**Labprogram 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.**

**What is the Wisconsin Breast Cancer Dataset (WBCD)?**

It’s a **famous dataset** used in machine learning and medical research to **predict whether a breast tumor is benign or malignant** based on features from cell samples.

| **Attribute** | **Description** |
| --- | --- |

|  |  |
| --- | --- |
| **Number of instances** | 569 |

|  |  |
| --- | --- |
| **Number of features** | 30 real-valued input features |

|  |  |
| --- | --- |
| **Classes** | 2 (Benign = 0, Malignant = 1) |

|  |  |
| --- | --- |
| **Feature types** | Radius, Texture, Perimeter, Area, Smoothness, etc. |

|  |  |
| --- | --- |
| **Use case** | Classification (benign or malignant tumor prediction) |

**Program:**

**import matplotlib.pyplot as plt**

**import numpy as np**

**from sklearn.datasets import load\_breast\_cancer**

**from sklearn.tree import DecisionTreeClassifier, plot\_tree**

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

**from sklearn.metrics import accuracy\_score**

**# Step 1: Load the breast cancer dataset**

**data = load\_breast\_cancer()**

**X = data.data**

**y = data.target**

**# Step 2: Split 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)**

**# Step 3: Train the Decision Tree classifier**

**clf = DecisionTreeClassifier(criterion=’gini’,random\_state=42)**

**clf.fit(X\_train, y\_train)**

**# Step 4: Evaluate the classifier**

**y\_pred = clf.predict(X\_test)**

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

**print(f"Accuracy on test data: {accuracy\*100:.2f}%")**

**# Step 5: Classify a new sample (Example: taking the first test sample)**

**new\_sample = X\_test[0].reshape(1, -1) # Reshapes it into the 2D shape expected by the model.**

**predicted\_class = clf.predict(new\_sample)[0]# Since predict returns an array (even for one sample), [0] extracts the single prediction**

