6.Decision Tree - Exercise

November 9, 2021

Classification using Decision Tree - Exercise

[1]:
```
# Import necessary package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

0.0.1 Step 1: Load the dataset

[3]:
```
df = pd.read_csv("E:\\MY LECTURES\\8.2021-09-03 DATA SCIENCE (KNU)\\3.
↪Programs\\dataset\\titanic.csv")
df.head()
```

[3]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[4]:
```
# Extract features only Survived, Pclass, Sex, Age, Fare
filtered_df = df.
    ↪drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked'],axis='columns')
filtered_df.head()
```

[4]:

	Survived	Pclass	Sex	Age	Fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0	3	male	35.0	8.0500

0.0.2 Step 2: Apply EDA

[5]:
```
filtered_df['Survived'].unique()
```

[5]:
```
array([0, 1], dtype=int64)
```

[6]:
```
filtered_df['Pclass'].unique()
```

[6]:
```
array([3, 1, 2], dtype=int64)
```

[7]:
```
filtered_df['Sex'].unique()
```

[7]:
```
array(['male', 'female'], dtype=object)
```

[8]:
```
filtered_df['Age'].unique()
```

[8]:
```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

[9]:
```
filtered_df['Fare'].unique()
```

[9]:
```
array([  7.25  ,  71.2833,   7.925 ,  53.1   ,   8.05  ,   8.4583,
        51.8625,  21.075 ,  11.1333,  30.0708,  16.7   ,  26.55  ,
        31.275 ,   7.8542,  16.    ,  29.125 ,  13.    ,  18.    ,
         7.225 ,  26.    ,   8.0292,  35.5   ,  31.3875, 263.    ,
         7.8792,   7.8958,  27.7208, 146.5208,   7.75  ,  10.5   ,
```

```
 82.1708,  52.    ,   7.2292,  11.2417,   9.475 ,  21.    ,
 41.5792,  15.5   ,  21.6792,  17.8   ,  39.6875,   7.8   ,
 76.7292,  61.9792,  27.75  ,  46.9   ,  80.    ,  83.475 ,
 27.9   ,  15.2458,   8.1583,   8.6625,  73.5   ,  14.4542,
 56.4958,   7.65  ,  29.    ,  12.475 ,   9.    ,   9.5   ,
  7.7875,  47.1   ,  15.85  ,  34.375 ,  61.175 ,  20.575 ,
 34.6542,  63.3583,  23.    ,  77.2875,   8.6542,   7.775 ,
 24.15  ,   9.825 ,  14.4583, 247.5208,   7.1417,  22.3583,
  6.975 ,   7.05  ,  14.5   ,  15.0458,  26.2833,   9.2167,
 79.2   ,   6.75  ,  11.5   ,  36.75  ,   7.7958,  12.525 ,
 66.6   ,   7.3125,  61.3792,   7.7333,  69.55  ,  16.1   ,
 15.75  ,  20.525 ,  55.    ,  25.925 ,  33.5   ,  30.6958,
 25.4667,  28.7125,   0.    ,  15.05  ,  39.    ,  22.025 ,
 50.    ,   8.4042,   6.4958,  10.4625,  18.7875,  31.    ,
113.275 ,  27.    ,  76.2917,  90.    ,   9.35  ,  13.5   ,
  7.55  ,  26.25  ,  12.275 ,   7.125 ,  52.5542,  20.2125,
 86.5   , 512.3292,  79.65  , 153.4625, 135.6333,  19.5   ,
 29.7   ,  77.9583,  20.25  ,  78.85  ,  91.0792,  12.875 ,
  8.85  , 151.55  ,  30.5   ,  23.25  ,  12.35  , 110.8833,
108.9   ,  24.    ,  56.9292,  83.1583, 262.375 ,  14.    ,
164.8667, 134.5   ,   6.2375,  57.9792,  28.5   , 133.65  ,
 15.9   ,   9.225 ,  35.    ,  75.25  ,  69.3   ,  55.4417,
211.5   ,   4.0125, 227.525 ,  15.7417,   7.7292,  12.    ,
120.    ,  12.65  ,  18.75  ,   6.8583,  32.5   ,   7.875 ,
 14.4   ,  55.9   ,   8.1125,  81.8583,  19.2583,  19.9667,
 89.1042,  38.5   ,   7.725 ,  13.7917,   9.8375,   7.0458,
  7.5208,  12.2875,   9.5875,  49.5042,  78.2667,  15.1   ,
  7.6292,  22.525 ,  26.2875,  59.4   ,   7.4958,  34.0208,
 93.5   , 221.7792, 106.425 ,  49.5   ,  71.    ,  13.8625,
  7.8292,  39.6   ,  17.4   ,  51.4792,  26.3875,  30.    ,
 40.125 ,   8.7125,  15.    ,  33.    ,  42.4   ,  15.55  ,
 65.    ,  32.3208,   7.0542,   8.4333,  25.5875,   9.8417,
  8.1375,  10.1708, 211.3375,  57.    ,  13.4167,   7.7417,
  9.4833,   7.7375,   8.3625,  23.45  ,  25.9292,   8.6833,
  8.5167,   7.8875,  37.0042,   6.45  ,   6.95  ,   8.3   ,
  6.4375,  39.4   ,  14.1083,  13.8583,  50.4958,   5.    ,
  9.8458,  10.5167])
```

