

Experiment 4

Aim

To implement and analyze **Decision Tree** and **Random Forest** classification models on a real-world heart disease dataset using Python, and evaluate their performance using appropriate classification metrics.

1. Dataset Source

Dataset – Decision Tree & Random Forest Classification
Heart Disease Dataset

Target Variable: **target** (0 = no heart disease, 1 = heart disease)

Source:

<https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset>

2. Dataset Description

Features:

- **age** — age in years
- **sex** — gender (0 = female, 1 = male)
- **cp** — chest pain type
- **trestbps** — resting blood pressure
- **chol** — serum cholesterol (mg/dl)
- **fbs** — fasting blood sugar > 120 mg/dl
- **restecg** — resting ECG results
- **thalach** — maximum heart rate achieved
- **exang** — exercise-induced angina
- **oldpeak** — ST depression induced by exercise

- **slope** — slope of peak exercise ST segment
 - **ca** — number of major vessels colored by fluoroscopy
 - **thal** — thalassemia status
 - **target** — presence (1) or absence (0) of heart disease
-

3. Mathematical Formulation of the Algorithm

1. Decision Tree (Classification)

A decision tree splits the dataset recursively by selecting features that best reduce impurity.

Gini Impurity:

$$Gini = 1 - \sum_{i=1}^C (p_i)^2$$

Information Gain (Entropy):

$$IG = Entropy_{parent} - Entropy_{children}$$

Entropy:

$$Entropy = - \sum p_i \log_2(p_i)$$

2. Random Forest (Ensemble of Trees)

Random Forest builds multiple decision trees using bootstrap samples and averages their predictions.

Majority Voting:

$$\hat{y} = \text{mode}(Tree_1(x), Tree_2(x), \dots, Tree_n(x))$$

4. Algorithm Limitations

Decision Tree

- Prone to overfitting
- Sensitive to small changes in data
- Can create very deep trees without pruning

Random Forest

- Requires more computational resources
 - Less interpretable than a single decision tree
 - Training is slower than simple models
-

5. Methodology / Workflow

Common Workflow

- Load dataset using Pandas
 - Select independent (X) and dependent (y) variables
 - Split dataset into training and testing sets
 - Train Decision Tree and Random Forest models
 - Predict on test data
 - Evaluate using classification metrics
-

6. Performance Analysis

Evaluation Metrics

- Accuracy Score
- Confusion Matrix
- Precision
- Recall
- F1-Score

Interpretation:

- Higher accuracy → better classification performance
 - Confusion matrix shows TP, TN, FP, FN
 - Precision and recall help evaluate class-specific performance
 - F1-score balances precision and recall
-

7. Hyperparameter Tuning

Decision Tree

- `criterion`: “gini” / “entropy”
- `max_depth`
- `min_samples_split`

Random Forest

- `n_estimators`: number of trees
 - `max_depth`
 - `max_features`
 - `min_samples_split`
-

Exercise 1: Decision Tree Classification

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Load dataset
df = pd.read_csv('heart.csv') # dataset from Kaggle

# Feature selection
X = df.drop('target', axis=1)
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Model training
dt_model = DecisionTreeClassifier(criterion='gini',
random_state=42)
dt_model.fit(X_train, y_train)

# Prediction
y_pred_dt = dt_model.predict(X_test)

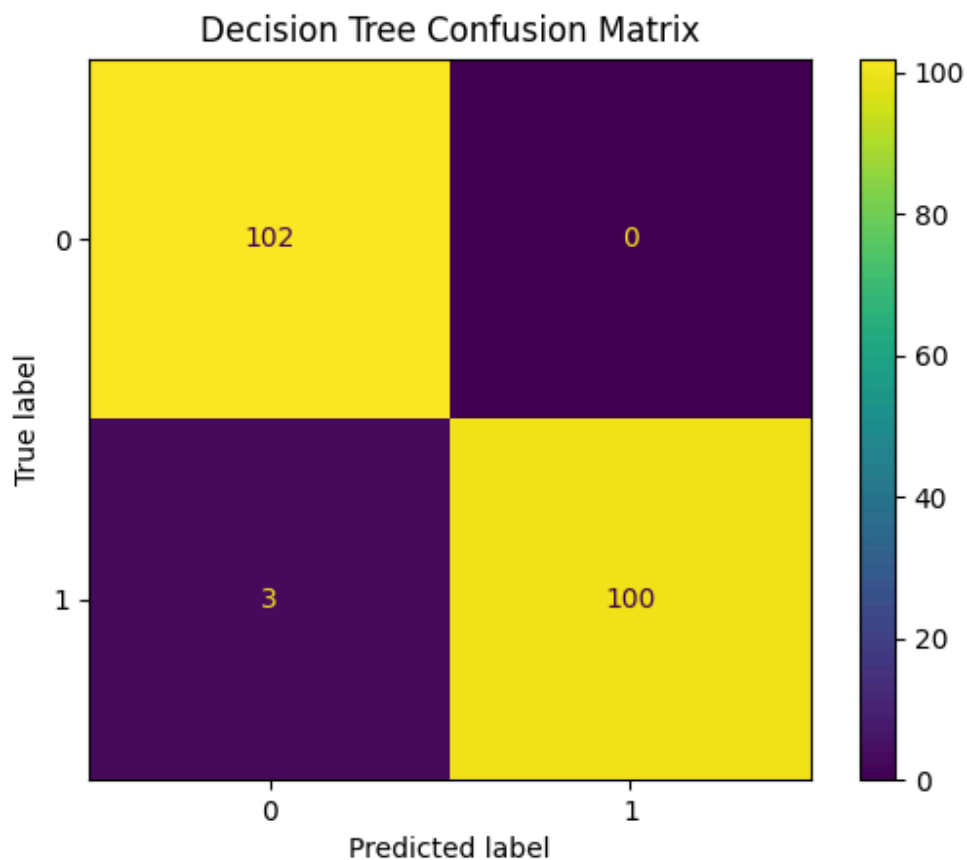
# Evaluation
print("Decision Tree Accuracy:", accuracy_score(y_test,
y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test,
y_pred_dt))
```

Output

