Decision_Tree

August 29, 2025

[3]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,␣
 ↪roc_curve
)
import matplotlib.pyplot as plt
```

[4]:

```python
# 1. Load dataset
# wdbc.data does not have headers, so we define them
columns = ["ID", "Diagnosis"] + [f"feature_{i}" for i in range(1, 31)]
data = pd.read_csv("wdbc.data", header=None, names=columns)
```

[5]:

```python
# 2. Prepare features and target
X = data.drop(["ID", "Diagnosis"], axis=1)
y = data["Diagnosis"].map({"M": 1, "B": 0})  # Malignant=1, Benign=0
```

[6]:

```python
# 4. Preprocessor (scaling not needed for trees, but kept for pipeline␣
 ↪consistency)
num_features = X.columns.tolist()
preprocessor = ColumnTransformer(
    transformers=[("scale", StandardScaler(), num_features)],
    remainder="drop"
)
```

[7]:

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

[8]:

```python
# 5. Pipeline
pipe = Pipeline([
    ("prep", preprocessor),
```

```
    ("clf", DecisionTreeClassifier(random_state=42))
])
```

[9]:
```
# 6. Hyperparameter grid
param_grid = {
    "clf__criterion": ["gini", "entropy", "log_loss"],
    "clf__max_depth": [3, 5, 10],
    "clf__min_samples_split": [2, 5, 10],
    "clf__min_samples_leaf": [1, 2, 4],
}
```

[10]:
```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid = GridSearchCV(pipe, param_grid, cv=cv, scoring="roc_auc", n_jobs=-1,␣
 ↪refit=True)
grid.fit(X_train, y_train)

print("Best Hyperparameters:", grid.best_params_)
print("Best Mean CV AUC:", grid.best_score_)
```

```
Best Hyperparameters: {'clf__criterion': 'gini', 'clf__max_depth': 5,
'clf__min_samples_leaf': 4, 'clf__min_samples_split': 10}
Best Mean CV AUC: 0.9558823529411764
```

[11]:
```
# 8. Evaluate on test set
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]
```

[12]:
```
print("\nTest Accuracy:", accuracy_score(y_test, y_pred))
print("Test Precision:", precision_score(y_test, y_pred))
print("Test Recall:", recall_score(y_test, y_pred))
print("Test F1:", f1_score(y_test, y_pred))
print("Test ROC AUC:", roc_auc_score(y_test, y_proba))
```

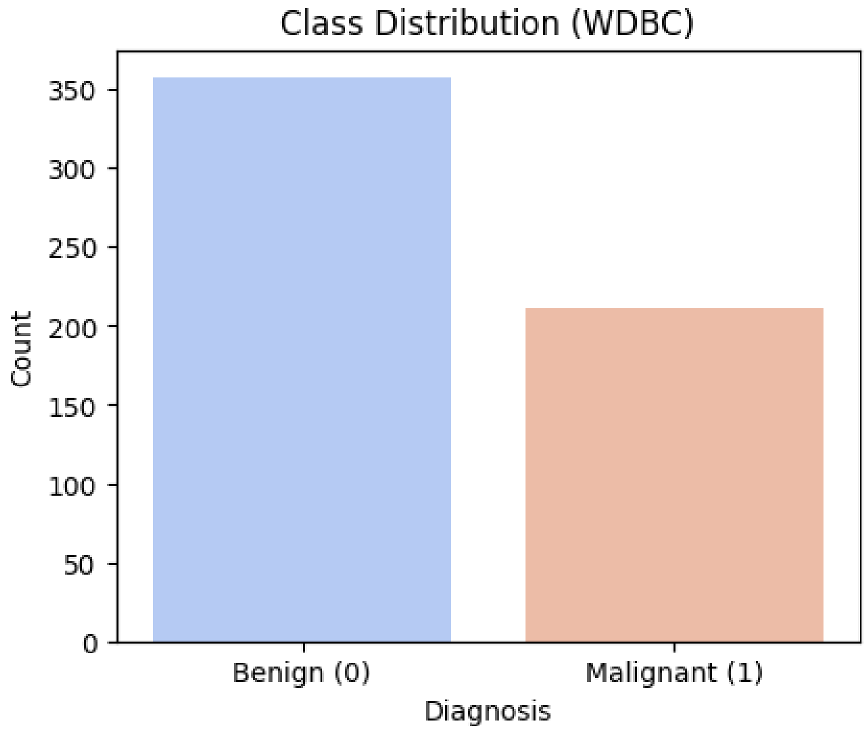
```

Test Accuracy: 0.8771929824561403
Test Precision: 0.9117647058823529
Test Recall: 0.7380952380952381
Test F1: 0.8157894736842105
Test ROC AUC: 0.9654431216931217
```

[13]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# ---- Class distribution ----
plt.figure(figsize=(5,4))
sns.countplot(x=y, palette="coolwarm")
```