0.0.3 Step 3. Pre-process and extract the features

[10]:
```
X = filtered_df.drop('Survived',axis='columns')
Y = filtered_df['Survived']
```

[11]:
```
# Input feature set
X.head(10)
```

[11]:

	Pclass	Sex	Age	Fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500
5	3	male	NaN	8.4583
6	1	male	54.0	51.8625
7	3	male	2.0	21.0750
8	3	female	27.0	11.1333
9	2	female	14.0	30.0708

[12]:

```
# Do lable encoding
X.Sex = X.Sex.map({'male': 1, 'female': 2})
X.head()
```

[12]:

	Pclass	Sex	Age	Fare
0	3	1	22.0	7.2500
1	1	2	38.0	71.2833
2	3	2	26.0	7.9250
3	1	2	35.0	53.1000
4	3	1	35.0	8.0500

[13]:

```
# Treat NaN in age feature
X.Age = X.Age.fillna(X.Age.mean())
X.head(10)
```

[13]:

	Pclass	Sex	Age	Fare
0	3	1	22.000000	7.2500
1	1	2	38.000000	71.2833
2	3	2	26.000000	7.9250
3	1	2	35.000000	53.1000
4	3	1	35.000000	8.0500
5	3	1	29.699118	8.4583
6	1	1	54.000000	51.8625
7	3	1	2.000000	21.0750
8	3	2	27.000000	11.1333
9	2	2	14.000000	30.0708

[14]:

```
# Output feature
Y.head()
```

[14]:

```
0    0
1    1
2    1
3    1
4    0
```

```
Name: Survived, dtype: int64
```

0.0.4 Step 4. Split the data for training and testing

[15]:
```
# Splitting dataset into training and testing set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 0)
```

0.0.5 Step 5. Training the model

[16]:
```
from sklearn import tree
DT_model = tree.DecisionTreeClassifier()
```

[17]:
```
DT_model.fit(x_train, y_train)
```

[17]:
```
DecisionTreeClassifier()
```

[18]:
```
y_train_pred = DT_model.predict(x_train)
```

[19]:
```
train_predicted_prob = DT_model.predict_proba(x_train)
train_predicted_prob
```

[19]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       …,
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

Performance score for logistic regression

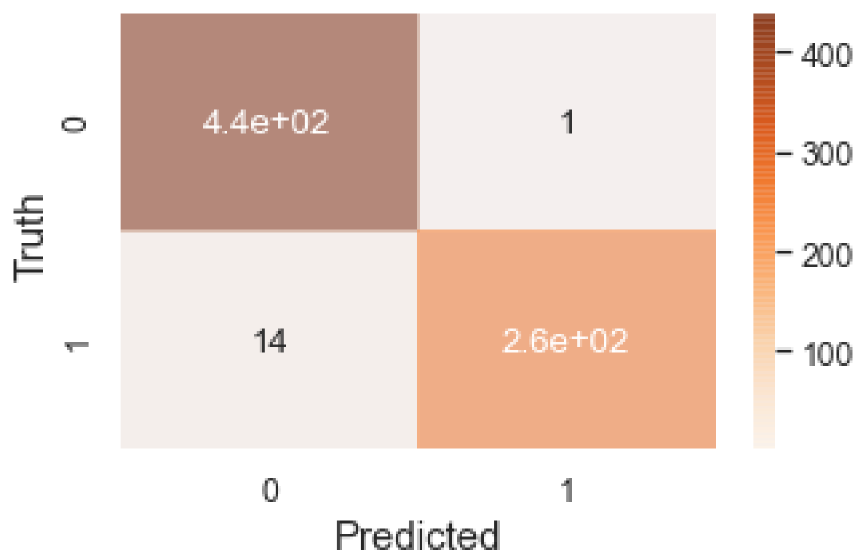
[20]:
```
out = DT_model.score(x_train, y_train)
DT_Train_RS = np.round(out,2)*100
print("Performance score for training set :",DT_Train_RS,"%")
```

```
Performance score for training set : 98.0 %
```

Confusion matrix R2 score says the performance of logistic regression over simple probability that does not feature Age. We are interested to know how many has been correctly and wrongly classified.

