Web Mining Digital Assignment 3

Devang Mehrotra

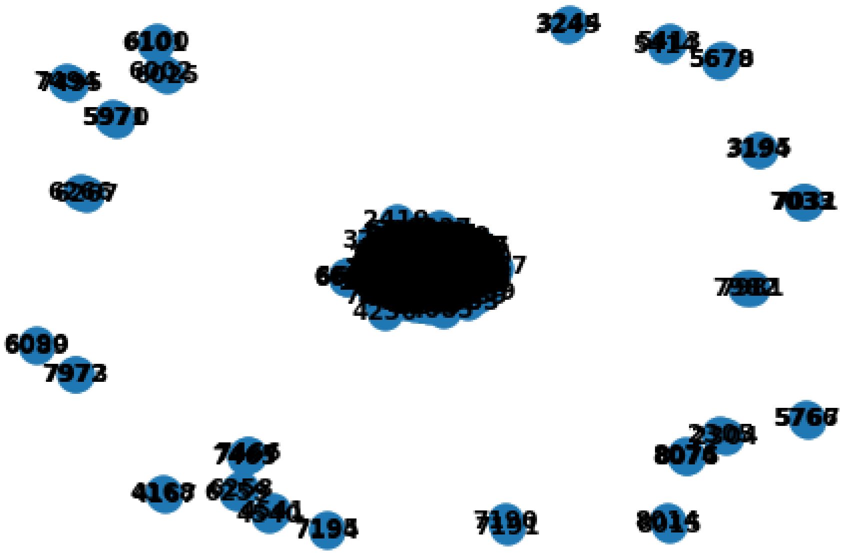
18BCE0763

Q1

In [2]:

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

d = nx.Graph()
edges = nx.read_edgelist('Wiki-Vote.txt')
d.add_edges_from(edges.edges())
nx.draw(d, with_labels=True, font_weight='bold')
plt.savefig("filename.png")
```



In [3]:

```
import networkx as nx
import numpy as np
n_nodes=7114
degree_prestige = dict((v,len(d.edges(v))/(n_nodes-1)) for v in d.nodes())
print("DEGREE PRESTIGE :\n")

for i in degree_prestige:
    print(i, " : ", degree_prestige[i])
```

```
DEGREE PRESTIGE :

30  :  0.0039364543793054966
1412  :  0.004077042035709265
3352  :  0.06706031210459722
5254  :  0.04091100801349642
5543  :  0.036974553634190915
7478  :  0.012934064389146634
3  :  0.007169970476592155
25  :  0.012652889076339097
4  :  0.004077042035709265
5  :  0.0032335160972866584
6  :  0.044144524110783075
7  :  0.003374103753690426
8  :  0.030507521439617602
9  :  0.011387600168705188
10  :  0.013918177983973008
11  :  0.10445662870799943
12  :  0.009138197666244904
13  :  0.0015464643204414453
```

In [4]:

```
x=list(d.nodes)
distance=[]

for i in range(0,500) :
    temp_dis = 0
    n = 0
    for j in range(0,500):
        if(nx.has_path(d,x[i],x[j]) == True):
            temp_dis = temp_dis + nx.shortest_path_length(d,source = x[j],target = x[i])
            n = n + 1
    if temp_dis == 0:
        distance.append([x[i], 0])
    else:
        distance.append([x[i], temp_dis/(n - 1)])

print("\nPROXIMITY PRESTIGE :\n")
for i in distance:
    print(str(i[0]) + " : " + str(i[1]))
```

```

PROXIMITY PRESTIGE :

30 : 1.9438877755511021
1412 : 2.2725450901803605
3352 : 1.1182364729458918
5254 : 1.8877755511022045
5543 : 1.9639278557114228
7478 : 2.038076152304609
3 : 2.4869739478957915
25 : 2.408817635270541
4 : 2.7334669338677355
5 : 2.68937875751503
6 : 2.030060120240481
7 : 2.68937875751503
8 : 2.1382765531062122
9 : 2.585170340681363
10 : 2.2725450901803605
11 : 1.593186372745491
12 : 2.302605210420842
```

In [12]:

```
prominance = np.random.randint(1, 4, size=n_nodes)
rank_prestige = np.zeros([n_nodes], dtype = int)
path_matrix = np.zeros([n_nodes, n_nodes], dtype = int)
i = 0
j = 0
for src in d.nodes:
    for dest in d.nodes:
        if d.has_edge(dest, src):
            path_matrix[i-1][j-1] = 1
        j = j+1
    j = 0
    i = i+1
for i in range(n_nodes):
    pr_i = 0
    for j in range(n_nodes):
        pr_i = pr_i + path_matrix[i][j] * prominance[j]
    rank_prestige[i] = pr_i

print("\nRANK PRESTIGE :\n")
for i in rank_prestige:
    print(i, " : ", rank_prestige[i])
```

```
RANK PRESTIGE :

53  :  3
933  :  3
603  :  8
534  :  72
174  :  579
96  :  359
177  :  820
56  :  285
46  :  3
616  :  104
47  :  3
421  :  113
159  :  782
197  :  87
1482  :  78
123  :  88
22  :  104
```

Q2

In [13]:

```
def hits(A,H,L,Lt):
    hits.count=getattr(hits,"count",1)
    Ak=np.dot(np.dot(Lt,L),A)
    Hk=np.dot(np.dot(L,Lt),H)
    As=[]
    for i in Ak:
        As.append(i**2)
    Ak=Ak/float(math.sqrt(sum(As)))
    Hs=[]
    for i in Hk:
        Hs.append(i**2)
    Hk=Hk/float(math.sqrt(sum(Hs)))
    print("Authority Score in Iteration",hits.count,":\n",Ak,"\n")
    print("Hub Score in Iteration ",hits.count,":\n",Hk,"\n")
    temp=0
    for i in range(len(Ak)):
        if(abs(Ak[i]-A[i])<0.05 and abs(Hk[i]-H[i])<0.05):
            temp=temp+1
    if(temp<len(Ak)):
        hits.count+=1
        hits(Ak,Hk,L,Lt)
    else:
        print("Total number of Iterations are: ",hits.count)
        print("\nAuthority and Hub score in last iteration are the final scores")
```

In [14]:

```
import numpy as np
import math
L=np.array([[0,1,1,1,0,0,0],[0,0,0,1,1,0,0],[0,0,0,0,0,1,0],[0,0,1,0,0,1,1],[0,0,0,1,0,0,1]
print("Adjency Matrix L: \n\n",L,"\n\n")
Lt=L.transpose()
Ain=np.ones(7)
Hin=np.ones(7)
hits(Ain,Hin,L,Lt)
```

```
Adjency Matrix L: 

 [[0 1 1 1 0 0 0]
 [0 0 0 1 1 0 0]
 [0 0 0 0 0 1 0]
 [0 0 1 0 0 1 1]
 [0 0 0 1 0 0 1]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0]] 


Authority Score in Iteration 1 :
 [0.         0.24659848 0.49319696 0.57539646 0.16439899 0.41099747
 0.41099747]

Hub Score in Iteration  1 :
 [0.5        0.33333333 0.25       0.58333333 0.41666667 0.
 0.25      ]

Authority Score in Iteration 2 :
 [0.         0.2471798  0.49435959 0.57160328 0.13903864 0.40166717
 0.43256464]

Hub Score in Iteration  2 :
 [0.53571547 0.29937041 0.20483238 0.59874082 0.42542111 0.
 0.20483238]

Total number of Iterations are:  2

Authority and Hub score in last iteration are the final scores
```

Q3

In [15]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
```

In [16]:

```
dataset = pd.read_excel('Credit card approval.xls')
```

In [17]:

```
dataset.head()
```

Out[17]:

