**Lab 3**

**Classification**

1. Write a program to implement the decision tree induction, test the model, and show the evaluation report.

Decision tree induction is a supervised learning technique used for classification tasks, where a model is trained to make predictions based on feature values. This program implements a decision tree classifier, tests the model on a dataset, and provides an evaluation report showing metrics like accuracy, precision, recall, and F1-score. It serves as a fundamental tool for decision-making and predictive analysis.

Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

from sklearn import tree

# Load the dataset (Pima Indians Diabetes Database)

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=columns)

# Prepare the input and output

spltd\_data = data.values

X = spltd\_data[:, 0:8] # Features (all except 'Outcome')

Y = spltd\_data[:, 8] # Target ('Outcome')

# Split the dataset into training and testing sets (80-20 split)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Initialize and train the Decision Tree classifier with specific parameters

algo = DecisionTreeClassifier(criterion="entropy", random\_state=42, max\_depth=4, min\_samples\_leaf=5)

algo.fit(X\_train, Y\_train)

# Model accuracy

result = algo.score(X\_test, Y\_test)

print("The Decision Tree model has given Accuracy of: %.3f%%" % (result \* 100.0))

# Visualize the Decision Tree with attribute names and decisions at leaf nodes

plt.figure(figsize=(30, 30))

tree.plot\_tree(algo, filled=True, feature\_names=columns[:-1], class\_names=["No Diabetes", "Diabetes"],

rounded=True, precision=2)

plt.show()

# Make predictions on the test set

Y\_pred = algo.predict(X\_test)

print("Predictions:", Y\_pred)

# Confusion Matrix

cf\_mtrx = metrics.confusion\_matrix(Y\_test, Y\_pred)

cm\_display = ConfusionMatrixDisplay(confusion\_matrix=cf\_mtrx, display\_labels=["No Diabetes", "Diabetes"])

cm\_display.plot()

plt.show()

# Classification report

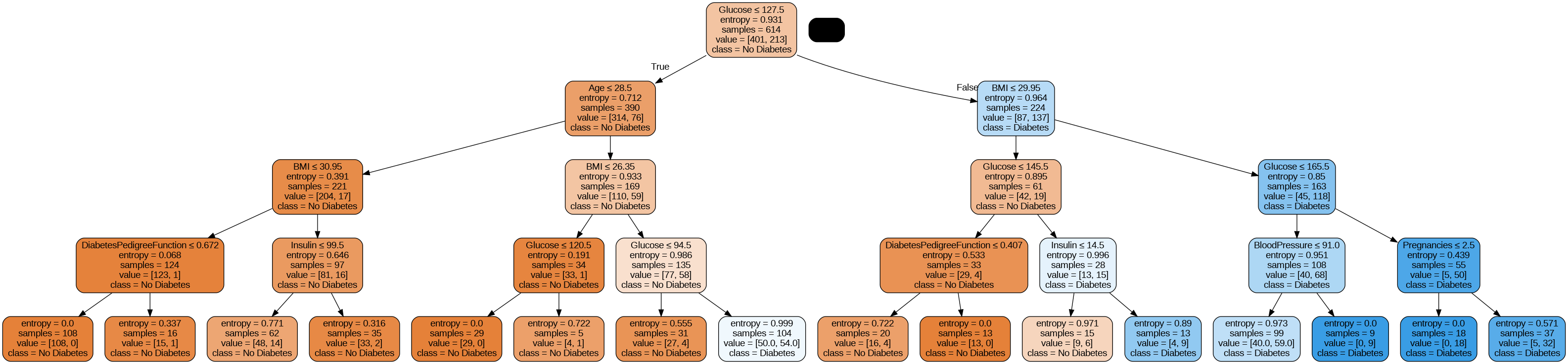
print(classification\_report(Y\_test, Y\_pred))