```
@jaykerkar0405 → /workspaces/MLDL-Lab (main) $ python3 main.py
===== Decision Tree Results =====
Accuracy: 0.9853658536585366
Classification Report:
              precision    recall  f1-score   support

     0       0.97       1.00       0.99       102
     1       1.00       0.97       0.99       103

 accuracy          0.99
 macro avg       0.99       0.99       0.99
weighted avg       0.99       0.99       0.99
```



Exercise 2: Random Forest Classification

Code

```
from sklearn.ensemble import RandomForestClassifier

# Random Forest model
rf_model = RandomForestClassifier(n_estimators=100,
random_state=42)
rf_model.fit(X_train, y_train)

# Prediction
y_pred_rf = rf_model.predict(X_test)

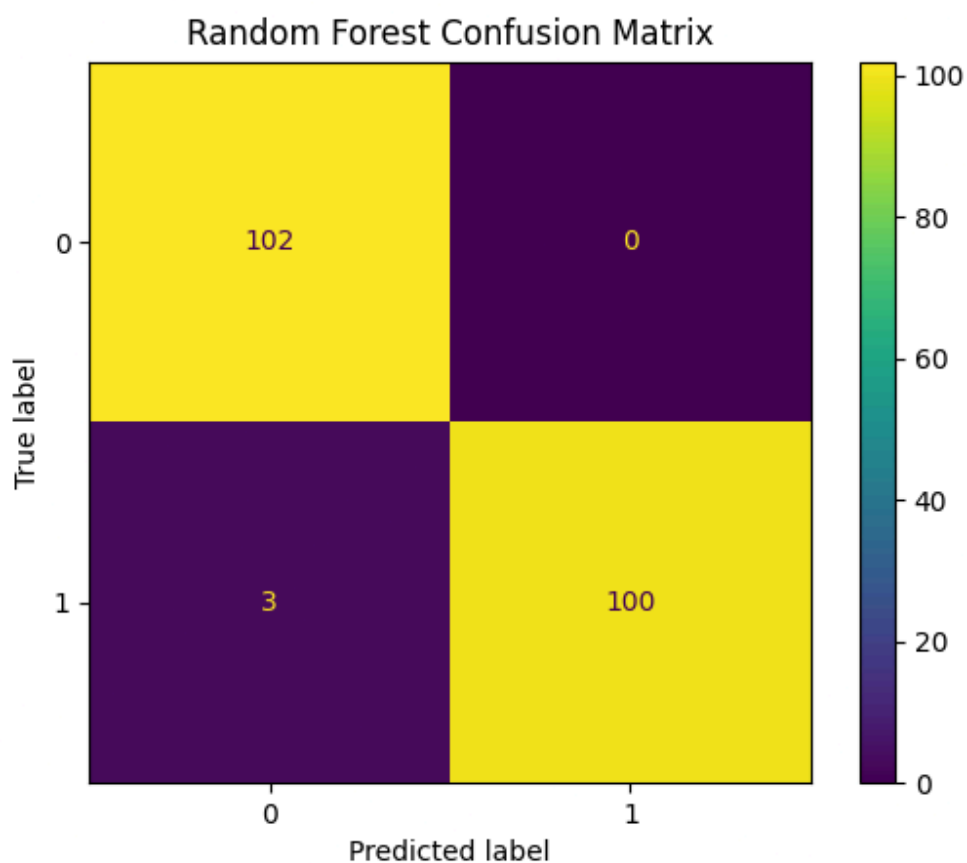
# Evaluation
print("Random Forest Accuracy:", accuracy_score(y_test,
y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test,
y_pred_rf))
```

Output

```
===== Random Forest Results =====
Accuracy: 0.9853658536585366
Classification Report:
              precision    recall  f1-score   support

         0       0.97        1.00        0.99         102
         1       1.00        0.97        0.99         103

   accuracy                0.99         205
  macro avg              0.99        0.99        0.99         205
weighted avg              0.99        0.99        0.99         205
```



Conclusion

This experiment successfully implemented **Decision Tree** and **Random Forest** classification models on the Heart Disease dataset. While a Decision Tree provides a simple interpretable model, the Random Forest classifier leverages ensemble learning to reduce overfitting and improve performance. The evaluation metrics such as accuracy, confusion matrix, precision, recall, and F1-score help in understanding the quality of predictions and model generalization when applied to real-world classification tasks.