DATE :

EXPERIMENT – 11

Aim :

Interpret and analyze the performance measures of different classification algorithms for the same dataset.

Requirements :

Importing the libraries

1. import numpy as nm
2. Python
3. Scikit-learn

Procedure :

1. Install Python and the required libraries (scikit-learn and NumPy).
2. Save the provided code in a Python file (e.g., "classification_analysis.py") in the same directory as the iris dataset.
3. Open a terminal or command prompt and navigate to the directory where the Python file is saved.
4. Run the code using the command: python classification_analysis.py
5. Analyze the printed performance measures, including accuracy, precision, recall, F1 score, and AUC-ROC score, for each classifier.
6. Examine the generated confusion matrix and classification report for each classifier to gain further insights into their performance on the iris dataset.

Code :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix, classification_report
# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize and train the classifiers
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
# Make predictions on the testing set
y_pred_logreg = logreg.predict(X_test)
y_pred_dt = dt.predict(X_test)
y_pred_rf = rf.predict(X_test)
# Calculate performance measures
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_logreg = precision_score(y_test, y_pred_logreg, average='weighted')
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
```

```
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_logreg = recall_score(y_test, y_pred_logreg, average='weighted')
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')
f1_logreg = f1_score(y_test, y_pred_logreg, average='weighted')
f1_dt = f1_score(y_test, y_pred_dt, average='weighted')
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')
auc_logreg = roc_auc_score(y_test, y_pred_logreg, average='weighted',
multi_class='ovr') auc_dt = roc_auc_score(y_test, y_pred_dt, average='weighted',
multi_class='ovr')
auc_rf = roc_auc_score(y_test, y_pred_rf, average='weighted',
multi_class='ovr') # Print the performance measures
print("Performance measures for Logistic Regression:")
print("Accuracy:", accuracy_logreg)
print("Precision:", precision_logreg)
print("Recall:", recall_logreg)
print("F1 Score:", f1_logreg)
print("AUC-ROC Score:", auc_logreg)
print()
print("Performance measures for Decision Tree:")
print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("F1 Score:", f1_dt)
print("AUC-ROC Score:", auc_dt)
print()
print("Performance measures for Random Forest:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
```

```
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
print("AUC-ROC Score:", auc_rf)
print()
# Generate confusion matrix and classification report
print("Confusion Matrix for Logistic Regression:")
print(confusion_matrix(y_test, y_pred_logreg))
print()
print("Confusion Matrix for Decision Tree:")
print(confusion_matrix(y_test, y_pred_dt))
print()
print("Confusion Matrix for Random Forest:")
print(confusion_matrix(y_test, y_pred_rf))
print()
print("Classification Report for Logistic Regression:")
print(classification_report(y_test, y_pred_logreg))
print()
print("Classification Report for Decision Tree:")
print(classification_report(y_test, y_pred_dt))
print()
print("Classification Report for Random Forest:")
print(classification_report(y_test, y_pred_rf))
```

OUTPUT :

