Ex No. 4	**Measuring the Performance of Machine Learning Model**
Date:	

Aim

To measure the performance of machine learning models in python.

Definition

Accuracy

Accuracy is what its literal meaning says, a measure of how accurate your model is.

Accuracy = Correct Predictions / Total Predictions

By using confusion matrix, Accuracy = (TP + TN)/(TP+TN+FP+FN)
Precision & Recall

Precision: It is the ratio of True Positives (TP) and the total positive predictions. Basically, it tells us how many times your positive prediction was actually positive.

$$\text{Precision} = \frac{tp}{tp + fp}$$

Recall : It is nothing but TPR (True Positive Rate explained above). It tells us about out of all the positive points how many were predicted positive.

$$\textbf{Recall} = \frac{tp}{tp + fn}$$

F-Measure: Harmonic mean of precision and recall.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Confusion Matrix

A **confusion matrix** is a correlation between the predictions of a model and the actual class labels of the data points.

Procedure

Open PyCharm Community Edition.

Go to File menu → New Project → Specify the project name → Press “Create” button.

Right Click on Project name → New → Python File → Specify the file name → Press Enter.

Type the following codes. Right click on file name or coding window → Select “Run” to view the result.

1. **R – Squared**
   Rsquared.py

```
from sklearn.metrics import r2_score

### Assume y is the actual value and f is the predicted values
y =[10, 20, 30]
f =[10, 20, 30]
r2 = r2_score(y, f)
print('r2 score for perfect model is', r2)

### Assume y is the actual value and f is the predicted values
y =[10, 20, 30]
f =[20, 20, 20]
r2 = r2_score(y, f)
print('r2 score for a model which predicts mean value always is', r2)

### Assume y is the actual value and f is the predicted values
y = [10, 20, 30]
f = [30, 10, 20]
r2 = r2_score(y, f)
print('r2 score for a worse model is', r2)
```

Output

```
r2 score for perfect model is 1.0
r2 score for a model which predicts mean value always is 0.0
r2 score for a worse model is -2.0
```

2. **Adjusted R-Squared**
   adjRsquared.py

```
from sklearn.linear_model import LinearRegression

import pandas as pd
```

```
#define URL where dataset is located

url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

#read in data

data = pd.read_csv(url)

#fit regression model

model = LinearRegression()

X, y = data[["mpg", "wt", "drat", "qsec"]], data.hp

model.fit(X, y)

#display adjusted R-squared

print(1 - (1-model.score(X, y))*(len(y)-1)/(len(y)-X.shape[1]-1))
```

Output
0.7787005290062519

3. **Mean Absolute Error**
   MeanAbsError.py

```
# Python program for calculating Mean Absolute

# Error using sklearn

# import the module

from sklearn.metrics import mean_absolute_error as mae

# list of integers of actual and calculated

actual = [2, 3, 5, 5, 9]

calculated = [3, 3, 8, 7, 6]

# calculate MAE

error = mae(actual, calculated)
```

```
# display

print("Mean absolute error : " + str(error))
```

Output

Mean absolute error : 1.8

4. **Mean Squared Error**
   MeanSqaError.py

```
from sklearn.metrics import mean_squared_error


# Given values

Y_true = [1,1,2,2,4] # Y_true = Y (original values)


# calculated values

Y_pred = [0.6,1.29,1.99,2.69,3.4] # Y_pred = Y'


# Calculation of Mean Squared Error (MSE)

MSE = mean_squared_error(Y_true,Y_pred)


print(MSE)
```

Output

0.21606

5. **Confusion Matrix and Related Metrics**
   confusionmatrix.py

```
import matplotlib.pyplot as plt

import numpy

from sklearn import metrics

actual = numpy.random.binomial(1,.9,size = 1000)

predicted = numpy.random.binomial(1,.9,size = 1000)
```

