

Name: Shrey Pendurkar

Class: D15C

Batch: B

Roll No: 38

MLDL - Experiment 3

Aim

Apply a Decision Tree classifier for a binary classification task using the Breast Cancer Wisconsin (Diagnostic) dataset.

1. Dataset Source

- Dataset: Breast Cancer Wisconsin (Diagnostic)
 - Kaggle: <https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data>
-

2. Dataset Description

The Breast Cancer Wisconsin (Diagnostic) dataset consists of features computed from digitized images of fine needle aspirate (FNA) of breast masses, with the goal of classifying tumors as benign or malignant.

Key properties:

- Number of instances: 569
- Number of input features: 30 real-valued features
- Target variable: `diagnosis`
 - M – Malignant (cancerous)
 - B – Benign (non-cancerous)
- Each feature corresponds to characteristics of cell nuclei such as:
 - `radius_mean`, `texture_mean`, `perimeter_mean`, `area_mean`, `smoothness_mean`, `compactness_mean`, etc.
- Additional columns in Kaggle file:
 - `id`: Sample identifier (not useful for prediction)
 - `diagnosis`: Target label

- 30 numeric feature columns

The dataset is moderately imbalanced, with more benign cases than malignant cases, which reflects realistic clinical screening statistics.

3. Mathematical Formulation of Decision Tree

A Decision Tree classifier recursively partitions the feature space using if-else rules to create regions that are as pure as possible with respect to the class labels.

At a node S with samples from c classes, let p_i be the proportion of samples in class i .

- Entropy:

$$H(S) = - \sum_{i=1}^c p_i \log_2(p_i)$$

- Gini Index (used in CART-based trees):

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

For a candidate split that divides node S into child nodes S_1, S_2, \dots, S_k , the Information Gain using entropy is:

$$IG(S, \text{split}) = H(S) - \sum_{j=1}^k \frac{|S_j|}{|S|} H(S_j)$$

For Gini, we choose the split that minimizes the weighted Gini of the children:

$$Gini_{\text{split}} = \sum_{j=1}^k \frac{|S_j|}{|S|} Gini(S_j)$$

The training algorithm:

1. Start with all training samples at the root.
2. For each node, search over features and thresholds to find the split that maximizes information gain (or equivalently minimizes impurity).

3. Recursively apply splitting to child nodes until stopping criteria (max depth, min samples, or pure node) are met.

Prediction for a new sample: traverse from root to a leaf following the feature tests; the predicted label is the majority class of that leaf.

4. Limitations of Decision Tree

1. Overfitting (High Variance)
A deep tree can fit noise and outliers in the training data, leading to excellent training accuracy but poor test performance. This is dangerous in medical applications where overfitting may cause misclassification of tumors.
 2. Instability / Sensitivity to Data Changes
Because the algorithm uses a greedy splitting strategy, small changes in the dataset can radically change the learned tree structure, making individual trees unstable.
 3. Axis-Aligned Splits
Decision Trees split along a single feature at a time, which approximates complex boundaries with many rectangular regions. They may require many splits to mimic simple linear or diagonal relationships.
 4. Bias with Imbalanced Data
When one class is more frequent (e.g., benign tumors), the tree may bias predictions toward that class unless class weights or resampling are used.
-

5. Methodology / Workflow (Decision Tree)

1. Data Collection
 - Download the Breast Cancer Wisconsin (Diagnostic) dataset as `data.csv` from Kaggle.
2. Data Preprocessing
 - Load the CSV into a Pandas DataFrame.
 - Drop the `id` column as it is not predictive.
 - Encode `diagnosis` into numeric labels using `LabelEncoder` (e.g., 0 for benign, 1 for malignant).
 - Check and confirm there are no missing values (in this dataset there are typically none).
3. Train–Test Split