```
Performance measures for Logistic Regression:
Accuracy: 0.9666666666666667
Precision: 0.970522792022792
Recall: 0.9666666666666667
```

F1 Score: 0.9665831244778613

AUC-ROC Score: 0.9816666666666667

Performance measures for Decision Tree:

Accuracy: 1.0

Precision: 1.0

Recall: 1.0

F1 Score: 1.0

AUC-ROC Score: 1.0

Performance measures for Random Forest:

Accuracy: 0.9666666666666667

Precision: 0.9722222222222222

Recall: 0.9666666666666667

F1 Score: 0.9665831244778613

AUC-ROC Score: 0.9816666666666667

Confusion Matrix for Logistic Regression:

[[10 0 0]

[0 9 1]

[0 0 10]]

Confusion Matrix for Decision Tree:

[[10 0 0]

[0 10 0]

[0 0 10]]

Confusion Matrix for Random Forest:

[[10 0 0]

[0 9 1]

[0 0 10]]

Classification Report for Logistic Regression:

precision recall f1-score support

0 1.00 1.00 1.00 10

1 1.00 0.90 0.95 10

2 0.91 1.00 0.95 10

accuracy 0.97 30

macro avg 0.97 0.97 0.97 30

weighted avg 0.97 0.97 0.97 30

Classification Report for Decision Tree:

precision recall f1-score support

0 1.00 1.00 1.00 10

1 1.00 1.00 1.00 10

2 1.00 1.00 1.00 10

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

Classification Report for Random Forest:

precision recall f1-score support

0 1.00 1.00 1.00 10

1 1.00 0.90 0.95 10

2 0.91 1.00 0.95 10

accuracy 0.97 30

macro avg 0.97 0.97 0.97 30

weighted avg 0.97 0.97 0.97 30

Result :

The above program is successfully executed. The provided code analyzes the performance of different classification algorithms on the iris dataset. It calculates accuracy, precision, recall, F1 score, and AUC- ROC score for each classifier. The confusion matrix shows true positives, true negatives, false positives, and false negatives, while the classification report provides precision, recall, F1 score, and support for each class. This analysis helps evaluate the effectiveness of the algorithms in classifying instances.

DATE :

EXPERIMENT – 12

Aim :

Write a program to implement the K-mean, Hierarchial Clustering for the given dataset and compute the accuracy of the model and compare.

Requirements :

1. scikit-learn - used to load the iris dataset.
2. numpy - handling arrays.
3. matplotlib - to create scatter plots and visualizing.
4. seaborn - to set the plot style and color palette.

Procedure :

STEP 1: Import the scikit-learn, numpy, matplotlib, and seaborn libraries.
STEP 2: Load the Iris dataset using the 'load-iris' function from scikit-learn.
STEP 3: Apply the K-means hierarchial clustering algorithms on the dataset using appropriate functions.
STEP 4: Compute the accuracy of the clustering models using the 'accuracy_score' function from scikit-learn.
STEP 5: Visualize the clusters and ploting the axes labels, titles and color Palettes.

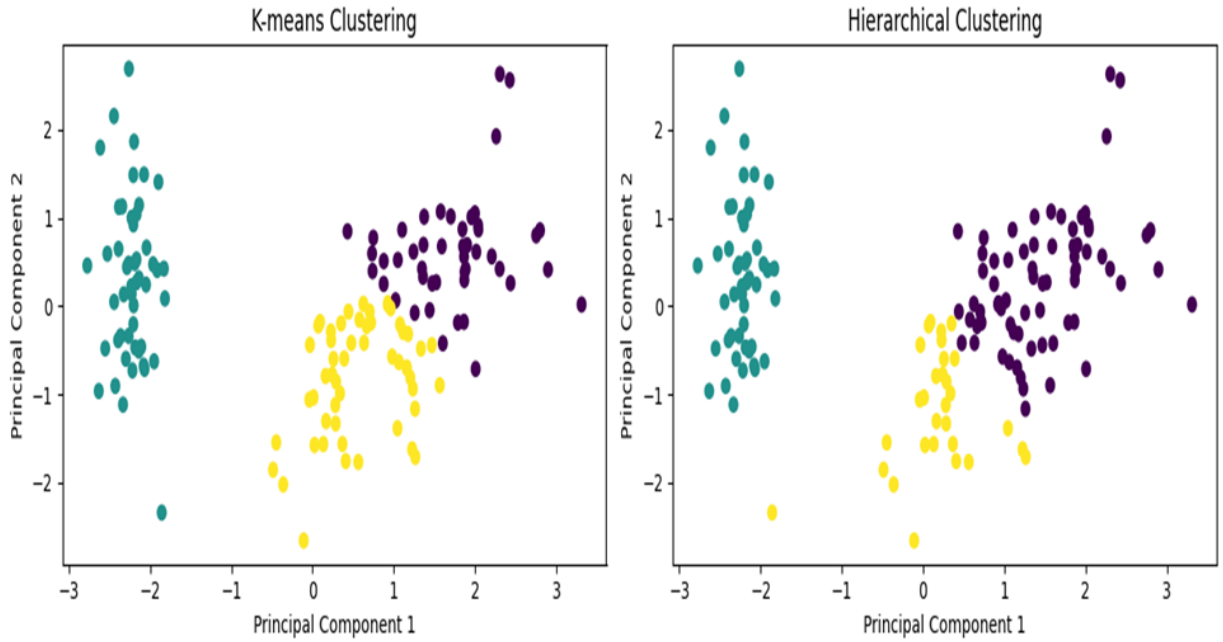
Code :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans, AgglomerativeClustering
```

```
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
iris = load_iris()
X = iris.data
y = iris.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
kmeans_labels = kmeans.labels_
hierarchical = AgglomerativeClustering(n_clusters=3)
hierarchical.fit(X_scaled)
hierarchical_labels = hierarchical.labels_
kmeans_accuracy = accuracy_score(y, kmeans_labels)
hierarchical_accuracy = accuracy_score(y, hierarchical_labels)
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis')
plt.title('K-means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=hierarchical_labels, cmap='viridis')
plt.title('Hierarchical Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
plt.tight_layout()
plt.show()
print("K-means Accuracy:", kmeans_accuracy)
print("Hierarchical Accuracy:", hierarchical_accuracy)
```

OUTPUT :



Result :

Therefore, the code applies K-means and hierarchial clustering on the Iris dataset, computes the accuracy of the models and visualizes the clusters using scatter plots.