	A1:	A2:	A3:	A4:	A5:	A6:	A7:	A8:	A9:	A10:	A11:	A12:	A13:	A14:	A15:	A1((c attril
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	202	0	
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	43	560	
2	a	24.5	0.500	u	g	q	h	1.50	t	f	0	f	g	280	824	
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	100	3	
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120	0	

Handle Missing Values

In [18]:

```
dataset.columns = dataset.columns.str.replace(' ', '')
```

In [19]:

```
dataset["A1:"].value_counts()
```

Out[19]:

```
b    468
a    210
?     12
Name: A1:, dtype: int64
```

In [20]:

```
dataset["A2:"].value_counts()
```

Out[20]:

```
?        12
22.67     9
20.42     7
19.17     6
25        6
         ..
39.83     1
35.42     1
42.17     1
17.83     1
56.75     1
Name: A2:, Length: 350, dtype: int64
```

In [21]:

```
dataset["A4:"].value_counts()
```

Out[21]:

```
u    519
y    163
?      6
l      2
Name: A4:, dtype: int64
```

In [22]:

```
dataset["A5:"].value_counts()
```

Out[22]:

```
g     519
p     163
?       6
gg      2
Name: A5:, dtype: int64
```

In [23]:

```
dataset["A6:"].value_counts()
```

Out[23]:

```
c     137
q      78
w      64
i      59
aa     54
ff     53
k      51
cc     41
m      38
x      38
d      30
e      25
j      10
?       9
r       3
Name: A6:, dtype: int64
```

In [24]:

```
dataset["A7:"].value_counts()
```

Out[24]:

```
v     399
h     138
bb     59
ff     57
?       9
z       8
j       8
dd      6
n       4
o       2
Name: A7:, dtype: int64
```

In [25]:

```
#Deleting Null Values
dataset = dataset.replace(to_replace = '?',value = np.nan)
dataset = dataset.dropna()
```

In [ ]:

```
#Encoding Categorical Data
dataset['A1:']=dataset['A1:'].replace(["a","b"],[0,1])
dataset["A4:"]=dataset['A4:'].replace(["u","y","l"],[0,1,2])
dataset["A5:"]=dataset['A5:'].replace(["g","p","gg"],[0,1,2])
dataset["A6:"]=dataset['A6:'].replace(["c","q","w","i","aa","ff","k","cc","m","x","d","e","
dataset["A7:"]=dataset['A7:'].replace(["v","h","bb","ff","z","j","dd","n","o"],[1,2,3,4,5,6
dataset["A9:"]=dataset['A9:'].replace(["f","t"],[0,1])
dataset["A10:"]=dataset['A10:'].replace(["f","t"],[0,1])
dataset["A12:"]=dataset['A12:'].replace(["f","t"],[0,1])
dataset["A13:"]=dataset['A13:'].replace(["g","s","p"],[1,2,3])
dataset["A16:+,-(classattribute)"]=dataset['A16:+,-(classattribute)'].replace(["-","+"],[0,
```

In [27]:

```
dataset.head()
```

Out[27]:

	A1:	A2:	A3:	A4:	A5:	A6:	A7:	A8:	A9:	A10:	A11:	A12:	A13:	A14:	A15:	(cla
0	1	30.83	0.000	0	0	3	1	1.25	1	1	1	0	1	202.0	0	
1	0	58.67	4.460	0	0	2	2	3.04	1	1	6	0	1	43.0	560	
2	0	24.50	0.500	0	0	2	2	1.50	1	0	0	0	1	280.0	824	
3	1	27.83	1.540	0	0	3	1	3.75	1	1	5	1	1	100.0	3	
4	1	20.17	5.625	0	0	3	1	1.71	1	0	0	0	2	120.0	0	

In [28]:

```
X = dataset.iloc[:, [0, 14]].values
y = dataset.iloc[:, 15].values
```