[21]:
```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train,y_train_pred)

plt.figure(figsize = (5,3))
sns.set(font_scale=1.1)

axes = plt.gca()
axes.xaxis.label.set_size(15)
axes.yaxis.label.set_size(15)

sns.heatmap(cm, annot=True,cmap=plt.cm.Oranges, alpha=0.5)

plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[21]: Text(19.5, 0.5, 'Truth')



Precison, Recall, F1, Accuracy

[22]:
```python
# Total report
from sklearn import metrics
print(metrics.classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       439
           1       1.00      0.95      0.97       273
```

```
    accuracy                           0.98       712
   macro avg       0.98      0.97      0.98       712
weighted avg       0.98      0.98      0.98       712
```

[23]:

```
# Accuracy score
temp = metrics.accuracy_score(y_train,y_train_pred)
DT_Train_Accuracy = np.round(temp,2)*100
print("Accuracy score : ",DT_Train_Accuracy,"%")
```

```
Accuracy score :  98.0 %
```

[24]:

```
# Precision score
temp = metrics.precision_score(y_train,y_train_pred)
DT_Train_Precision = np.round(temp,2)*100
print("Precision score : ",DT_Train_Precision,"%")
```

```
Precision score :  100.0 %
```

[25]:

```
# Recall score
temp = metrics.recall_score(y_train,y_train_pred)
DT_Train_Recall = np.round(temp,2)*100
print("Recall score : ",DT_Train_Recall,"%")
```

```
Recall score :  95.0 %
```

[26]:

```
# F1 score
temp = metrics.f1_score(y_train,y_train_pred)
DT_Train_F1 = np.round(temp,2)*100
print("F1 score : ",DT_Train_F1,"%")
```

```
F1 score :  97.0 %
```

[27]:

```
# Cohen Kappa score
temp = metrics.cohen_kappa_score(y_train,y_train_pred)
DT_Train_CK = np.round(temp,2)*100
print("Cohen Kappa score : ",DT_Train_CK,"%")
```

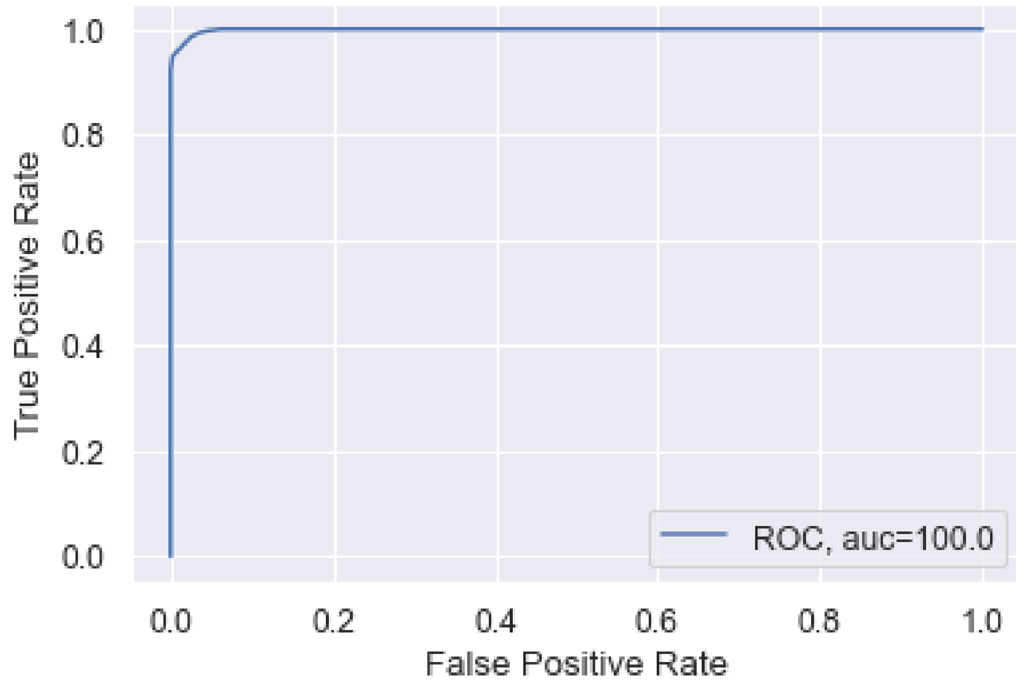
```
Cohen Kappa score :  96.0 %
```

ROC

[28]:

```
prob = train_predicted_prob[::,1]
fpr, tpr, _ = metrics.roc_curve(y_train, prob)
DT_Train_AUC = np.round(metrics.roc_auc_score(y_train, prob),2)*100
plt.plot(fpr,tpr,label="ROC, auc="+str(DT_Train_AUC))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.legend(loc=4)
plt.show()
```



0.0.6 Step 6. Testing the model

[29]:
```
y_test_pred = DT_model.predict(x_test)
y_test_pred
```

[29]:
```
array([0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 1], dtype=int64)
```

[30]:
```
test_predicted_prob = DT_model.predict_proba(x_test)
```

Performance score for logistic regression

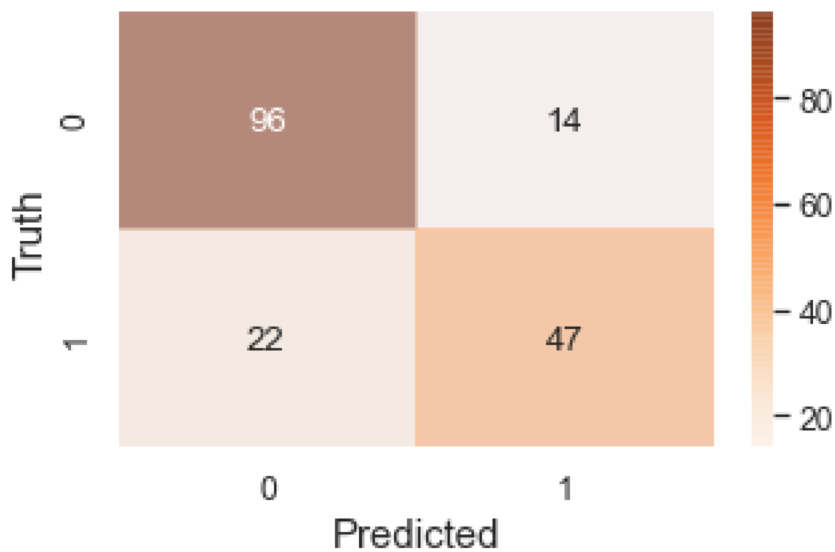
[31]:
```
out = DT_model.score(x_test, y_test)
DT_Test_RS = np.round(out,2)*100
print("Performance score for training set :",DT_Test_RS,"%")
```

```
Performance score for training set : 80.0 %
```

Confusion matrix R2 score says the performance of logistic regression over simple probability that does not feature Age. We are interested to know how many has been correctly and wrongly classified.

[32]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_test_pred)

plt.figure(figsize = (5,3))
sns.set(font_scale=1.1)

axes = plt.gca()
axes.xaxis.label.set_size(15)
axes.yaxis.label.set_size(15)

sns.heatmap(cm, annot=True,cmap=plt.cm.Oranges, alpha=0.5)

plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[32]:
```
Text(19.5, 0.5, 'Truth')
```



Precison, Recall, F1, Accuracy

[33]:
```
# Total report
from sklearn import metrics
print(metrics.classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       110
           1       0.77      0.68      0.72        69

    accuracy                           0.80       179
   macro avg       0.79      0.78      0.78       179
weighted avg       0.80      0.80      0.80       179
```

[34]:
```
# Accuracy score
temp = metrics.accuracy_score(y_test,y_test_pred)
DT_Test_Accuracy = np.round(temp,2)*100
print("Accuracy score : ",DT_Test_Accuracy,"%")
```

```
Accuracy score :  80.0 %
```

[35]:
```
# Precision score
temp = metrics.precision_score(y_test,y_test_pred)
DT_Test_Precision = np.round(temp,2)*100
print("Precision score : ",DT_Test_Precision,"%")
```

```
Precision score :  77.0 %
```

[36]:
```
# Recall score
temp = metrics.recall_score(y_test,y_test_pred)
DT_Test_Recall = np.round(temp,2)*100
print("Recall score : ",DT_Test_Recall,"%")
```