**print("Predicted class for the new sample:",**

**data.target\_names[predicted\_class])**

**print("Actual class:", data.target\_names[y\_test[0]])**

**# Step 6: Visualize the Decision Tree**

**plt.figure(figsize=(20, 10))**

**plot\_tree(clf, filled=True, feature\_names=data.feature\_names,**

**class\_names=data.target\_names)**

**plt.title("Decision Tree for Breast Cancer Classification")**

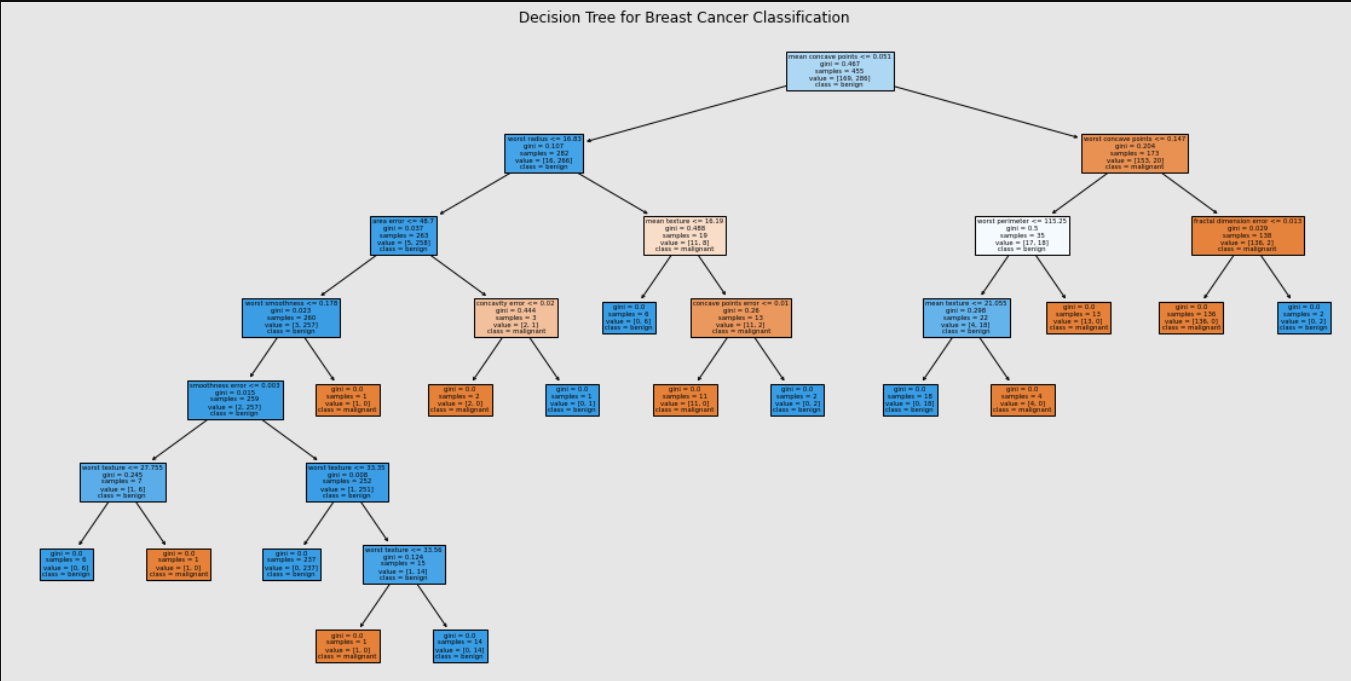
**plt.show()**

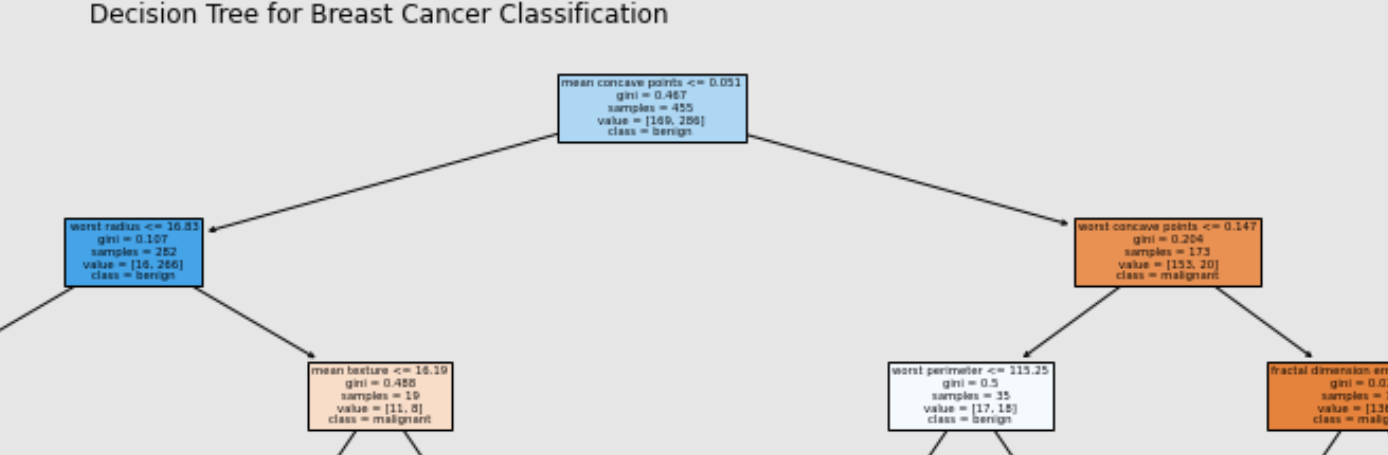
**Output:**

Accuracy on test data: 94.74%

Predicted class for the new sample: benign

Actual class: benign





**Labprogram 9: Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.**

**What is the Olivetti Faces Dataset?**

* **Created by:** AT&T Laboratories Cambridge (formerly Olivetti Research Laboratory)
* **Purpose:**  
  Mainly used for **face recognition** and **unsupervised learning** (like PCA, clustering, etc.).
* **Data:**
  + **400 grayscale images**
  + **40 different people** (subjects)
  + **Each person:** 10 images (with different expressions, lighting, angles)

