**Program-8: Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.**

import numpy as np

from sklearn.datasets import load\_breast\_cancer

from sklearn.tree import DecisionTreeClassifier, export\_text, plot\_tree

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

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load Breast Cancer Dataset

data = load\_breast\_cancer()

X = data.data # Features

y = data.target # Labels

feature\_names = data.feature\_names

target\_names = data.target\_names

# Split dataset into training and test sets

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

# Build the Decision Tree model

model = DecisionTreeClassifier(criterion="entropy", max\_depth=7, random\_state=42)

model.fit(X\_train, y\_train)

# Evaluate the model

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Decision Tree Accuracy: {accuracy:.2f}")

# Visualize the Decision Tree

plt.figure(figsize=(16, 10))

plot\_tree(model, feature\_names=feature\_names, class\_names=target\_names, filled=True, rounded=True)

plt.title("Decision Tree Visualization")

plt.show()

# Print Text Representation of the Tree

tree\_rules = export\_text(model, feature\_names=list(feature\_names))

print("\nDecision Tree Rules:")

print(tree\_rules)

# Classify a New Sample

new\_sample = np.array([20.57, 17.77, 132.9, 1326.0, 0.08474, 0.07864, 0.0869, 0.07017, 0.1812,

0.05667, 0.5435, 0.7339, 3.398, 74.08, 0.005225, 0.01308, 0.0186, 0.0134,

0.01389, 0.003532, 25.38, 24.99, 166.1, 2019.0, 0.1622, 0.6656, 0.7119,

0.2654, 0.4601, 0.1189]).reshape(1, -1)

predicted\_class = model.predict(new\_sample)

print("\nNew Sample Classification:")

print(f"Predicted Class: {target\_names[predicted\_class[0]]}")

**Explanation**

1. **Dataset**:
   * **Breast Cancer Dataset**: A built-in dataset in sklearn with 30 numerical features used to classify tumors as malignant or benign.
2. **Model**:
   * **Decision Tree Classifier**: A DecisionTreeClassifier is used with max\_depth=3 to control tree size and prevent overfitting.
   * **Criterion**: Uses "gini" (Gini impurity) as the splitting criterion.
3. **Evaluation**:
   * Splits the dataset into training and test sets (80% training, 20% testing).
   * Calculates the accuracy of the model on the test set.
4. **Visualization**:
   * Visualizes the decision tree using plot\_tree.
   * Outputs a textual representation of the decision rules with export\_text.
5. **Classifying a New Sample**:
   * The program classifies a new sample (example values provided) and outputs the predicted class (malignant or benign).

**Output**

1. **Console Output**:
   * Model accuracy on the test dataset.
   * Text-based decision tree rules.
   * Predicted class for the new sample.

Example:

plaintext

CopyEdit

Decision Tree Accuracy: 0.92

Decision Tree Rules:

|--- mean concave points <= 0.05

| |--- worst concave points <= 0.13

| | |--- worst radius <= 16.30

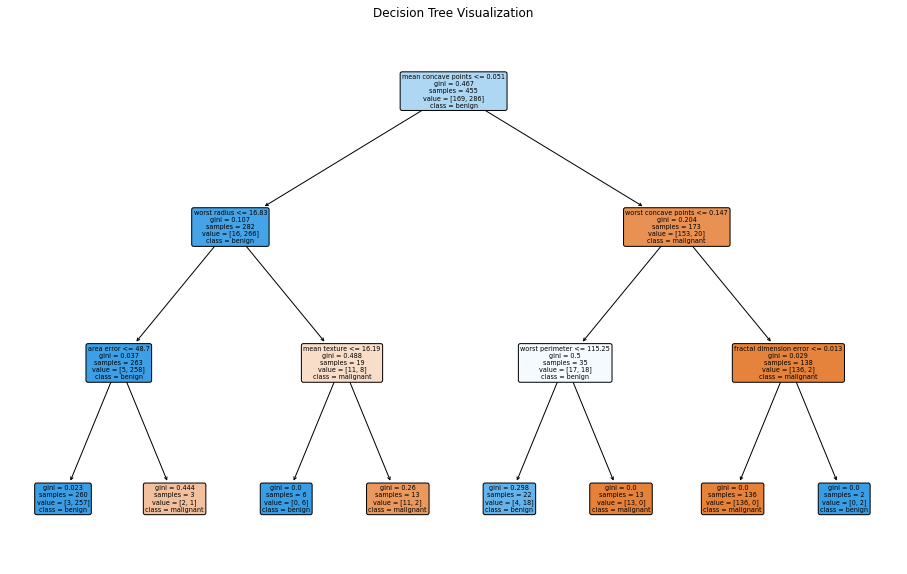
| | | |--- class: benign

...

New Sample Classification:

Predicted Class: malignant

1. **Graph Output**:
   * A visualization of the decision tree with feature names, class names, and splitting criteria.



**How to Use**

1. Run the program to see the tree built from the dataset and evaluate its performance.

Modify the new\_sample values to classify other samples

**Experiment 9:**

Develop a program to implement the Naive Bayesian classifier, considering the Olivetti Face Data set for training.

Compute the accuracy of the classifier, considering a few test data set.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_olivetti\_faces

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

# Step 1: Load the Olivetti Face dataset

data = fetch\_olivetti\_faces()

X = data.data # features: flattened 64x64 images

y = data.target # labels: person ID (0 to 39)

print(f"Dataset Shape: {X.shape}")

print(f"Number of Classes: {len(np.unique(y))}")

# Step 2: 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)

# Step 3: Build and train the Naive Bayes Classifier

model = GaussianNB()

model.fit(X\_train, y\_train)

# Step 4: Predict on the test data

y\_pred = model.predict(X\_test)

# Step 5: Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"\nAccuracy of Naive Bayes Classifier on Olivetti Faces: {accuracy:.2f}")

# Optional: Print detailed classification report

print("\nClassification Report:")

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

# ---------------------------------------------------

# Step 6: Display all 40 unique faces (one face per person)

# ---------------------------------------------------

print("\nDisplaying 40 unique faces from the dataset...")

# Create a figure

fig, axes = plt.subplots(5, 8, figsize=(16, 10)) # 5 rows × 8 columns = 40 images

fig.suptitle("Olivetti Faces - One Sample per Person", fontsize=20)

# Keep track of already displayed labels

displayed\_labels = set()

# Plot one image per unique label

i = 0

for ax in axes.flat:

while y[i] in displayed\_labels:

i += 1

img = X[i].reshape(64, 64) # Reshape flattened array back to 64x64 image

ax.imshow(img, cmap='gray')

ax.set\_title(f"Person {y[i]}")

ax.axis('off')

displayed\_labels.add(y[i])

i += 1

plt.tight\_layout()

plt.subplots\_adjust(top=0.88)

plt.show()