```
Recall score :  68.0 %
```

[37]:
```
# F1 score
temp = metrics.f1_score(y_test,y_test_pred)
DT_Test_F1 = np.round(temp,2)*100
print("F1 score : ",DT_Test_F1,"%")
```

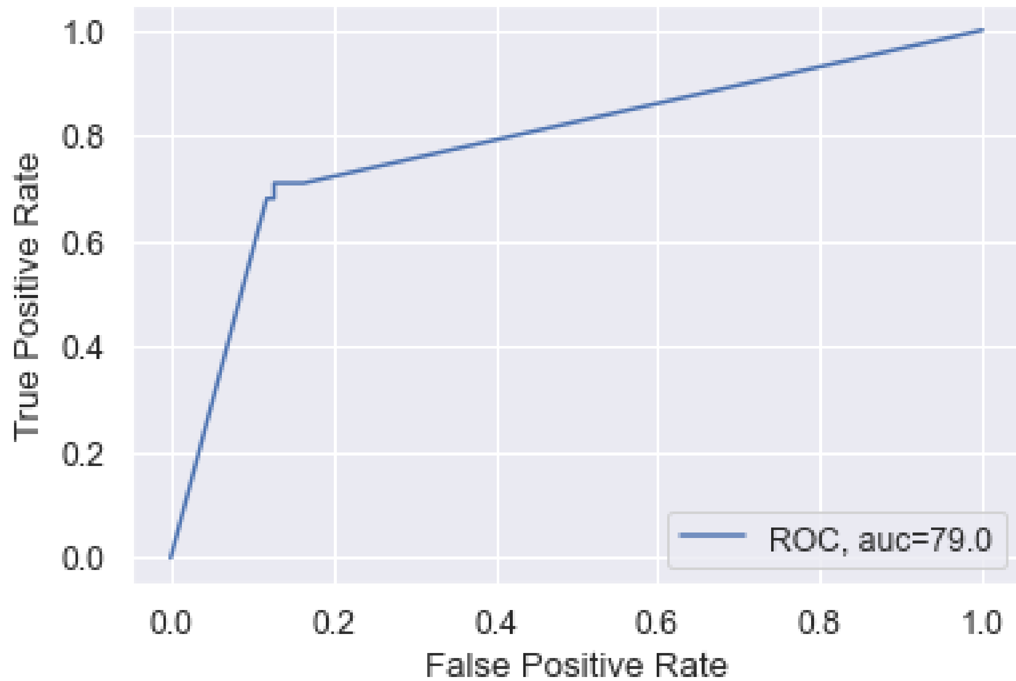
```
F1 score :  72.0 %
```

[38]:
```
# Cohen Kappa score
temp = metrics.cohen_kappa_score(y_test,y_test_pred)
DT_Test_CK = np.round(temp,2)*100
print("Cohen Kappa score : ",DT_Test_CK,"%")
```

```
Cohen Kappa score :  56.99999999999999 %
```

ROC

[39]:
```
prob = test_predicted_prob[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, prob)
DT_Test_AUC = np.round(metrics.roc_auc_score(y_test, prob),2)*100
plt.plot(fpr,tpr,label="ROC, auc="+str(DT_Test_AUC))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



0.0.7 Step 7. Prediction using the model

For Pclass: 1, Sex: 0, Age: 43, Fare: 50, passenger survived?

[42]:
```
DT_model.predict([[1,0,43,50]])
```

[42]:
```
array([0], dtype=int64)
```

0.0.8 Step 8. Summary

[43]:
```
print("              Decision Tree      ")
print("=====================================================")
print("\t\tTraining phase       Testing phase ")
print("=====================================================")
print("RS\t\t   ",DT_Train_RS,"%\t\t", DT_Test_RS,"%")
print("Accuracy\t   ",DT_Train_Accuracy,"%\t\t", DT_Test_Accuracy,"%")
print("Precision\t   ",DT_Train_Precision,"%\t\t", DT_Test_Precision,"%")
print("Recall\t\t   ",DT_Train_Recall,"%\t\t", DT_Test_Recall,"%")
print("F1\t\t   ",DT_Train_F1,"%\t\t", DT_Test_F1,"%")
print("CK\t\t   ",DT_Train_CK,"%\t\t", DT_Test_CK,"%")
print("AUC\t\t   ",DT_Train_AUC,"%\t\t", DT_Test_AUC,"%")
print("=====================================================")
```

```
             Decision Tree
=====================================================
                Training phase       Testing phase
=====================================================
RS                  98.0 %              80.0 %
Accuracy            98.0 %              80.0 %
Precision           100.0 %             77.0 %
Recall              95.0 %              68.0 %
F1                  97.0 %              72.0 %
CK                  96.0 %              56.99999999999999 %
AUC                 100.0 %             79.0 %
=====================================================
```