5 Fold Cross Validation

In [29]:

```
from sklearn.model_selection import KFold

kf = KFold(n_splits = 5, shuffle = True, random_state = 2)
result = next(kf.split(dataset), None)

train = dataset.iloc[result[0]]
test =  dataset.iloc[result[1]]

print(train)
print(test)
```

```
     A1:    A2:     A3:  A4:  A5:  A6:  A7:    A8:  A9:  A10:  A11:  A12:  \
0      1  30.83   0.000    0    0    3    1  1.250    1     1     1     0
2      0  24.50   0.500    0    0    2    2  1.500    1     0     0     0
4      1  20.17   5.625    0    0    3    1  1.710    1     0     0     0
5      1  32.08   4.000    0    0    9    1  2.500    1     0     0     1
6      1  33.17   1.040    0    0   14    2  6.500    1     0     0     1
..   ...    ...     ...  ...  ...  ...  ...    ...  ...   ...   ...   ...
683    1  36.42   0.750    1    1   11    1  0.585    0     0     0     0
684    1  40.58   3.290    0    0    9    1  3.500    0     0     0     1
685    1  21.08  10.085    1    1   12    2  1.250    0     0     0     0
687    0  25.25  13.500    1    1    6    4  2.000    0     1     1     1
689    1  35.00   3.375    0    0    1    2  8.290    0     0     0     1

     A13:   A14:   A15:  A16:+,-(classattribute)
0       1  202.0      0                        1
2       1  280.0    824                        1
4       2  120.0      0                        1
5       1  360.0      0                        1
6       1  164.0  31285                        1
..    ...    ...    ...                      ...
683     1  240.0      3                        0
684     2  400.0      0                        0
685     1  260.0      0                        0
687     1  200.0      1                        0
689     1    0.0      0                        0

[522 rows x 16 columns]
     A1:    A2:    A3:  A4:  A5:  A6:  A7:    A8:  A9:  A10:  A11:  A12:  \
1      0  58.67  4.460    0    0    2    2  3.040    1     1     6     0
3      1  27.83  1.540    0    0    3    1  3.750    1     1     5     1
10     1  22.08  0.830    0    0    1    2  2.165    0     0     0     1
13     1  48.08  6.040    0    0    7    1  0.040    0     0     0     0
16     1  28.25  0.875    0    0    9    1  0.960    1     1     3     1
..   ...    ...    ...  ...  ...  ...  ...    ...  ...   ...   ...   ...
659    0  28.58  3.750    0    0    1    1  0.250    0     1     1     1
665    1  31.83  0.040    1    1    9    1  0.040    0     0     0     0
680    1  19.50  0.290    0    0    7    1  0.290    0     0     0     0
686    0  22.67  0.750    0    0    1    1  2.000    0     1     2     1
688    1  17.92  0.205    0    0    5    1  0.040    0     0     0     0

     A13:   A14:  A15:  A16:+,-(classattribute)
1       1   43.0   560                        1
3       1  100.0     3                        1
10      1  128.0     0                        1
13      1    0.0  2690                        1
16      1  396.0     0                        1
..    ...    ...   ...                      ...
659     1   40.0   154                        0
```

```
665      1    0.0     0                        0
680      1  280.0   364                        0
686      1  200.0   394                        0
688      1  280.0   750                        0

[131 rows x 16 columns]
```

In [30]:

```
#Splitting into Traing and Test Sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state =
```

Decision Tree

In [31]:

```
#Training
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', max_depth=3)
classifier.fit(X_train, y_train)
```

Out[31]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [32]:

```
#Predicting the test set results
y_pred = classifier.predict(X_test)
```

In [33]:

```
#Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n",cm)
```

```
Confusion Matrix:
 [[93 17]
 [45 41]]
```

In [34]:

```
#Precision, Recall and F1-Score
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.85      0.75       110
           1       0.71      0.48      0.57        86

    accuracy                           0.68       196
   macro avg       0.69      0.66      0.66       196
weighted avg       0.69      0.68      0.67       196
```