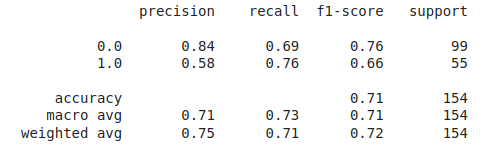
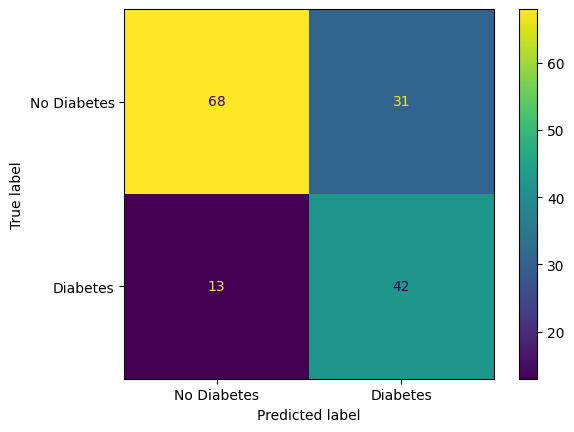
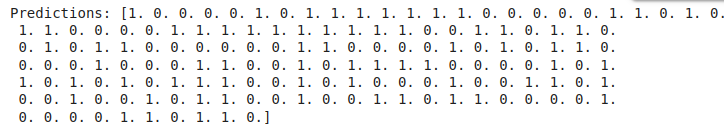
Explanation:

This code builds and evaluates a Decision Tree classifier on the Pima Indians Diabetes dataset. After loading the dataset and preparing the features (X) and target (Y), it splits the data into training and testing sets (80% for training, 20% for testing). The Decision Tree classifier is trained using the entropy criterion, with a maximum depth of 4 and a minimum leaf sample size of 5. The model's accuracy is evaluated on the test set, and the decision tree is visualized with labeled features and class predictions. Predictions are made on the test set, followed by the display of a confusion matrix and a detailed classification report to assess the model’s performance.

Output:







1. Write a program to implement the Naïve Bayes classifier, test the model, and show the evaluation report.

The Naïve Bayes classifier is a probabilistic model used for classification tasks based on Bayes' theorem, assuming independence between features. This program implements a Naïve Bayes classifier, tests it on a sample dataset, and generates an evaluation report that includes accuracy, precision, recall, and F1-score. It is widely used due to its simplicity and efficiency in handling large datasets.

Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

# Load the dataset (Pima Indians Diabetes Database)

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=columns)

# Prepare the input and output

spltd\_data = data.values

X = spltd\_data[:, 0:8] # Features (all except 'Outcome')

Y = spltd\_data[:, 8] # Target ('Outcome')

# Split the dataset into training and testing sets (80-20 split)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Initialize and train the Naïve Bayes classifier

nb\_classifier = GaussianNB()

nb\_classifier.fit(X\_train, Y\_train)

# Model accuracy

result = nb\_classifier.score(X\_test, Y\_test)

print("The Naïve Bayes model has given Accuracy of: %.3f%%" % (result \* 100.0))

# Make predictions on the test set

Y\_pred = nb\_classifier.predict(X\_test)

print("Predictions:", Y\_pred)

# Confusion Matrix

cf\_mtrx = metrics.confusion\_matrix(Y\_test, Y\_pred)

cm\_display = ConfusionMatrixDisplay(confusion\_matrix=cf\_mtrx, display\_labels=[False, True])

cm\_display.plot()

plt.show()

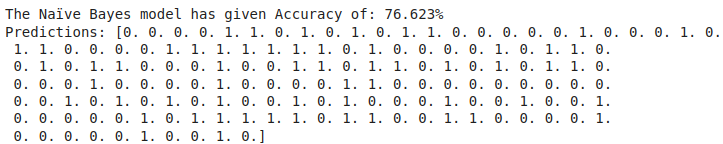
# Classification report

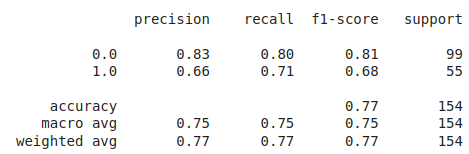
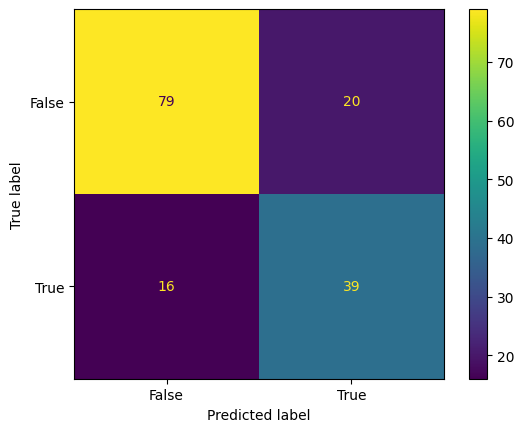
print(classification\_report(Y\_test, Y\_pred))

Explanation:

This code trains and evaluates a Naïve Bayes classifier using the Pima Indians Diabetes dataset. After loading the dataset and preparing the feature variables (X) and target variable (Y), it splits the data into training and testing sets (80% training and 20% testing). The GaussianNB classifier is used, which assumes that the features follow a Gaussian distribution. The accuracy of the model is evaluated on the test set, and predictions are made. A confusion matrix is displayed to visualize true vs. predicted classifications, and a classification report is generated to assess performance, including metrics like precision, recall, and F1-score.