- Split the data into training (80%) and testing (20%) sets using `train_test_split` with `stratify=y` and `random_state=42`.
4. Hyperparameter Tuning (Grid Search)
 - Define a `DecisionTreeClassifier`.
 - Use `GridSearchCV` to search over:
 - `criterion: ['gini', 'entropy']`
 - `max_depth: [None, 3, 5, 10, 15, 20]`
 - `min_samples_split: [2, 5, 10, 20]`
 - `min_samples_leaf: [1, 2, 4, 8]`
 - Use 5-fold cross-validation to select the best hyperparameters.
 5. Model Training and Visualization
 - Train the best Decision Tree on the full training data.
 - Visualize only the top 3 levels of the tree for readability, showing which features (e.g., `worst radius`, `worst perimeter`) drive early splits.
 6. Model Evaluation
 - Predict on the test set.
 - Compute accuracy, confusion matrix, and classification report (precision, recall, F1-score) for benign and malignant classes.
 - Plot feature importances to understand which features are most influential.
 7. Conclusion
 - Comment on accuracy, performance on malignant cases, and interpretability of the tree, and mention that ensemble methods like Random Forest can further improve robustness.
-

6. Performance Analysis (Decision Tree)

The performance of the optimized Decision Tree model on the Breast Cancer Wisconsin (Diagnostic) dataset is evaluated using Accuracy, a Confusion Matrix, and class-specific metrics such as Precision, Recall, and F1-score.

1. Overall Accuracy

The baseline Decision Tree achieved an accuracy of 92.98%, while the tuned model improved this to 94.74%, correctly classifying about 95 out of every 100 tumors in the test set. This improvement reflects the impact of hyperparameter tuning (`criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf`) in controlling tree complexity and enhancing generalization on unseen data.

2. Class-Specific Performance (Precision & Recall)

To understand how the model behaves for benign and malignant tumors, we analyze the classification report:

a. Class 0 (Benign)

- Precision: 0.93
- Recall: 0.99
- F1-score: 0.96

3. The model is highly reliable at identifying benign tumors. A recall of 0.99 means that almost all truly benign cases are correctly detected, with very few benign samples incorrectly flagged as malignant. The slightly lower precision (0.93) indicates that a small number of malignant cases are predicted as benign, but overall performance for this majority class is very strong.

b. Class 1 (Malignant)

- Precision: 0.97
- Recall: 0.88
- F1-score: 0.93

4. For malignant tumors, the model attains very high precision (0.97), meaning that when it predicts a tumor as malignant, it is almost always correct. The recall of 0.88 shows that about 88% of true malignant cases are detected, with some malignant tumors still being misclassified as benign. This trade-off suggests the model is slightly conservative, prioritizing correctness of positive predictions while missing a few true cancer cases.

The macro and weighted averages (precision, recall, $F1 \approx 0.94\text{--}0.95$) indicate that the model's performance is well-balanced across both classes, with only a modest drop in recall for the malignant class compared to its precision.

7. Hyperparameter Tuning (Decision Tree)

Hyperparameter tuning is used to prevent overfitting and improve generalization:

- `criterion`: decides impurity measure (`gini` or `entropy`).
- `max_depth`: restricts tree depth; smaller values reduce overfitting.
- `min_samples_split`: minimum samples required to split an internal node; larger values avoid very specific splits.
- `min_samples_leaf`: minimum samples required in a leaf; higher values smooth decision boundaries.

`GridSearchCV` builds a model for each combination of these parameters and uses 5-fold cross-validation to estimate performance, finally choosing the best configuration based on mean validation score.