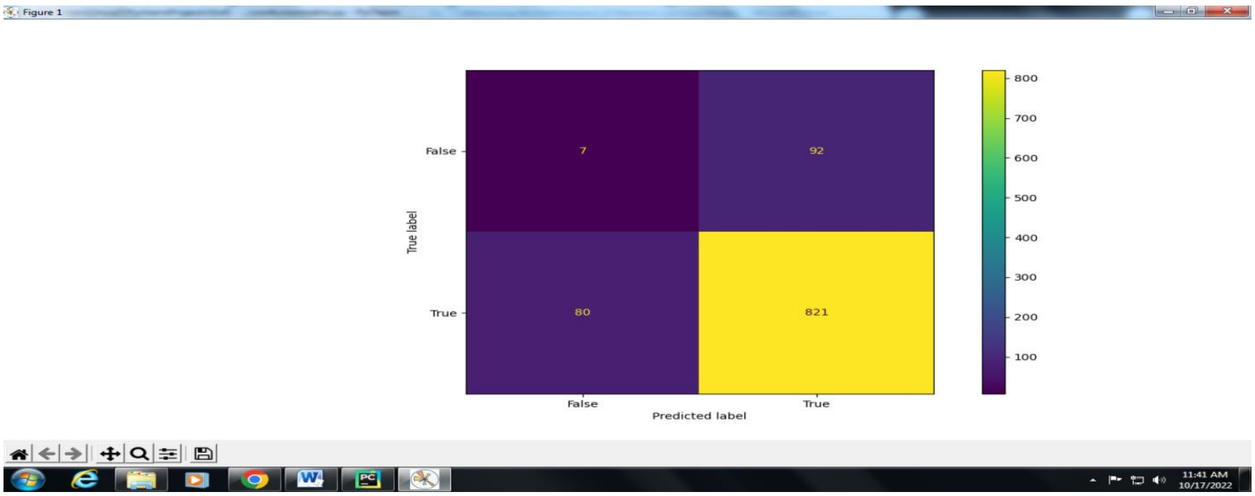
```
confusion_matrix = metrics.confusion_matrix(actual, predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])

cm_display.plot()

plt.show()
```

Output



relatedmetrics.py

```
import numpy

from sklearn import metrics


actual = numpy.random.binomial(1,.9,size = 1000)

predicted = numpy.random.binomial(1,.9,size = 1000)


Accuracy = metrics.accuracy_score(actual, predicted)

Precision = metrics.precision_score(actual, predicted)

Sensitivity_recall = metrics.recall_score(actual, predicted)

Specificity = metrics.recall_score(actual, predicted, pos_label=0)

F1_score = metrics.f1_score(actual, predicted)
```

```
#metrics:

print({"Accuracy":Accuracy,"Precision":Precision,"Sensitivity_recall":Sensitivity_recall,"Specif
icity":Specificity,"F1_score":F1_score})
```

Output
{'Accuracy': 0.836, 'Precision': 0.9074889867841409, 'Sensitivity_recall': 0.911504424778761, 'Specificity': 0.125, 'F1_score': 0.9094922737306844}

6. **F1-Score**
   F1score.py

```
import numpy as np

from sklearn.metrics import f1_score

#define array of actual classes

actual = np.repeat([1, 0], repeats=[160, 240])

#define array of predicted classes

pred = np.repeat([1, 0, 1, 0], repeats=[120, 40, 70, 170])

#calculate F1 score

print(f1_score(actual, pred))
```

Output
0.6857142857142857

7. **AUC-ROC Curve**
   AUCROC.py

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline

#
# Load the breast cancer data set
#
```

```
bc = datasets.load_breast_cancer()
X, y = bc.data, bc.target
#
# Create training and test split
#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1,
stratify=y)
#
# Create the estimator - pipeline
#
pipeline = make_pipeline(StandardScaler(), LogisticRegression(random_state=1))
#
# Create training test splits using two features
#
pipeline.fit(X_train[:, [2, 13]], y_train)
probs = pipeline.predict_proba(X_test[:, [2, 13]])
fpr1, tpr1, thresholds = roc_curve(y_test, probs[:, 1], pos_label=1)
roc_auc1 = auc(fpr1, tpr1)
#
# Create training test splits using two different features
#
pipeline.fit(X_train[:, [4, 14]], y_train)
probs2 = pipeline.predict_proba(X_test[:, [4, 14]])
fpr2, tpr2, thresholds = roc_curve(y_test, probs2[:, 1], pos_label=1)
roc_auc2 = auc(fpr2, tpr2)
#
# Create training test splits using all features
#
pipeline.fit(X_train, y_train)
probs3 = pipeline.predict_proba(X_test)
fpr3, tpr3, thresholds = roc_curve(y_test, probs3[:, 1], pos_label=1)
roc_auc3 = auc(fpr3, tpr3)

