Decision Tree

AIM:

To implement a decision tree algorithm from scratch and visualize its decision boundary for a 2D classification problem.

ALGORITHM:

Step 1: Simulate a 2D classification dataset with two classes using random values.
Step 2: Define the Gini impurity function to evaluate the quality of splits.
Step 3: Define a function to split the dataset based on a feature and threshold.
Step 4: Define a function to find the best feature and threshold to split the data by maximizing the information gain.
Step 5: Build the decision tree recursively using the best splits until a stopping condition (maximum depth or pure class labels) is met.
Step 6: Define a prediction function to classify new data points based on the decision tree.
Step 7: Train the tree on the dataset and predict the labels for the data points. Evaluate accuracy by comparing predictions with actual labels.
Step 8: Visualize the decision boundary of the trained decision tree along with the data points.

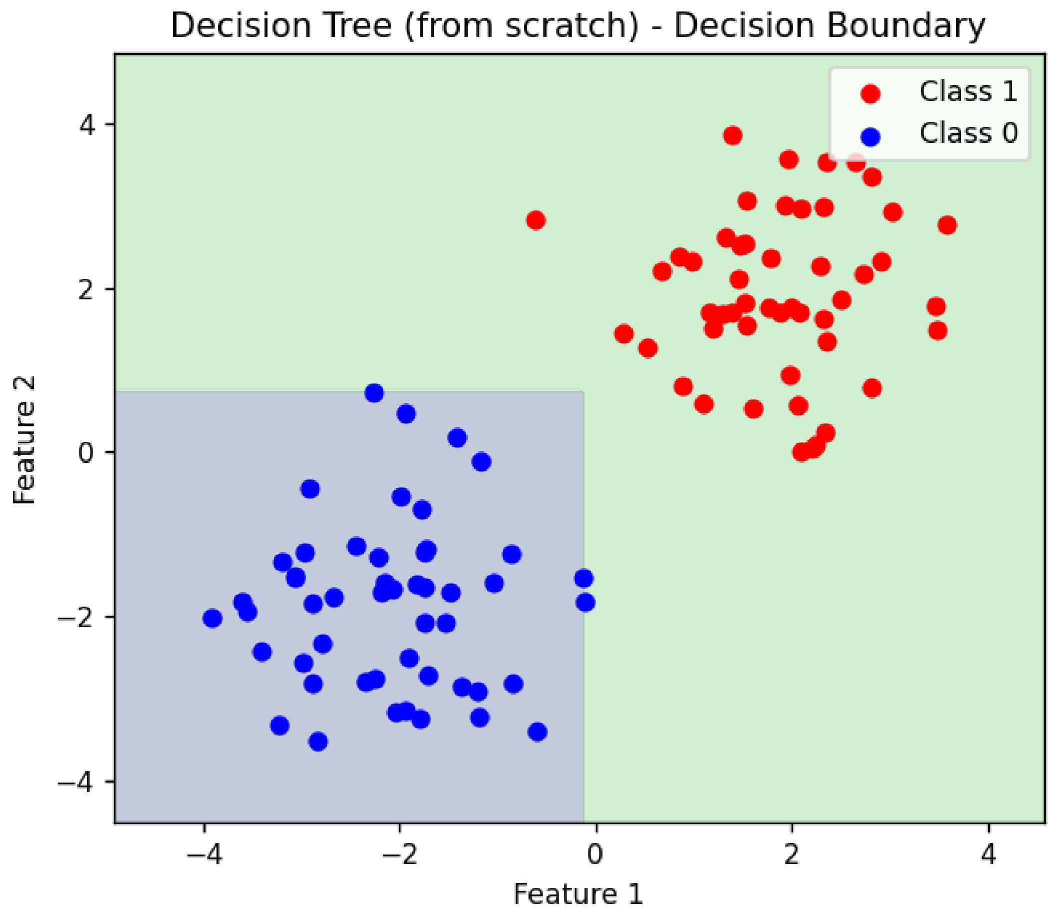
SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt

# 1. Simulate 2D classification data
np.random.seed(42)
X1 = np.random.randn(50, 2) + np.array([2, 2])
X2 = np.random.randn(50, 2) + np.array([-2, -2])
X = np.vstack([X1, X2])
y = np.hstack([np.ones(50), np.zeros(50)])

# 2. Gini Impurity
def gini(y):
    classes, counts = np.unique(y, return_counts=True)
```

```
    probs = counts / len(y)
    return 1 - np.sum(probs ** 2)

# 3. Split dataset
def split(X, y, feature, threshold):
    left_mask = X[:, feature] <= threshold
    right_mask = ~left_mask
    return X[left_mask], y[left_mask], X[right_mask], y[right_mask]

# 4. Best split
def best_split(X, y):
    best_feat, best_thresh, best_gain = None, None, -1
    base_impurity = gini(y)
    for feature in range(X.shape[1]):
        thresholds = np.unique(X[:, feature])
        for t in thresholds:
            _, y_left, _, y_right = split(X, y, feature, t)
            if len(y_left) == 0 or len(y_right) == 0:
                continue
            g = base_impurity - (len(y_left)/len(y)) * gini(y_left) - (len(y_right)/len(y)) * gini(y_right)
            if g > best_gain:
                best_feat, best_thresh, best_gain = feature, t, g
    return best_feat, best_thresh

# 5. Build the Tree
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, *, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value  # for leaf

def build_tree(X, y, depth=0, max_depth=5):
    if len(np.unique(y)) == 1 or depth >= max_depth:
        value = np.argmax(np.bincount(y.astype(int)))
        return Node(value=value)

    feature, threshold = best_split(X, y)
    if feature is None:
        value = np.argmax(np.bincount(y.astype(int)))
        return Node(value=value)

    X_left, y_left, X_right, y_right = split(X, y, feature, threshold)
    left = build_tree(X_left, y_left, depth+1, max_depth)
```

```
    right = build_tree(X_right, y_right, depth+1, max_depth)
    return Node(feature, threshold, left, right)

# 6. Predict with tree
def predict_tree(x, node):
    if node.value is not None:
        return node.value
    if x[node.feature] <= node.threshold:
        return predict_tree(x, node.left)
    else:
        return predict_tree(x, node.right)

# 7. Train & Predict
tree = build_tree(X, y)
y_pred = np.array([predict_tree(x, tree) for x in X])
acc = np.mean(y_pred == y)
print(f"\nAccuracy: {acc * 100:.2f}%")

# 8. Decision Boundary Visualization
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
grid = np.c_[xx.ravel(), yy.ravel()]
preds = np.array([predict_tree(pt, tree) for pt in grid])
Z = preds.reshape(xx.shape)

plt.figure(figsize=(6, 5))
plt.contourf(xx, yy, Z, alpha=0.3, levels=1)
plt.scatter(X1[:, 0], X1[:, 1], color='red', label='Class 1')
plt.scatter(X2[:, 0], X2[:, 1], color='blue', label='Class 0')
plt.title("Decision Tree (from scratch) - Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

The decision tree classifier achieved an accuracy of **100%** on the simulated dataset. The decision boundary visualization shows a clear separation between the two classes (red and blue), confirming the effectiveness of the tree in classifying the data