125.422,'entropy = 0.188\nsamples = 124\nvalue = [241.869, 28.473]'),
 Text(71.2083,129.823,'entropy = 0.202\nsamples = 154\nvalue = [289.606, 37.371]'),
 Text(73.5819,134.224,'entropy = 0.415\nsamples = 333\nvalue = [76.38, 183.298]'),
 Text(75.9556,138.625,'entropy = 0.238\nsamples = 207\nvalue = [19.095, 119.233]'),
 Text(83.0764,143.025,'X[25178] <= 0.148\nentropy = 0.459\nsamples = 597\nvalue = [178.219,
320.92]'),
 Text(80.7028,138.625,'X[34845] <= 0.057\nentropy = 0.438\nsamples = 584\nvalue = [152.759,
317.954]'),
 Text(78.3292,134.224,'X[27786] <= 0.097\nentropy = 0.414\nsamples = 574\nvalue = [130.482,
316.174]'),
 Text(75.9556,129.823,'X[38936] <= 0.042\nentropy = 0.393\nsamples = 562\nvalue = [114.569,
312.022]'),
 Text(73.5819,125.422,'X[23555] <= 0.06\nentropy = 0.377\nsamples = 559\nvalue = [105.022,
312.022]'),
 Text(71.2083,121.022,'X[30731] <= 0.155\nentropy = 0.359\nsamples = 556\nvalue = [95.474,
312.022]'),
 Text(68.8347,116.621,'X[31595] <= 0.221\nentropy = 0.34\nsamples = 550\nvalue = [85.927,
310.242]'),
 Text(66.4611,112.22,'X[26081] <= 0.132\nentropy = 0.318\nsamples = 544\nvalue = [76.38,
308.463]'),
 Text(64.0875,107.819,'entropy = 0.26\nsamples = 490\nvalue = [50.92, 281.176]'),
 Text(68.8347,107.819,'entropy = 0.499\nsamples = 54\nvalue = [25.46, 27.287]'),
 Text(71.2083,112.22,'entropy = 0.265\nsamples = 6\nvalue = [9.547, 1.78]'),
 Text(73.5819,116.621,'entropy = 0.265\nsamples = 6\nvalue = [9.547, 1.78]'),
 Text(75.9556,121.022,'entropy = -0.0\nsamples = 3\nvalue = [9.547, 0.0]'),
 Text(78.3292,125.422,'entropy = -0.0\nsamples = 3\nvalue = [9.547, 0.0]'),
 Text(80.7028,129.823,'entropy = 0.328\nsamples = 12\nvalue = [15.912, 4.152]'),
 Text(83.0764,134.224,'entropy = 0.137\nsamples = 10\nvalue = [22.277, 1.78]'),
 Text(85.45,138.625,'entropy = 0.187\nsamples = 13\nvalue = [25.46, 2.966]'),
 Text(83.0764,147.426,'entropy = 0.169\nsamples = 166\nvalue = [340.526, 34.999]'),
 Text(85.45,151.827,'entropy = 0.292\nsamples = 336\nvalue = [41.372, 191.603]'),
 Text(87.8236,156.228,'entropy = 0.118\nsamples = 242\nvalue = [9.547, 141.774]'),
 Text(90.1972,160.629,'entropy = 0.25\nsamples = 355\nvalue = [35.007, 204.06]'),
```

```
 Text(92.5708,165.029,'entropy = 0.246\nsamples = 396\nvalue = [38.19, 227.788]'),
 Text(94.9444,169.43,'entropy = 0.155\nsamples = 353\nvalue = [19.095, 205.839]'),
 Text(97.3181,173.831,'entropy = 0.21\nsamples = 448\nvalue = [35.007, 259.227]'),
 Text(111.56,178.232,'X[22747] <= 0.104\nentropy = 0.326\nsamples = 763\nvalue = [111.387,
431.848]'),
 Text(109.186,173.831,'X[27186] <= 0.105\nentropy = 0.278\nsamples = 749\nvalue = [85.927,
428.288]'),
 Text(106.813,169.43,'X[43277] <= 0.221\nentropy = 0.227\nsamples = 736\nvalue = [63.65,
424.729]'),
 Text(104.439,165.029,'X[11442] <= 0.029\nentropy = 0.184\nsamples = 721\nvalue = [47.737,
418.797]'),
 Text(102.065,160.629,'X[19688] <= 0.099\nentropy = 0.153\nsamples = 717\nvalue = [38.19,
418.204]'),
 Text(99.6917,156.228,'X[25178] <= 0.117\nentropy = 0.131\nsamples = 715\nvalue = [31.825,
418.204]'),
 Text(97.3181,151.827,'X[30412] <= 0.111\nentropy = 0.097\nsamples = 703\nvalue = [22.277,
412.865]'),
 Text(94.9444,147.426,'X[13932] <= 0.099\nentropy = 0.072\nsamples = 698\nvalue = [15.912,
411.086]'),
 Text(92.5708,143.025,'X[39042] <= 0.319\nentropy = 0.045\nsamples = 690\nvalue = [9.547,
407.527]'),
 Text(90.1972,138.625,'X[33420] <= 0.034\nentropy = 0.03\nsamples = 689\nvalue = [6.365,
407.527]'),
 Text(87.8236,134.224,'X[39127] <= 0.252\nentropy = 0.015\nsamples = 688\nvalue = [3.182,
407.527]'),
 Text(85.45,129.823,'entropy = 0.0\nsamples = 687\nvalue = [0.0, 407.527]'),
 Text(90.1972,129.823,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(92.5708,134.224,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(94.9444,138.625,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(97.3181,143.025,'entropy = 0.46\nsamples = 8\nvalue = [6.365, 3.559]'),
 Text(99.6917,147.426,'entropy = 0.342\nsamples = 5\nvalue = [6.365, 1.78]'),
 Text(102.065,151.827,'entropy = 0.46\nsamples = 12\nvalue = [9.547, 5.339]'),
 Text(104.439,156.228,'entropy = -0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(106.813,160.629,'entropy = 0.11\nsamples = 4\nvalue = [9.547, 0.593]'),
 Text(109.186,165.029,'entropy = 0.396\nsamples = 15\nvalue = [15.912, 5.932]'),
 Text(111.56,169.43,'entropy = 0.238\nsamples = 13\nvalue = [22.277, 3.559]'),
 Text(113.933,173.831,'entropy = 0.215\nsamples = 14\nvalue = [25.46, 3.559]'),
 Text(123.428,182.633,'X[25745] <= 0.104\nentropy = 0.17\nsamples = 580\nvalue = [35.007,
337.529]'),
 Text(121.054,178.232,'X[16525] <= 0.152\nentropy = 0.132\nsamples = 569\nvalue = [25.46,
332.784]'),
 Text(118.681,173.831,'X[2917] <= 0.167\nentropy = 0.103\nsamples = 565\nvalue = [19.095,
331.597]'),
 Text(116.307,169.43,'X[37909] <= 0.081\nentropy = 0.056\nsamples = 548\nvalue = [9.547,
323.293]'),
 Text(113.933,165.029,'X[33994] <= 0.075\nentropy = 0.038\nsamples = 547\nvalue = [6.365,
323.293]'),
 Text(111.56,160.629,'X[3801] <= 0.219\nentropy = 0.019\nsamples = 546\nvalue = [3.182,
323.293]'),
 Text(109.186,156.228,'entropy = 0.0\nsamples = 545\nvalue = [0.0, 323.293]'),
 Text(113.933,156.228,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(116.307,160.629,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(118.681,165.029,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(121.054,169.43,'entropy = 0.498\nsamples = 17\nvalue = [9.547, 8.305]'),
 Text(123.428,173.831,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(125.801,178.232,'entropy = 0.444\nsamples = 11\nvalue = [9.547, 4.746]'),
 Text(118.087,187.033,'entropy = 0.125\nsamples = 183\nvalue = [420.088, 30.253]'),
 Text(137.669,191.434,'X[28047] <= 0.118\nentropy = 0.216\nsamples = 1059\nvalue = [85.927, 612.18
]'),
 Text(135.296,187.033,'X[4574] <= 0.051\nentropy = 0.193\nsamples = 1043\nvalue = [73.197, 605.061
]'),
 Text(132.922,182.633,'X[41100] <= 0.179\nentropy = 0.173\nsamples = 1035\nvalue = [63.65,
602.095]'),
 Text(130.549,178.232,'X[29115] <= 0.043\nentropy = 0.159\nsamples = 1033\nvalue = [57.285, 602.09
5]'),
 Text(128.175,173.831,'X[22753] <= 0.133\nentropy = 0.144\nsamples = 1031\nvalue = [50.92,
602.095]'),
 Text(125.801,169.43,'X[1127] <= 0.185\nentropy = 0.128\nsamples = 1029\nvalue = [44.555,
602.095]'),
 Text(123.428,165.029,'X[43525] <= 0.06\nentropy = 0.112\nsamples = 1024\nvalue = [38.19,
600.316]'),
 Text(121.054,160.629,'X[4280] <= 0.037\nentropy = 0.096\nsamples = 1018\nvalue = [31.825, 597.943
]'),
 Text(118.681,156.228,'X[5827] <= 0.109\nentropy = 0.079\nsamples = 1011\nvalue = [25.46,
594.977]'),
 Text(116.307,151.827,'X[31038] <= 0.112\nentropy = 0.061\nsamples = 1003\nvalue = [19.095, 591.41
8]'),
```

```
5]'),
 Text(113.933,147.426,'X[20276] <= 0.159\nentropy = 0.051\nsamples = 1002\nvalue = [15.912, 591.41
8]'),
 Text(111.56,143.025,'X[38898] <= 0.124\nentropy = 0.041\nsamples = 1001\nvalue = [12.73,
591.418]'),
 Text(109.186,138.625,'X[16611] <= 0.051\nentropy = 0.031\nsamples = 1000\nvalue = [9.547,
591.418]'),
 Text(106.813,134.224,'X[26725] <= 0.094\nentropy = 0.021\nsamples = 999\nvalue = [6.365,
591.418]'),
 Text(104.439,129.823,'X[25745] <= 0.263\nentropy = 0.011\nsamples = 998\nvalue = [3.182,
591.418]'),
 Text(102.065,125.422,'entropy = 0.0\nsamples = 997\nvalue = [0.0, 591.418]'),
 Text(106.813,125.422,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(109.186,129.823,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(111.56,134.224,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(113.933,138.625,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(116.307,143.025,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(118.681,147.426,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(121.054,151.827,'entropy = 0.46\nsamples = 8\nvalue = [6.365, 3.559]'),
 Text(123.428,156.228,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(125.801,160.629,'entropy = 0.396\nsamples = 6\nvalue = [6.365, 2.373]'),
 Text(128.175,165.029,'entropy = 0.342\nsamples = 5\nvalue = [6.365, 1.78]'),
 Text(130.549,169.43,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(132.922,173.831,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(135.296,178.232,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(137.669,182.633,'entropy = 0.362\nsamples = 8\nvalue = [9.547, 2.966]'),
 Text(140.043,187.033,'entropy = 0.46\nsamples = 16\nvalue = [12.73, 7.118]'),
 Text(149.538,195.835,'X[41032] <= 0.139\nentropy = 0.203\nsamples = 1064\nvalue = [79.562, 616.33
2]'),
 Text(147.164,191.434,'X[2750] <= 0.12\nentropy = 0.177\nsamples = 1055\nvalue = [66.832,
613.366]'),
 Text(144.79,187.033,'X[41905] <= 0.071\nentropy = 0.143\nsamples = 1035\nvalue = [50.92,
604.468]'),
 Text(142.417,182.633,'X[22478] <= 0.219\nentropy = 0.128\nsamples = 1033\nvalue = [44.555, 604.46
8]'),
 Text(140.043,178.232,'X[2847] <= 0.114\nentropy = 0.112\nsamples = 1030\nvalue = [38.19,
603.875]'),
 Text(137.669,173.831,'X[30412] <= 0.139\nentropy = 0.079\nsamples = 1005\nvalue = [25.46,
591.418]'),
 Text(135.296,169.43,'X[34808] <= 0.085\nentropy = 0.061\nsamples = 999\nvalue = [19.095,
589.045]'),
 Text(132.922,165.029,'X[45436] <= 0.19\nentropy = 0.051\nsamples = 998\nvalue = [15.912,
589.045]'),
 Text(130.549,160.629,'X[31595] <= 0.167\nentropy = 0.041\nsamples = 997\nvalue = [12.73,
589.045]'),
 Text(128.175,156.228,'X[23029] <= 0.062\nentropy = 0.021\nsamples = 985\nvalue = [6.365,
583.113]'),
 Text(125.801,151.827,'X[7683] <= 0.227\nentropy = 0.011\nsamples = 984\nvalue = [3.182,
583.113]'),
 Text(123.428,147.426,'entropy = 0.0\nsamples = 983\nvalue = [0.0, 583.113]'),
 Text(128.175,147.426,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(130.549,151.827,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(132.922,156.228,'entropy = 0.499\nsamples = 12\nvalue = [6.365, 5.932]'),
 Text(135.296,160.629,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(137.669,165.029,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(140.043,169.43,'entropy = 0.396\nsamples = 6\nvalue = [6.365, 2.373]'),
 Text(142.417,173.831,'entropy = 0.5\nsamples = 25\nvalue = [12.73, 12.457]'),
 Text(144.79,178.232,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(147.164,182.633,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(149.538,187.033,'entropy = 0.46\nsamples = 20\nvalue = [15.912, 8.898]'),
 Text(151.911,191.434,'entropy = 0.307\nsamples = 9\nvalue = [12.73, 2.966]'),
 Text(161.406,200.236,'X[2903] <= 0.127\nentropy = 0.353\nsamples = 2553\nvalue = [426.453, 1434.9
44]'),
 Text(159.032,195.835,'X[2847] <= 0.058\nentropy = 0.337\nsamples = 2522\nvalue = [388.263, 1423.6
73]'),
 Text(156.658,191.434,'X[4594] <= 0.184\nentropy = 0.317\nsamples = 2462\nvalue = [343.708, 1396.3
86]'),
 Text(154.285,187.033,'X[40151] <= 0.227\nentropy = 0.291\nsamples = 2360\nvalue = [289.606, 1345.
965]'),
 Text(151.911,182.633,'X[40164] <= 0.094\nentropy = 0.278\nsamples = 2338\nvalue = [267.328, 1337.
067]'),
 Text(149.538,178.232,'X[40533] <= 0.225\nentropy = 0.269\nsamples = 2334\nvalue = [254.599, 1337.
067]'),
 Text(147.164,173.831,'X[2768] <= 0.165\nentropy = 0.259\nsamples = 2330\nvalue = [241.869, 1337.0
67]'),
 Text(144.79,169.43,'X[34938] <= 0.198\nentropy = 0.25\nsamples = 2324\nvalue = [229.139,
1335.88]'),
 Text(142.417,165.029,'X[40933] <= 0.135\nentropy = 0.238\nsamples = 2307\nvalue = [213.226, 1328.
```

```
 Text(142.417,165.029,'X[16955] <= 0.153\nentropy = 0.236\nsamples = 2307\nvalue = [213.226, 1326.
762]'),
 Text(140.043,160.629,'X[31506] <= 0.309\nentropy = 0.224\nsamples = 2276\nvalue = [194.131, 1313.
932]'),
 Text(137.669,156.228,'X[22594] <= 0.201\nentropy = 0.219\nsamples = 2274\nvalue = [187.766, 1313.
932]'),
 Text(135.296,151.827,'X[19207] <= 0.091\nentropy = 0.213\nsamples = 2272\nvalue = [181.401, 1313.
932]'),
 Text(132.922,147.426,'X[34803] <= 0.107\nentropy = 0.207\nsamples = 2270\nvalue = [175.037, 1313.
932]'),
 Text(130.549,143.025,'X[26459] <= 0.107\nentropy = 0.202\nsamples = 2268\nvalue = [168.672, 1313.
932]'),
 Text(128.175,138.625,'X[28152] <= 0.227\nentropy = 0.196\nsamples = 2266\nvalue = [162.307, 1313.
932]'),
 Text(125.801,134.224,'X[25688] <= 0.201\nentropy = 0.182\nsamples = 2235\nvalue = [146.394, 1298.
509]'),
 Text(123.428,129.823,'X[29442] <= 0.221\nentropy = 0.176\nsamples = 2233\nvalue = [140.029, 1298.
509]'),
 Text(121.054,125.422,'X[44498] <= 0.23\nentropy = 0.169\nsamples = 2229\nvalue = [133.664,
1297.322]'),
 Text(118.681,121.022,'X[43274] <= 0.413\nentropy = 0.163\nsamples = 2225\nvalue = [127.299, 1296.
136]'),
 Text(116.307,116.621,'X[34048] <= 0.146\nentropy = 0.156\nsamples = 2221\nvalue = [120.934, 1294.
95]'),
 Text(113.933,112.22,'X[14952] <= 0.214\nentropy = 0.147\nsamples = 2206\nvalue = [111.387, 1287.8
31]'),
 Text(111.56,107.819,'X[44424] <= 0.236\nentropy = 0.136\nsamples = 2191\nvalue = [101.839, 1280.7
13]'),
 Text(109.186,103.418,'X[6007] <= 0.177\nentropy = 0.129\nsamples = 2186\nvalue = [95.474, 1278.93
3]'),
 Text(106.813,99.0176,'X[23067] <= 0.381\nentropy = 0.122\nsamples = 2181\nvalue = [89.109, 1277.1
54]'),
 Text(104.439,94.6169,'X[23454] <= 0.239\nentropy = 0.108\nsamples = 2150\nvalue = [76.38,
1261.137]'),
 Text(102.065,90.2161,'X[5710] <= 0.325\nentropy = 0.1\nsamples = 2143\nvalue = [70.015,
1258.171]'),
 Text(99.6917,85.8153,'X[44230] <= 0.23\nentropy = 0.092\nsamples = 2136\nvalue = [63.65,
1255.205]'),
 Text(97.3181,81.4145,'X[12255] <= 0.23\nentropy = 0.084\nsamples = 2129\nvalue = [57.285,
1252.239]'),
 Text(94.9444,77.0137,'X[8238] <= 0.272\nentropy = 0.075\nsamples = 2118\nvalue = [50.92,
1246.901]'),
 Text(92.5708,72.6129,'X[38472] <= 0.207\nentropy = 0.071\nsamples = 2117\nvalue = [47.737, 1246.9
01]'),
 Text(90.1972,68.2122,'X[12136] <= 0.129\nentropy = 0.067\nsamples = 2116\nvalue = [44.555, 1246.9
01]'),
 Text(87.8236,63.8114,'X[33709] <= 0.112\nentropy = 0.062\nsamples = 2115\nvalue = [41.372, 1246.9
01]'),
 Text(85.45,59.4106,'X[20492] <= 0.224\nentropy = 0.058\nsamples = 2114\nvalue = [38.19,
1246.901]'),
 Text(83.0764,55.0098,'X[31921] <= 0.236\nentropy = 0.053\nsamples = 2113\nvalue = [35.007, 1246.9
01]'),
 Text(80.7028,50.609,'X[14094] <= 0.225\nentropy = 0.049\nsamples = 2112\nvalue = [31.825, 1246.90
1]'),
 Text(78.3292,46.2082,'X[33926] <= 0.102\nentropy = 0.044\nsamples = 2111\nvalue = [28.642, 1246.9
01]'),
 Text(75.9556,41.8075,'X[12094] <= 0.09\nentropy = 0.039\nsamples = 2110\nvalue = [25.46,
1246.901]'),
 Text(73.5819,37.4067,'X[1894] <= 0.121\nentropy = 0.034\nsamples = 2109\nvalue = [22.277, 1246.90
1]'),
 Text(71.2083,33.0059,'X[41823] <= 0.25\nentropy = 0.03\nsamples = 2108\nvalue = [19.095,
1246.901]'),
 Text(68.8347,28.6051,'X[39385] <= 0.283\nentropy = 0.025\nsamples = 2107\nvalue = [15.912, 1246.9
01]'),
 Text(66.4611,24.2043,'X[7013] <= 0.272\nentropy = 0.02\nsamples = 2106\nvalue = [12.73,
1246.901]'),
 Text(64.0875,19.8035,'X[25496] <= 0.078\nentropy = 0.015\nsamples = 2105\nvalue = [9.547,
1246.901]'),
 Text(61.7139,15.4027,'X[4730] <= 0.461\nentropy = 0.01\nsamples = 2104\nvalue = [6.365,
1246.901]'),
 Text(59.3403,11.002,'X[23717] <= 0.121\nentropy = 0.005\nsamples = 2103\nvalue = [3.182,
1246.901]'),
 Text(56.9667,6.60118,'entropy = 0.0\nsamples = 2101\nvalue = [0.0, 1246.307]'),
 Text(61.7139,6.60118,'entropy = 0.265\nsamples = 2\nvalue = [3.182, 0.593]'),
 Text(64.0875,11.002,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(66.4611,15.4027,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(68.8347,19.8035,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(71.2083,24.2043,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
```