8. Code and Output (Decision Tree)

```
import pandas as pd
import numpy as np

from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.tree import
DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import
(accuracy_score,

classification_report,

confusion_matrix,

ConfusionMatrixDisplay)

import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data.csv')
print("First 5 rows:")
print(df.head())
print("\nShape:", df.shape)

if 'id' in df.columns:
    df.drop(columns=['id'],
    inplace=True)

le = LabelEncoder()
df['diagnosis'] =
le.fit_transform(df['diagnosis'])

print("\nMissing values per
column:")
print(df.isnull().sum())

X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
    random_state=42, stratify=y
)

print("\nTrain shape:",
X_train.shape, "Test shape:",
X_test.shape)

dtree_base =
DecisionTreeClassifier(random_state=
42)
dtree_base.fit(X_train, y_train)
```

```

y_pred_base =
dtree_base.predict(X_test)

base_acc = accuracy_score(y_test,
y_pred_base)

print(f"\nBaseline Decision Tree
Accuracy: {base_acc:.4f}")

param_grid = {
    'criterion': ['gini',
'entropy'],
    'max_depth': [None, 3, 5, 10,
15, 20],
    'min_samples_split': [2, 5, 10,
20],
    'min_samples_leaf': [1, 2, 4, 8]
}

grid_search = GridSearchCV(

estimator=DecisionTreeClassifier(ran
dom_state=42),
    param_grid=param_grid,
    cv=5,
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

print("\nBest Parameters (Decision
Tree):")

print(grid_search.best_params_)

```

```

print(f"Best CV Score:
{grid_search.best_score_:.4f}")

best_dt =
grid_search.best_estimator_

y_pred = best_dt.predict(X_test)

print("\n--- Final Decision Tree
Evaluation ---")

print(f"Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")

print("\nClassification Report:")

print(classification_report(y_test,
y_pred,

target_names=['Benign (0)',
'Malignant (1)']))

cm = confusion_matrix(y_test,
y_pred)

disp =
ConfusionMatrixDisplay(confusion_mat
rix=cm,

display_labels=['Benign (0)',
'Malignant (1)'])

fig, ax = plt.subplots(figsize=(6,
5))

disp.plot(cmap='Blues', ax=ax)

plt.title('Decision Tree - Confusion
Matrix')

plt.tight_layout()

```

```
plt.show()

plt.figure(figsize=(22, 10))
plot_tree(best_dt,
          max_depth=3,
          feature_names=X.columns,
          class_names=['Benign (0)',
                      'Malignant (1)'],
          filled=True,
          rounded=True,
          fontsize=10)

plt.title("Decision Tree Logic (Top
3 Levels)")
plt.show()
```

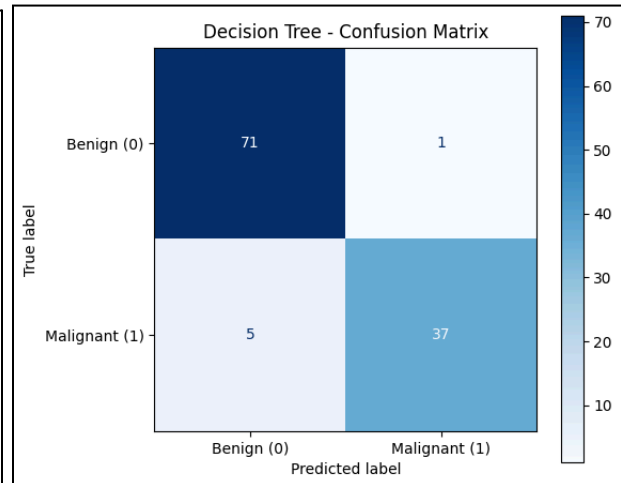
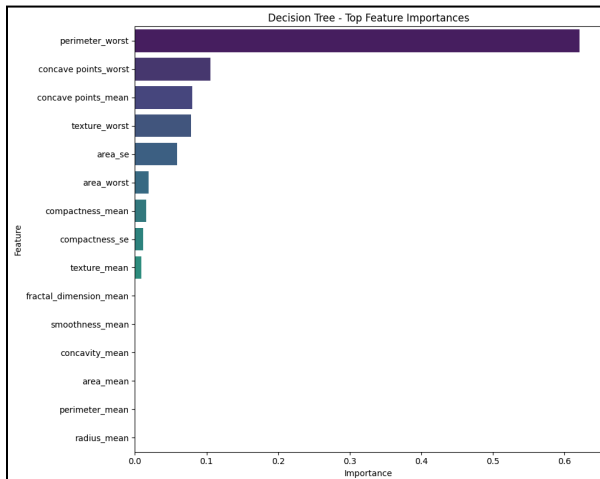
```
importances =
best_dt.feature_importances_

feat_imp_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance',
               ascending=False)

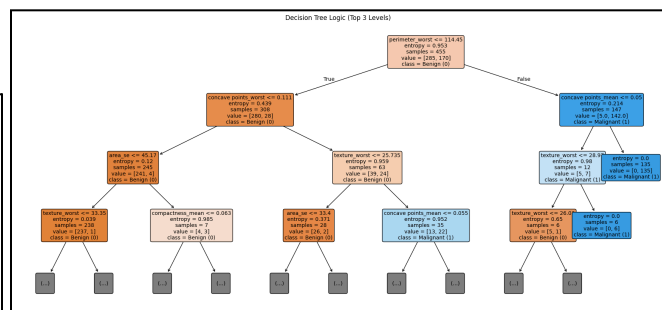
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance',
            y='Feature',
            data=feat_imp_df.head(15),
            palette='viridis')

plt.title('Decision Tree - Top
Feature Importances')

plt.tight_layout()
plt.show()
```