Output:





1. Write a program to implement the Back propagation algorithm, test the model, and show the evaluation report.

The Backpropagation algorithm is a supervised learning technique used for training artificial neural networks. It involves adjusting weights based on the error between the predicted and actual outputs, typically using gradient descent. This program implements the Backpropagation algorithm, tests the model on a dataset, and provides an evaluation report to assess its performance with metrics such as accuracy, precision, recall, and F1-score.

Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neural\_network import MLPClassifier

from sklearn import metrics

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

# Load the dataset (Pima Indians Diabetes Database)

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=columns)

# Prepare the input and output

spltd\_data = data.values

X = spltd\_data[:, 0:8] # Features (all except 'Outcome')

Y = spltd\_data[:, 8] # Target ('Outcome')

# Split the dataset into training and testing sets (80-20 split)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Initialize and train the MLP classifier (Backpropagation)

mlp\_classifier = MLPClassifier(hidden\_layer\_sizes=(10, 10), max\_iter=1000, random\_state=42)

mlp\_classifier.fit(X\_train, Y\_train)

# Model accuracy

result = mlp\_classifier.score(X\_test, Y\_test)

print("The MLP Classifier model has given Accuracy of: %.3f%%" % (result \* 100.0))

# Make predictions on the test set

Y\_pred = mlp\_classifier.predict(X\_test)

print("Predictions:", Y\_pred)

# Confusion Matrix

cf\_mtrx = metrics.confusion\_matrix(Y\_test, Y\_pred)

cm\_display = ConfusionMatrixDisplay(confusion\_matrix=cf\_mtrx, display\_labels=[False, True])

cm\_display.plot()

plt.show()

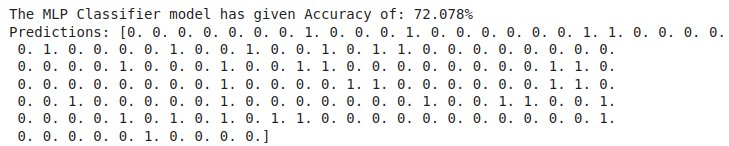
# Classification report

print(classification\_report(Y\_test, Y\_pred))

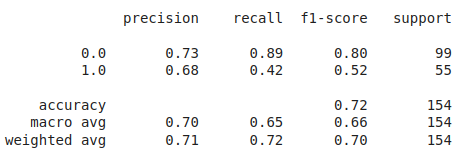
Explanation:

This code trains and evaluates a Multi-Layer Perceptron (MLP) classifier on the Pima Indians Diabetes dataset. After loading and preparing the data, it splits it into training (80%) and testing (20%) sets. The MLP classifier is initialized with two hidden layers, each containing 10 neurons, and it is trained using backpropagation over a maximum of 1000 iterations. The model's accuracy is then assessed on the test set, and predictions are made. A confusion matrix is displayed to visualize the performance, and a classification report is generated to provide detailed metrics such as precision, recall, and F1-score.

Output:



data:image/png;base64,



1. Write a program to implement the SVM, test the model, and show the evaluation report.

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the hyperplane that best separates data points of different classes. This program implements the SVM classifier, tests it on a dataset, and generates an evaluation report including metrics like accuracy, precision, recall, and F1-score. SVM is highly effective in high-dimensional spaces and is widely used for classification problems.

Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn import metrics

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

# Load the dataset (Pima Indians Diabetes Database)

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=columns)

# Prepare the input and output

spltd\_data = data.values

X = spltd\_data[:, 0:8] # Features (all except 'Outcome')

Y = spltd\_data[:, 8] # Target ('Outcome')

# Split the dataset into training and testing sets (80-20 split)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Initialize and train the SVM classifier

svm\_classifier = SVC(kernel='linear', random\_state=42)

svm\_classifier.fit(X\_train, Y\_train)

# Model accuracy

result = svm\_classifier.score(X\_test, Y\_test)

print("The SVM model has given Accuracy of: %.3f%%" % (result \* 100.0))

# Make predictions on the test set

Y\_pred = svm\_classifier.predict(X\_test)

print("Predictions:", Y\_pred)