```
 Text(71.2083,24.2043,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(73.5819,28.6051,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(75.9556,33.0059,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(78.3292,37.4067,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(80.7028,41.8075,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(83.0764,46.2082,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(85.45,50.609,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(87.8236,55.0098,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(90.1972,59.4106,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(92.5708,63.8114,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(94.9444,68.2122,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(97.3181,72.6129,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(99.6917,77.0137,'entropy = 0.496\nsamples = 11\nvalue = [6.365, 5.339]'),
 Text(102.065,81.4145,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(104.439,85.8153,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(106.813,90.2161,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(109.186,94.6169,'entropy = 0.493\nsamples = 31\nvalue = [12.73, 16.016]'),
 Text(111.56,99.0176,'entropy = 0.342\nsamples = 5\nvalue = [6.365, 1.78]'),
 Text(113.933,103.418,'entropy = 0.342\nsamples = 5\nvalue = [6.365, 1.78]'),
 Text(116.307,107.819,'entropy = 0.489\nsamples = 15\nvalue = [9.547, 7.118]'),
 Text(118.681,112.22,'entropy = 0.489\nsamples = 15\nvalue = [9.547, 7.118]'),
 Text(121.054,116.621,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(123.428,121.022,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(125.801,125.422,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(128.175,129.823,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(130.549,134.224,'entropy = 0.5\nsamples = 31\nvalue = [15.912, 15.423]'),
 Text(132.922,138.625,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(135.296,143.025,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(137.669,147.426,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(140.043,151.827,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(142.417,156.228,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(144.79,160.629,'entropy = 0.492\nsamples = 31\nvalue = [19.095, 14.83]'),
 Text(147.164,165.029,'entropy = 0.427\nsamples = 17\nvalue = [15.912, 7.118]'),
 Text(149.538,169.43,'entropy = 0.156\nsamples = 6\nvalue = [12.73, 1.186]'),
 Text(151.911,173.831,'entropy = 0.0\nsamples = 4\nvalue = [12.73, 0.0]'),
 Text(154.285,178.232,'entropy = 0.0\nsamples = 4\nvalue = [12.73, 0.0]'),
 Text(156.658,182.633,'entropy = 0.408\nsamples = 22\nvalue = [22.277, 8.898]'),
 Text(159.032,187.033,'entropy = 0.499\nsamples = 102\nvalue = [54.102, 50.422]'),
 Text(161.406,191.434,'entropy = 0.471\nsamples = 60\nvalue = [44.555, 27.287]'),
 Text(163.779,195.835,'entropy = 0.352\nsamples = 31\nvalue = [38.19, 11.271]'),
 Text(173.274,204.636,'X[2903] <= 0.109\nentropy = 0.304\nsamples = 2937\nvalue = [385.08, 1670.44
4]'),
 Text(170.9,200.236,'X[38261] <= 0.155\nentropy = 0.275\nsamples = 2901\nvalue = [327.796,
1659.766]'),
 Text(168.526,195.835,'X[41032] <= 0.031\nentropy = 0.263\nsamples = 2893\nvalue = [305.518, 1659.
173]'),
 Text(166.153,191.434,'X[11442] <= 0.08\nentropy = 0.245\nsamples = 2853\nvalue = [273.693,
1641.377]'),
 Text(163.779,187.033,'X[28047] <= 0.045\nentropy = 0.237\nsamples = 2849\nvalue = [260.964, 1641.
377]'),
 Text(161.406,182.633,'X[29557] <= 0.272\nentropy = 0.226\nsamples = 2837\nvalue = [245.051, 1637.
224]'),
 Text(159.032,178.232,'X[40151] <= 0.17\nentropy = 0.218\nsamples = 2831\nvalue = [232.321,
1636.038]'),
 Text(156.658,173.831,'X[19359] <= 0.174\nentropy = 0.205\nsamples = 2808\nvalue = [213.226, 1625.
954]'),
 Text(154.285,169.43,'X[4105] <= 0.163\nentropy = 0.192\nsamples = 2784\nvalue = [194.131,
1615.276]'),
 Text(151.911,165.029,'X[28716] <= 0.246\nentropy = 0.184\nsamples = 2780\nvalue = [184.584, 1614.
683]'),
 Text(149.538,160.629,'X[31595] <= 0.295\nentropy = 0.177\nsamples = 2775\nvalue = [175.037, 1613.
497]'),
 Text(147.164,156.228,'X[22747] <= 0.17\nentropy = 0.169\nsamples = 2767\nvalue = [165.489,
1610.531]'),
 Text(144.79,151.827,'X[10089] <= 0.268\nentropy = 0.159\nsamples = 2749\nvalue = [152.759, 1602.2
26]'),
 Text(142.417,147.426,'X[29126] <= 0.208\nentropy = 0.153\nsamples = 2747\nvalue = [146.394, 1602.
226]'),
 Text(140.043,143.025,'X[3255] <= 0.224\nentropy = 0.148\nsamples = 2745\nvalue = [140.029, 1602.2
26]'),
 Text(137.669,138.625,'X[35751] <= 0.087\nentropy = 0.142\nsamples = 2743\nvalue = [133.664, 1602.
226]'),
 Text(135.296,134.224,'X[4247] <= 0.065\nentropy = 0.136\nsamples = 2741\nvalue = [127.299, 1602.2
26]'),
 Text(132.922,129.823,'X[2917] <= 0.409\nentropy = 0.131\nsamples = 2739\nvalue = [120.934, 1602.2
26]'),
 Text(130.549,125.422,'X[37020] <= 0.198\nentropy = 0.125\nsamples = 2736\nvalue = [114.569, 1601.
633]'),
```

```
655]'),
 Text(128.175,121.022,'X[19249] <= 0.179\nentropy = 0.119\nsamples = 2733\nvalue = [108.204, 1601.
039]'),
 Text(125.801,116.621,'X[2768] <= 0.216\nentropy = 0.112\nsamples = 2730\nvalue = [101.839, 1600.4
46]'),
 Text(123.428,112.22,'X[9197] <= 0.06\nentropy = 0.106\nsamples = 2726\nvalue = [95.474,
1599.26]'),
 Text(121.054,107.819,'X[24971] <= 0.207\nentropy = 0.097\nsamples = 2708\nvalue = [85.927, 1590.3
62]'),
 Text(118.681,103.418,'X[44230] <= 0.187\nentropy = 0.091\nsamples = 2703\nvalue = [79.562, 1588.5
82]'),
 Text(116.307,99.0176,'X[45662] <= 0.107\nentropy = 0.084\nsamples = 2697\nvalue = [73.197, 1586.2
09]'),
 Text(113.933,94.6169,'X[41653] <= 0.121\nentropy = 0.081\nsamples = 2696\nvalue = [70.015, 1586.2
09]'),
 Text(111.56,90.2161,'X[34727] <= 0.134\nentropy = 0.078\nsamples = 2695\nvalue = [66.832, 1586.20
9]'),
 Text(109.186,85.8153,'X[5440] <= 0.431\nentropy = 0.074\nsamples = 2694\nvalue = [63.65,
1586.209]'),
 Text(106.813,81.4145,'X[44598] <= 0.302\nentropy = 0.071\nsamples = 2693\nvalue = [60.467, 1586.2
09]'),
 Text(104.439,77.0137,'X[11647] <= 0.088\nentropy = 0.067\nsamples = 2692\nvalue = [57.285, 1586.2
09]'),
 Text(102.065,72.6129,'X[41722] <= 0.1\nentropy = 0.064\nsamples = 2691\nvalue = [54.102,
1586.209]'),
 Text(99.6917,68.2122,'X[23067] <= 0.644\nentropy = 0.06\nsamples = 2690\nvalue = [50.92,
1586.209]'),
 Text(97.3181,63.8114,'X[41663] <= 0.088\nentropy = 0.057\nsamples = 2689\nvalue = [47.737, 1586.2
09]'),
 Text(94.9444,59.4106,'X[12106] <= 0.067\nentropy = 0.053\nsamples = 2688\nvalue = [44.555, 1586.2
09]'),
 Text(92.5708,55.0098,'X[18166] <= 0.385\nentropy = 0.05\nsamples = 2687\nvalue = [41.372,
1586.209]'),
 Text(90.1972,50.609,'X[382] <= 0.167\nentropy = 0.046\nsamples = 2686\nvalue = [38.19,
1586.209]'),
 Text(87.8236,46.2082,'X[37152] <= 0.09\nentropy = 0.042\nsamples = 2685\nvalue = [35.007,
1586.209]'),
 Text(85.45,41.8075,'X[3885] <= 0.139\nentropy = 0.039\nsamples = 2684\nvalue = [31.825,
1586.209]'),
 Text(83.0764,37.4067,'X[15320] <= 0.094\nentropy = 0.035\nsamples = 2683\nvalue = [28.642, 1586.2
09]'),
 Text(80.7028,33.0059,'X[32585] <= 0.091\nentropy = 0.031\nsamples = 2682\nvalue = [25.46,
1586.209]'),
 Text(78.3292,28.6051,'X[35188] <= 0.186\nentropy = 0.027\nsamples = 2681\nvalue = [22.277, 1586.2
09]'),
 Text(75.9556,24.2043,'X[31004] <= 0.118\nentropy = 0.024\nsamples = 2680\nvalue = [19.095, 1586.2
09]'),
 Text(73.5819,19.8035,'X[18878] <= 0.335\nentropy = 0.02\nsamples = 2679\nvalue = [15.912,
1586.209]'),
 Text(71.2083,15.4027,'X[43825] <= 0.449\nentropy = 0.016\nsamples = 2678\nvalue = [12.73,
1586.209]'),
 Text(68.8347,11.002,'X[28386] <= 0.158\nentropy = 0.012\nsamples = 2677\nvalue = [9.547,
1586.209]'),
 Text(66.4611,6.60118,'X[21430] <= 0.088\nentropy = 0.008\nsamples = 2676\nvalue = [6.365,
1586.209]'),
 Text(64.0875,2.20039,'entropy = 0.004\nsamples = 2675\nvalue = [3.182, 1586.209]'),
 Text(68.8347,2.20039,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(71.2083,6.60118,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(73.5819,11.002,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(75.9556,15.4027,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(78.3292,19.8035,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(80.7028,24.2043,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(83.0764,28.6051,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(85.45,33.0059,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(87.8236,37.4067,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(90.1972,41.8075,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(92.5708,46.2082,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(94.9444,50.609,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(97.3181,55.0098,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(99.6917,59.4106,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(102.065,63.8114,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(104.439,68.2122,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(106.813,72.6129,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(109.186,77.0137,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(111.56,81.4145,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(113.933,85.8153,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(116.307,90.2161,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(118.681,94.6169,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
```

```
 Text(118.681,94.6169,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(121.054,99.0176,'entropy = 0.396\nsamples = 6\nvalue = [6.365, 2.373]'),
 Text(123.428,103.418,'entropy = 0.342\nsamples = 5\nvalue = [6.365, 1.78]'),
 Text(125.801,107.819,'entropy = 0.499\nsamples = 18\nvalue = [9.547, 8.898]'),
 Text(128.175,112.22,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(130.549,116.621,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(132.922,121.022,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(135.296,125.422,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(137.669,129.823,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(140.043,134.224,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(142.417,138.625,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(144.79,143.025,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(147.164,147.426,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(149.538,151.827,'entropy = 0.478\nsamples = 18\nvalue = [12.73, 8.305]'),
 Text(151.911,156.228,'entropy = 0.362\nsamples = 8\nvalue = [9.547, 2.966]'),
 Text(154.285,160.629,'entropy = 0.197\nsamples = 5\nvalue = [9.547, 1.186]'),
 Text(156.658,165.029,'entropy = 0.11\nsamples = 4\nvalue = [9.547, 0.593]'),
 Text(159.032,169.43,'entropy = 0.46\nsamples = 24\nvalue = [19.095, 10.678]'),
 Text(161.406,173.831,'entropy = 0.452\nsamples = 23\nvalue = [19.095, 10.084]'),
 Text(163.779,178.232,'entropy = 0.156\nsamples = 6\nvalue = [12.73, 1.186]'),
 Text(166.153,182.633,'entropy = 0.328\nsamples = 12\nvalue = [15.912, 4.152]'),
 Text(168.526,187.033,'entropy = 0.0\nsamples = 4\nvalue = [12.73, 0.0]'),
 Text(170.9,191.434,'entropy = 0.46\nsamples = 40\nvalue = [31.825, 17.796]'),
 Text(173.274,195.835,'entropy = 0.051\nsamples = 8\nvalue = [22.277, 0.593]'),
 Text(175.647,200.236,'entropy = 0.265\nsamples = 36\nvalue = [57.285, 10.678]'),
 Text(185.142,209.037,'X[11440] <= 0.096\nentropy = 0.223\nsamples = 2827\nvalue = [238.686, 1632.
479]'),
 Text(182.768,204.636,'X[42813] <= 0.038\nentropy = 0.196\nsamples = 2803\nvalue = [200.496, 1625.
361]'),
 Text(180.394,200.236,'X[4650] <= 0.193\nentropy = 0.179\nsamples = 2784\nvalue = [178.219, 1618.2
42]'),
 Text(178.021,195.835,'X[4681] <= 0.119\nentropy = 0.159\nsamples = 2754\nvalue = [152.759, 1605.1
92]'),
 Text(175.647,191.434,'X[38261] <= 0.044\nentropy = 0.148\nsamples = 2747\nvalue = [140.029, 1603.
412]'),
 Text(173.274,187.033,'X[45172] <= 0.048\nentropy = 0.136\nsamples = 2739\nvalue = [127.299, 1601.
039]'),
 Text(170.9,182.633,'X[26459] <= 0.162\nentropy = 0.125\nsamples = 2731\nvalue = [114.569,
1598.667]'),
 Text(168.526,178.232,'X[37414] <= 0.099\nentropy = 0.119\nsamples = 2729\nvalue = [108.204, 1598.
667]'),
 Text(166.153,173.831,'X[2903] <= 0.168\nentropy = 0.113\nsamples = 2727\nvalue = [101.839, 1598.6
67]'),
 Text(163.779,169.43,'X[45379] <= 0.141\nentropy = 0.103\nsamples = 2716\nvalue = [92.292, 1593.92
1]'),
 Text(161.406,165.029,'X[4105] <= 0.187\nentropy = 0.094\nsamples = 2705\nvalue = [82.745, 1589.17
5]'),
 Text(159.032,160.629,'X[11127] <= 0.11\nentropy = 0.088\nsamples = 2702\nvalue = [76.38,
1588.582]'),
 Text(156.658,156.228,'X[7004] <= 0.173\nentropy = 0.081\nsamples = 2698\nvalue = [70.015, 1587.39
6]'),
 Text(154.285,151.827,'X[14049] <= 0.099\nentropy = 0.074\nsamples = 2693\nvalue = [63.65,
1585.616]'),
 Text(151.911,147.426,'X[34896] <= 0.11\nentropy = 0.067\nsamples = 2685\nvalue = [57.285,
1582.057]'),
 Text(149.538,143.025,'X[14376] <= 0.241\nentropy = 0.061\nsamples = 2676\nvalue = [50.92,
1577.905]'),
 Text(147.164,138.625,'X[40164] <= 0.227\nentropy = 0.057\nsamples = 2675\nvalue = [47.737, 1577.9
05]'),
 Text(144.79,134.224,'X[13256] <= 0.297\nentropy = 0.053\nsamples = 2674\nvalue = [44.555, 1577.90
5]'),
 Text(142.417,129.823,'X[36694] <= 0.115\nentropy = 0.05\nsamples = 2673\nvalue = [41.372,
1577.905]'),
 Text(140.043,125.422,'X[39927] <= 0.105\nentropy = 0.046\nsamples = 2672\nvalue = [38.19,
1577.905]'),
 Text(137.669,121.022,'X[4704] <= 0.059\nentropy = 0.042\nsamples = 2671\nvalue = [35.007, 1577.90
5]'),
 Text(135.296,116.621,'X[31021] <= 0.031\nentropy = 0.039\nsamples = 2670\nvalue = [31.825, 1577.9
05]'),
 Text(132.922,112.22,'X[860] <= 0.071\nentropy = 0.035\nsamples = 2669\nvalue = [28.642,
1577.905]'),
 Text(130.549,107.819,'X[30235] <= 0.261\nentropy = 0.031\nsamples = 2668\nvalue = [25.46,
1577.905]'),
 Text(128.175,103.418,'X[33928] <= 0.309\nentropy = 0.027\nsamples = 2667\nvalue = [22.277, 1577.9
05]'),
 Text(125.801,99.0176,'X[36111] <= 0.112\nentropy = 0.024\nsamples = 2666\nvalue = [19.095, 1577.9
05]'),
```

```
 Text(123.428,94.6169,'X[4038] <= 0.112\nentropy = 0.02\nsamples = 2665\nvalue = [15.912,
1577.905]'),
 Text(121.054,90.2161,'X[5989] <= 0.221\nentropy = 0.016\nsamples = 2664\nvalue = [12.73,
1577.905]'),
 Text(118.681,85.8153,'X[3801] <= 0.219\nentropy = 0.012\nsamples = 2663\nvalue = [9.547,
1577.905]'),
 Text(116.307,81.4145,'X[44657] <= 0.239\nentropy = 0.008\nsamples = 2662\nvalue = [6.365,
1577.905]'),
 Text(113.933,77.0137,'X[2433] <= 0.135\nentropy = 0.004\nsamples = 2661\nvalue = [3.182,
1577.905]'),
 Text(111.56,72.6129,'entropy = 0.0\nsamples = 2660\nvalue = [0.0, 1577.905]'),
 Text(116.307,72.6129,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(118.681,77.0137,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(121.054,81.4145,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(123.428,85.8153,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(125.801,90.2161,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(128.175,94.6169,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(130.549,99.0176,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(132.922,103.418,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(135.296,107.819,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(137.669,112.22,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(140.043,116.621,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(142.417,121.022,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(144.79,125.422,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(147.164,129.823,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(149.538,134.224,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(151.911,138.625,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(154.285,143.025,'entropy = 0.478\nsamples = 9\nvalue = [6.365, 4.152]'),
 Text(156.658,147.426,'entropy = 0.46\nsamples = 8\nvalue = [6.365, 3.559]'),
 Text(159.032,151.827,'entropy = 0.342\nsamples = 5\nvalue = [6.365, 1.78]'),
 Text(161.406,156.228,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(163.779,160.629,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(166.153,165.029,'entropy = 0.444\nsamples = 11\nvalue = [9.547, 4.746]'),
 Text(168.526,169.43,'entropy = 0.444\nsamples = 11\nvalue = [9.547, 4.746]'),
 Text(170.9,173.831,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(173.274,178.232,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(175.647,182.633,'entropy = 0.265\nsamples = 8\nvalue = [12.73, 2.373]'),
 Text(178.021,187.033,'entropy = 0.265\nsamples = 8\nvalue = [12.73, 2.373]'),
 Text(180.394,191.434,'entropy = 0.215\nsamples = 7\nvalue = [12.73, 1.78]'),
 Text(182.768,195.835,'entropy = 0.448\nsamples = 30\nvalue = [25.46, 13.05]'),
 Text(185.142,200.236,'entropy = 0.367\nsamples = 19\nvalue = [22.277, 7.118]'),
 Text(187.515,204.636,'entropy = 0.265\nsamples = 24\nvalue = [38.19, 7.118]'),
 Text(197.01,213.438,'X[11440] <= 0.071\nentropy = 0.225\nsamples = 7399\nvalue = [633.314, 4271.0
2]'),
 Text(194.636,209.037,'X[19359] <= 0.073\nentropy = 0.212\nsamples = 7348\nvalue = [582.394, 4250.
258]'),
 Text(192.263,204.636,'X[42813] <= 0.091\nentropy = 0.193\nsamples = 7193\nvalue = [506.015, 4172.
55]'),
 Text(189.889,200.236,'X[41032] <= 0.055\nentropy = 0.182\nsamples = 7165\nvalue = [467.825, 4163.
058]'),
 Text(187.515,195.835,'X[26775] <= 0.082\nentropy = 0.166\nsamples = 7025\nvalue = [410.54, 4090.6
88]'),
 Text(185.142,191.434,'X[2768] <= 0.139\nentropy = 0.151\nsamples = 6866\nvalue = [359.62, 4005.86
1]'),
 Text(182.768,187.033,'X[35635] <= 0.141\nentropy = 0.146\nsamples = 6857\nvalue = [343.708, 4003.
488]'),
 Text(180.394,182.633,'X[40533] <= 0.153\nentropy = 0.141\nsamples = 6845\nvalue = [330.978, 3998.
743]'),
 Text(178.021,178.232,'X[38261] <= 0.147\nentropy = 0.137\nsamples = 6833\nvalue = [318.248, 3993.
997]'),
 Text(175.647,173.831,'X[45288] <= 0.143\nentropy = 0.132\nsamples = 6820\nvalue = [305.518, 3988.
658]'),
 Text(173.274,169.43,'X[37582] <= 0.121\nentropy = 0.129\nsamples = 6814\nvalue = [295.971, 3986.8
79]'),
 Text(170.9,165.029,'X[10882] <= 0.066\nentropy = 0.125\nsamples = 6806\nvalue = [286.423,
3983.913]'),
 Text(168.526,160.629,'X[33251] <= 0.082\nentropy = 0.123\nsamples = 6804\nvalue = [280.058, 3983.
913]'),
 Text(166.153,156.228,'X[41755] <= 0.153\nentropy = 0.12\nsamples = 6802\nvalue = [273.693,
3983.913]'),
 Text(163.779,151.827,'X[16244] <= 0.171\nentropy = 0.116\nsamples = 6781\nvalue = [260.964, 3973.
828]'),
 Text(161.406,147.426,'X[28384] <= 0.156\nentropy = 0.113\nsamples = 6779\nvalue = [254.599, 3973.
828]'),
 Text(159.032,143.025,'X[13107] <= 0.048\nentropy = 0.111\nsamples = 6777\nvalue = [248.234, 3973.
828]'),
 Text(156.658,138.625,'X[45785] <= 0.208\nentropy = 0.108\nsamples = 6775\nvalue = [241.869, 3973.
```

```
828]'),
 Text(154.285,134.224,'X[4105] <= 0.366\nentropy = 0.106\nsamples = 6773\nvalue = [235.504, 3973.8
28]'),
 Text(151.911,129.823,'X[6579] <= 0.126\nentropy = 0.103\nsamples = 6771\nvalue = [229.139, 3973.8
28]'),
 Text(149.538,125.422,'X[11442] <= 0.093\nentropy = 0.099\nsamples = 6761\nvalue = [219.591, 3969.
676]'),
 Text(147.164,121.022,'X[44545] <= 0.039\nentropy = 0.097\nsamples = 6758\nvalue = [213.226, 3969.
083]'),
 Text(144.79,116.621,'X[34516] <= 0.133\nentropy = 0.093\nsamples = 6746\nvalue = [203.679, 3963.7
44]'),
 Text(142.417,112.22,'X[19518] <= 0.156\nentropy = 0.09\nsamples = 6743\nvalue = [197.314,
3963.151]'),
 Text(140.043,107.819,'X[22109] <= 0.149\nentropy = 0.088\nsamples = 6739\nvalue = [190.949, 3961.
965]'),
 Text(137.669,103.418,'X[12083] <= 0.195\nentropy = 0.085\nsamples = 6733\nvalue = [184.584, 3959.
592]'),
 Text(135.296,99.0176,'X[43544] <= 0.219\nentropy = 0.082\nsamples = 6726\nvalue = [178.219, 3956.
626]'),
 Text(132.922,94.6169,'X[38339] <= 0.264\nentropy = 0.08\nsamples = 6719\nvalue = [171.854,
3953.66]'),
 Text(130.549,90.2161,'X[32651] <= 0.114\nentropy = 0.077\nsamples = 6711\nvalue = [165.489, 3950.
101]'),
 Text(128.175,85.8153,'X[16297] <= 0.16\nentropy = 0.075\nsamples = 6701\nvalue = [159.124,
3945.355]'),
 Text(125.801,81.4145,'X[14551] <= 0.348\nentropy = 0.071\nsamples = 6674\nvalue = [149.577, 3931.
118]'),
 Text(123.428,77.0137,'X[10127] <= 0.594\nentropy = 0.068\nsamples = 6663\nvalue = [143.212, 3925.
78]'),
 Text(121.054,72.6129,'X[43499] <= 0.059\nentropy = 0.067\nsamples = 6662\nvalue = [140.029, 3925.
78]'),
 Text(118.681,68.2122,'X[3732] <= 0.034\nentropy = 0.065\nsamples = 6661\nvalue = [136.847, 3925.7
8]'),
 Text(116.307,63.8114,'X[17715] <= 0.082\nentropy = 0.064\nsamples = 6660\nvalue = [133.664, 3925.
78]'),
 Text(113.933,59.4106,'X[43047] <= 0.066\nentropy = 0.062\nsamples = 6659\nvalue = [130.482, 3925.
78]'),
 Text(111.56,55.0098,'X[23922] <= 0.151\nentropy = 0.061\nsamples = 6658\nvalue = [127.299, 3925.7
8]'),
 Text(109.186,50.609,'X[14430] <= 0.125\nentropy = 0.059\nsamples = 6657\nvalue = [124.117, 3925.7
8]'),
 Text(106.813,46.2082,'X[21899] <= 0.338\nentropy = 0.058\nsamples = 6656\nvalue = [120.934, 3925.
78]'),
 Text(104.439,41.8075,'X[30305] <= 0.362\nentropy = 0.057\nsamples = 6655\nvalue = [117.752, 3925.
78]'),
 Text(102.065,37.4067,'X[22406] <= 0.226\nentropy = 0.055\nsamples = 6654\nvalue = [114.569, 3925.
78]'),
 Text(99.6917,33.0059,'X[17092] <= 0.286\nentropy = 0.054\nsamples = 6653\nvalue = [111.387, 3925.
78]'),
 Text(97.3181,28.6051,'X[43830] <= 0.064\nentropy = 0.052\nsamples = 6652\nvalue = [108.204, 3925.
78]'),
 Text(94.9444,24.2043,'X[28647] <= 0.115\nentropy = 0.051\nsamples = 6651\nvalue = [105.022, 3925.
78]'),
 Text(92.5708,19.8035,'X[23855] <= 0.547\nentropy = 0.049\nsamples = 6650\nvalue = [101.839, 3925.
78]'),
 Text(90.1972,15.4027,'X[32726] <= 0.141\nentropy = 0.048\nsamples = 6649\nvalue = [98.657, 3925.7
8]'),
 Text(87.8236,11.002,'X[22913] <= 0.074\nentropy = 0.046\nsamples = 6648\nvalue = [95.474, 3925.78
]'),
 Text(85.45,6.60118,'X[34673] <= 0.154\nentropy = 0.045\nsamples = 6647\nvalue = [92.292,
3925.78]'),
 Text(83.0764,2.20039,'entropy = 0.043\nsamples = 6646\nvalue = [89.109, 3925.78]'),
 Text(87.8236,2.20039,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(90.1972,6.60118,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(92.5708,11.002,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(94.9444,15.4027,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(97.3181,19.8035,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(99.6917,24.2043,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(102.065,28.6051,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(104.439,33.0059,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(106.813,37.4067,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(109.186,41.8075,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(111.56,46.2082,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(113.933,50.609,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(116.307,55.0098,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(118.681,59.4106,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(121.054,63.8114,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
```