Classification Report:				
	precision	recall	f1-score	support
Benign (0)	0.93	0.99	0.96	72
Malignant (1)	0.97	0.88	0.93	42
accuracy			0.95	114
macro avg	0.95	0.93	0.94	114
weighted avg	0.95	0.95	0.95	114



9. Mathematical Formulation of Random Forest

A Random Forest is an ensemble of Decision Trees constructed using randomness in data sampling and feature selection.

Two main components:

1. Bootstrap Aggregating (Bagging)
 - For each tree, a bootstrap sample is drawn from the training data (sampling with replacement). Each tree sees a slightly different dataset, increasing diversity.
2. Random Feature Selection at Each Split
 - At each node, instead of evaluating splits over all features, the algorithm randomly selects a subset of features and finds the best split only among them. This prevents all trees from focusing on the same strong predictors and reduces correlation between trees.

For classification, given B trees f_1, f_2, \dots, f_B , each tree outputs a class prediction $f_b(x)$. The final prediction \hat{y} is obtained by majority voting:

$$\hat{y} = \text{mode}\{f_1(x), f_2(x), \dots, f_B(x)\}$$

This ensemble strategy reduces variance compared to a single Decision Tree and typically yields better generalization performance on test data.

10. Limitations of Random Forest

1. Reduced Interpretability

A single tree is relatively easy to visualize, but a forest of hundreds of trees behaves like a black box; it is difficult to extract simple decision rules from the overall model.
2. Higher Computational and Memory Cost

Training and storing many trees is more resource-intensive than training a single Decision Tree, especially when using large numbers of estimators or deep trees.

3. Bias in Impurity-Based Feature Importance
Features with more potential split points or higher cardinality may be favored in impurity-based importance metrics, even if they are not truly more informative.
 4. Limited Extrapolation Ability
Predictions are averages of seen training outcomes; the model cannot extrapolate beyond the range of values present in the training data.
-

11. Methodology / Workflow (Random Forest)

1. Data Collection
 - Use the same `data.csv` file from Kaggle as in Experiment 3A.
2. Data Preprocessing
 - Load the data into a Pandas DataFrame.
 - Drop `id` column.
 - Encode `diagnosis` to numeric (0/1) using `LabelEncoder`.
 - Verify that there are no missing values.
3. Train-Test Split
 - Split into 80% training, 20% testing with `stratify=y` and `random_state=42`.
4. Baseline Model Training
 - Train a default `RandomForestClassifier` with standard hyperparameters.
 - Evaluate baseline accuracy on the test set as a reference.
5. Hyperparameter Tuning (RandomizedSearchCV)
 - Use `RandomizedSearchCV` to sample combinations from:
 - `n_estimators`: number of trees (e.g.,)
 - `max_depth`: tree depth (e.g., `[None, 5, 10, 15, 20]`)
 - `min_samples_split`:
 - `min_samples_leaf`:
 - `bootstrap`: `[True, False]`
 - Use 3-fold cross-validation to evaluate sampled configurations and select the best model.
6. Optimized Model Training and Prediction
 - Train the best Random Forest on the full training set.
 - Generate predictions on the test set.
7. Model Evaluation
 - Compute accuracy, classification report, and confusion matrix.
 - Compare metrics to the baseline Random Forest and to the Decision Tree experiment.