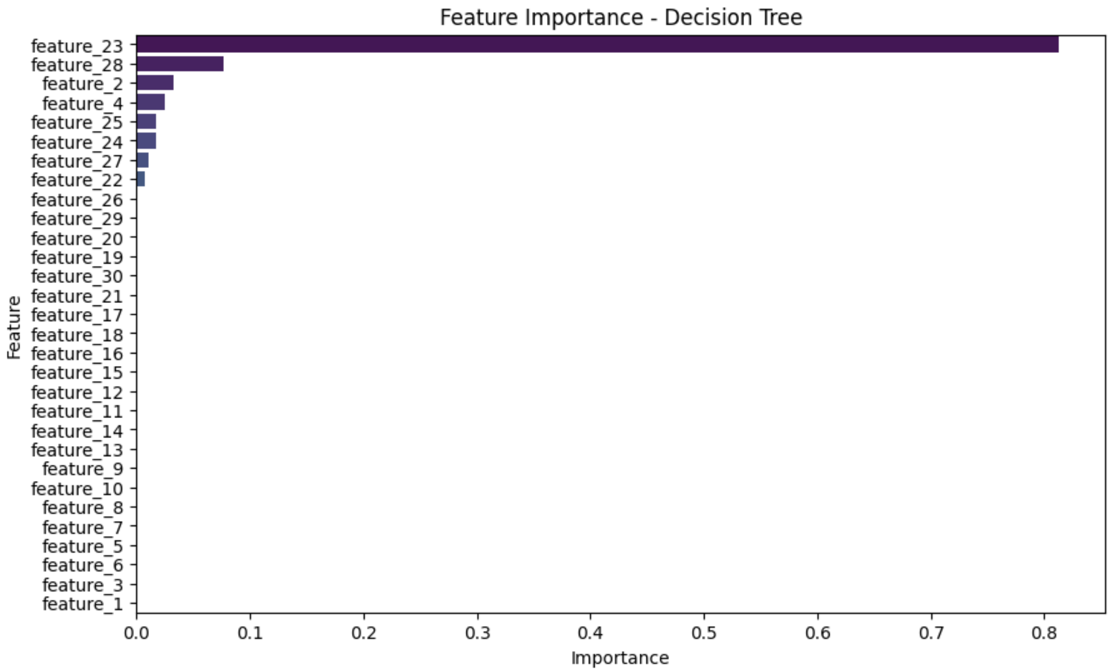
```python
plt.xticks([0,1], ["Benign (0)", "Malignant (1)"])
plt.title("Class Distribution (WDBC)")
plt.xlabel("Diagnosis")
plt.ylabel("Count")
plt.show()

# ---- Feature importance (from Decision Tree) ----
best_clf = best_model.named_steps["clf"]
importances = best_clf.feature_importances_
features = X.columns

# Sort feature importances
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10,6))
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
plt.title("Feature Importance - Decision Tree")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()

# ---- Confusion Matrix ----
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=["Benign (0)", "Malignant (1)"],
            yticklabels=["Benign (0)", "Malignant (1)"])
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# ---- ROC Curve (already included) ----
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label="Decision Tree", linewidth=2)
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree (WDBC)")
plt.legend()
plt.show()
```