```
 Text(123.428,68.2122,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(125.801,72.6129,'entropy = 0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(128.175,77.0137,'entropy = 0.496\nsamples = 11\nvalue = [6.365, 5.339]'),
 Text(130.549,81.4145,'entropy = 0.481\nsamples = 27\nvalue = [9.547, 14.237]'),
 Text(132.922,85.8153,'entropy = 0.489\nsamples = 10\nvalue = [6.365, 4.746]'),
 Text(135.296,90.2161,'entropy = 0.46\nsamples = 8\nvalue = [6.365, 3.559]'),
 Text(137.669,94.6169,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(140.043,99.0176,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(142.417,103.418,'entropy = 0.396\nsamples = 6\nvalue = [6.365, 2.373]'),
 Text(144.79,107.819,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(147.164,112.22,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(149.538,116.621,'entropy = 0.46\nsamples = 12\nvalue = [9.547, 5.339]'),
 Text(151.911,121.022,'entropy = 0.156\nsamples = 3\nvalue = [6.365, 0.593]'),
 Text(154.285,125.422,'entropy = 0.422\nsamples = 10\nvalue = [9.547, 4.152]'),
 Text(156.658,129.823,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(159.032,134.224,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(161.406,138.625,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(163.779,143.025,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(166.153,147.426,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(168.526,151.827,'entropy = 0.493\nsamples = 21\nvalue = [12.73, 10.084]'),
 Text(170.9,156.228,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(173.274,160.629,'entropy = 0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(175.647,165.029,'entropy = 0.362\nsamples = 8\nvalue = [9.547, 2.966]'),
 Text(178.021,169.43,'entropy = 0.265\nsamples = 6\nvalue = [9.547, 1.78]'),
 Text(180.394,173.831,'entropy = 0.416\nsamples = 13\nvalue = [12.73, 5.339]'),
 Text(182.768,178.232,'entropy = 0.396\nsamples = 12\nvalue = [12.73, 4.746]'),
 Text(185.142,182.633,'entropy = 0.396\nsamples = 12\nvalue = [12.73, 4.746]'),
 Text(187.515,187.033,'entropy = 0.226\nsamples = 9\nvalue = [15.912, 2.373]'),
 Text(189.889,191.434,'entropy = 0.469\nsamples = 159\nvalue = [50.92, 84.827]'),
 Text(192.263,195.835,'entropy = 0.493\nsamples = 140\nvalue = [57.285, 72.37]'),
 Text(194.636,200.236,'entropy = 0.319\nsamples = 28\nvalue = [38.19, 9.491]'),
 Text(197.01,204.636,'entropy = 0.5\nsamples = 155\nvalue = [76.38, 77.709]'),
 Text(199.383,209.037,'entropy = 0.411\nsamples = 51\nvalue = [50.92, 20.762]'),
 Text(313.817,217.839,'X[17569] <= 0.057\nentropy = 0.478\nsamples = 32245\nvalue = [22700.645, 14
896.37]'),
 Text(296.516,213.438,'X[10560] <= 0.016\nentropy = 0.454\nsamples = 25004\nvalue = [20616.119, 10
989.573]'),
 Text(284.462,209.037,'X[3744] <= 0.056\nentropy = 0.442\nsamples = 23213\nvalue = [20285.141, 998
8.849]'),
 Text(272.224,204.636,'X[29622] <= 0.046\nentropy = 0.426\nsamples = 20584\nvalue = [19381.316, 85
97.801]'),
 Text(259.614,200.236,'X[23595] <= 0.118\nentropy = 0.416\nsamples = 19549\nvalue = [19180.82, 802
1.213]'),
 Text(246.262,195.835,'X[23609] <= 0.041\nentropy = 0.399\nsamples = 17201\nvalue = [18041.491, 68
40.751]'),
 Text(231.427,191.434,'X[26852] <= 0.081\nentropy = 0.387\nsamples = 16106\nvalue = [17637.316, 62
66.536]'),
 Text(213.625,187.033,'X[17183] <= 0.12\nentropy = 0.375\nsamples = 15106\nvalue = [17226.776, 574
9.861]'),
 Text(200.57,182.633,'X[45177] <= 0.117\nentropy = 0.348\nsamples = 11836\nvalue = [14744.44, 4272
.8]'),
 Text(198.197,178.232,'X[18895] <= 0.076\nentropy = 0.342\nsamples = 11558\nvalue = [14696.702, 41
16.789]'),
 Text(195.823,173.831,'X[13900] <= 0.039\nentropy = 0.335\nsamples = 11246\nvalue = [14610.775, 39
47.728]'),
 Text(193.449,169.43,'X[14617] <= 0.056\nentropy = 0.326\nsamples = 10827\nvalue = [14435.739, 373
1.804]'),
 Text(188.702,165.029,'X[1285] <= 0.052\nentropy = 0.316\nsamples = 10278\nvalue = [14136.586, 346
1.899]'),
 Text(186.328,160.629,'X[38723] <= 0.07\nentropy = 0.311\nsamples = 10104\nvalue = [14098.396, 336
5.801]'),
 Text(183.955,156.228,'X[27139] <= 0.29\nentropy = 0.304\nsamples = 9758\nvalue = [13923.359, 3193
.181]'),
 Text(175.647,151.827,'X[1233] <= 0.032\nentropy = 0.343\nsamples = 6506\nvalue = [8252.176, 2321.
181]'),
 Text(173.274,147.426,'X[40199] <= 0.026\nentropy = 0.33\nsamples = 6074\nvalue = [8000.76,
2111.782]'),
 Text(170.9,143.025,'X[11440] <= 0.079\nentropy = 0.321\nsamples = 5783\nvalue = [7838.453,
1969.415]'),
 Text(168.526,138.625,'X[31572] <= 0.073\nentropy = 0.335\nsamples = 5535\nvalue = [7182.862, 1944
.501]'),
 Text(163.779,134.224,'X[34162] <= 0.032\nentropy = 0.363\nsamples = 4404\nvalue = [5241.548, 1635
.445]'),
 Text(161.406,129.823,'X[21842] <= 0.037\nentropy = 0.358\nsamples = 4322\nvalue = [5225.636, 1589
.769]'),
 Text(159.032,125.422,'X[45829] <= 0.097\nentropy = 0.348\nsamples = 4097\nvalue = [5104.701, 1478
.841]'),
```

```
 Text(156.658,121.022,'X[14943] <= 0.173\nentropy = 0.345\nsamples = 4051\nvalue = [5101.519, 1452
.147]'),
 Text(154.285,116.621,'X[10127] <= 0.063\nentropy = 0.337\nsamples = 3900\nvalue = [5028.322, 1376
.218]'),
 Text(151.911,112.22,'X[2762] <= 0.118\nentropy = 0.325\nsamples = 3617\nvalue = [4837.373,
1243.935]'),
 Text(149.538,107.819,'X[39677] <= 0.053\nentropy = 0.323\nsamples = 3584\nvalue = [4834.19, 1224.
952]'),
 Text(147.164,103.418,'X[12609] <= 0.064\nentropy = 0.311\nsamples = 3319\nvalue = [4630.511, 1105
.72]'),
 Text(144.79,99.0176,'X[32813] <= 0.043\nentropy = 0.307\nsamples = 3272\nvalue = [4617.782, 1080.
212]'),
 Text(142.417,94.6169,'X[37866] <= 0.025\nentropy = 0.305\nsamples = 3251\nvalue = [4617.782, 1067
.755]'),
 Text(140.043,90.2161,'X[43716] <= 0.047\nentropy = 0.301\nsamples = 3206\nvalue = [4605.052, 1043
.434]'),
 Text(137.669,85.8153,'X[16218] <= 0.091\nentropy = 0.296\nsamples = 3136\nvalue = [4573.227, 1007
.842]'),
 Text(135.296,81.4145,'X[35122] <= 0.032\nentropy = 0.293\nsamples = 3107\nvalue = [4570.044, 991.
233]'),
 Text(132.922,77.0137,'X[2903] <= 0.063\nentropy = 0.291\nsamples = 3088\nvalue = [4570.044, 979.9
62]'),
 Text(130.549,72.6129,'X[36593] <= 0.064\nentropy = 0.305\nsamples = 2894\nvalue = [4108.584, 950.
895]'),
 Text(128.175,68.2122,'X[17123] <= 0.148\nentropy = 0.295\nsamples = 2697\nvalue = [3946.278, 864.
288]'),
 Text(125.801,63.8114,'X[18726] <= 0.125\nentropy = 0.292\nsamples = 2676\nvalue = [3946.278, 851.
831]'),
 Text(123.428,59.4106,'X[3391] <= 0.026\nentropy = 0.29\nsamples = 2660\nvalue = [3946.278,
842.34]'),
 Text(121.054,55.0098,'X[4002] <= 0.093\nentropy = 0.287\nsamples = 2638\nvalue = [3943.095, 829.8
83]'),
 Text(118.681,50.609,'X[19188] <= 0.027\nentropy = 0.278\nsamples = 2507\nvalue = [3841.256, 771.1
56]'),
 Text(116.307,46.2082,'X[41798] <= 0.128\nentropy = 0.276\nsamples = 2485\nvalue = [3838.073, 758.
699]'),
 Text(113.933,41.8075,'X[14481] <= 0.091\nentropy = 0.274\nsamples = 2472\nvalue = [3838.073, 750.
988]'),
 Text(111.56,37.4067,'X[43308] <= 0.084\nentropy = 0.27\nsamples = 2443\nvalue = [3828.526,
735.565]'),
 Text(109.186,33.0059,'X[26601] <= 0.078\nentropy = 0.269\nsamples = 2431\nvalue = [3828.526, 728.
446]'),
 Text(106.813,28.6051,'X[29631] <= 0.115\nentropy = 0.261\nsamples = 2338\nvalue = [3755.329, 686.
922]'),
 Text(104.439,24.2043,'X[23607] <= 0.169\nentropy = 0.259\nsamples = 2325\nvalue = [3755.329, 679.
211]'),
 Text(102.065,19.8035,'X[38715] <= 0.286\nentropy = 0.258\nsamples = 2315\nvalue = [3755.329, 673.
279]'),
 Text(99.6917,15.4027,'X[7741] <= 0.132\nentropy = 0.256\nsamples = 2305\nvalue = [3755.329, 667.3
47]'),
 Text(97.3181,11.002,'X[3443] <= 0.011\nentropy = 0.255\nsamples = 2295\nvalue = [3755.329,
661.415]'),
 Text(94.9444,6.60118,'X[12625] <= 0.212\nentropy = 0.252\nsamples = 2274\nvalue = [3748.964, 650.
144]'),
 Text(92.5708,2.20039,'entropy = 0.25\nsamples = 2258\nvalue = [3745.781, 641.246]'),
 Text(97.3181,2.20039,'entropy = 0.388\nsamples = 16\nvalue = [3.182, 8.898]'),
 Text(99.6917,6.60118,'entropy = 0.461\nsamples = 21\nvalue = [6.365, 11.271]'),
 Text(102.065,11.002,'entropy = -0.0\nsamples = 10\nvalue = [0.0, 5.932]'),
 Text(104.439,15.4027,'entropy = -0.0\nsamples = 10\nvalue = [0.0, 5.932]'),
 Text(106.813,19.8035,'entropy = -0.0\nsamples = 10\nvalue = [0.0, 5.932]'),
 Text(109.186,24.2043,'entropy = -0.0\nsamples = 13\nvalue = [0.0, 7.712]'),
 Text(111.56,28.6051,'entropy = 0.462\nsamples = 93\nvalue = [73.197, 41.524]'),
 Text(113.933,33.0059,'entropy = -0.0\nsamples = 12\nvalue = [0.0, 7.118]'),
 Text(116.307,37.4067,'entropy = 0.472\nsamples = 29\nvalue = [9.547, 15.423]'),
 Text(118.681,41.8075,'entropy = -0.0\nsamples = 13\nvalue = [0.0, 7.712]'),
 Text(121.054,46.2082,'entropy = 0.324\nsamples = 22\nvalue = [3.182, 12.457]'),
 Text(123.428,50.609,'entropy = 0.464\nsamples = 131\nvalue = [101.839, 58.727]'),
 Text(125.801,55.0098,'entropy = 0.324\nsamples = 22\nvalue = [3.182, 12.457]'),
 Text(128.175,59.4106,'entropy = -0.0\nsamples = 16\nvalue = [0.0, 9.491]'),
 Text(130.549,63.8114,'entropy = -0.0\nsamples = 21\nvalue = [0.0, 12.457]'),
 Text(132.922,68.2122,'entropy = 0.454\nsamples = 197\nvalue = [162.307, 86.607]'),
 Text(135.296,72.6129,'entropy = 0.111\nsamples = 194\nvalue = [461.46, 29.067]'),
 Text(137.669,77.0137,'entropy = -0.0\nsamples = 19\nvalue = [0.0, 11.271]'),
 Text(140.043,81.4145,'entropy = 0.27\nsamples = 29\nvalue = [3.182, 16.61]'),
 Text(142.417,85.8153,'entropy = 0.498\nsamples = 70\nvalue = [31.825, 35.592]'),
 Text(144.79,90.2161,'entropy = 0.451\nsamples = 45\nvalue = [12.73, 24.321]'),
 Text(147.164,94.6169,'entropy = -0.0\nsamples = 21\nvalue = [0.0, 12.457]'),
```

```
 Text(149.538,99.0176,'entropy = 0.444\nsamples = 47\nvalue = [12.73, 25.507]'),
 Text(151.911,103.418,'entropy = 0.466\nsamples = 265\nvalue = [203.679, 119.233]'),
 Text(154.285,107.819,'entropy = 0.246\nsamples = 33\nvalue = [3.182, 18.982]'),
 Text(156.658,112.22,'entropy = 0.484\nsamples = 283\nvalue = [190.949, 132.283]'),
 Text(159.032,116.621,'entropy = 0.5\nsamples = 151\nvalue = [73.197, 75.929]'),
 Text(161.406,121.022,'entropy = 0.19\nsamples = 46\nvalue = [3.182, 26.694]'),
 Text(163.779,125.422,'entropy = 0.499\nsamples = 225\nvalue = [120.934, 110.928]'),
 Text(166.153,129.823,'entropy = 0.383\nsamples = 82\nvalue = [15.912, 45.676]'),
 Text(173.274,134.224,'X[12625] <= 0.081\nentropy = 0.237\nsamples = 1131\nvalue = [1941.314, 309.
056]'),
 Text(170.9,129.823,'X[44564] <= 0.115\nentropy = 0.225\nsamples = 1091\nvalue = [1934.949,
286.514]'),
 Text(168.526,125.422,'X[18272] <= 0.13\nentropy = 0.211\nsamples = 1026\nvalue = [1887.212,
256.854]'),
 Text(166.153,121.022,'X[39088] <= 0.047\nentropy = 0.202\nsamples = 994\nvalue = [1871.299,
240.838]'),
 Text(163.779,116.621,'X[2762] <= 0.035\nentropy = 0.198\nsamples = 984\nvalue = [1871.299,
234.906]'),
 Text(161.406,112.22,'X[26561] <= 0.068\nentropy = 0.195\nsamples = 975\nvalue = [1871.299,
229.567]'),
 Text(159.032,107.819,'X[18726] <= 0.105\nentropy = 0.191\nsamples = 966\nvalue = [1871.299,
224.229]'),
 Text(156.658,103.418,'X[21842] <= 0.107\nentropy = 0.188\nsamples = 958\nvalue = [1871.299,
219.483]'),
 Text(154.285,99.0176,'X[27854] <= 0.098\nentropy = 0.182\nsamples = 941\nvalue = [1864.935,
210.585]'),
 Text(151.911,94.6169,'X[30081] <= 0.022\nentropy = 0.179\nsamples = 934\nvalue = [1864.935,
206.433]'),
 Text(149.538,90.2161,'X[35114] <= 0.097\nentropy = 0.177\nsamples = 927\nvalue = [1864.935,
202.28]'),
 Text(147.164,85.8153,'X[42878] <= 0.025\nentropy = 0.174\nsamples = 920\nvalue = [1864.935,
198.128]'),
 Text(144.79,81.4145,'X[33217] <= 0.12\nentropy = 0.171\nsamples = 913\nvalue = [1864.935,
193.976]'),
 Text(142.417,77.0137,'X[3656] <= 0.048\nentropy = 0.168\nsamples = 907\nvalue = [1864.935,
190.416]'),
 Text(140.043,72.6129,'X[10127] <= 0.22\nentropy = 0.163\nsamples = 892\nvalue = [1858.57,
182.705]'),
 Text(137.669,68.2122,'X[449] <= 0.077\nentropy = 0.16\nsamples = 886\nvalue = [1858.57,
179.146]'),
 Text(135.296,63.8114,'X[18716] <= 0.024\nentropy = 0.158\nsamples = 880\nvalue = [1858.57,
175.586]'),
 Text(132.922,59.4106,'X[17123] <= 0.071\nentropy = 0.154\nsamples = 869\nvalue = [1855.387,
169.654]'),
 Text(130.549,55.0098,'X[14619] <= 0.039\nentropy = 0.15\nsamples = 859\nvalue = [1852.205,
164.316]'),
 Text(128.175,50.609,'X[22992] <= 0.086\nentropy = 0.147\nsamples = 854\nvalue = [1852.205,
161.35]'),
 Text(125.801,46.2082,'X[45797] <= 0.048\nentropy = 0.145\nsamples = 849\nvalue = [1852.205,
158.384]'),
 Text(123.428,41.8075,'X[5392] <= 0.164\nentropy = 0.143\nsamples = 844\nvalue = [1852.205,
155.418]'),
 Text(121.054,37.4067,'X[29844] <= 0.037\nentropy = 0.141\nsamples = 839\nvalue = [1852.205,
152.452]'),
 Text(118.681,33.0059,'X[24337] <= 0.167\nentropy = 0.138\nsamples = 834\nvalue = [1852.205,
149.486]'),
 Text(116.307,28.6051,'X[640] <= 0.062\nentropy = 0.136\nsamples = 830\nvalue = [1852.205,
147.113]'),
 Text(113.933,24.2043,'X[23692] <= 0.068\nentropy = 0.134\nsamples = 826\nvalue = [1852.205,
144.74]'),
 Text(111.56,19.8035,'X[14539] <= 0.163\nentropy = 0.133\nsamples = 822\nvalue = [1852.205,
142.367]'),
 Text(109.186,15.4027,'X[17379] <= 0.181\nentropy = 0.131\nsamples = 818\nvalue = [1852.205,
139.995]'),
 Text(106.813,11.002,'X[13649] <= 0.168\nentropy = 0.129\nsamples = 814\nvalue = [1852.205,
137.622]'),
 Text(104.439,6.60118,'X[23548] <= 0.08\nentropy = 0.127\nsamples = 810\nvalue = [1852.205,
135.249]'),
 Text(102.065,2.20039,'entropy = 0.125\nsamples = 806\nvalue = [1852.205, 132.876]'),
 Text(106.813,2.20039,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(109.186,6.60118,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(111.56,11.002,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(113.933,15.4027,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(116.307,19.8035,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(118.681,24.2043,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(121.054,28.6051,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(123.428,33.0059,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
```