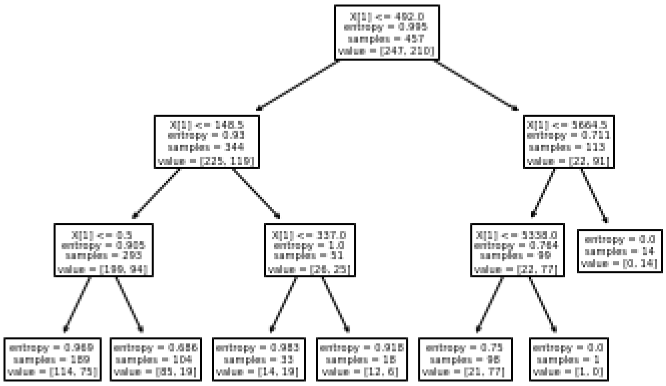
In [35]:

```
#Prediction accuracy
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("Accuracy:",acc)
```

```
Accuracy: 0.6836734693877551
```

In [36]:

```
#Decision Tree
from sklearn import tree
tree.plot_tree(classifier);
```



In [37]:

```
#ROC and AUC Curves
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
y_pred = classifier.predict_proba(X_test)
y_pred = y_pred[:, 1]
fpr, tpr, thresholds = roc_curve(y_test,y_pred)
auc = roc_auc_score(y_test,y_pred)
print('AUC Score: %.2f' % auc)
```

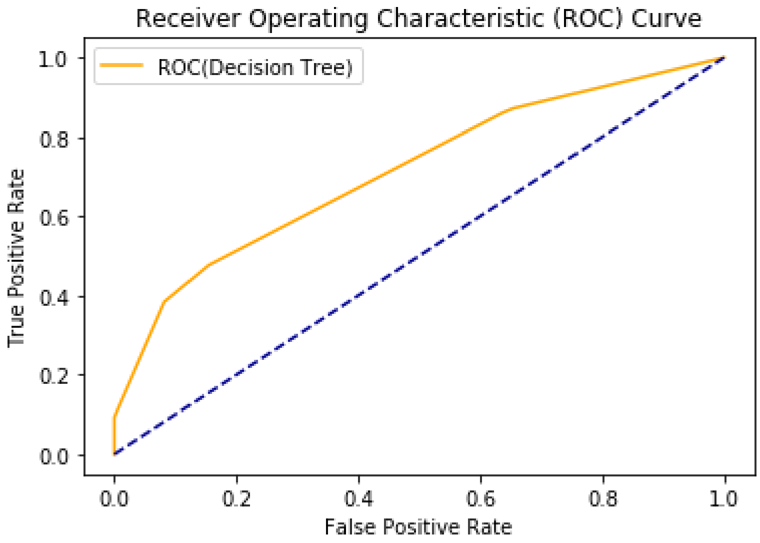
```
AUC Score: 0.71
```

In [38]:

```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC(Decision Tree)')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

In [39]:

```
plot_roc_curve(fpr, tpr)
```



Naive Bayes

In [40]:

```
#Training
from sklearn.naive_bayes import GaussianNB
classifier1 = GaussianNB()
classifier1.fit(X_train, y_train)
```

Out[40]:

```
GaussianNB()
```

In [41]:

```
#Predicting the test set results
y_pred1 = classifier1.predict(X_test)
```

In [42]:

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test, y_pred1)
print("Confusion Matrix:\n",cm1)
```

```
Confusion Matrix:
 [[107   3]
 [ 67  19]]
```

In [43]:

```
#Precision, Recall and F1-Score
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.61      0.97      0.75       110
           1       0.86      0.22      0.35        86

    accuracy                           0.64       196
   macro avg       0.74      0.60      0.55       196
weighted avg       0.72      0.64      0.58       196
```

In [44]:

```
#Prediction Accuracy
from sklearn.metrics import accuracy_score
acc1 = accuracy_score(y_test, y_pred1)
print("Accuracy: ",acc1)
```

```
Accuracy:  0.6428571428571429
```