**Image Details:**

* **Image size:** 64 × 64 pixels
* **Color:** Grayscale (pixel values from 0 to 255)

**Program:**

**from sklearn.datasets import fetch\_olivetti\_faces**

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

**from sklearn.decomposition import PCA**

**from sklearn.naive\_bayes import GaussianNB**

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

**import matplotlib.pyplot as plt**

**# Step 1: Load the dataset**

**faces = fetch\_olivetti\_faces()**

**X = faces.data # Each image is flattened (64x64 -> 4096 features)**

**y = faces.target # Labels from 0 to 39 (40 people)**

**# Step 2: Split into train and test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y,**

**test\_size=0.25,random\_state=42, stratify=y) #** Makes sure the **proportion of classes** (benign/malignant) in training and testing stays the **same as in the original data**. (Important if classes are imbalanced!)

**# Step 3: Dimensionality Reduction using PCA**

**pca = PCA(n\_components=100, whiten=True, random\_state=42) #whiten=true, removes the correlation between components so that it becomes independent components**

**X\_train\_pca = pca.fit\_transform(X\_train)**

**X\_test\_pca = pca.transform(X\_test)**

**# Step 4: Train the Naive Bayes Classifier**

**gnb = GaussianNB()**

**gnb.fit(X\_train\_pca, y\_train)**

**# Step 5: Predict and Evaluate**

**y\_pred = gnb.predict(X\_test\_pca)**

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

**print("Accuracy of Naive Bayes on Olivetti Faces: {:.2f}%".format(accuracy \* 100))**

**print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))**

**Output:**

Accuracy of Naive Bayes on Olivetti Faces: 92.00%

Classification Report:

precision recall f1-score support

0 0.67 1.00 0.80 2

1 1.00 0.67 0.80 3

2 1.00 1.00 1.00 3

3 1.00 1.00 1.00 3

4 1.00 1.00 1.00 2

5 1.00 1.00 1.00 3

6 1.00 1.00 1.00 2

7 0.50 1.00 0.67 3

8 1.00 1.00 1.00 2

9 1.00 0.50 0.67 2

10 1.00 1.00 1.00 3

11 1.00 1.00 1.00 3

12 0.50 1.00 0.67 2

13 1.00 0.50 0.67 2

14 1.00 1.00 1.00 3

15 1.00 0.50 0.67 2

16 0.67 1.00 0.80 2

17 1.00 0.50 0.67 2

18 1.00 1.00 1.00 3

19 1.00 1.00 1.00 2

20 1.00 1.00 1.00 3

21 1.00 0.67 0.80 3

22 1.00 1.00 1.00 3

23 1.00 1.00 1.00 3

24 1.00 0.50 0.67 2

25 1.00 1.00 1.00 2

26 1.00 1.00 1.00 2

27 1.00 1.00 1.00 2

28 1.00 1.00 1.00 3

29 1.00 1.00 1.00 3

30 1.00 1.00 1.00 3

31 0.67 1.00 0.80 2

32 1.00 1.00 1.00 3

33 1.00 1.00 1.00 2

34 1.00 1.00 1.00 2

35 1.00 1.00 1.00 3

36 1.00 1.00 1.00 3

37 1.00 1.00 1.00 2

38 1.00 1.00 1.00 3

39 1.00 0.50 0.67 2

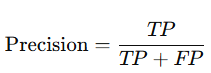
accuracy 0.92 100

macro avg 0.95 0.91 0.91 100

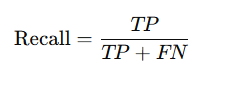
weighted avg 0.95 0.92 0.92 100

**Explanation:**

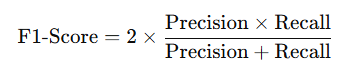
**Precision**: The ratio of correctly predicted positive observations to the total predicted positives. It tells us how many of the predicted positive cases are actually positive.