```
 Text(125.801,37.4067,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(128.175,41.8075,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(130.549,46.2082,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(132.922,50.609,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(135.296,55.0098,'entropy = 0.468\nsamples = 10\nvalue = [3.182, 5.339]'),
 Text(137.669,59.4106,'entropy = 0.454\nsamples = 11\nvalue = [3.182, 5.932]'),
 Text(140.043,63.8114,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(142.417,68.2122,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(144.79,72.6129,'entropy = 0.495\nsamples = 15\nvalue = [6.365, 7.712]'),
 Text(147.164,77.0137,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(149.538,81.4145,'entropy = 0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(151.911,85.8153,'entropy = 0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(154.285,90.2161,'entropy = 0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(156.658,94.6169,'entropy = 0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(159.032,99.0176,'entropy = 0.486\nsamples = 17\nvalue = [6.365, 8.898]'),
 Text(161.406,103.418,'entropy = 0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(163.779,107.819,'entropy = 0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(166.153,112.22,'entropy = 0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(168.526,116.621,'entropy = 0.0\nsamples = 10\nvalue = [0.0, 5.932]'),
 Text(170.9,121.022,'entropy = 0.5\nsamples = 32\nvalue = [15.912, 16.016]'),
 Text(173.274,125.422,'entropy = 0.473\nsamples = 65\nvalue = [47.737, 29.66]'),
 Text(175.647,129.823,'entropy = 0.343\nsamples = 40\nvalue = [6.365, 22.541]'),
 Text(173.274,138.625,'entropy = 0.071\nsamples = 248\nvalue = [655.591, 24.914]'),
 Text(175.647,143.025,'entropy = 0.498\nsamples = 291\nvalue = [162.307, 142.367]'),
 Text(178.021,147.426,'entropy = 0.496\nsamples = 432\nvalue = [251.416, 209.399]'),
 Text(192.263,151.827,'X[2762] <= 0.134\nentropy = 0.231\nsamples = 3252\nvalue = [5671.183, 872.0
]'),
 Text(189.889,147.426,'X[30411] <= 0.017\nentropy = 0.228\nsamples = 3224\nvalue = [5671.183, 855.
39]'),
 Text(187.515,143.025,'X[3391] <= 0.084\nentropy = 0.225\nsamples = 3203\nvalue = [5671.183, 842.9
33]'),
 Text(185.142,138.625,'X[40199] <= 0.051\nentropy = 0.222\nsamples = 3177\nvalue = [5668.001, 828.
103]'),
 Text(182.768,134.224,'X[23597] <= 0.216\nentropy = 0.214\nsamples = 3029\nvalue = [5527.971, 766.
411]'),
 Text(180.394,129.823,'X[13779] <= 0.042\nentropy = 0.212\nsamples = 3009\nvalue = [5524.789, 755.
14]'),
 Text(178.021,125.422,'X[12625] <= 0.039\nentropy = 0.202\nsamples = 2847\nvalue = [5359.3, 689.88
8]'),
 Text(175.647,121.022,'X[21977] <= 0.024\nentropy = 0.196\nsamples = 2759\nvalue = [5282.92, 651.9
24]'),
 Text(173.274,116.621,'X[14952] <= 0.206\nentropy = 0.194\nsamples = 2743\nvalue = [5279.738, 643.
026]'),
 Text(170.9,112.22,'X[45105] <= 0.211\nentropy = 0.192\nsamples = 2733\nvalue = [5279.738,
637.094]'),
 Text(168.526,107.819,'X[12779] <= 0.025\nentropy = 0.19\nsamples = 2718\nvalue = [5276.555,
628.789]'),
 Text(166.153,103.418,'X[38699] <= 0.195\nentropy = 0.187\nsamples = 2688\nvalue = [5260.643, 613.
959]'),
 Text(163.779,99.0176,'X[19188] <= 0.037\nentropy = 0.186\nsamples = 2679\nvalue = [5260.643, 608.
62]'),
 Text(161.406,94.6169,'X[12609] <= 0.105\nentropy = 0.185\nsamples = 2670\nvalue = [5260.643, 603.
282]'),
 Text(159.032,90.2161,'X[40640] <= 0.147\nentropy = 0.183\nsamples = 2661\nvalue = [5260.643, 597.
943]'),
 Text(156.658,85.8153,'X[13859] <= 0.181\nentropy = 0.182\nsamples = 2652\nvalue = [5260.643, 592.
604]'),
 Text(154.285,81.4145,'X[3792] <= 0.076\nentropy = 0.181\nsamples = 2643\nvalue = [5260.643, 587.2
65]'),
 Text(151.911,77.0137,'X[45633] <= 0.152\nentropy = 0.18\nsamples = 2635\nvalue = [5260.643,
582.52]'),
 Text(149.538,72.6129,'X[32358] <= 0.156\nentropy = 0.178\nsamples = 2627\nvalue = [5260.643, 577.
774]'),
 Text(147.164,68.2122,'X[25474] <= 0.129\nentropy = 0.177\nsamples = 2619\nvalue = [5260.643, 573.
029]'),
 Text(144.79,63.8114,'X[43274] <= 0.327\nentropy = 0.176\nsamples = 2611\nvalue = [5260.643, 568.2
83]'),
 Text(142.417,59.4106,'X[25210] <= 0.148\nentropy = 0.174\nsamples = 2598\nvalue = [5257.46, 561.1
65]'),
 Text(140.043,55.0098,'X[44982] <= 0.01\nentropy = 0.173\nsamples = 2591\nvalue = [5257.46,
557.012]'),
 Text(137.669,50.609,'X[32814] <= 0.046\nentropy = 0.172\nsamples = 2584\nvalue = [5257.46, 552.86
]'),
 Text(135.296,46.2082,'X[35366] <= 0.111\nentropy = 0.171\nsamples = 2577\nvalue = [5257.46, 548.7
07]'),
 Text(132.922,41.8075,'X[45829] <= 0.177\nentropy = 0.17\nsamples = 2570\nvalue = [5257.46,
544.555]'),
```

```
 Text(130.549,37.4067,'X[16898] <= 0.255\nentropy = 0.169\nsamples = 2563\nvalue = [5257.46, 540.4
03]'),
 Text(128.175,33.0059,'X[12298] <= 0.057\nentropy = 0.168\nsamples = 2556\nvalue = [5257.46, 536.2
5]'),
 Text(125.801,28.6051,'X[13288] <= 0.146\nentropy = 0.167\nsamples = 2549\nvalue = [5257.46, 532.0
98]'),
 Text(123.428,24.2043,'X[9490] <= 0.028\nentropy = 0.165\nsamples = 2533\nvalue = [5251.096, 523.7
93]'),
 Text(121.054,19.8035,'X[21097] <= 0.105\nentropy = 0.163\nsamples = 2521\nvalue = [5247.913, 517.
268]'),
 Text(118.681,15.4027,'X[35696] <= 0.229\nentropy = 0.162\nsamples = 2506\nvalue = [5241.548, 509.
556]'),
 Text(116.307,11.002,'X[45592] <= 0.216\nentropy = 0.161\nsamples = 2500\nvalue = [5241.548, 505.9
97]'),
 Text(113.933,6.60118,'X[29752] <= 0.177\nentropy = 0.16\nsamples = 2494\nvalue = [5241.548,
502.438]'),
 Text(111.56,2.20039,'entropy = 0.159\nsamples = 2488\nvalue = [5241.548, 498.879]'),
 Text(116.307,2.20039,'entropy = -0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(118.681,6.60118,'entropy = -0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(121.054,11.002,'entropy = -0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(123.428,15.4027,'entropy = 0.495\nsamples = 15\nvalue = [6.365, 7.712]'),
 Text(125.801,19.8035,'entropy = 0.441\nsamples = 12\nvalue = [3.182, 6.525]'),
 Text(128.175,24.2043,'entropy = 0.491\nsamples = 16\nvalue = [6.365, 8.305]'),
 Text(130.549,28.6051,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(132.922,33.0059,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(135.296,37.4067,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(137.669,41.8075,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(140.043,46.2082,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(142.417,50.609,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(144.79,55.0098,'entropy = -0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(147.164,59.4106,'entropy = 0.427\nsamples = 13\nvalue = [3.182, 7.118]'),
 Text(149.538,63.8114,'entropy = -0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(151.911,68.2122,'entropy = -0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(154.285,72.6129,'entropy = -0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(156.658,77.0137,'entropy = -0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(159.032,81.4145,'entropy = -0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(161.406,85.8153,'entropy = -0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(163.779,90.2161,'entropy = -0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(166.153,94.6169,'entropy = -0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(168.526,99.0176,'entropy = -0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(170.9,103.418,'entropy = 0.499\nsamples = 30\nvalue = [15.912, 14.83]'),
 Text(173.274,107.819,'entropy = 0.401\nsamples = 15\nvalue = [3.182, 8.305]'),
 Text(175.647,112.22,'entropy = -0.0\nsamples = 10\nvalue = [0.0, 5.932]'),
 Text(178.021,116.621,'entropy = 0.388\nsamples = 16\nvalue = [3.182, 8.898]'),
 Text(180.394,121.022,'entropy = 0.444\nsamples = 88\nvalue = [76.38, 37.965]'),
 Text(182.768,125.422,'entropy = 0.406\nsamples = 162\nvalue = [165.489, 65.252]'),
 Text(185.142,129.823,'entropy = 0.343\nsamples = 20\nvalue = [3.182, 11.271]'),
 Text(187.515,134.224,'entropy = 0.425\nsamples = 148\nvalue = [140.029, 61.693]'),
 Text(189.889,138.625,'entropy = 0.291\nsamples = 26\nvalue = [3.182, 14.83]'),
 Text(192.263,143.025,'entropy = -0.0\nsamples = 21\nvalue = [0.0, 12.457]'),
 Text(194.636,147.426,'entropy = -0.0\nsamples = 28\nvalue = [0.0, 16.61]'),
 Text(188.702,156.228,'entropy = 0.5\nsamples = 346\nvalue = [175.037, 172.62]'),
 Text(191.076,160.629,'entropy = 0.407\nsamples = 174\nvalue = [38.19, 96.098]'),
 Text(198.197,165.029,'X[13285] <= 0.016\nentropy = 0.499\nsamples = 549\nvalue = [299.153,
269.905]'),
 Text(195.823,160.629,'X[22747] <= 0.064\nentropy = 0.495\nsamples = 505\nvalue = [299.153,
243.804]'),
 Text(193.449,156.228,'entropy = 0.5\nsamples = 475\nvalue = [248.234, 235.499]'),
 Text(198.197,156.228,'entropy = 0.241\nsamples = 30\nvalue = [50.92, 8.305]'),
 Text(200.57,160.629,'entropy = 0.0\nsamples = 44\nvalue = [0.0, 26.101]'),
 Text(198.197,169.43,'entropy = 0.495\nsamples = 419\nvalue = [175.037, 215.924]'),
 Text(200.57,173.831,'entropy = 0.447\nsamples = 312\nvalue = [85.927, 169.061]'),
 Text(202.944,178.232,'entropy = 0.359\nsamples = 278\nvalue = [47.737, 156.011]'),
 Text(226.68,182.633,'X[27139] <= 0.283\nentropy = 0.468\nsamples = 3270\nvalue = [2482.336, 1477.
061]'),
 Text(218.372,178.232,'X[40143] <= 0.093\nentropy = 0.494\nsamples = 2128\nvalue = [1276.175, 1024
.452]'),
 Text(213.625,173.831,'X[11440] <= 0.084\nentropy = 0.5\nsamples = 1604\nvalue = [833.81,
796.071]'),
 Text(211.251,169.43,'X[14617] <= 0.017\nentropy = 0.5\nsamples = 1554\nvalue = [757.431,
780.648]'),
 Text(208.878,165.029,'X[17885] <= 0.134\nentropy = 0.5\nsamples = 1462\nvalue = [751.066,
727.26]'),
 Text(206.504,160.629,'X[14943] <= 0.144\nentropy = 0.5\nsamples = 1443\nvalue = [709.694,
723.701]'),
 Text(204.131,156.228,'X[12625] <= 0.089\nentropy = 0.5\nsamples = 1348\nvalue = [700.146,
669.127]'),
```

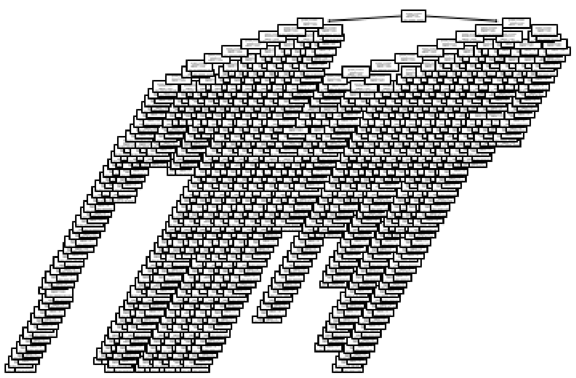
```
 Text(201.757,151.827,'X[25886] <= 0.031\nentropy = 0.499\nsamples = 1285\nvalue = [696.964, 632.3
48]'),
 Text(199.383,147.426,'X[17183] <= 0.396\nentropy = 0.498\nsamples = 1244\nvalue = [696.964, 608.0
27]'),
 Text(197.01,143.025,'X[40533] <= 0.064\nentropy = 0.495\nsamples = 1168\nvalue = [687.416, 564.72
4]'),
 Text(194.636,138.625,'X[18878] <= 0.09\nentropy = 0.497\nsamples = 1157\nvalue = [655.591,
564.131]'),
 Text(192.263,134.224,'X[31572] <= 0.123\nentropy = 0.495\nsamples = 1103\nvalue = [652.409, 532.6
91]'),
 Text(189.889,129.823,'X[44811] <= 0.012\nentropy = 0.5\nsamples = 919\nvalue = [480.555,
455.576]'),
 Text(187.515,125.422,'X[41231] <= 0.095\nentropy = 0.499\nsamples = 878\nvalue = [480.555,
431.254]'),
 Text(185.142,121.022,'X[40199] <= 0.127\nentropy = 0.494\nsamples = 763\nvalue = [455.095,
367.782]'),
 Text(182.768,116.621,'X[26956] <= 0.137\nentropy = 0.491\nsamples = 730\nvalue = [455.095,
348.207]'),
 Text(180.394,112.22,'X[42401] <= 0.079\nentropy = 0.498\nsamples = 668\nvalue = [375.533,
326.258]'),
 Text(178.021,107.819,'X[1830] <= 0.107\nentropy = 0.5\nsamples = 631\nvalue = [324.613,
313.801]'),
 Text(175.647,103.418,'X[13285] <= 0.11\nentropy = 0.499\nsamples = 603\nvalue = [324.613,
297.192]'),
 Text(173.274,99.0176,'X[1135] <= 0.025\nentropy = 0.498\nsamples = 578\nvalue = [324.613,
282.362]'),
 Text(170.9,94.6169,'X[3771] <= 0.093\nentropy = 0.496\nsamples = 556\nvalue = [324.613,
269.312]'),
 Text(168.526,90.2161,'entropy = 0.491\nsamples = 497\nvalue = [311.883, 236.686]'),
 Text(173.274,90.2161,'entropy = 0.404\nsamples = 59\nvalue = [12.73, 32.626]'),
 Text(175.647,94.6169,'entropy = 0.0\nsamples = 22\nvalue = [0.0, 13.05]'),
 Text(178.021,99.0176,'entropy = 0.0\nsamples = 25\nvalue = [0.0, 14.83]'),
 Text(180.394,103.418,'entropy = 0.0\nsamples = 28\nvalue = [0.0, 16.61]'),
 Text(182.768,107.819,'entropy = 0.316\nsamples = 37\nvalue = [50.92, 12.457]'),
 Text(185.142,112.22,'entropy = 0.339\nsamples = 62\nvalue = [79.562, 21.948]'),
 Text(187.515,116.621,'entropy = 0.0\nsamples = 33\nvalue = [0.0, 19.576]'),
 Text(189.889,121.022,'entropy = 0.409\nsamples = 115\nvalue = [25.46, 63.472]'),
 Text(192.263,125.422,'entropy = 0.0\nsamples = 41\nvalue = [0.0, 24.321]'),
 Text(194.636,129.823,'entropy = 0.428\nsamples = 184\nvalue = [171.854, 77.116]'),
 Text(197.01,134.224,'entropy = 0.167\nsamples = 54\nvalue = [3.182, 31.439]'),
 Text(199.383,138.625,'entropy = 0.036\nsamples = 11\nvalue = [31.825, 0.593]'),
 Text(201.757,143.025,'entropy = 0.296\nsamples = 76\nvalue = [9.547, 43.303]'),
 Text(204.131,147.426,'entropy = 0.0\nsamples = 41\nvalue = [0.0, 24.321]'),
 Text(206.504,151.827,'entropy = 0.147\nsamples = 63\nvalue = [3.182, 36.778]'),
 Text(208.878,156.228,'entropy = 0.253\nsamples = 95\nvalue = [9.547, 54.574]'),
 Text(211.251,160.629,'entropy = 0.146\nsamples = 19\nvalue = [41.372, 3.559]'),
 Text(213.625,165.029,'entropy = 0.19\nsamples = 92\nvalue = [6.365, 53.388]'),
 Text(215.999,169.43,'entropy = 0.28\nsamples = 50\nvalue = [76.38, 15.423]'),
 Text(223.119,173.831,'X[13900] <= 0.029\nentropy = 0.449\nsamples = 524\nvalue = [442.365,
228.381]'),
 Text(220.746,169.43,'X[18272] <= 0.02\nentropy = 0.44\nsamples = 501\nvalue = [442.365,
214.737]'),
 Text(218.372,165.029,'entropy = 0.432\nsamples = 482\nvalue = [442.365, 203.467]'),
 Text(223.119,165.029,'entropy = 0.0\nsamples = 19\nvalue = [0.0, 11.271]'),
 Text(225.493,169.43,'entropy = 0.0\nsamples = 23\nvalue = [0.0, 13.644]'),
 Text(234.988,178.232,'X[17183] <= 0.281\nentropy = 0.397\nsamples = 1142\nvalue = [1206.161, 452.
61]'),
 Text(232.614,173.831,'X[41029] <= 0.142\nentropy = 0.364\nsamples = 855\nvalue = [1012.029,
318.547]'),
 Text(230.24,169.43,'X[36700] <= 0.047\nentropy = 0.354\nsamples = 825\nvalue = [1008.847,
301.344]'),
 Text(227.867,165.029,'X[38723] <= 0.078\nentropy = 0.342\nsamples = 786\nvalue = [999.299,
279.989]'),
 Text(225.493,160.629,'X[14617] <= 0.044\nentropy = 0.334\nsamples = 768\nvalue = [999.299,
269.312]'),
 Text(223.119,156.228,'X[38699] <= 0.061\nentropy = 0.324\nsamples = 736\nvalue = [989.752,
252.109]'),
 Text(220.746,151.827,'X[25285] <= 0.065\nentropy = 0.318\nsamples = 724\nvalue = [989.752,
244.99]'),
 Text(218.372,147.426,'X[26601] <= 0.093\nentropy = 0.311\nsamples = 706\nvalue = [986.569,
234.906]'),
 Text(215.999,143.025,'X[34938] <= 0.157\nentropy = 0.301\nsamples = 679\nvalue = [977.022,
220.669]'),
 Text(213.625,138.625,'X[23364] <= 0.11\nentropy = 0.293\nsamples = 663\nvalue = [973.84,
211.771]'),
 Text(211.251,134.224,'X[9380] <= 0.068\nentropy = 0.284\nsamples = 642\nvalue = [967.475,
200.501]'),
```

```
 Text(208.878,129.823,'X[32757] <= 0.169\nentropy = 0.279\nsamples = 633\nvalue = [967.475,
195.162]'),
 Text(206.504,125.422,'X[30038] <= 0.028\nentropy = 0.275\nsamples = 625\nvalue = [967.475,
190.416]'),
 Text(204.131,121.022,'X[33324] <= 0.187\nentropy = 0.27\nsamples = 617\nvalue = [967.475,
185.671]'),
 Text(201.757,116.621,'X[37335] <= 0.042\nentropy = 0.265\nsamples = 609\nvalue = [967.475,
180.925]'),
 Text(199.383,112.22,'X[2675] <= 0.137\nentropy = 0.261\nsamples = 601\nvalue = [967.475,
176.18]'),
 Text(197.01,107.819,'X[2480] <= 0.147\nentropy = 0.256\nsamples = 594\nvalue = [967.475,
172.027]'),
 Text(194.636,103.418,'X[2762] <= 0.16\nentropy = 0.252\nsamples = 587\nvalue = [967.475,
167.875]'),
 Text(192.263,99.0176,'X[27966] <= 0.1\nentropy = 0.248\nsamples = 581\nvalue = [967.475,
164.316]'),
 Text(189.889,94.6169,'X[19291] <= 0.231\nentropy = 0.244\nsamples = 575\nvalue = [967.475,
160.756]'),
 Text(187.515,90.2161,'X[4784] <= 0.023\nentropy = 0.24\nsamples = 569\nvalue = [967.475,
157.197]'),
 Text(185.142,85.8153,'X[16601] <= 0.075\nentropy = 0.237\nsamples = 563\nvalue = [967.475,
153.638]'),
 Text(182.768,81.4145,'X[41463] <= 0.139\nentropy = 0.233\nsamples = 558\nvalue = [967.475,
150.672]'),
 Text(180.394,77.0137,'X[17568] <= 0.044\nentropy = 0.23\nsamples = 553\nvalue = [967.475,
147.706]'),
 Text(178.021,72.6129,'X[7517] <= 0.04\nentropy = 0.226\nsamples = 548\nvalue = [967.475,
144.74]'),
 Text(175.647,68.2122,'X[31391] <= 0.03\nentropy = 0.223\nsamples = 543\nvalue = [967.475,
141.774]'),
 Text(173.274,63.8114,'X[31141] <= 0.161\nentropy = 0.219\nsamples = 538\nvalue = [967.475,
138.808]'),
 Text(170.9,59.4106,'X[25652] <= 0.071\nentropy = 0.216\nsamples = 533\nvalue = [967.475,
135.842]'),
 Text(168.526,55.0098,'X[2980] <= 0.071\nentropy = 0.212\nsamples = 528\nvalue = [967.475,
132.876]'),
 Text(166.153,50.609,'X[35696] <= 0.171\nentropy = 0.209\nsamples = 523\nvalue = [967.475,
129.91]'),
 Text(163.779,46.2082,'X[32358] <= 0.149\nentropy = 0.205\nsamples = 518\nvalue = [967.475,
126.944]'),
 Text(161.406,41.8075,'X[13859] <= 0.082\nentropy = 0.201\nsamples = 513\nvalue = [967.475,
123.978]'),
 Text(159.032,37.4067,'X[45797] <= 0.066\nentropy = 0.195\nsamples = 503\nvalue = [964.292,
118.639]'),
 Text(156.658,33.0059,'entropy = 0.192\nsamples = 499\nvalue = [964.292, 116.267]'),
 Text(161.406,33.0059,'entropy = 0.0\nsamples = 4\nvalue = [0.0, 2.373]'),
 Text(163.779,37.4067,'entropy = 0.468\nsamples = 10\nvalue = [3.182, 5.339]'),
 Text(166.153,41.8075,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(168.526,46.2082,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(170.9,50.609,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(173.274,55.0098,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(175.647,59.4106,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(178.021,63.8114,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(180.394,68.2122,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(182.768,72.6129,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(185.142,77.0137,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(187.515,81.4145,'entropy = 0.0\nsamples = 5\nvalue = [0.0, 2.966]'),
 Text(189.889,85.8153,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(192.263,90.2161,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(194.636,94.6169,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(197.01,99.0176,'entropy = 0.0\nsamples = 6\nvalue = [0.0, 3.559]'),
 Text(199.383,103.418,'entropy = 0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(201.757,107.819,'entropy = 0.0\nsamples = 7\nvalue = [0.0, 4.152]'),
 Text(204.131,112.22,'entropy = 0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(206.504,116.621,'entropy = 0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(208.878,121.022,'entropy = 0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(211.251,125.422,'entropy = 0.0\nsamples = 8\nvalue = [0.0, 4.746]'),
 Text(213.625,129.823,'entropy = 0.0\nsamples = 9\nvalue = [0.0, 5.339]'),
 Text(215.999,134.224,'entropy = 0.461\nsamples = 21\nvalue = [6.365, 11.271]'),
 Text(218.372,138.625,'entropy = 0.388\nsamples = 16\nvalue = [3.182, 8.898]'),
 Text(220.746,143.025,'entropy = 0.481\nsamples = 27\nvalue = [9.547, 14.237]'),
 Text(223.119,147.426,'entropy = 0.365\nsamples = 18\nvalue = [3.182, 10.084]'),
 Text(225.493,151.827,'entropy = 0.0\nsamples = 12\nvalue = [0.0, 7.118]'),
 Text(227.867,156.228,'entropy = 0.459\nsamples = 32\nvalue = [9.547, 17.203]'),
 Text(230.24,160.629,'entropy = 0.0\nsamples = 18\nvalue = [0.0, 10.678]'),
 Text(232.614,165.029,'entropy = 0.427\nsamples = 39\nvalue = [9.547, 21.355]'),
 Text(234.988,169.43,'entropy = 0.263\nsamples = 30\nvalue = [3.182, 17.203]'),
```