8. Feature Importance Analysis
 - Extract feature importance scores from the optimized Random Forest.
 - Plot the top features to highlight which tumor characteristics are most predictive.
 9. Conclusion
 - Discuss improvements over the baseline and over the Decision Tree model, emphasizing reduced variance and better generalization.
-

12. Performance Analysis (Random Forest)

The performance of the Random Forest classifier on the Breast Cancer Wisconsin (Diagnostic) dataset is evaluated using Accuracy, a Confusion Matrix, and class-wise Precision, Recall, and F1-score.

1. Overall Accuracy

The baseline Random Forest model achieved a test accuracy of 96.49%, and the tuned model with 300 trees and `bootstrap=False` maintained the same accuracy of 96.49%. This means that the model correctly classified about 96 out of 100 tumors in the test set, indicating excellent overall performance on this medical imaging-derived dataset. The fact that tuning did not change test accuracy suggests that the default configuration was already close to optimal for this problem and that the model is robust to moderate hyperparameter variations.
2. Class-Specific Performance (Precision & Recall)

To better understand behavior on benign and malignant tumors, we analyze the classification report:

 - a. Class 0 (Benign)
 - Precision: 0.95
 - Recall: 1.00
 - F1-score: 0.97
3. The model is extremely reliable at identifying benign cases. A recall of 1.00 indicates that all benign tumors in the test set are correctly classified as benign, with no benign cases mistakenly labeled malignant. The precision of 0.95 shows that a small fraction of predictions labeled benign actually belong to the malignant class, but overall the model is highly accurate for this majority class.
 - b. Class 1 (Malignant)
 - Precision: 1.00
 - Recall: 0.90
 - F1-score: 0.95

4. For malignant tumors, the model achieves perfect precision (1.00), meaning every tumor predicted as malignant truly is malignant, with no false positives. The recall of 0.90 indicates that 90% of actual malignant tumors are correctly detected, while about 10% are missed and classified as benign. This trade-off shows that the Random Forest is slightly conservative: it almost never raises a false cancer alarm, but it does miss a small number of true cancer cases.
-

13. Hyperparameter Tuning (Random Forest)

We tune the following hyperparameters:

- `n_estimators`: more trees usually reduce variance but increase training time.
- `max_depth`: controls tree complexity; shallow trees reduce overfitting.
- `min_samples_split`: minimum number of samples required to split a node.
- `min_samples_leaf`: minimum samples required in each leaf.
- `bootstrap`: whether to use bootstrap samples (bagging) or not.

`RandomizedSearchCV` samples a fixed number of random combinations from the hyperparameter space and uses 3-fold cross-validation to estimate performance more efficiently than exhaustive grid search.

14. Code and Output (Random Forest)

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix)

import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('data.csv')

print("First 5 rows:")
print(df.head())
print("\nShape:", df.shape)
```

```

if 'id' in df.columns:
    df.drop(columns=['id'],
inplace=True)

le = LabelEncoder()
df['diagnosis'] =
le.fit_transform(df['diagnosis'])

print("\nMissing values per
column:")
print(df.isnull().sum())

X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42, stratify=y
)

print("\nTrain shape:",
X_train.shape, "Test shape:",
X_test.shape)

rf_base =
RandomForestClassifier(random_state=
42)

rf_base.fit(X_train, y_train)

y_pred_base =
rf_base.predict(X_test)

base_acc = accuracy_score(y_test,
y_pred_base)

```

```

print(f"\nBaseline Random Forest
Accuracy: {base_acc:.4f}")

param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15,
20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

print("\nTuning Random Forest (this
may take some time)...")