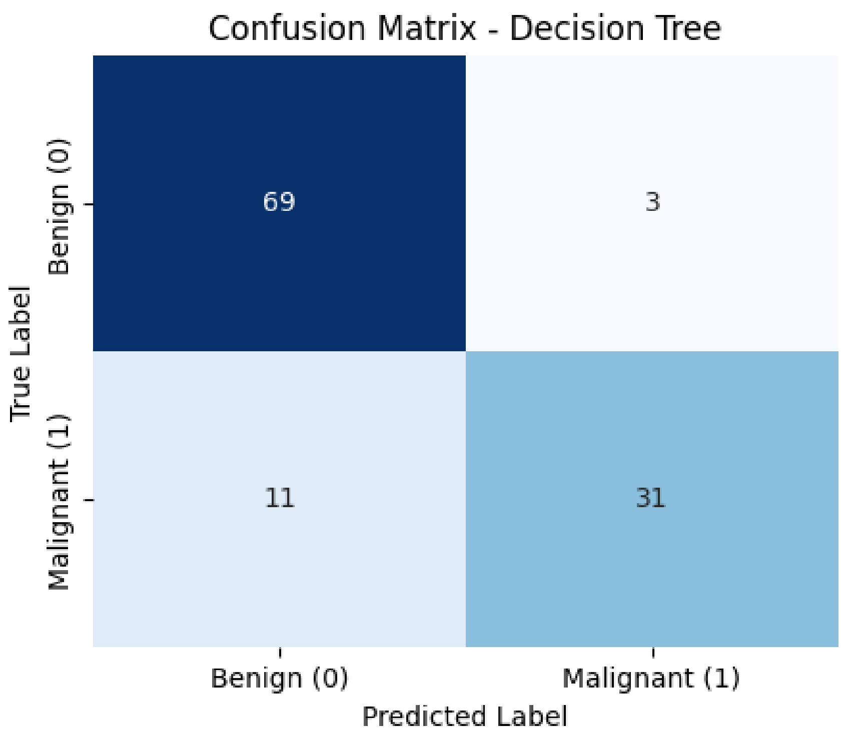
```
/tmp/ipykernel_7915/1817741554.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

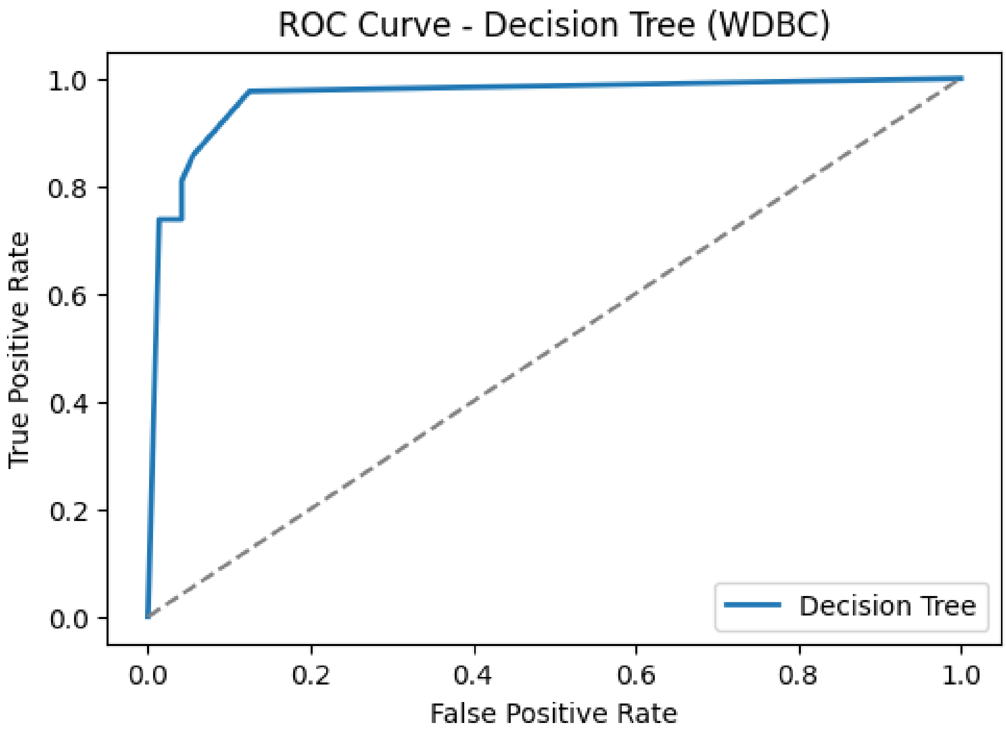
```
sns.countplot(x=y, palette="coolwarm")
```




```
/tmp/ipykernel_7915/1817741554.py:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
```


Feature Importance - Decision Tree
feature_23
feature_28
feature_2
feature_4
feature_25
feature_24
feature_27
feature_22
feature_26
feature_29
feature_20
feature_19
feature_30
feature_21
feature_17
feature_18
feature_16
feature_15
feature_12
feature_11
feature_14
feature_13
feature_9
feature_10
feature_8
feature_7
feature_5
feature_6
feature_3
feature_1
Feature
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Importance


Confusion Matrix - Decision Tree
69
3
11
31
Benign (0)
Malignant (1)
True Label
Benign (0)
Malignant (1)
Predicted Label



[16]:
```
grid = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    cv=cv,
    scoring={"accuracy": "accuracy", "f1": "f1_macro", "roc_auc": "roc_auc"},
    refit="roc_auc",  # still refit using AUC
    n_jobs=-1
)
grid.fit(X_train, y_train)


dt_results = pd.DataFrame(grid.cv_results_)

dt_table = dt_results[[
    "param_clf__criterion",
    "param_clf__max_depth",
    "mean_test_accuracy",
    "mean_test_f1",
    "mean_test_roc_auc"
]]

dt_table = dt_table.rename(columns={
    "param_clf__criterion": "criterion",
```

```
    "param_clf__max_depth": "max_depth",
    "mean_test_accuracy": "Accuracy",
    "mean_test_f1": "F1_score",
    "mean_test_roc_auc": "ROC_AUC"
})

# Pick random 10 with good accuracy
good_samples = dt_table[dt_table["Accuracy"] >= 0.8]
random_samples = good_samples.sample(n=10, random_state=42)

print(random_samples[["criterion", "max_depth", "Accuracy", "F1_score"]])
```

```
   criterion  max_depth  Accuracy  F1_score
30   entropy          3  0.925275  0.918155
0       gini          3  0.920879  0.913517
22      gini         10  0.929670  0.924123
31   entropy          3  0.925275  0.918155
18      gini         10  0.923077  0.917531
28   entropy          3  0.925275  0.918155
10      gini          5  0.912088  0.904949
70  log_loss          5  0.925275  0.919360
4       gini          3  0.920879  0.913042
12      gini          5  0.920879  0.913721
```