```
 Text(237.361,173.831,'entropy = 0.483\nsamples = 287\nvalue = [194.131, 134.063]'),
 Text(249.229,187.033,'X[25745] <= 0.08\nentropy = 0.493\nsamples = 1000\nvalue = [410.54,
516.675]'),
 Text(246.856,182.633,'X[27139] <= 0.253\nentropy = 0.487\nsamples = 974\nvalue = [365.985,
509.556]'),
 Text(244.482,178.232,'X[20365] <= 0.089\nentropy = 0.437\nsamples = 624\nvalue = [162.307,
339.902]'),
 Text(242.108,173.831,'X[40151] <= 0.091\nentropy = 0.418\nsamples = 614\nvalue = [143.212,
337.529]'),
 Text(239.735,169.43,'X[14722] <= 0.104\nentropy = 0.396\nsamples = 600\nvalue = [124.117,
332.784]'),
 Text(237.361,165.029,'X[14696] <= 0.117\nentropy = 0.377\nsamples = 593\nvalue = [111.387,
331.004]'),
 Text(234.988,160.629,'X[2750] <= 0.117\nentropy = 0.36\nsamples = 589\nvalue = [101.839,
330.411]'),
 Text(232.614,156.228,'X[34257] <= 0.12\nentropy = 0.332\nsamples = 572\nvalue = [85.927,
323.293]'),
 Text(230.24,151.827,'X[11442] <= 0.12\nentropy = 0.317\nsamples = 570\nvalue = [79.562,
323.293]'),
 Text(227.867,147.426,'X[24675] <= 0.117\nentropy = 0.301\nsamples = 568\nvalue = [73.197,
323.293]'),
 Text(225.493,143.025,'X[18477] <= 0.043\nentropy = 0.284\nsamples = 566\nvalue = [66.832,
323.293]'),
 Text(223.119,138.625,'X[17137] <= 0.107\nentropy = 0.265\nsamples = 564\nvalue = [60.467,
323.293]'),
 Text(220.746,134.224,'X[34908] <= 0.213\nentropy = 0.237\nsamples = 554\nvalue = [50.92,
319.14]'),
 Text(218.372,129.823,'X[2917] <= 0.288\nentropy = 0.205\nsamples = 544\nvalue = [41.372,
314.988]'),
 Text(215.999,125.422,'X[8905] <= 0.11\nentropy = 0.181\nsamples = 540\nvalue = [35.007,
313.801]'),
 Text(213.625,121.022,'X[22579] <= 0.202\nentropy = 0.141\nsamples = 526\nvalue = [25.46,
307.276]'),
 Text(211.251,116.621,'X[2768] <= 0.116\nentropy = 0.126\nsamples = 525\nvalue = [22.277,
307.276]'),
 Text(208.878,112.22,'X[36352] <= 0.163\nentropy = 0.11\nsamples = 524\nvalue = [19.095,
307.276]'),
 Text(206.504,107.819,'X[15735] <= 0.159\nentropy = 0.094\nsamples = 523\nvalue = [15.912,
307.276]'),
 Text(204.131,103.418,'X[40570] <= 0.101\nentropy = 0.076\nsamples = 522\nvalue = [12.73,
307.276]'),
 Text(201.757,99.0176,'X[24933] <= 0.18\nentropy = 0.058\nsamples = 521\nvalue = [9.547,
307.276]'),
 Text(199.383,94.6169,'X[4784] <= 0.179\nentropy = 0.04\nsamples = 520\nvalue = [6.365,
307.276]'),
 Text(197.01,90.2161,'X[29580] <= 0.232\nentropy = 0.02\nsamples = 519\nvalue = [3.182,
307.276]'),
 Text(194.636,85.8153,'entropy = 0.0\nsamples = 518\nvalue = [0.0, 307.276]'),
 Text(199.383,85.8153,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(201.757,90.2161,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(204.131,94.6169,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(206.504,99.0176,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(208.878,103.418,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(211.251,107.819,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(213.625,112.22,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(215.999,116.621,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(218.372,121.022,'entropy = 0.482\nsamples = 14\nvalue = [9.547, 6.525]'),
 Text(220.746,125.422,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(223.119,129.823,'entropy = 0.422\nsamples = 10\nvalue = [9.547, 4.152]'),
 Text(225.493,134.224,'entropy = 0.422\nsamples = 10\nvalue = [9.547, 4.152]'),
 Text(227.867,138.625,'entropy = -0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(230.24,143.025,'entropy = -0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(232.614,147.426,'entropy = -0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(234.988,151.827,'entropy = -0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(237.361,156.228,'entropy = 0.427\nsamples = 17\nvalue = [15.912, 7.118]'),
 Text(239.735,160.629,'entropy = 0.11\nsamples = 4\nvalue = [9.547, 0.593]'),
 Text(242.108,165.029,'entropy = 0.215\nsamples = 7\nvalue = [12.73, 1.78]'),
 Text(244.482,169.43,'entropy = 0.319\nsamples = 14\nvalue = [19.095, 4.746]'),
 Text(246.856,173.831,'entropy = 0.197\nsamples = 10\nvalue = [19.095, 2.373]'),
 Text(249.229,178.232,'entropy = 0.496\nsamples = 350\nvalue = [203.679, 169.654]'),
 Text(251.603,182.633,'entropy = 0.238\nsamples = 26\nvalue = [44.555, 7.118]'),
 Text(261.097,191.434,'X[16525] <= 0.068\nentropy = 0.485\nsamples = 1095\nvalue = [404.175, 574.2
15]'),
 Text(258.724,187.033,'X[45311] <= 0.17\nentropy = 0.471\nsamples = 1055\nvalue = [343.708,
561.758]'),
 Text(256.35,182.633,'X[7004] <= 0.122\nentropy = 0.44\nsamples = 940\nvalue = [248.234,
511.336]'),
```

```
 Text(253.976,178.232,'X[27139] <= 0.305\nentropy = 0.424\nsamples = 927\nvalue = [222.774,
508.37]'),
 Text(251.603,173.831,'X[25745] <= 0.124\nentropy = 0.359\nsamples = 723\nvalue = [124.117,
405.747]'),
 Text(249.229,169.43,'X[28047] <= 0.08\nentropy = 0.328\nsamples = 711\nvalue = [105.022,
402.188]'),
 Text(246.856,165.029,'X[19359] <= 0.102\nentropy = 0.304\nsamples = 704\nvalue = [92.292,
400.408]'),
 Text(244.482,160.629,'X[33923] <= 0.026\nentropy = 0.272\nsamples = 687\nvalue = [76.38,
393.29]'),
 Text(242.108,156.228,'X[41087] <= 0.107\nentropy = 0.249\nsamples = 683\nvalue = [66.832,
392.697]'),
 Text(239.735,151.827,'X[42323] <= 0.076\nentropy = 0.223\nsamples = 678\nvalue = [57.285,
391.51]'),
 Text(237.361,147.426,'X[14013] <= 0.066\nentropy = 0.195\nsamples = 671\nvalue = [47.737,
389.137]'),
 Text(234.988,143.025,'X[27948] <= 0.106\nentropy = 0.174\nsamples = 669\nvalue = [41.372,
389.137]'),
 Text(232.614,138.625,'X[10269] <= 0.082\nentropy = 0.142\nsamples = 656\nvalue = [31.825,
383.205]'),
 Text(230.24,134.224,'X[3850] <= 0.19\nentropy = 0.117\nsamples = 652\nvalue = [25.46, 382.019]'),
 Text(227.867,129.823,'X[17287] <= 0.204\nentropy = 0.091\nsamples = 645\nvalue = [19.095,
379.053]'),
 Text(225.493,125.422,'X[439] <= 0.188\nentropy = 0.064\nsamples = 635\nvalue = [12.73,
374.307]'),
 Text(223.119,121.022,'X[37628] <= 0.07\nentropy = 0.049\nsamples = 634\nvalue = [9.547,
374.307]'),
 Text(220.746,116.621,'X[23847] <= 0.112\nentropy = 0.033\nsamples = 633\nvalue = [6.365,
374.307]'),
 Text(218.372,112.22,'X[8090] <= 0.105\nentropy = 0.017\nsamples = 632\nvalue = [3.182,
374.307]'),
 Text(215.999,107.819,'entropy = 0.0\nsamples = 631\nvalue = [0.0, 374.307]'),
 Text(220.746,107.819,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(223.119,112.22,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(225.493,116.621,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(227.867,121.022,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(230.24,125.422,'entropy = 0.489\nsamples = 10\nvalue = [6.365, 4.746]'),
 Text(232.614,129.823,'entropy = 0.434\nsamples = 7\nvalue = [6.365, 2.966]'),
 Text(234.988,134.224,'entropy = 0.265\nsamples = 4\nvalue = [6.365, 1.186]'),
 Text(237.361,138.625,'entropy = 0.473\nsamples = 13\nvalue = [9.547, 5.932]'),
 Text(239.735,143.025,'entropy = -0.0\nsamples = 2\nvalue = [6.365, 0.0]'),
 Text(242.108,147.426,'entropy = 0.319\nsamples = 7\nvalue = [9.547, 2.373]'),
 Text(244.482,151.827,'entropy = 0.197\nsamples = 5\nvalue = [9.547, 1.186]'),
 Text(246.856,156.228,'entropy = 0.11\nsamples = 4\nvalue = [9.547, 0.593]'),
 Text(249.229,160.629,'entropy = 0.427\nsamples = 17\nvalue = [15.912, 7.118]'),
 Text(251.603,165.029,'entropy = 0.215\nsamples = 7\nvalue = [12.73, 1.78]'),
 Text(253.976,169.43,'entropy = 0.265\nsamples = 12\nvalue = [19.095, 3.559]'),
 Text(256.35,173.831,'entropy = 0.5\nsamples = 204\nvalue = [98.657, 102.623]'),
 Text(258.724,178.232,'entropy = 0.187\nsamples = 13\nvalue = [25.46, 2.966]'),
 Text(261.097,182.633,'entropy = 0.452\nsamples = 115\nvalue = [95.474, 50.422]'),
 Text(263.471,187.033,'entropy = 0.283\nsamples = 40\nvalue = [60.467, 12.457]'),
 Text(272.965,195.835,'X[41032] <= 0.055\nentropy = 0.5\nsamples = 2348\nvalue = [1139.329,
1180.463]'),
 Text(270.592,191.434,'X[40143] <= 0.095\nentropy = 0.497\nsamples = 2227\nvalue = [967.475, 1140.
718]'),
 Text(268.218,187.033,'X[19359] <= 0.124\nentropy = 0.482\nsamples = 1797\nvalue = [646.044, 945.5
56]'),
 Text(265.844,182.633,'X[2768] <= 0.096\nentropy = 0.473\nsamples = 1734\nvalue = [572.847, 921.82
9]'),
 Text(263.471,178.232,'X[44327] <= 0.034\nentropy = 0.466\nsamples = 1721\nvalue = [541.022, 920.0
49]'),
 Text(261.097,173.831,'X[31572] <= 0.091\nentropy = 0.459\nsamples = 1707\nvalue = [509.197, 917.6
76]'),
 Text(258.724,169.43,'X[40151] <= 0.137\nentropy = 0.431\nsamples = 1438\nvalue = [359.62, 785.986
]'),
 Text(256.35,165.029,'X[38261] <= 0.111\nentropy = 0.42\nsamples = 1421\nvalue = [334.161,
780.648]'),
 Text(253.976,160.629,'X[7004] <= 0.162\nentropy = 0.41\nsamples = 1412\nvalue = [315.066,
778.868]'),
 Text(251.603,156.228,'X[39677] <= 0.209\nentropy = 0.403\nsamples = 1408\nvalue = [302.336, 778.8
68]'),
 Text(249.229,151.827,'X[27835] <= 0.126\nentropy = 0.387\nsamples = 1366\nvalue = [270.511, 759.8
86]'),
 Text(246.856,147.426,'X[44230] <= 0.161\nentropy = 0.375\nsamples = 1352\nvalue = [251.416, 755.1
4]'),
 Text(244.482,143.025,'X[38894] <= 0.29\nentropy = 0.365\nsamples = 1346\nvalue = [238.686,
753.954]'),
```

```
 Text(242.108,138.625,'X[7296] <= 0.164\nentropy = 0.358\nsamples = 1343\nvalue = [229.139, 753.95
4]'),
 Text(239.735,134.224,'X[34845] <= 0.133\nentropy = 0.349\nsamples = 1340\nvalue = [219.591, 753.9
54]'),
 Text(237.361,129.823,'X[35696] <= 0.144\nentropy = 0.339\nsamples = 1331\nvalue = [206.861, 750.9
88]'),
 Text(234.988,125.422,'X[22370] <= 0.062\nentropy = 0.325\nsamples = 1315\nvalue = [190.949, 744.4
63]'),
 Text(232.614,121.022,'X[20569] <= 0.132\nentropy = 0.315\nsamples = 1310\nvalue = [181.401, 743.2
76]'),
 Text(230.24,116.621,'X[37836] <= 0.036\nentropy = 0.3\nsamples = 1289\nvalue = [165.489,
733.785]'),
 Text(227.867,112.22,'X[32887] <= 0.129\nentropy = 0.29\nsamples = 1283\nvalue = [155.942,
732.005]'),
 Text(225.493,107.819,'X[24575] <= 0.145\nentropy = 0.275\nsamples = 1268\nvalue = [143.212, 725.4
8]'),
 Text(223.119,103.418,'X[27139] <= 0.26\nentropy = 0.263\nsamples = 1260\nvalue = [133.664,
722.514]'),
 Text(220.746,99.0176,'X[16599] <= 0.202\nentropy = 0.173\nsamples = 828\nvalue = [50.92,
481.676]'),
 Text(218.372,94.6169,'X[15681] <= 0.175\nentropy = 0.155\nsamples = 825\nvalue = [44.555,
481.083]'),
 Text(215.999,90.2161,'X[21324] <= 0.129\nentropy = 0.136\nsamples = 822\nvalue = [38.19,
480.49]'),
 Text(213.625,85.8153,'X[24675] <= 0.137\nentropy = 0.117\nsamples = 818\nvalue = [31.825,
479.303]'),
 Text(211.251,81.4145,'X[23855] <= 0.133\nentropy = 0.096\nsamples = 814\nvalue = [25.46,
478.117]'),
 Text(208.878,77.0137,'X[30085] <= 0.13\nentropy = 0.074\nsamples = 807\nvalue = [19.095,
475.151]'),
 Text(206.504,72.6129,'X[7683] <= 0.154\nentropy = 0.051\nsamples = 796\nvalue = [12.73,
469.812]'),
 Text(204.131,68.2122,'X[9925] <= 0.077\nentropy = 0.039\nsamples = 795\nvalue = [9.547,
469.812]'),
 Text(201.757,63.8114,'X[19826] <= 0.1\nentropy = 0.026\nsamples = 794\nvalue = [6.365,
469.812]'),
 Text(199.383,59.4106,'X[7670] <= 0.209\nentropy = 0.013\nsamples = 793\nvalue = [3.182,
469.812]'),
 Text(197.01,55.0098,'entropy = 0.0\nsamples = 792\nvalue = [0.0, 469.812]'),
 Text(201.757,55.0098,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(204.131,59.4106,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(206.504,63.8114,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(208.878,68.2122,'entropy = -0.0\nsamples = 1\nvalue = [3.182, 0.0]'),
 Text(211.251,72.6129,'entropy = 0.496\nsamples = 11\nvalue = [6.365, 5.339]'),
 ...]
```



In [92]:

```
# I am not able to make sense of the tree . Hence I am exporting
```

In [95]:

```
a3
```

Out[95]:

```
<61441x46009 sparse matrix of type '<class 'numpy.float64'>'
 with 2063308 stored elements in COOrdinate format>
```

In [119]:

```
from sklearn.tree import export_graphviz
target = ['negative','positive']
export_graphviz(om_bow,out_file='bow_dt.dot.',class_names=target,rounded = True, proportion = False
,max_depth=3,feature_names=features)
```

In [121]:

```
from graphviz import Source
from sklearn import tree
```

FEATURE IMPORTANCE FOR BOW

In [97]:

```
om_bow.get_params
```

Out[97]:

```
<bound method BaseEstimator.get_params of DecisionTreeClassifier(class_weight='balanced',
criterion='gini', max_depth=50,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')>
```

In [98]:

```
count_vect.get_params
```

Out[98]:

```
<bound method BaseEstimator.get_params of CountVectorizer(analyzer='word', binary=False,
decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)>
```

In [112]:

```
features = count_vect.get_feature_names()
```

In [118]:

```
#features=np.argsort(features)[::-1]
```

In [114]:

```
feat_importances = om_bow.feature_importances_
```

In [115]:

```
len(features),len(feat_importances)
features.append('zzzzzzzzza')
```

In [116]:

```
len(features)
```

Out[116]:

```
46009
```

In [117]:

```
cf = pd.DataFrame({'Word' : features, 'Coefficient' : feat_importances})
cf_new = cf.sort_values("Coefficient", ascending = False)
print('***** Top 20 IMPORTANT FEATURES  *****')
print('\n')
print(cf_new.head(20))
#print('\n')
#print('***** Top 10 IMPORTANT FEATURES FOR NEGATIVE CLASS *****')
#print('\n')
#print(cf_new.tail(10))
```

```
***** Top 20 IMPORTANT FEATURES  *****


              Word  Coefficient
27139          not     0.140460
17569        great     0.089926
10560    delicious     0.040452
3744          best     0.040385
23595         love     0.029933
29622      perfect     0.024847
17183         good     0.024308
23609        loves     0.024093
11440  disappointed    0.021146
26852         nice     0.016022
13900    excellent     0.014375
14617     favorite     0.013908
18895       highly     0.013534
45177    wonderful     0.012340
2903           bad     0.012234
38261        stale     0.010604
41032      thought     0.009971
25745        money     0.009113
12625         easy     0.009022
2768         awful     0.007347
```

Observations :

1) We have found that not and great are the top 2 words that are having the highest influence.

PERFORMANCE MEASURMENTS FOR BOW (DECISION TREE)

In [125]:

```
precision_bow = precision_score(y_test1, ompredictions_bow, pos_label = 1)
recall_bow = recall_score(y_test1, ompredictions_bow, pos_label = 1)
f1score_bow = f1_score(y_test1, ompredictions_bow, pos_label = 1)
```

In [126]:

```
print('\nThe Test Precision for optimal depth and split values for Decision Tree (BOW)   is %f' %
(precision_bow))
print('\nThe Test Recall for optimal depth and split values for Decision Tree (BOW)  is %f' % (rec
all_bow))
print('\nThe Test F1-Scorefor optimal depth and split values for Decision Tree (BOW)  is %f' %  (f
1score_bow))
```

```
The Test Precision for optimal depth and split values for Decision Tree (BOW)   is 0.939356

The Test Recall for optimal depth and split values for Decision Tree (BOW)  is 0.693359

The Test F1-Scorefor optimal depth and split values for Decision Tree (BOW)  is 0.797826
```

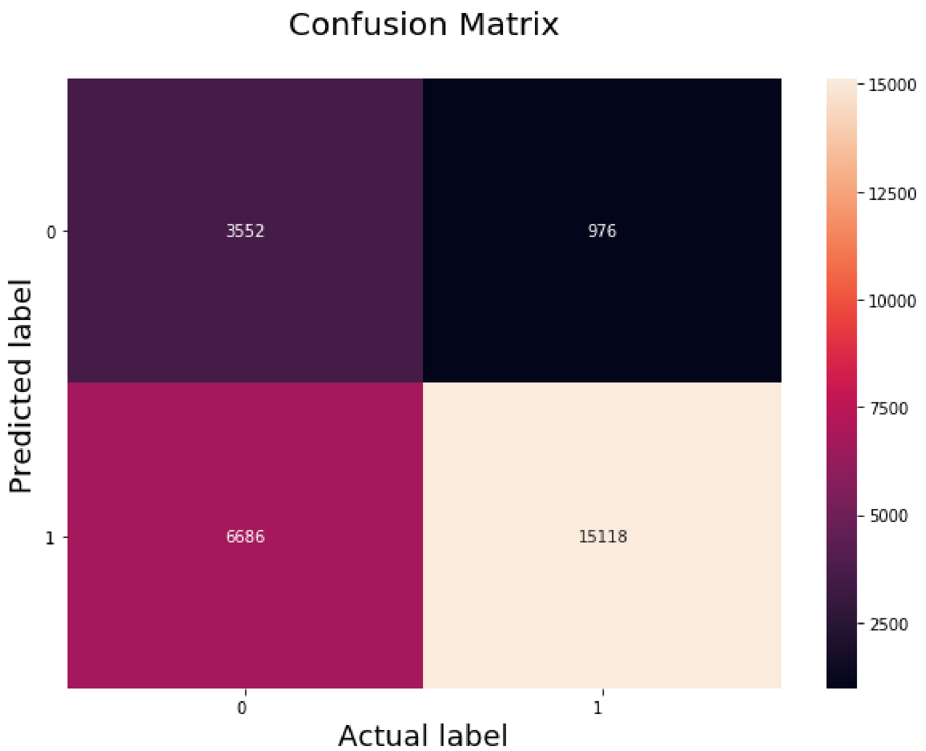
CONFUSION MATRIX

In [132]:

```
# Reference Links
# https://datatofish.com/confusion-matrix-python/
```

In [127]:

```
# Code for drawing seaborn heatmaps
class_names = [ 0,1]
df_heatmap = pd.DataFrame(confusion_matrix(y_test1, ompredictions_bow), index=class_names, columns
=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)#
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)
plt.ylabel('Predicted label',size=18)
plt.xlabel('Actual label',size=18)
plt.title("Confusion Matrix\n",size=20)
plt.show()
```



In [128]:

```
TrueNeg,FalseNeg,FalsePos, TruePos = confusion_matrix(y_test1, ompredictions_bow).ravel()
TPR = TruePos/(FalseNeg + TruePos)
FPR = FalsePos/(TrueNeg + FalsePos)
TNR = TrueNeg/(TrueNeg + FalsePos)
FNR = FalseNeg/(FalseNeg + TruePos)
```

In [129]:

```
print("TPR of the Decision Tree  (BOW)   is :  %f" % (TPR))
print("FPR of the Decision Tree  (BOW)   is :  %f" % (FPR))
print("TNR of the Decision Tree (BOW)   is :  %f" % (TNR))
print("FNR of the Decision Tree  (BOW)  is :  %f" % (FNR))
```

```
TPR of the Decision Tree  (BOW)   is :  0.939356
FPR of the Decision Tree  (BOW)   is :  0.653057
TNR of the Decision Tree (BOW)   is :  0.346943
FNR of the Decision Tree  (BOW)  is :  0.060644
```

FNR of the Decision Tree (BOW) is : 0.060644

PLOTTING THE ROC CURVE (BOW) ---- > FOR BOTH TRAIN AND TEST DATA

In [130]:

```
len(y_train1)
```

Out[130]:

```
61441
```

In [131]:

```
len(probs1)
```

Out[131]:

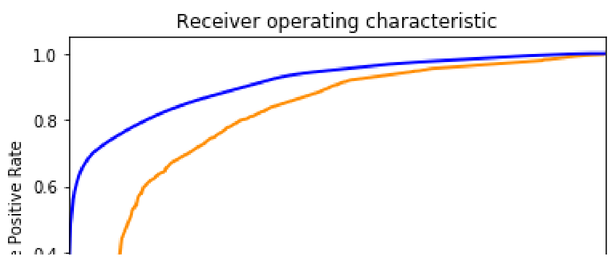
```
61441
```

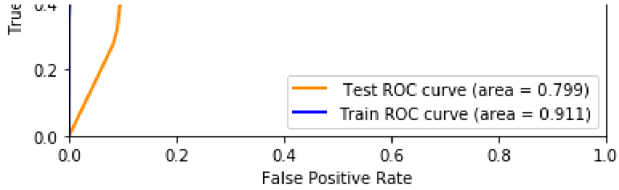
In [132]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()

fpr1 = dict()
tpr1 = dict()
roc_auc1 = dict()

#for i in range(26331):
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test1,probs)
    roc_auc[i] = auc(fpr[i], tpr[i])

#for i in range(61441):
for i in range(4):
    fpr1[i], tpr1[i], _ = roc_curve(y_train1,probs1)
    roc_auc1[i] = auc(fpr1[i], tpr1[i])
```

In [133]:

```
#print(roc_auc_score(y_test1,ompredictions_bow))
plt.figure()
#plt.plot(fpr[1], tpr[1])
lw = 2
plt.plot(fpr[0], tpr[0], color='darkorange',lw=lw, label=' Test ROC curve (area = %0.3f)' % roc_auc[0])
plt.plot(fpr1[0], tpr1[0], color='blue',lw=lw, label='Train ROC curve (area = %0.3f)' % roc_auc1[0])
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.title('Receiver operating characteristic')
plt.show()
```





Observations

1) We observe that AUC for train data is 0.91 even though the test data is 0.79. This implied that the model is overfitted.