# Confusion Matrix

cf\_mtrx = metrics.confusion\_matrix(Y\_test, Y\_pred)

cm\_display = ConfusionMatrixDisplay(confusion\_matrix=cf\_mtrx, display\_labels=[False, True])

cm\_display.plot()

plt.show()

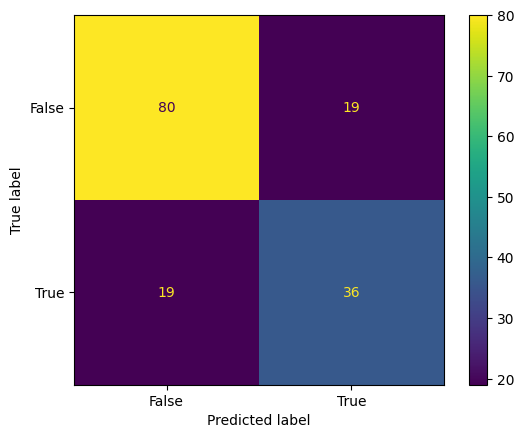
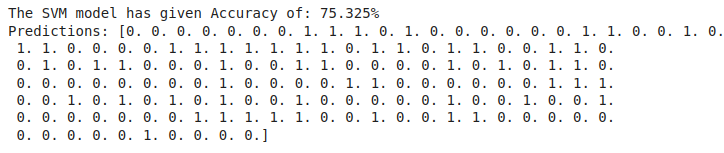
# Classification report

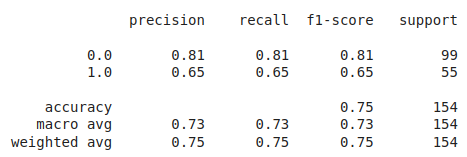
print(classification\_report(Y\_test, Y\_pred))

Explanation:

This code trains and evaluates a Support Vector Machine (SVM) classifier using the Pima Indians Diabetes dataset. After loading and preparing the data, it splits it into training (80%) and testing (20%) sets. The SVM classifier is initialized with a linear kernel and trained on the data. The model's accuracy is calculated on the test set, and predictions are made. A confusion matrix is displayed to assess the classification performance, and a classification report is generated to provide detailed evaluation metrics like precision, recall, and F1-score.

Output:





1. Write a program to implement the linear regression, test the model, and show the evaluation report.

Linear regression is a statistical method used for modeling the relationship between a dependent variable and one or more independent variables. This program implements a linear regression model, tests it on a dataset, and generates an evaluation report showing key metrics such as mean squared error (MSE), R-squared, and others. It is commonly used for predicting continuous values based on input features.

Program:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset (Pima Indians Diabetes Database)

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=columns)

# Prepare the input and output

spltd\_data = data.values

X = spltd\_data[:, 0:8] # Features (all except 'Outcome')

Y = spltd\_data[:, 8] # Target ('Outcome')

# Split the dataset into training and testing sets (80-20 split)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Initialize and train the Linear Regression model

linear\_regressor = LinearRegression()

linear\_regressor.fit(X\_train, Y\_train)

# Make predictions on the test set

Y\_pred = linear\_regressor.predict(X\_test)

# Evaluate the model using Mean Squared Error (MSE) and R² Score

mse = mean\_squared\_error(Y\_test, Y\_pred)

print(f'Mean Squared Error (MSE): {mse:.3f}')

r2 = r2\_score(Y\_test, Y\_pred)

print(f'R² Score: {r2:.3f}')

# Optional: Scatter plot of actual vs predicted values

plt.scatter(Y\_test, Y\_pred)

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Actual vs Predicted Diabetes Outcome')

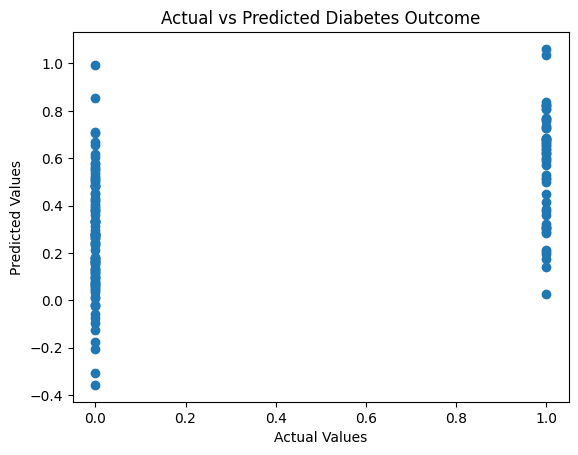
plt.show()

Explanation:

This code applies a Linear Regression model to predict the outcome of diabetes in the Pima Indians Diabetes dataset. After loading and preparing the data, it splits it into training (80%) and testing (20%) sets. The model is trained on the training set, and predictions are made on the test set. The performance of the model is evaluated using two metrics: Mean Squared Error (MSE) and R² Score. A scatter plot is also generated to visually compare the actual versus predicted values, which helps assess how well the model fits the data.