fig, ax = plt.subplots(figsize=(7.5, 7.5))

plt.plot(fpr1, tpr1, label='ROC Curve 1 (AUC = %0.2f)' % (roc_auc1))
plt.plot(fpr2, tpr2, label='ROC Curve 2 (AUC = %0.2f)' % (roc_auc2))
plt.plot(fpr3, tpr3, label='ROC Curve 3 (AUC = %0.2f)' % (roc_auc3))
plt.plot([0, 1], [0, 1], linestyle='--', color='red', label='Random Classifier')
plt.plot([0, 0, 1], [0, 1, 1], linestyle=':', color='green', label='Perfect Classifier')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.legend(loc="lower right")
plt.show()
```

Output

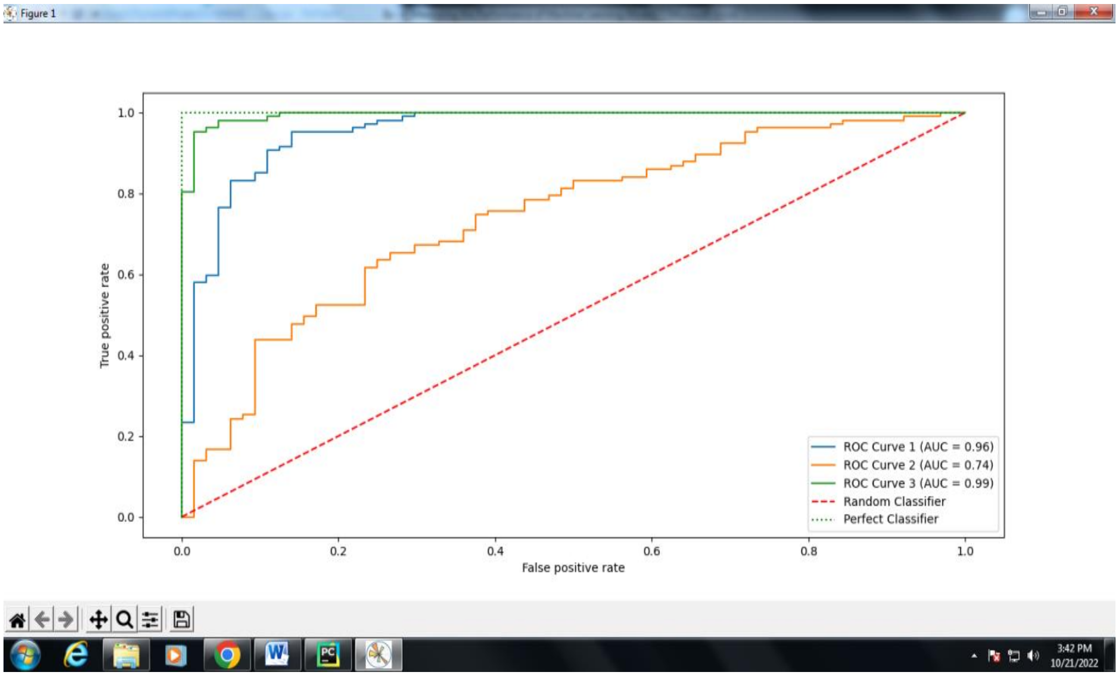


Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
ROC Curve 1 (AUC = 0.96)
ROC Curve 2 (AUC = 0.74)
ROC Curve 3 (AUC = 0.99)
Random Classifier
Perfect Classifier
3:42 PM
10/21/2022

Result

Thus, the performance of machine learning models has been measured successfully.