**Breast Cancer Dataset Tutorial**

**1. Introduction**

The Breast Cancer dataset is a classic binary classification problem. It contains 569 samples with 30 features, such as radius, texture, perimeter, etc. The goal is to predict whether a tumor is **malignant (1)** or **benign (0)**.

**2. Setup**

Install the required libraries:

pip install numpy pandas scikit-learn xgboost matplotlib

**3. Code Implementation**

# Import libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.neighbors import KNeighborsClassifier

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

import matplotlib.pyplot as plt

# Load dataset

data = load\_breast\_cancer()

X = data.data # Features

y = data.target #Labels (0 = malignant, 1 = benign)

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Function to evaluate models

def evaluate\_model(model, X\_test, y\_test):

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

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

roc\_auc = roc\_auc\_score(y\_test, model.predict\_proba(X\_test)[:, 1])

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

print(f"ROC AUC: {roc\_auc:.2f}")

print("Classification Report:")

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

# 1. Decision Tree

print("Decision Tree:")

dt\_model = DecisionTreeClassifier(max\_depth=5, random\_state=42)

dt\_model.fit(X\_train, y\_train)

evaluate\_model(dt\_model, X\_test, y\_test)

# 2. Random Forest

print("Random Forest:")

rf\_model = RandomForestClassifier(n\_estimators=100, max\_depth=5, random\_state=42)

rf\_model.fit(X\_train, y\_train)

evaluate\_model(rf\_model, X\_test, y\_test)

# 3. XGBoost

print("XGBoost:")

xgb\_model = XGBClassifier(random\_state=42, use\_label\_encoder=False, eval\_metric='logloss')

xgb\_model.fit(X\_train, y\_train)

evaluate\_model(xgb\_model, X\_test, y\_test)

# 4. K-Nearest Neighbors (KNN)

print("K-Nearest Neighbors (KNN):")

knn\_model = KNeighborsClassifier(n\_neighbors=5)

knn\_model.fit(X\_train, y\_train)

evaluate\_model(knn\_model, X\_test, y\_test)

**4. Explanation of the Code**

1. **Decision Tree**:
   * A single decision tree is trained with a maximum depth of 5 to prevent overfitting.
2. **Random Forest**:
   * An ensemble of 100 decision trees is trained, each with a maximum depth of 5.
3. **XGBoost**:
   * A gradient boosting model is trained with default parameters, using logloss as the evaluation metric.
4. **K-Nearest Neighbors (KNN)**:
   * A KNN model is trained with k=5 neighbors.

**5. Exercises**

1. **Hyperparameter Tuning**:

GridSearchCV is a powerful tool in scikit-learn for hyperparameter tuning. It performs an exhaustive search over a specified parameter grid to find the best combination of hyperparameters for a model. It uses cross-validation to evaluate each combination, ensuring that the selected hyperparameters generalize well to unseen data.

### ****How GridSearchCV Works****

1. **Define a Parameter Grid**:
   * Specify the hyperparameters and their possible values as a dictionary.
   * Example: For a Random Forest, you might tune n\_estimators, max\_depth, and min\_samples\_split.
2. **Cross-Validation**:
   * GridSearchCV splits the data into training and validation sets multiple times (based on the cv parameter) to evaluate each combination of hyperparameters.
3. **Best Model**:
   * After evaluating all combinations, GridSearchCV returns the model with the best hyperparameters.

### ****Using GridSearchCV****

Here’s how to use GridSearchCV with the **Breast Cancer dataset**:

#### ****1. Import Libraries****

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

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

from sklearn.ensemble import RandomForestClassifier

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

#### ****2. Load Dataset****

data = load\_breast\_cancer()

X = data.data

y = data.target

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

#### ****3. Define the Model and Parameter Grid****

# Define the model

model = RandomForestClassifier(random\_state=42)

# Define the parameter grid

param\_grid = {

'n\_estimators': [50, 100, 200], # Number of trees in the forest

'max\_depth': [None, 5, 10], # Maximum depth of the tree

'min\_samples\_split': [2, 5, 10] # Minimum number of samples required to split a node

}

#### ****4. Perform Grid Search****

# Set up GridSearchCV

grid\_search = GridSearchCV(

estimator=model,

param\_grid=param\_grid,

scoring='accuracy', # Metric to evaluate

cv=5, # Number of cross-validation folds

verbose=1, # Print progress

n\_jobs=-1 # Use all available CPU cores

)

# Fit the model

grid\_search.fit(X\_train, y\_train)

#### ****5. Evaluate the Best Model****

# Get the best model

best\_model = grid\_search.best\_estimator\_

# Evaluate on the test set

y\_pred = best\_model.predict(X\_test)

print(f"Best Parameters: {grid\_search.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}")

print("Classification Report:")

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

### ****Example Output****

Fitting 5 folds for each of 27 candidates, totalling 135 fits

Best Parameters: {'max\_depth': 10, 'min\_samples\_split': 2, 'n\_estimators': 200}

Accuracy: 0.97

Classification Report:

precision recall f1-score support

0 0.97 0.94 0.95 63

1 0.96 0.98 0.97 108

### ****Key Parameters of GridSearchCV****

* estimator: The model to tune.
* param\_grid: Dictionary with hyperparameters and their possible values.
* scoring: Metric to evaluate (e.g., accuracy, roc\_auc, f1).
* cv: Number of cross-validation folds.
* verbose: Controls the amount of output (higher values = more output).
* n\_jobs: Number of CPU cores to use (-1 = use all cores).

Use GridSearchCV to find the best hyperparameters for each model.

Example: Tune max\_depth, min\_samples\_split for Decision Trees, n\_estimators for Random Forest, learning\_rate for XGBoost, and n\_neighbors for KNN.

1. **Compare Performance**:
   * Compare the accuracy, ROC AUC, and training time of all four models.
2. **ROC Curve**:
   * Plot the ROC curve for each model to visualize the trade-off between true positive rate and false positive rate.
3. **Feature Importance:**

# Plot feature importance for Random Forest

importances = rf\_model.feature\_importances\_

feature\_names = data.feature\_names

indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))

plt.title("Feature Importances (Random Forest)")

plt.bar(range(X.shape[1]), importances[indices], align="center")

plt.xticks(range(X.shape[1]), [feature\_names[i] for i in indices], rotation=90)

plt.xlabel("Feature")

plt.ylabel("Importance")

plt.show()

**7. Expected Output**

You should see the accuracy, ROC AUC, and classification report for each model. For example:

Decision Tree:

Accuracy: 0.95

ROC AUC: 0.96

Classification Report:

precision recall f1-score support

0 0.94 0.93 0.93 63

1 0.96 0.96 0.96 108

1. **Compare Feature Importance Across Models**:
   * Compare the feature importance of Decision Trees, Random Forests, and XGBoost. Are the top features consistent across models?
2. **Feature Selection**:
   * Use feature importance to select the top 10 features and retrain the models. Does the performance improve or stay the same?