Output:



This scatter plot compares the actual and predicted outcomes for diabetes classification. The x-axis represents the actual values (0 for no diabetes and 1 for diabetes), while the y-axis shows the predicted probabilities or outcomes from the model. Most points are clustered around 0 and 1 on both axes, indicating that the model generally performs well in predicting the correct outcomes. However, some deviation from these clusters suggests occasional misclassification or uncertain predictions. Overall, the plot highlights the relationship between actual and predicted values, providing insight into the model's performance.

1. Write a program to illustrate the k-fold cross validation.

K-fold cross-validation is a technique used to assess the performance of a machine learning model by splitting the dataset into ‘k’ subsets (folds) and training and testing the model k times, each time with a different fold as the test set. This program demonstrates the implementation of k-fold cross-validation, helping to ensure the model's reliability by evaluating its performance across different subsets of the dataset.

Program:

import pandas as pd

import numpy as np

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

# Load the dataset (Pima Indians Diabetes Database)

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=columns)

# Prepare the input and output

spltd\_data = data.values

X = spltd\_data[:, 0:8] # Features (all except 'Outcome')

Y = spltd\_data[:, 8] # Target ('Outcome')

# Set up K-Fold Cross Validation (10 folds)

kfold = KFold(n\_splits=10, random\_state=42, shuffle=True)

# Evaluate Decision Tree Classifier

dtree = DecisionTreeClassifier(random\_state=42)

dtree\_cv\_scores = cross\_val\_score(dtree, X, Y, cv=kfold, scoring='accuracy')

# Print the mean and standard deviation of the accuracy scores

print(f'Decision Tree - Mean Accuracy: {dtree\_cv\_scores.mean():.3f}, Std Dev: {dtree\_cv\_scores.std():.3f}')

# Visualize the results (mean accuracy with error bars)

plt.bar(['Decision Tree'], [dtree\_cv\_scores.mean()], yerr=[dtree\_cv\_scores.std()], capsize=5, color='red')

plt.title('Decision Tree Performance with K-Fold Cross Validation')

plt.ylabel('Accuracy')

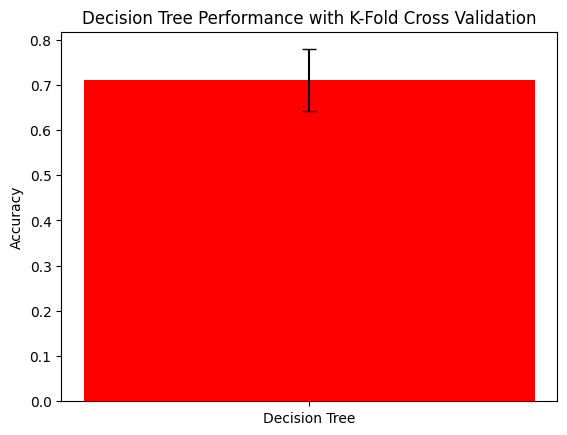
plt.show()

Explanation:

This code evaluates the performance of a Decision Tree Classifier using K-Fold Cross Validation on the Pima Indians Diabetes dataset. After preparing the feature and target variables, it sets up a 10-fold cross-validation and applies it to the Decision Tree model. The accuracy scores from each fold are collected, and both the mean and standard deviation of these scores are printed. A bar chart is then generated to visually represent the mean accuracy of the model, with error bars showing the standard deviation, providing a clear view of the model's stability across different folds.

Output:





This bar plot shows the performance of a Decision Tree model evaluated using K-Fold Cross Validation, with accuracy as the metric. The red bar represents the average accuracy across all folds, while the black error bar indicates the variability (standard deviation or confidence interval) in accuracy across the folds. The y-axis shows the accuracy range, and the x-axis labels the evaluated model as "Decision Tree." The error bar highlights how consistent the model's performance is when trained and tested on different data splits. Overall, this plot summarizes both the model's effectiveness and reliability.