TFIDF WITH FEATURE ENGINEERING

In [134]:

```
tf_idf_vect = TfidfVectorizer(min_df=10)
c1 = tf_idf_vect.fit_transform(X_trainbow['Cleaned Text'].values)
d1 = tf_idf_vect.transform(X_testbow['Cleaned Text'])
print("the type of count vectorizer :",type(c1))
print("the shape of out text TFIDF vectorizer : ",c1.get_shape())
print("the number of unique words :", c1.get_shape()[1])
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (61441, 9723)
the number of unique words : 9723
```

In [135]:

```
c1 = preprocessing.normalize(c1)
```

In [136]:

```
c2 = sparse.csr_matrix(X_train1['Length'].values)
c2 = preprocessing.normalize(c2)
```

In [137]:

```
c1
```

Out[137]:

```
<61441x9723 sparse matrix of type '<class 'numpy.float64'>'
 with 1925265 stored elements in Compressed Sparse Row format>
```

In [138]:

```
c2.T
```

Out[138]:

```
<61441x1 sparse matrix of type '<class 'numpy.float64'>'
 with 61271 stored elements in Compressed Sparse Column format>
```

In [139]:

```
c3 = sparse.hstack([c1, c2.T])
```

In [140]:

```
d1 = preprocessing.normalize(d1)
d2 = sparse.csr_matrix(X_test1['Length'].values)
d2 = preprocessing.normalize(d2)
d3 = sparse.hstack([d1, d2.T])
```

Decision Tree - TFIDF

In [142]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model_tfidf = GridSearchCV(DecisionTreeClassifier(max_features="log2",class_weight = 'balanced'),
tree_para, scoring = 'roc_auc', cv=5, return_train_score= True)
model_tfidf.fit(c3, y_train1)
print(model_tfidf.best_estimator_)
print(model_tfidf.score(d3, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
                       max_depth=1000, max_features='log2', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
0.698775066112614
```

In [144]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model1_tfidf = GridSearchCV(DecisionTreeClassifier(class_weight = 'balanced'), tree_para, scoring
= 'roc_auc', cv=5, return_train_score= True)
model1_tfidf.fit(c3, y_train1)
print(model1_tfidf.best_estimator_)
print(model1_tfidf.score(d3, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=50,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
0.7890400242231952
```

Observations :

1) We found that the accuracy has enhanced when we used all features . However computation time is more in this case.

OPTIMAL ALPHA FOR TFIDF - THROUGH PLOTTING APPROACH

In [182]:

```
alph_depth = [1,1,1,1,5,5,5,5,10,10,10,10,50,50,50,50,100,100,100,100,500,500,500,500,1000,1000,1000
,1000]
alph_split = [5,10,100,500]*7
```

In [183]:

```
model1_tfidf.cv_results_
```

Out[183]:

```
{'mean_fit_time': array([ 0.19246554,  0.195573  ,  0.20659676,  0.19630585,  0.87291446,
         0.85693283,  0.84818764,  0.83197412,  2.28739405,  2.19863219,
         1.8935554 ,  1.78212557, 13.81681061, 12.98860955, 10.56166449,
         7.01649137, 17.25311985, 16.61338453, 14.23996224,  8.92628307,
        24.87175694, 23.92781   , 21.08874831, 15.29480433, 23.11316381,
        22.61035819, 19.76628156, 15.70002403]),
 'std_fit_time': array([0.0057352 , 0.01443049, 0.00586341, 0.01227734, 0.03220714,
        0.0185391 , 0.01418578, 0.02134509, 0.04734477, 0.05749157,
        0.02101704, 0.06127148, 0.4814207 , 0.20046402, 0.22480554,
        0.216491  , 0.38337132, 0.47797433, 0.432881  , 0.45170812,
        0.62076266, 0.95960972, 1.03346876, 0.99033401, 0.64358931,
```

```
       0.78540116, 0.56460919, 1.57292037]),
 'mean_score_time': array([0.01374974, 0.01396151, 0.00751915, 0.01257119, 0.01176481,
        0.01231503, 0.01176686, 0.00844193, 0.01418328, 0.01196971,
        0.00937085, 0.01250205, 0.01429033, 0.01176858, 0.01508913,
        0.01476297, 0.00923882, 0.01528497, 0.01562486, 0.00937328,
        0.01548734, 0.0125061 , 0.01562066, 0.00937095, 0.0096776 ,
        0.01534457, 0.01250544, 0.01187878]),
 'std_score_time': array([7.28999866e-03, 2.05741645e-03, 6.36827443e-03, 6.28686869e-03,
        6.05021657e-03, 6.16748856e-03, 6.05137857e-03, 7.09904150e-03,
        1.85998713e-03, 6.07071983e-03, 7.65126828e-03, 6.25104167e-03,
        9.23870622e-04, 6.05491385e-03, 1.04689946e-03, 1.10992761e-03,
        7.54735822e-03, 6.70173437e-04, 1.17234079e-05, 7.65325449e-03,
        2.49709044e-04, 6.25306342e-03, 1.96167353e-05, 7.65134641e-03,
        5.75354922e-03, 3.50796409e-03, 2.93626427e-03, 1.80974777e-03]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50,
                    100, 100, 100, 100, 500, 500, 500, 500, 1000, 1000,
                    1000, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500,
5,
                    10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10,
                    100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 500},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 500},
  {'max_depth': 1000, 'min_samples_split': 5},
  {'max_depth': 1000, 'min_samples_split': 10},
  {'max_depth': 1000, 'min_samples_split': 100},
  {'max_depth': 1000, 'min_samples_split': 500}],
 'split0_test_score': array([0.6344704 , 0.6344704 , 0.6344704 , 0.6344704 , 0.74248159,
        0.74248159, 0.74239932, 0.74239962, 0.77861964, 0.78071638,
        0.78183638, 0.78535456, 0.71500907, 0.72356729, 0.76923563,
        0.80570576, 0.7173176 , 0.7257107 , 0.76317587, 0.79472817,
        0.69396854, 0.70319544, 0.72974796, 0.76371481, 0.69938096,
        0.70447705, 0.7376502 , 0.76660007]),
 'split1_test_score': array([0.61832067, 0.61832067, 0.61832067, 0.61832067, 0.72021626,
        0.72021626, 0.72045509, 0.72179638, 0.7604717 , 0.76105066,
        0.765846  , 0.7688843 , 0.71117694, 0.71838795, 0.77165884,
        0.80221834, 0.71777487, 0.72739847, 0.77076485, 0.794711  ,
        0.69101943, 0.70397281, 0.72733327, 0.75363326, 0.69185582,
        0.69810474, 0.73309702, 0.75609373]),
 'split2_test_score': array([0.63139012, 0.63139012, 0.63139012, 0.63139012, 0.73027043,
        0.73027043, 0.73062816, 0.73315482, 0.76580962, 0.76738177,
        0.77356802, 0.77811468, 0.72330763, 0.7308136 , 0.7647024 ,
        0.7992643 , 0.70710238, 0.71319459, 0.75128827, 0.78562756,
```

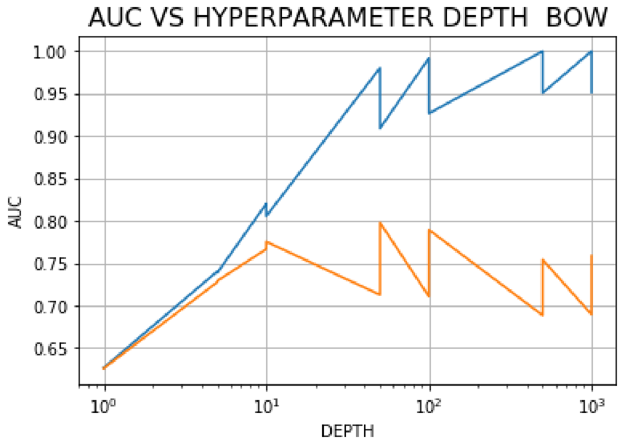
```
       0.68899016, 0.69269194, 0.7245832 , 0.75237392, 0.69477488,
       0.69687497, 0.72406218, 0.76045067]),
 'split3_test_score': array([0.62641364, 0.62641364, 0.62641364, 0.62641364, 0.72789868,
       0.72789868, 0.72850239, 0.7314886 , 0.77108946, 0.77035468,
       0.7733794 , 0.77816105, 0.69963485, 0.70868052, 0.7549997 ,
       0.79697286, 0.70447896, 0.71635271, 0.75330997, 0.79317765,
       0.68442357, 0.68931398, 0.72440961, 0.76238813, 0.68146772,
       0.68745798, 0.73008768, 0.76102889]),
 'split4_test_score': array([0.61934616, 0.61934616, 0.61934616, 0.61934616, 0.71736767,
       0.71736767, 0.71717957, 0.71742833, 0.7566339 , 0.75692178,
       0.75667692, 0.76573507, 0.71436189, 0.71957886, 0.75688076,
       0.7854472 , 0.70616721, 0.70537093, 0.74923621, 0.77830585,
       0.68431551, 0.68521753, 0.72478494, 0.73856679, 0.68031994,
       0.68800571, 0.72020655, 0.74856721]),
 'mean_test_score': array([0.6259884 , 0.6259884 , 0.6259884 , 0.6259884 , 0.72764725,
       0.72764725, 0.72783323, 0.72925386, 0.76652504, 0.76728529,
       0.77026168, 0.77525015, 0.71269845, 0.72020604, 0.76349596,
       0.79792213, 0.71056855, 0.71760592, 0.75755544, 0.78931028,
       0.68854371, 0.69487884, 0.7261719 , 0.75413562, 0.68956043,
       0.69498456, 0.72902098, 0.75854836]),
 'std_test_score': array([0.00639098, 0.00639098, 0.00639098, 0.00639098, 0.00880794,
       0.00880794, 0.00881334, 0.00881455, 0.00777307, 0.00819879,
       0.00846949, 0.00706879, 0.00766321, 0.00721595, 0.00658748,
       0.00688997, 0.0057611 , 0.00815235, 0.00815594, 0.00645181,
       0.00375791, 0.00749607, 0.00208283, 0.0090065 , 0.0074803 ,
       0.00646168, 0.00623697, 0.00600408]),
 'rank_test_score': array([25, 25, 25, 25, 14, 14, 13, 11,  6,  5,  4,  3, 19, 17,  7,  1, 20,
       18,  9,  2, 24, 22, 16, 10, 23, 21, 12,  8]),
 'split0_train_score': array([0.62465503, 0.62465503, 0.62465503, 0.62465503, 0.73849001,
       0.73846588, 0.73797607, 0.73762778, 0.81647917, 0.81499752,
       0.80619969, 0.80382044, 0.97924974, 0.97282043, 0.94410523,
       0.91111782, 0.99195075, 0.98771482, 0.96427441, 0.92765697,
       0.99990018, 0.99898847, 0.98332812, 0.95165075, 0.99989612,
       0.99902033, 0.98421522, 0.95107   ]),
 'split1_train_score': array([0.6283462 , 0.6283462 , 0.6283462 , 0.6283462 , 0.74135079,
       0.74135079, 0.74067044, 0.73901935, 0.81856764, 0.81810642,
       0.80994568, 0.8030658 , 0.98013029, 0.97461319, 0.94370622,
       0.90765334, 0.99069148, 0.98667446, 0.96036205, 0.92765228,
       0.99989909, 0.99893055, 0.9834167 , 0.95083277, 0.99989919,
       0.99891947, 0.98290772, 0.95468662]),
 'split2_train_score': array([0.62545502, 0.62545502, 0.62545502, 0.62545502, 0.73983665,
       0.73983665, 0.73926325, 0.73856382, 0.8211086 , 0.82029097,
       0.81158376, 0.8069031 , 0.98119692, 0.97636515, 0.94900379,
       0.90442   , 0.99341158, 0.99030372, 0.96796645, 0.92213343,
       0.99990604, 0.99910743, 0.98480817, 0.94453156, 0.99990384,
       0.99905242, 0.98495835, 0.94462176]),
 'split3_train_score': array([0.62670423, 0.62670423, 0.62670423, 0.62670423, 0.7403446 ,
       0.7403446 , 0.73985656, 0.73814718, 0.82225512, 0.82116241,
       0.81264153, 0.80703569, 0.98184068, 0.97542483, 0.94715298,
       0.90982202, 0.99239514, 0.98826168, 0.96518859, 0.92792347,
       0.99988819, 0.99892529, 0.98369175, 0.95152349, 0.99989855,
       0.9989023 , 0.98385711, 0.95194054]),
 'split4_train_score': array([0.62808921, 0.62808921, 0.62808921, 0.62808921, 0.74178904,
       0.74178904, 0.74148481, 0.74059895, 0.82229407, 0.82138737,
       0.81389305, 0.80626422, 0.97841341, 0.97264122, 0.94623127,
       0.91113755, 0.99049209, 0.9866569 , 0.96368989, 0.92716842,
       0.99988797, 0.99897175, 0.98535704, 0.95408047, 0.99988253,
       0.99897896, 0.98529596, 0.95401866]),
 'mean_train_score': array([0.62664994, 0.62664994, 0.62664994, 0.62664994, 0.74036222,
       0.74035739, 0.73985023, 0.73879142, 0.82014092, 0.81918894,
       0.81085274, 0.80541785, 0.98016621, 0.97437297, 0.9460399 ,
       0.90883015, 0.99178821, 0.98792232, 0.96429628, 0.92650691,
       0.99989629, 0.9989847 , 0.98412036, 0.95052381, 0.99989605,
       0.9989747 , 0.98424687, 0.95126752]),
 'std_train_score': array([1.43939068e-03, 1.43939068e-03, 1.43939068e-03, 1.43939068e-03,
       1.16574277e-03, 1.17350678e-03, 1.20013928e-03, 1.01390695e-03,
       2.27710213e-03, 2.39507725e-03, 2.66275222e-03, 1.65066992e-03,
       1.24737336e-03, 1.45204848e-03, 1.96210382e-03, 2.54462749e-03,
       1.08746176e-03, 1.34072528e-03, 2.45408727e-03, 2.20027643e-03,
       7.11228927e-06, 6.58862330e-05, 8.13506779e-04, 3.19129393e-03,
       7.20549799e-06, 5.73263112e-05, 8.42975156e-04, 3.57539049e-03])}
```

In [185]:

```
train_auc_tfidf = model1_tfidf.cv_results_['mean_train_score']
cv_auc_tfidf = model1_tfidf.cv_results_['mean_test_score']
```

In [187]:

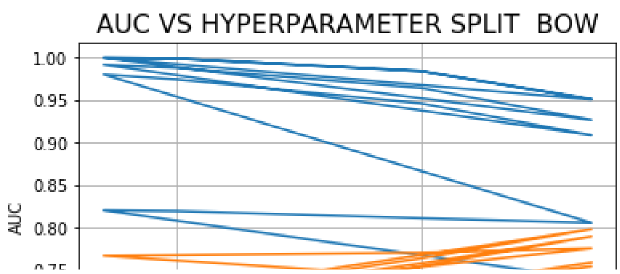
```
plt.plot(alph_depth,train_auc_tfidf)
plt.plot(alph_depth,cv_auc_tfidf)
plt.xlabel('DEPTH',size=10)
plt.ylabel('AUC',size=10)
plt.title('AUC VS HYPERPARAMETER DEPTH  BOW',size=16)
plt.xscale('log')
plt.grid()
plt.show()
print("\n\n Depth Values :\n", alph_depth)
print("\n Train AUC for each  value is :\n ", np.round(train_auc_tfidf,5))
print("\n CV AUC for each value is :\n ", np.round(cv_auc_tfidf,5))
```

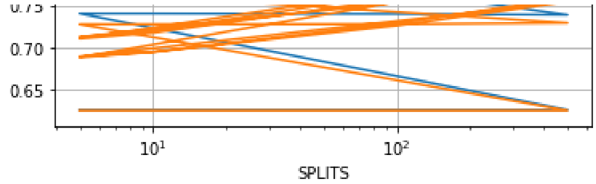



```
 Depth Values :
 [1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50, 100, 100, 100, 100, 500, 500, 500, 500,
1000, 1000, 1000, 1000]

 Train AUC for each  value is :
  [0.62665 0.62665 0.62665 0.62665 0.74036 0.74036 0.73985 0.73879 0.82014
 0.81919 0.81085 0.80542 0.98017 0.97437 0.94604 0.90883 0.99179 0.98792
 0.9643  0.92651 0.9999  0.99898 0.98412 0.95052 0.9999  0.99897 0.98425
 0.95127]

 CV AUC for each value is :
  [0.62599 0.62599 0.62599 0.62599 0.72765 0.72765 0.72783 0.72925 0.76653
 0.76729 0.77026 0.77525 0.7127  0.72021 0.7635  0.79792 0.71057 0.71761
 0.75756 0.78931 0.68854 0.69488 0.72617 0.75414 0.68956 0.69498 0.72902
 0.75855]
```

In [188]:

```
plt.plot(alph_split,train_auc_tfidf)
plt.plot(alph_split,cv_auc_tfidf)
plt.xlabel('SPLITS',size=10)
plt.ylabel('AUC',size=10)
plt.title('AUC VS HYPERPARAMETER SPLIT  BOW',size=16)
plt.xscale('log')
plt.grid()
plt.show()
print("\n\n Split Values :\n", alph_split)
print("\n Train AUC for each alpha value is :\n ", np.round(train_auc_tfidf,5))
print("\n CV AUC for each alpha value is :\n ", np.round(cv_auc_tfidf,5))
```






```
Split Values :
[5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100,
500, 5, 10, 100, 500]

Train AUC for each alpha value is :
 [0.62665 0.62665 0.62665 0.62665 0.74036 0.74036 0.73985 0.73879 0.82014
0.81919 0.81085 0.80542 0.98017 0.97437 0.94604 0.90883 0.99179 0.98792
0.9643  0.92651 0.9999  0.99898 0.98412 0.95052 0.9999  0.99897 0.98425
0.95127]

CV AUC for each alpha value is :
 [0.62599 0.62599 0.62599 0.62599 0.72765 0.72765 0.72783 0.72925 0.76653
0.76729 0.77026 0.77525 0.7127  0.72021 0.7635  0.79792 0.71057 0.71761
0.75756 0.78931 0.68854 0.69488 0.72617 0.75414 0.68956 0.69498 0.72902
0.75855]
```

Observations :

1) We found that optimal value for depth is 50 and min_samples_split is 500 as the cv accuracy is highest for that point.

Training the model with the best hyper parameter for TFIDF

In [164]:

```
om_tfidf = DecisionTreeClassifier(class_weight = 'balanced', max_depth = 50 , min_samples_split = 5
00 )
```

In [165]:

```
om_tfidf.fit(c3, y_train1)
ompredictions_tfidf  = om_tfidf.predict(d3)
```

In [166]:

```
probs2 = om_tfidf.predict_proba(c3)
probs3 = om_tfidf.predict_proba(d3)
probs2= probs2[:, 1]
probs3 = probs3[:, 1]
```

Feature Importance for TFIDF and exporting Decision Tree

In [243]:

```
features = tf_idf_vect.get_feature_names()
```

In [247]:

```
from sklearn.tree import export_graphviz
target = ['negative','positive']
export_graphviz(om_tfidf,out_file='tfidf_dt.dot.',class_names=target,rounded = True, proportion = F
alse,max_depth=3,feature_names=features)
```

In [168]:

```
feat_importances = om_tfidf.feature_importances_
```

In [169]:

```
len(features)
```

Out[169]:

```
9723
```

In [250]:

```
features.append('zzzzzzzzzzzaaaaaa')
```

In [171]:

```
cf = pd.DataFrame({'Word' : features, 'Coefficient' : feat_importances})
cf_new = cf.sort_values("Coefficient", ascending = False)
print('***** Top 20 IMPORTANT FEATURES  *****')
print('\n')
print(cf_new.head(20))
```

```
***** Top 20 IMPORTANT FEATURES  *****


             Word  Coefficient
5710          not     0.135897
3778        great     0.089494
718          best     0.040796
2241    delicious     0.038182
5003         love     0.033816
3698         good     0.026103
6176      perfect     0.025101
5008        loves     0.024190
2443  disappointed     0.024006
2965    excellent     0.015996
5660         nice     0.015340
3139     favorite     0.014419
9582    wonderful     0.013318
4045       highly     0.012381
546           bad     0.012284
2719         easy     0.009321
5465        money     0.009194
8750      thought     0.008516
7175      reviews     0.007863
3237         find     0.007627
```

Observations :

1) We found that the top 2 most important features affecting positive class are not and great.