In [45]:

```
#ROC and AUC Curves
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
y_pred1 = classifier1.predict_proba(X_test)
y_pred1 = y_pred1[:, 1]
fpr1, tpr1, thresholds1 = roc_curve(y_test,y_pred1)
auc1 = roc_auc_score(y_test,y_pred1)
print('AUC Score: %.2f' % auc)
```

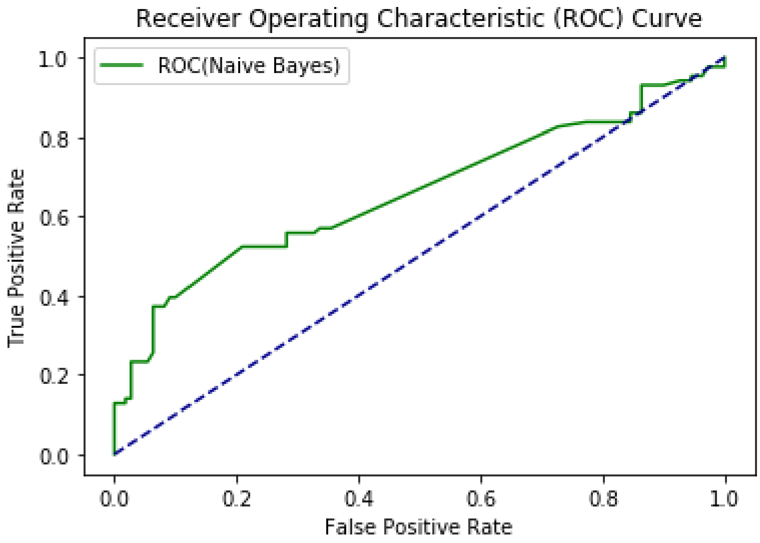
```
AUC Score: 0.71
```

In [46]:

```
def plot_roc_curve(fpr1, tpr1):
    plt.plot(fpr1, tpr1, color='green', label='ROC(Naive Bayes)')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

In [47]:

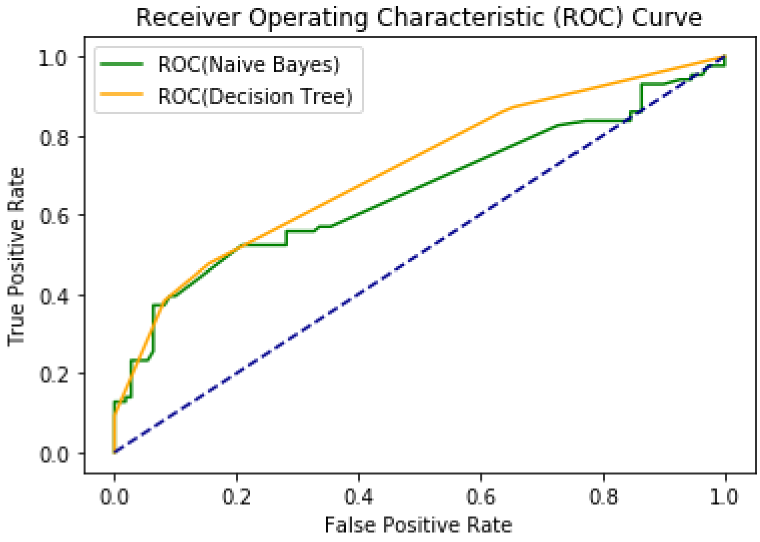
```
plot_roc_curve(fpr1, tpr1)
```



Comparision of ROC Curves of Decision Tree and Naive Bayes

In [48]:

```
plt.plot(fpr1, tpr1, color='green', label='ROC(Naive Bayes)')
plt.plot(fpr, tpr, color='orange', label='ROC(Decision Tree)')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
ROC(Naive Bayes)
ROC(Decision Tree)
True Positive Rate
False Positive Rate