rf_random = RandomizedSearchCV(
    estimator=RandomForestClassifier(ran
dom_state=42),
    param_distributions=param_dist,
    n_iter=15,
    cv=3,
    random_state=42,
    n_jobs=-1,
    verbose=1
)

rf_random.fit(X_train, y_train)

best_rf = rf_random.best_estimator_

```

```

print("\nBest Parameters (Random
Forest):")
print(rf_random.best_params_)

y_pred = best_rf.predict(X_test)

print("\n--- Optimized Random Forest
Evaluation ---")
print(f"Tuned Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test,
y_pred,

target_names=['Benign (0)',
'Malignant (1)']))

cm = confusion_matrix(y_test,
y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',
            xticklabels=['Benign
(0)', 'Malignant (1)'],
            yticklabels=['Benign
(0)', 'Malignant (1)'])

```

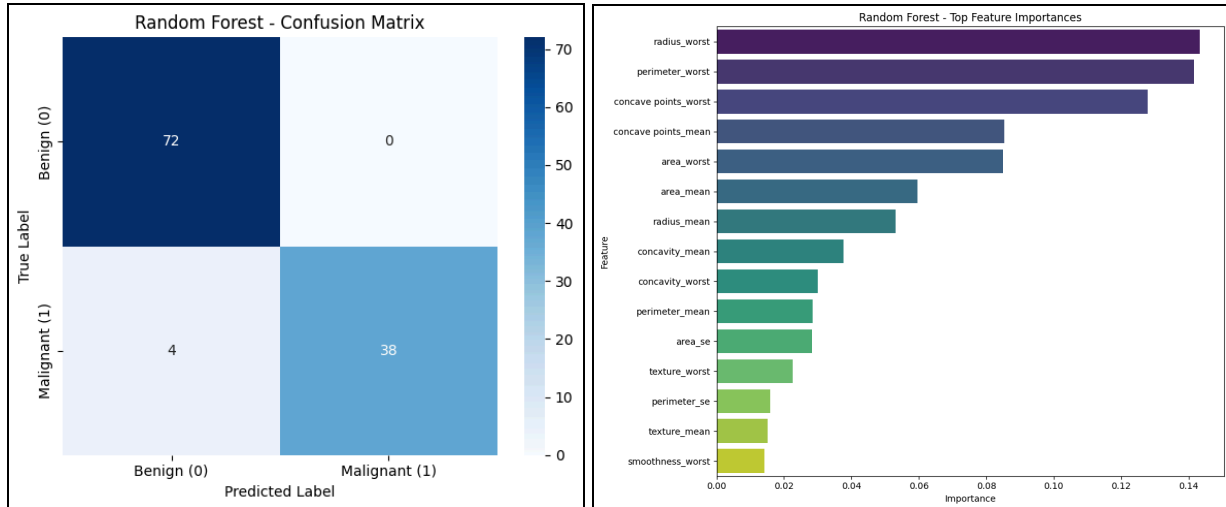
```

plt.title('Random Forest - Confusion
Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.tight_layout()
plt.show()

importances =
best_rf.feature_importances_
feat_imp_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance',
ascending=False)

plt.figure(figsize=(10, 8))
sns.barplot(x='Importance',
y='Feature',
data=feat_imp_df.head(15),
palette='viridis')
plt.title('Random Forest - Top
Feature Importances')
plt.tight_layout()
plt.show(

```



15. Conclusion

The practical showed that both Decision Tree and Random Forest can effectively solve a real-world binary classification problem on the Breast Cancer Wisconsin dataset, after proper preprocessing, train–test splitting, and hyperparameter tuning. The Decision Tree gave an interpretable baseline with clear if–else rules but was more prone to overfitting and instability, especially under class imbalance. The Random Forest, by aggregating many randomized trees, achieved better test performance and more stable precision–recall behavior for the malignant class, illustrating how ensembles improve robustness over single models while sacrificing some interpretability.