PERFORMANCE MEASURMENTS FOR TFIDF

In [172]:

```
precision_tfidf = precision_score(y_test1, ompredictions_tfidf, pos_label = 1)
recall_tfidf = recall_score(y_test1, ompredictions_tfidf, pos_label = 1)
f1score_tfidf = f1_score(y_test1, ompredictions_tfidf, pos_label = 1)
```

In [174]:

```
print('\nThe Test Precision for optimal c for LR (TFIDF) is %f' % (precision_tfidf))
print('\nThe Test Recall for optimal c for LR (TFIDF) is  %f' % (recall_tfidf))
print('\nThe Test F1-Score for optimal c for LR (TFIDF) is  %f' % (f1score_tfidf))
```

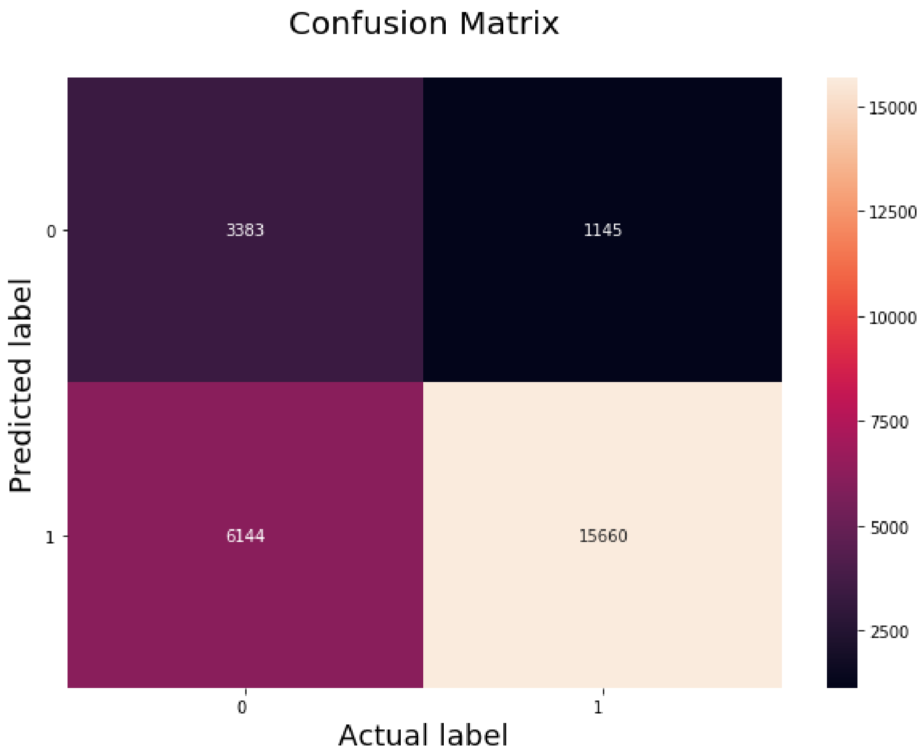
```
The Test Precision for optimal c for LR (TFIDF) is 0.931866

The Test Recall for optimal c for LR (TFIDF) is  0.718217
```

The Test F1-Score for optimal c for LR (TFIDF) is 0.811210

CONFUSION MATRIX (TFIDF)

In [175]:

```
# Code for drawing seaborn heatmaps
class_names = [ 0,1]
df_heatmap = pd.DataFrame(confusion_matrix(y_test1, ompredictions_tfidf), index=class_names, column
s=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)#
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)
plt.ylabel('Predicted label',size=18)
plt.xlabel('Actual label',size=18)
plt.title("Confusion Matrix\n",size=20)
plt.show()
```



In [176]:

```
TrueNeg,FalseNeg,FalsePos, TruePos = confusion_matrix(y_test1, ompredictions_tfidf).ravel()
TPR = TruePos/(FalseNeg + TruePos)
FPR = FalsePos/(TrueNeg + FalsePos)
TNR = TrueNeg/(TrueNeg + FalsePos)
FNR = FalseNeg/(FalseNeg + TruePos)
print("TPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (TPR))
print("FPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (FPR))
print("TNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (TNR))
print("FNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (FNR))
```

```
TPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.931866
FPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.644904
TNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.355096
FNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.068134
```

ROC CURVE FOR TFIDF

In [177]:

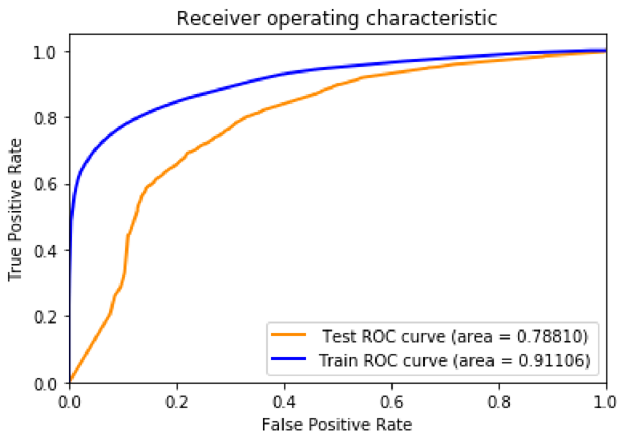
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()

fpr1 = dict()
tpr1 = dict()
roc_auc1 = dict()

for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test1,probs3)
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [178]:

```
from tqdm import tqdm
for i in range(4):
    fpr1[i], tpr1[i], _ = roc_curve(y_train1,probs2)
    roc_auc1[i] = auc(fpr1[i], tpr1[i])
```

In [180]:

```
#print(roc_auc_score(y_test1,ompredictions_bow))
plt.figure()
#plt.plot(fpr[1], tpr[1])
lw = 2
plt.plot(fpr[2], tpr[2], color='darkorange',lw=lw, label=' Test ROC curve (area = %0.5f)' % roc_auc
[0])
plt.plot(fpr1[2], tpr1[2], color='blue',lw=lw, label='Train ROC curve (area = %0.5f)' % roc_auc1[0]
)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.title('Receiver operating characteristic')
plt.show()
```



Observations:

1) We found that the training score has been good . However the test score has been less. It means that the model is over fitted to some extent

Word 2 Vector Data

Preparaing Training Data for Word to Vector

In [189]:

In [189]:

```
i=0
list_of_sentance=[]
for sentance in (X_trainbow['Cleaned Text'].values):
    list_of_sentance.append(sentance.split())
```

In [190]:

```
#WORD TO VECTOR

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.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")


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])
```

```
[('fantastic', 0.8406455516815186), ('awesome', 0.8259341716766357), ('good', 0.8041646480560303),
('excellent', 0.7943145632743835), ('terrific', 0.7926111817359924), ('amazing',
0.7851496934890747), ('wonderful', 0.7577135562896729), ('perfect', 0.752568244934082),
('fabulous', 0.6984073519706726), ('decent', 0.6744866371154785)]
==================================================
[('best', 0.733176589012146), ('greatest', 0.719762921333313), ('tastiest', 0.7155513763427734), (
'closest', 0.6296120882034302), ('disgusting', 0.6190059185028076), ('healthiest',
0.6187819242477417), ('experienced', 0.6123749017715454), ('coolest', 0.6008093953132629),
('awful', 0.5906195044517517), ('neapolitan', 0.5890594720840454)]
number of words that occured minimum 5 times  14706
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore',
'hard', 'find', 'products', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances',
'till', 'know', 'going', 'imports', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding',
'satisfied', 'safe', 'infestation', 'literally', 'everywhere', 'flying', 'around', 'kitchen',
'bought', 'hoping', 'least', 'get', 'rid', 'weeks', 'fly', 'stuck', 'buggers', 'success', 'rate',
'day']
```

In [191]:

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

```
100%|██████████████████████████████████████████████████| 61441/61441 [01:
28<00:00, 690.56it/s]
```

```
61441
50
```

In [192]:

```
sent_vectors[1]
```

Out[192]:

```
array([ 0.56022235,  0.67879919, -1.16433695,  0.65188279, -0.21017857,
       -0.23444822, -0.53972821, -1.61046622,  0.19608753, -0.25899484,
        0.04205897,  0.60934081,  0.15788466, -0.29382586,  0.3356308 ,
       -0.52329437, -0.27323416,  0.08312617, -0.88036611,  0.7756978 ,
        0.17861451, -0.04143068, -0.14238001, -0.43226893,  0.37092679,
        0.6208309 ,  1.28321907,  0.04711953, -0.05883861,  0.11775111,
        0.2887365 ,  0.1064998 ,  0.4384588 , -0.48808138,  0.04436555,
        0.19986581,  0.19658527,  0.94421775, -0.1201418 ,  1.26456587,
        0.42490968,  0.31975452,  0.30604115, -0.31224488,  0.63765167,
       -0.0106173 ,  0.56775195, -0.06896764,  0.65739828, -0.41883424])
```

Preparing Test Data for Word to Vector

In [193]:

```
X_test1.head(4)
```

Out[193]:

	Cleaned Text	Length
61441	used treat training reward dog loves easy brea...	66
61442	much fun watching puppies asking chicken treat...	134
61443	little shih tzu absolutely loves cesar softies...	181
61444	westie like picture package loves treats perfe...	162

In [194]:

```
i=0
list_of_sentance1=[]
for sentance in (X_test1['Cleaned Text'].values):
    list_of_sentance1.append(sentance.split())
```

In [195]:

```
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_model1=Word2Vec(list_of_sentance1,min_count=5,size=50, workers=4)
    print(w2v_model1.wv.most_similar('great'))
    print('='*50)
    print(w2v_model1.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_model1=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=T
rue)
        print(w2v_model1.wv.most_similar('great'))
        print(w2v_model1.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")


w2v_words1 = list(w2v_model1.wv.vocab)
```

```
w2v_words1 = list(w2v_model1.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words1))
print("sample words ", w2v_words1[0:50])
```

```
[('awesome', 0.8254234194755554), ('excellent', 0.8242385983467102), ('good', 0.7897971868515015), ('fantastic', 0.7889764308929443), ('wonderful', 0.7711959481239319), ('amazing', 0.7454652786254883), ('perfect', 0.7004916667938232), ('decent', 0.694219708442688), ('nice', 0.6861531734466553), ('terrific', 0.6521133184432983)]
==================================================
[('best', 0.7830518484115601), ('greatest', 0.7777789831161499), ('closest', 0.7717968225479126), ('nastiest', 0.7655762434005737), ('ever', 0.7230478525161743), ('tastiest', 0.7198038697242737), ('disgusting', 0.6892082691192627), ('hottest', 0.6884726881980896), ('beats', 0.6878105401992798), ('superior', 0.6872898936271667)]
number of words that occured minimum 5 times  9573
sample words  ['used', 'treat', 'training', 'reward', 'dog', 'loves', 'easy', 'break', 'smaller', 'pieces', 'buy', 'much', 'fun', 'watching', 'puppies', 'asking', 'chicken', 'treats', 'go', 'crazy', 'show', 'blue', 'package', 'small', 'eat', 'not', 'bad', 'smell', 'recommend', 'happy', 'little', 'shih', 'tzu', 'absolutely', 'tried', 'different', 'flavors', 'seems', 'enjoy', 'grilled', 'flavor', 'soft', 'enough', 'half', 'satisfy', 'westie', 'like', 'picture', 'perfect', 'size']
```

In [196]:

```
sent_vectors1 = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance1): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words1:
            vec = w2v_model1.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors1.append(sent_vec)
print(len(sent_vectors1))
print(len(sent_vectors1[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 26332/26332 [00:26<00:00, 981.69it/s]
```

```
26332
50
```

In [197]:

```
e3 = sent_vectors
f3 = sent_vectors1
```

In [198]:

```
len(y_test1)
```

Out[198]:

```
26332
```

In [199]:

```
e3 = preprocessing.normalize(e3)
e4 = sparse.csr_matrix(X_train1['Length'].values)
e4 = preprocessing.normalize(e4)
e5 = sparse.hstack([e3, e4.T])
```

In [200]:

```
f3 = preprocessing.normalize(f3)
f4 = sparse.csr_matrix(X_test1['Length'].values)
f4 = preprocessing.normalize(f4)
f5 = sparse.hstack([f3, f4.T])
```

Applying Decision Tree on Word to VECTOR

In [208]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model_w2v = GridSearchCV(DecisionTreeClassifier(max_features="log2",class_weight = 'balanced'), tr
ee_para, scoring = 'roc_auc', cv=5, return_train_score= True)
model_w2v.fit(e5, y_train1)
print(model_w2v.best_estimator_)
print(model_w2v.score(f5, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=50,
                       max_features='log2', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
0.6566681162985621
```

In [209]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model_w2v1 = GridSearchCV(DecisionTreeClassifier(class_weight = 'balanced'), tree_para, scoring =
'roc_auc', cv=5, return_train_score= True)
model_w2v1.fit(e5, y_train1)
print(model_w2v1.best_estimator_)
print(model_w2v1.score(f5, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
0.5923836419209885
```

Observations:

1) Both the above models have very less frequency . Surprisingly when we consider the log of features only the accuracy is high.

In [216]:

```
model3 = model_w2v
```

In [217]:

```
model_w2v.cv_results_
```

Out[217]:

```
{'mean_fit_time': array([0.09014406, 0.09032226, 0.09253793, 0.0874784 , 0.22807107,
        0.23079228, 0.23946481, 0.21244936, 0.61974654, 0.60959082,
        0.58819976, 0.42781701, 1.14950814, 1.10482087, 0.90074325,
        0.47751718, 1.15530291, 1.10393691, 0.90082531, 0.49071083,
        1.19088373, 1.13182101, 0.89601364, 0.49658022, 1.22367826,
        1.15857687, 0.8935699 , 0.50690885]),
 'std_fit_time': array([0.00608462, 0.01254362, 0.00812381, 0.00765249, 0.00765228,
        0.00847297, 0.01533399, 0.00765321, 0.02274793, 0.01626625,
        0.01521377, 0.00784872, 0.03355539, 0.03161787, 0.03846654,
        0.01627087, 0.03370975, 0.02757575, 0.02799729, 0.0112693 ,
        0.04492254, 0.05387487, 0.03493974, 0.01819977, 0.06924667,
        0.02345011, 0.02295924, 0.03388925]),
 'mean_score_time': array([0.00937328, 0.0126193 , 0.01509137, 0.00937309, 0.01249776,
        0.01475902, 0.01110325, 0.01562243, 0.01562853, 0.0123517 ,
        0.01506977, 0.01248875, 0.01559424, 0.01248717, 0.01356597,
        0.01249838, 0.0158669 , 0.01526828, 0.01214714, 0.01289358,
        0.01248937, 0.01296229, 0.01528168, 0.01195464, 0.01479869,
```

```
        0.0154798 , 0.01560559, 0.01248183]),
 'std_score_time': array([7.65325384e-03, 6.43229331e-03, 1.06335842e-03, 7.65309810e-03,
        6.24887947e-03, 1.10331943e-03, 5.63596166e-03, 1.08106461e-06,
        1.52613220e-05, 6.18080945e-03, 1.06941207e-03, 6.24438074e-03,
        1.18319048e-05, 6.24359446e-03, 3.31393970e-03, 6.24918990e-03,
        5.31372417e-04, 6.70293631e-04, 6.10794165e-03, 2.55005787e-03,
        6.24473185e-03, 4.48785901e-03, 6.76159126e-04, 6.06624072e-03,
        7.52025450e-04, 2.73346168e-04, 1.77063962e-05, 6.24092549e-03]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50,
                    100, 100, 100, 100, 500, 500, 500, 500, 1000, 1000,
                    1000, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500,
5,
                    10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10,
                    100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 500},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 500},
  {'max_depth': 1000, 'min_samples_split': 5},
  {'max_depth': 1000, 'min_samples_split': 10},
  {'max_depth': 1000, 'min_samples_split': 100},
  {'max_depth': 1000, 'min_samples_split': 500}],
 'split0_test_score': array([0.64102867, 0.64102867, 0.5989988 , 0.59098595, 0.78512717,
        0.76845708, 0.75101103, 0.78333961, 0.78363642, 0.76882408,
        0.80285442, 0.77835876, 0.6444169 , 0.67858778, 0.75893497,
        0.78914556, 0.64315881, 0.68830968, 0.75692743, 0.8015579 ,
        0.65001176, 0.68084036, 0.7757677 , 0.80015307, 0.65518618,
        0.66863876, 0.76055949, 0.79167592]),
 'split1_test_score': array([0.57782547, 0.65147014, 0.63285918, 0.65147014, 0.73077937,
        0.74320379, 0.73419688, 0.73301735, 0.76972462, 0.78373174,
        0.78164882, 0.77371166, 0.65298575, 0.65665011, 0.7677185 ,
        0.78448539, 0.64954664, 0.65689802, 0.76686706, 0.7935586 ,
        0.64269771, 0.66804377, 0.75668817, 0.78369926, 0.65574107,
        0.66203651, 0.76017414, 0.78867569]),
 'split2_test_score': array([0.65745513, 0.5751347 , 0.53698622, 0.65263024, 0.78428048,
        0.75211811, 0.74856532, 0.72906383, 0.79722983, 0.76505562,
        0.80015764, 0.78204467, 0.65434183, 0.689114  , 0.77424938,
        0.79438889, 0.65617161, 0.67708641, 0.77889562, 0.77744382,
        0.66584054, 0.66630208, 0.77284629, 0.79652536, 0.6579757 ,
        0.68225872, 0.78222238, 0.78731153]),
 'split3_test_score': array([0.6124661 , 0.59419551, 0.53928111, 0.59156141, 0.76320058,
        0.78763553, 0.7392671 , 0.77384078, 0.77189408, 0.76481689,
        0.7620195 , 0.77176821, 0.65113232, 0.66167632, 0.7737986 ,
        0.79643261, 0.64817212, 0.65857956, 0.76832672, 0.77518902,
```

```
        0.64281137, 0.67056423, 0.7620628 , 0.78724276, 0.63053943,
        0.66716776, 0.75470073, 0.77211   ]),
 'split4_test_score': array([0.55919395, 0.55609457, 0.55531287, 0.52188077, 0.7586393 ,
        0.73413826, 0.72781906, 0.72650819, 0.77121321, 0.75990274,
        0.78310579, 0.77488848, 0.6499582 , 0.66884133, 0.76052446,
        0.79710676, 0.64092639, 0.66720135, 0.76180344, 0.77235441,
        0.64470129, 0.6590596 , 0.75206994, 0.78595003, 0.64563733,
        0.67100412, 0.76643953, 0.76899114]),
 'mean_test_score': array([0.60959541, 0.60358657, 0.57268929, 0.60170863, 0.76440561,
        0.75711031, 0.74017231, 0.74915389, 0.7787401 , 0.76846661,
        0.78595811, 0.77615454, 0.650567  , 0.67097428, 0.76704517,
        0.79231155, 0.64759531, 0.66961544, 0.76656415, 0.78402142,
        0.64921289, 0.66896228, 0.76388742, 0.79071436, 0.64901665,
        0.67022125, 0.76481953, 0.78175358]),
 'std_test_score': array([0.03697705, 0.03700811, 0.0374315 , 0.04828945, 0.01994965,
        0.01900761, 0.00868387, 0.02431025, 0.01049442, 0.00814192,
        0.01474348, 0.00364317, 0.00342395, 0.01168955, 0.00642272,
        0.00480762, 0.00532639, 0.01178699, 0.00735946, 0.01145329,
        0.00872827, 0.00706595, 0.00912379, 0.0064316 , 0.01016218,
        0.00669843, 0.00946164, 0.00930725]),
 'rank_test_score': array([25, 26, 28, 27, 12, 14, 16, 15,  6,  8,  3,  7, 21, 17,  9,  1, 24,
        19, 10,  4, 22, 20, 13,  2, 23, 18, 11,  5]),
 'split0_train_score': array([0.64082312, 0.64082312, 0.58616772, 0.58575124, 0.78569659,
        0.78184217, 0.75711575, 0.79130052, 0.89341075, 0.88366998,
        0.87302611, 0.82411549, 0.99915952, 0.99443527, 0.90914662,
        0.82910863, 0.99913668, 0.99391495, 0.91019718, 0.83900127,
        0.99915732, 0.99434919, 0.9112529 , 0.83736276, 0.99918115,
        0.99408595, 0.9077147 , 0.83871735]),
 'split1_train_score': array([0.6125196 , 0.66166024, 0.64187972, 0.66166024, 0.75525069,
        0.77172472, 0.76659873, 0.76638669, 0.89286181, 0.88715238,
        0.87472172, 0.8291986 , 0.99920984, 0.99414299, 0.92067296,
        0.84145043, 0.99914452, 0.99376672, 0.91251142, 0.85110622,
        0.99915885, 0.9944729 , 0.90817039, 0.83312013, 0.99920214,
        0.99446856, 0.91492833, 0.8374888 ]),
 'split2_train_score': array([0.63580179, 0.5799958 , 0.54289679, 0.66130997, 0.78861575,
        0.7608016 , 0.7518067 , 0.74589812, 0.88983462, 0.88051587,
        0.87525488, 0.8259874 , 0.999103  , 0.9942688 , 0.90956396,
        0.8400764 , 0.99923774, 0.99406585, 0.91640637, 0.82894838,
        0.99907579, 0.9940419 , 0.90962151, 0.83655883, 0.99913174,
        0.99402347, 0.91405817, 0.8422759 ]),
 'split3_train_score': array([0.6221701 , 0.60823297, 0.55738473, 0.58755126, 0.78565581,
        0.80794882, 0.76747939, 0.78939342, 0.88076635, 0.88982081,
        0.86995927, 0.83040974, 0.99915191, 0.99413768, 0.91775024,
        0.84615583, 0.99911319, 0.99428241, 0.91462906, 0.83647724,
        0.99920578, 0.9940732 , 0.91301812, 0.83882246, 0.99916837,
        0.99425988, 0.910456  , 0.82459772]),
 'split4_train_score': array([0.56510058, 0.57482914, 0.57872995, 0.55714813, 0.77156142,
        0.75066115, 0.75102096, 0.74888221, 0.88514784, 0.88372964,
        0.87373672, 0.8331583 , 0.99917515, 0.99457809, 0.91413625,
        0.85137641, 0.99917404, 0.99412451, 0.91769956, 0.83319818,
        0.99915191, 0.99424598, 0.91434422, 0.83850773, 0.99923419,
        0.99433411, 0.91450919, 0.83252687]),
 'mean_train_score': array([0.61528304, 0.61310825, 0.58141178, 0.61068417, 0.77735605,
        0.77459569, 0.75880431, 0.76837219, 0.88840428, 0.88497774,
        0.87333974, 0.82857391, 0.99915988, 0.99431257, 0.91425401,
        0.84163354, 0.99916123, 0.99403089, 0.91428872, 0.83774625,
        0.99914993, 0.99423663, 0.91128143, 0.83687438, 0.99918352,
        0.99423439, 0.91233328, 0.83512133]),
 'std_train_score': array([2.70061570e-02, 3.37961629e-02, 3.39024676e-02, 4.28588871e-02,
        1.25495508e-02, 1.96790447e-02, 7.04852088e-03, 1.92690749e-02,
        4.81502967e-03, 3.20501790e-03, 1.85801687e-03, 3.20452067e-03,
        3.47199928e-05, 1.71355608e-04, 4.50604842e-03, 7.41223421e-03,
        4.29118863e-05, 1.76903275e-04, 2.68825434e-03, 7.48013032e-03,
        4.18410555e-05, 1.63225065e-04, 2.22851840e-03, 2.04427464e-03,
        3.41475320e-05, 1.62452339e-04, 2.80355284e-03, 6.11885385e-03])}
```

In [218]:

```
train_auc_w2v = model3.cv_results_['mean_train_score']
cv_auc_w2v = model3.cv_results_['mean_test_score']
```

In [219]:

```
alph_depth = [1,1,1,1,5,5,5,5,10,10,10,10,50,50,50,50,100,100,100,100,500,500,500,500,1000,1000,1000
,1000]
```

```
alph_split = [5,10,100,500]*7
```