****

**Recall (Sensitivity)**: The ratio of correctly predicted positive observations to all observations in the actual class. It tells us how many actual positive cases are correctly identified by the model.

****

**F1-Score**: The harmonic mean of precision and recall. It balances both the precision and recall, especially when there is an uneven class distribution.

****

**Support**: The number of actual occurrences of the class in the dataset.

In addition, it also calculates the **macro average** and **weighted average** of these metrics, which provide an overall summary of the model's performance.

**Labprogram 10: Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.**

**# Step 1: Import required libraries**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.datasets import load\_breast\_cancer**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.cluster import KMeans**

**from sklearn.decomposition import PCA**

**from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report**

**# Step 2: Load the Breast Cancer dataset**

**data = load\_breast\_cancer()**

**# Step 3: Feature Scaling (Normalization)**

**# Standardize features to have mean = 0 and variance = 1**

**X = StandardScaler().fit\_transform(data.data)**

**# Step 4: Extract the target labels (malignant = 0, benign = 1)**

**y = data.target**

**# Step 5: Apply K-Means Clustering**

**# Create KMeans model with 2 clusters and fit it to the scaled datakmeans = KMeans(n\_clusters=2, random\_state=42)**

**labels = kmeans.fit\_predict(X)**

**# Step 6: Evaluate the clustering performance**

**print(f"Accuracy of K-Means clustering: {accuracy\_score(y, labels) \* 100:.2f}%")**

**print("\nConfusion Matrix:\n", confusion\_matrix(y, labels))**

**print("\nClassification Report:\n", classification\_report(y, labels))**

**#Step 7: Dimensionality Reduction using PCA- Reduce to 2D using PCA**

**pca = PCA(n\_components=2)**

**X\_pca = pca.fit\_transform(X)**

**# Step 8: Project cluster centroids into PCA space**

**centroids\_pca = pca.transform(kmeans.cluster\_centers\_)**

**# Step 9: Visualize the True Labels and K-Means Clustering**

**fig, axes = plt.subplots(1, 2, figsize=(12, 6))**

**titles = ['True Labels', 'K-Means Clustering']**

**# Plot each scatter plot (True labels and Clustering labels)**

**for ax, data\_labels, title in zip(axes, [y, labels], titles):**

**ax.scatter(X\_pca[:, 0], X\_pca[:, 1], c=data\_labels, cmap='coolwarm', alpha=0.7)**

**ax.set(title=title, xlabel='PCA Component 1', ylabel='PCA Component 2')**

**ax.grid(True)**

**# Plot the cluster centroids on the K-Means clustering plot**

**axes[1].scatter(\*centroids\_pca.T, marker='X', s=200, c='black', label='Centroids')**

**axes[1].legend()**

**# Step 10: Final plot adjustments and display**

**plt.tight\_layout()**

**plt.show()**

**Output:**

Accuracy of K-Means clustering: 91.04%

Confusion Matrix:

[[175 37]

[ 14 343]]

Classification Report:

precision recall f1-score support

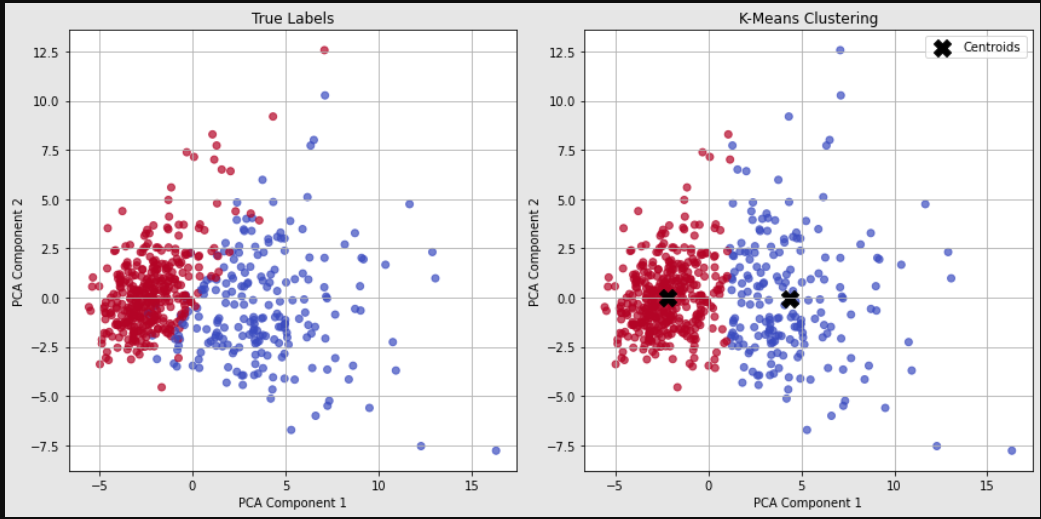
0 0.93 0.83 0.87 212

1 0.90 0.96 0.93 357

accuracy 0.91 569

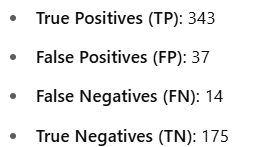
macro avg 0.91 0.89 0.90 569

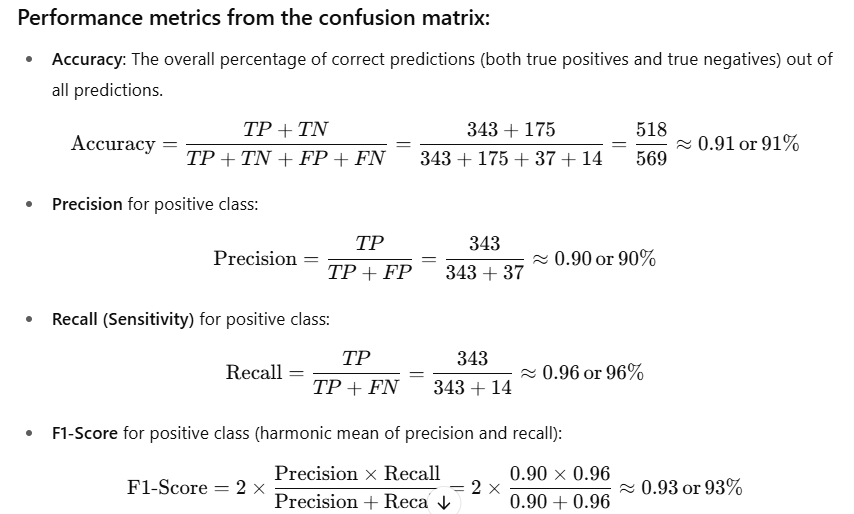
weighted avg 0.91 0.91 0.91 569

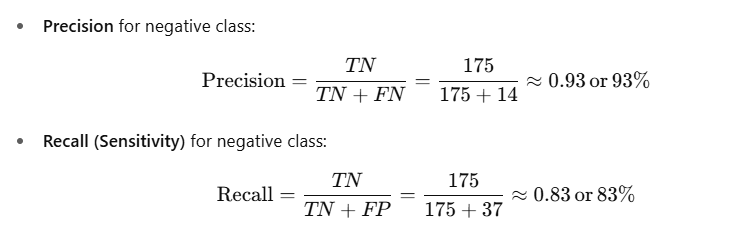
****

**Explanation:**

A **confusion matrix** is a useful tool for evaluating the performance of a classification model. It compares the true labels (ground truth) with the predicted labels to determine how well the model is performing. Let's break down the confusion matrix you've provided:



****

****