In [220]:

```
train_auc_w2v
```

Out[220]:

```
array([0.61528304, 0.61310825, 0.58141178, 0.61068417, 0.77735605,
       0.77459569, 0.75880431, 0.76837219, 0.88840428, 0.88497774,
       0.87333974, 0.82857391, 0.99915988, 0.99431257, 0.91425401,
       0.84163354, 0.99916123, 0.99403089, 0.91428872, 0.83774625,
       0.99914993, 0.99423663, 0.91128143, 0.83687438, 0.99918352,
       0.99423439, 0.91233328, 0.83512133])
```

In [221]:

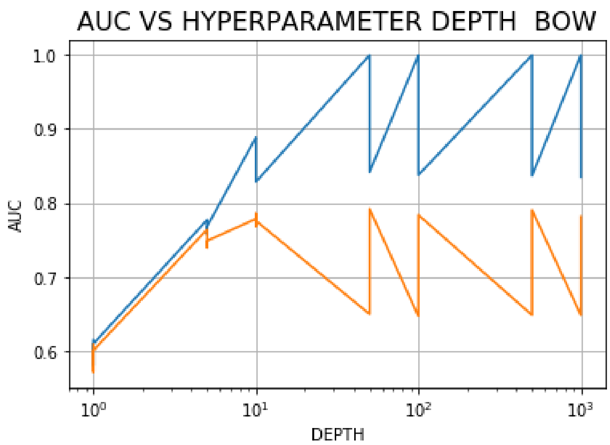
```
cv_auc_w2v
```

Out[221]:

```
array([0.60959541, 0.60358657, 0.57268929, 0.60170863, 0.76440561,
       0.75711031, 0.74017231, 0.74915389, 0.7787401 , 0.76846661,
       0.78595811, 0.77615454, 0.650567  , 0.67097428, 0.76704517,
       0.79231155, 0.64759531, 0.66961544, 0.76656415, 0.78402142,
       0.64921289, 0.66896228, 0.76388742, 0.79071436, 0.64901665,
       0.67022125, 0.76481953, 0.78175358])
```

In [222]:

```
plt.plot(alph_depth,train_auc_w2v)
plt.plot(alph_depth,cv_auc_w2v)
plt.xlabel('DEPTH',size=10)
plt.ylabel('AUC',size=10)
plt.title('AUC VS HYPERPARAMETER DEPTH  BOW',size=16)
plt.xscale('log')
plt.grid()
plt.show()
print("\n\n Depth Values :\n", alph_depth)
print("\n Train AUC for each alpha value is :\n ", np.round(train_auc_w2v,5))
print("\n CV AUC for each alpha value is :\n ", np.round(cv_auc_w2v,5))
```

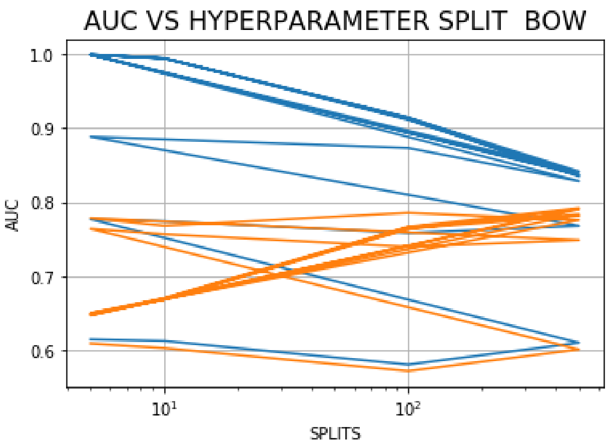



```
 Depth Values :
 [1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50, 100, 100, 100, 100, 500, 500, 500, 500,
1000, 1000, 1000, 1000]

 Train AUC for each alpha value is :
  [0.61528 0.61311 0.58141 0.61068 0.77736 0.7746  0.7588  0.76837 0.8884
 0.88498 0.87334 0.82857 0.99916 0.99431 0.91425 0.84163 0.99916 0.99403
 0.91429 0.83775 0.99915 0.99424 0.91128 0.83687 0.99918 0.99423 0.91233
 0.83512]

 CV AUC for each alpha value is :
  [0.6096  0.60359 0.57269 0.60171 0.76441 0.75711 0.74017 0.74915 0.77874
```

```
0.76847 0.78596 0.77615 0.65057 0.67097 0.76705 0.79231 0.6476  0.66962
0.76656 0.78402 0.64921 0.66896 0.76389 0.79071 0.64902 0.67022 0.76482
0.78175]
```

In [224]:

```
plt.plot(alph_split,train_auc_w2v)
plt.plot(alph_split,cv_auc_w2v)
plt.xlabel('SPLITS',size=10)
plt.ylabel('AUC',size=10)
plt.title('AUC VS HYPERPARAMETER SPLIT  BOW',size=16)
plt.xscale('log')
plt.grid()
plt.show()
print("\n\n Split Values :\n", alph_split)
print("\n Train AUC for each alpha value is :\n ", np.round(train_auc_w2v,5))
print("\n CV AUC for each alpha value is :\n ", np.round(cv_auc_w2v,5))
```




```
 Split Values :
 [5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100,
500, 5, 10, 100, 500]

 Train AUC for each alpha value is :
  [0.61528 0.61311 0.58141 0.61068 0.77736 0.7746  0.7588  0.76837 0.8884
 0.88498 0.87334 0.82857 0.99916 0.99431 0.91425 0.84163 0.99916 0.99403
 0.91429 0.83775 0.99915 0.99424 0.91128 0.83687 0.99918 0.99423 0.91233
 0.83512]

 CV AUC for each alpha value is :
  [0.6096  0.60359 0.57269 0.60171 0.76441 0.75711 0.74017 0.74915 0.77874
 0.76847 0.78596 0.77615 0.65057 0.67097 0.76705 0.79231 0.6476  0.66962
 0.76656 0.78402 0.64921 0.66896 0.76389 0.79071 0.64902 0.67022 0.76482
 0.78175]
```

In [226]:

```
max(cv_auc_w2v)
```

Out[226]:

```
0.7923115528602512
```

Observations:

1) Optimal number of splits = 500 and max_depth = 50 as the cv_auc is high at that point

Running the model with the optimal hyperparameter

In [227]:

```
om_w2v = DecisionTreeClassifier(class_weight = 'balanced', max_depth = 50 , min_samples_split = 500
)
om_w2v.fit(e5, y_train1)
ompredictions_w2v  = om_w2v.predict(f5)
probs4 = om_w2v.predict_proba(e5)
probs5 = om_w2v.predict_proba(f5)
probs4= probs4[:, 1]
probs5 = probs5[:, 1]
```

PERFORMANCE MEASURMENTS FOR w2v Decision Tree

In [228]:

```
precision_w2v = precision_score(y_test1, ompredictions_w2v, pos_label = 1)
recall_w2v = recall_score(y_test1, ompredictions_w2v, pos_label = 1)
f1score_w2v = f1_score(y_test1, ompredictions_w2v, pos_label = 1)

print('\nThe Test Precision for optimal c for LR (TFIDF) is %f' % (precision_w2v))
print('\nThe Test Recall for optimal c for LR (TFIDF) is  %f' % (recall_w2v))
print('\nThe Test F1-Score for optimal c for LR (TFIDF) is  %f' % (f1score_w2v))
```

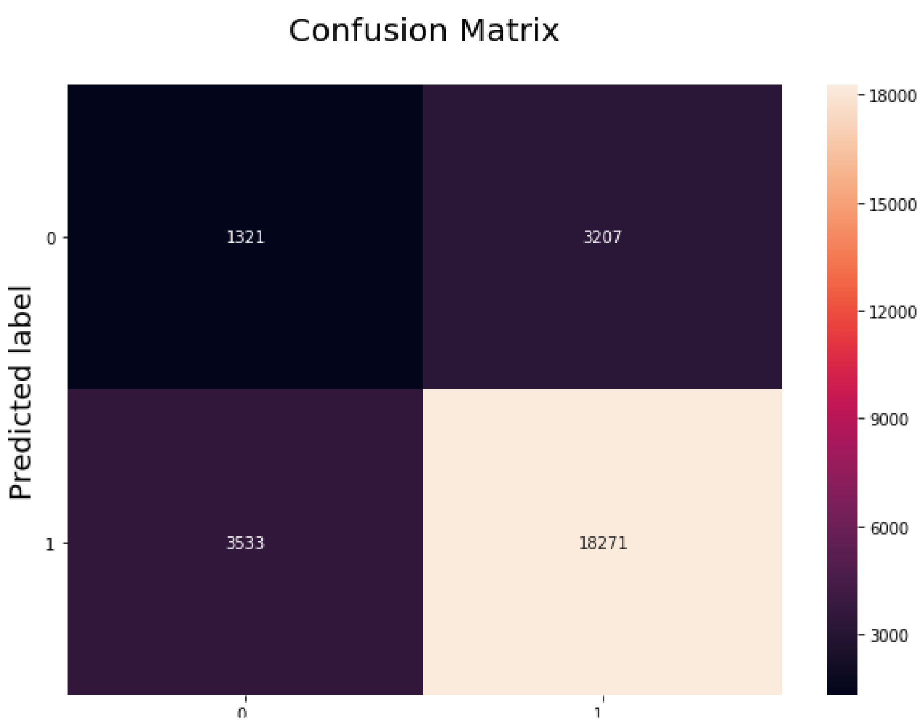
```
The Test Precision for optimal c for LR (TFIDF) is 0.850684

The Test Recall for optimal c for LR (TFIDF) is  0.837966

The Test F1-Score for optimal c for LR (TFIDF) is  0.844277
```

In [229]:

```
class_names = [ 0,1]
df_heatmap = pd.DataFrame(confusion_matrix(y_test1, ompredictions_w2v), index=class_names, columns
=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)#
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)
plt.ylabel('Predicted label',size=18)
plt.xlabel('Actual label',size=18)
plt.title("Confusion Matrix\n",size=20)
```

Out[229]:

```
Text(0.5,1,'Confusion Matrix\n')
```



Actual label

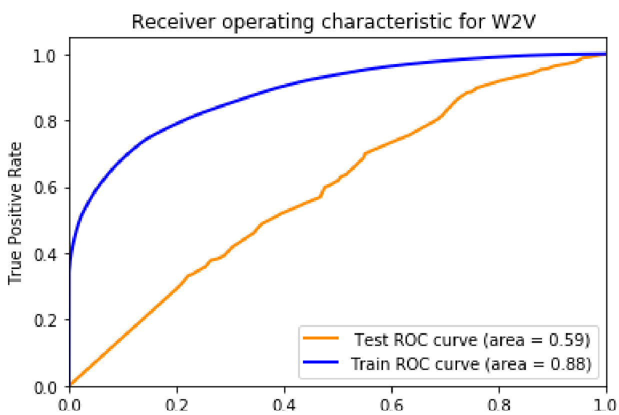
In [230]:

```
TrueNeg,FalseNeg,FalsePos, TruePos = confusion_matrix(y_test1, ompredictions_w2v).ravel()
TPR = TruePos/(FalseNeg + TruePos)
FPR = FalsePos/(TrueNeg + FalsePos)
TNR = TrueNeg/(TrueNeg + FalsePos)
FNR = FalseNeg/(FalseNeg + TruePos)
print("TPR of the Logistic Regression  (TFIDF) for optimal alpha   is :  %f" % (TPR))
print("FPR of the Logistic Regression (TFIDF) for optimal  alpha  is :  %f" % (FPR))
print("TNR of the Logistic Regression (TFIDF) for optimal alpha  is :  %f" % (TNR))
print("FNR of the Logistic Regression (TFIDF) for optimal alpha  is :  %f" % (FNR))
```

```
TPR of the Logistic Regression  (TFIDF) for optimal alpha   is :  0.850684
FPR of the Logistic Regression (TFIDF) for optimal  alpha  is :  0.727853
TNR of the Logistic Regression (TFIDF) for optimal alpha  is :  0.272147
FNR of the Logistic Regression (TFIDF) for optimal alpha  is :  0.149316
```

In [231]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()

fpr1 = dict()
tpr1 = dict()
roc_auc1 = dict()

for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test1,probs5)
    roc_auc[i] = auc(fpr[i], tpr[i])


from tqdm import tqdm
for i in range(4):
    fpr1[i], tpr1[i], _ = roc_curve(y_train1,probs4)
    roc_auc1[i] = auc(fpr1[i], tpr1[i])


#print(roc_auc_score(y_test1,ompredictions_bow))
plt.figure()
#plt.plot(fpr[1], tpr[1])
lw = 2
plt.plot(fpr[0], tpr[0], color='darkorange',lw=lw, label=' Test ROC curve (area = %0.2f)' % roc_auc
[0])
plt.plot(fpr1[0], tpr1[0], color='blue',lw=lw, label='Train ROC curve (area = %0.2f)' % roc_auc1[0]
)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.title('Receiver operating characteristic for W2V ')
plt.show()
```



Observations :

1) Word 2 vECTOR HAS NOT PERFORMED THAT EFFICIENTLY WHEN COMPARED TO BOW or TFIDF.

2) Test Acuuracy is very less when compared to train accuracy. Hence Overfitting would have been the issue here.

TFIDF AVERGE WORD TO VECTOR

Preparing Training Data for TFIDF-AVG W2V

In [248]:

```
model = TfidfVectorizer()
model.fit_transform(X_trainbow['Cleaned Text'].values)
# 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 [203]:

```
# 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 list
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%|██████████████████████████████████████████████████| 61441/61441 [14
:11<00:00, 72.17it/s]
```

Preparing Test Data for TFIDF- AVG W2V

In [204]:

```
# 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_vectors1 = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance1): # 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_words1 and word in tfidf_feat:
            vec = w2v_model1.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_vectors1.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████████████████████| 26332/26332 [17
:52<00:00, 24.54it/s]
```

In [205]:

```
g3 = tfidf_sent_vectors
h3 = tfidf_sent_vectors1
```

In [206]:

```
g3 = preprocessing.normalize(g3)
h3 = preprocessing.normalize(h3)
```

DECISION TREE ON TFIDF - AVG W2V

In [232]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model1 = GridSearchCV(DecisionTreeClassifier(max_features="log2",class_weight = 'balanced'),
tree_para, scoring = 'roc_auc', cv=5, return_train_score= True)
model1.fit(g3, y_train1)
print(model1.best_estimator_)
print(model1.score(h3, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                       max_features='log2', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
0.5848606783418351
```

In [233]:

```
tree_para = [{'max_depth':[1,5,10,50,100,500,1000],'min_samples_split' : [5, 10, 100, 500]}]
model2 = GridSearchCV(DecisionTreeClassifier(class_weight = 'balanced'), tree_para, scoring =
'roc_auc', cv=5, return_train_score= True)
model2.fit(g3, y_train1)
print(model2.best_estimator_)
print(model2.score(h3, y_test1))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
0.6099152441394033
```

Running the model with the optimal depth and split

In [234]:

```
om_w2vtfidf = DecisionTreeClassifier(class_weight = 'balanced', max_depth = 10 , min_samples_split
= 500 )
om_w2vtfidf.fit(g3, y_train1)
ompredictions_w2vtfidf = om_w2vtfidf.predict(h3)
```

```
ompredictions_w2vtfidf = om_w2vtfidf.predict(h3)
```

In [235]:

```
probs6 = om_w2vtfidf.predict_proba(g3)
probs7 = om_w2vtfidf.predict_proba(h3)
probs6= probs6[:, 1]
probs7 = probs7[:, 1]
```

In [236]:

```
precision_w2vtfidf = precision_score(y_test1, ompredictions_w2vtfidf, pos_label = 1)
recall_w2vtfidf = recall_score(y_test1, ompredictions_w2vtfidf, pos_label = 1)
f1score_w2vtfidf = f1_score(y_test1, ompredictions_w2vtfidf, pos_label = 1)
```

In [237]:

```
print('\nThe Test Precision for optimal c for LR (TFIDF) is %f' % (precision_w2vtfidf))
print('\nThe Test Recall for optimal c for LR (TFIDF) is  %f' % (recall_w2vtfidf))
print('\nThe Test F1-Score for optimal c for LR (TFIDF) is  %f' % (f1score_w2vtfidf))
```

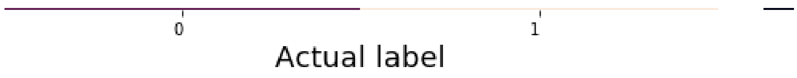
```
The Test Precision for optimal c for LR (TFIDF) is 0.854571

The Test Recall for optimal c for LR (TFIDF) is  0.746790

The Test F1-Score for optimal c for LR (TFIDF) is  0.797053
```

In [238]:

```
# Code for drawing seaborn heatmaps
class_names = [ 0,1]
df_heatmap = pd.DataFrame(confusion_matrix(y_test1, ompredictions_w2vtfidf), index=class_names,
columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)#
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=10)
plt.ylabel('Predicted label',size=18)
plt.xlabel('Actual label',size=18)
plt.title("Confusion Matrix\n",size=20)
```

Out[238]:

```
Text(0.5,1,'Confusion Matrix\n')
```



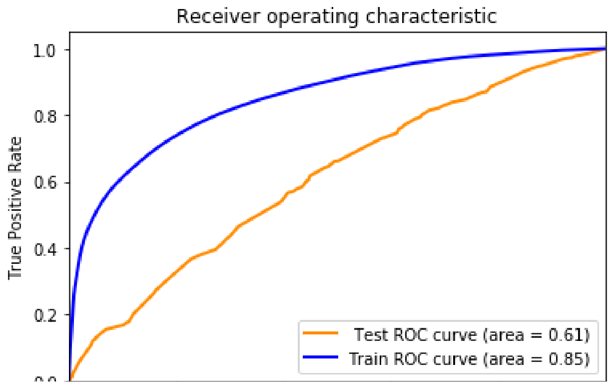
In [239]:

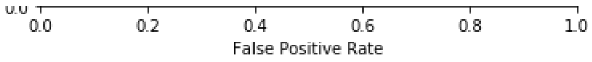
```
TrueNeg,FalseNeg,FalsePos, TruePos = confusion_matrix(y_test1, ompredictions_w2vtfidf).ravel()
TPR = TruePos/(FalseNeg + TruePos)
FPR = FalsePos/(TrueNeg + FalsePos)
TNR = TrueNeg/(TrueNeg + FalsePos)
FNR = FalseNeg/(FalseNeg + TruePos)
print("TPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (TPR))
print("FPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (FPR))
print("TNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (TNR))
print("FNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  %f" % (FNR))
```

```
TPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.854571
FPR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.758588
TNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.241412
FNR of the Multinomial naive Bayes classifier (TFIDF) for alpha  is :  0.145429
```

In [241]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()

fpr1 = dict()
tpr1 = dict()
roc_auc1 = dict()

for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test1,probs7)
    roc_auc[i] = auc(fpr[i], tpr[i])


from tqdm import tqdm
for i in range(4):
    fpr1[i], tpr1[i], _ = roc_curve(y_train1,probs6)
    roc_auc1[i] = auc(fpr1[i], tpr1[i])


#print(roc_auc_score(y_test1,ompredictions_bow))
plt.figure()
#plt.plot(fpr[1], tpr[1])
lw = 2
plt.plot(fpr[0], tpr[0], color='darkorange',lw=lw, label=' Test ROC curve (area = %0.2f)' % roc_auc
[0])
plt.plot(fpr1[0], tpr1[0], color='blue',lw=lw, label='Train ROC curve (area = %0.2f)' % roc_auc1[0]
)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.title('Receiver operating characteristic')
plt.show()
```





Observations

1) Both Training Accuracy and Test acccuracy has been very low in TFIDF- AVG W2V. The reason might be the case that I did not take feature engineering for this vectoriser and the model may be over fitted.

Conclusions

In [242]:

```
res = pd.DataFrame()
```

In [254]:

```
model_names = ["BOW","BOW","TF-IDF","TF-IDF","W2V","W2V", "TF-IDF AVGW2V" , "TF-IDF AVGW2V"]
max_depth = [1000,50,1000,1000,50,10,10,10]
min_samples_split = [500]*8
s1 = "YES"
s2 = "NO"
Feature_Engineering = [s1,s1,s1,s1,s1,s1,s2,s2]
AUC = [0.6838,0.8006,0.6988,0.7890,0.6567,0.5924,0.5849,0.6099]
```

In [258]:

```
max_features = ["log(N)", "N"]*4
```

In [259]:

```
res['Vectorizer'] = model_names
res['Max_Features'] = max_features
res['Maximum Depth'] =  max_depth
res['Minimum Samples Split'] = min_samples_split
res['Feature Engineering'] = Feature_Engineering
res['AUC'] = AUC
```

In [260]:

```
res
```

Out[260]:

	Vectorizer	Max_Features	Maximum Depth	Minimum Samples Split	Feature Engineering	AUC
0	BOW	log(N)	1000	500	YES	0.6838
1	BOW	N	50	500	YES	0.8006
2	TF-IDF	log(N)	1000	500	YES	0.6988
3	TF-IDF	N	1000	500	YES	0.7890
4	W2V	log(N)	50	500	YES	0.6567
5	W2V	N	10	500	YES	0.5924
6	TF-IDF AVGW2V	log(N)	10	500	NO	0.5849
7	TF-IDF AVGW2V	N	10	500	NO	0.6099

In [262]:

```
import tabulatehelper as th
```

DISPLAYING THE RESULTS IN TABULAR FORMAT

DISPLAYING THE RESULTS IN TABULAR FORMAT

In [264]:

```
print(th.md_table(res, formats={-1: 'c'}))
```

```
| Vectorizer     | Max_Features    |   Maximum Depth |   Minimum Samples Split | Feature Engineering
|    AUC |
|:---------------|:----------------|----------------:|------------------------:|:--------------------
:------:|
| BOW            | log(N)          |            1000 |                     500 | YES
| 0.6838 |
| BOW            | N               |              50 |                     500 | YES
| 0.8006 |
| TF-IDF         | log(N)          |            1000 |                     500 | YES
| 0.6988 |
| TF-IDF         | N               |            1000 |                     500 | YES
| 0.789  |
| W2V            | log(N)          |              50 |                     500 | YES
| 0.6567 |
| W2V            | N               |              10 |                     500 | YES
| 0.5924 |
| TF-IDF AVGW2V  | log(N)          |              10 |                     500 | NO
| 0.5849 |
| TF-IDF AVGW2V  | N               |              10 |                     500 | NO
| 0.6099 |
```

Final Observations :

1) The best models have come through BOW and TFIDF. In TFIDF the AUC has been slightly lower when compared to BOW.

2) IN THE CASE OF BAG OF WORDS, TFIDF considering N features PERFORMED significantly BETTER WHEN COMPARED TO Log(N) features.

3) IN THE CASE OF AVGW2V considering N features efficiency is less when compared to Log(N) features.

5) IN THE CASE OF TFIDF- AVG W2V, I HAVE NOT USED FEATURE ENGINEERING. considering N features PERFORMED significantly BETTER WHEN COMPARED TO Log(N) features.

6) As suggested I have added length of preprocessed reviews as one more feature which has been contributed for more accuracy. However if i would have used more features like length of common words or something else, the results would have been different(my assumption)

7) I have visualised the tree for BOW and TFIDF. As it is not clear in the jupyter notebook, I have exported.

8) Overall, the best model for the decision tree classifier is BOW with 0.80 AUC.

References

I have referred many links. However part of my code has been inspired from the following links

1) Applied AI Course - https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/

2) SKLEARN

3) STACK OVERFLOW - MANY