**Reg:** 231701039

**Decision Tree**

**Aim:**

To implement a **Decision Tree**  in Python from scratch to classify data based on input features, demonstrating understanding of the decision-making process in artificial intelligence using Gini Impurity and recursive tree construction.

**CODE**

import numpy as np

class Node:

def \_\_init\_\_(self, feature=None, threshold=None, left=None, right=None, value=None):

self.feature = feature # index of the feature to split on

self.threshold = threshold # threshold value for the split

self.left = left # left child node

self.right = right # right child node

self.value = value # class prediction (for leaf nodes)

def gini(y):

"""Calculate Gini Impurity"""

classes, counts = np.unique(y, return\_counts=True)

impurity = 1.0 - sum((count / len(y))\*\*2 for count in counts)

return impurity

def split(X, y, feature, threshold):

"""Split dataset based on a feature and threshold"""

left\_idx = np.where(X[:, feature] <= threshold)

right\_idx = np.where(X[:, feature] > threshold)

return X[left\_idx], y[left\_idx], X[right\_idx], y[right\_idx]

def best\_split(X, y):

"""Find the best split"""

best\_feature, best\_threshold, best\_gain = None, None, 0

parent\_gini = gini(y)

n\_features = X.shape[1]

for feature in range(n\_features):

thresholds = np.unique(X[:, feature])

for t in thresholds:

X\_left, y\_left, X\_right, y\_right = split(X, y, feature, t)

if len(y\_left) == 0 or len(y\_right) == 0:

continue

weighted\_gini = (len(y\_left) \* gini(y\_left) + len(y\_right) \* gini(y\_right)) / len(y)

gain = parent\_gini - weighted\_gini

if gain > best\_gain:

best\_gain = gain

best\_feature = feature

best\_threshold = t

return best\_feature, best\_threshold

def build\_tree(X, y, depth=0, max\_depth=3):

"""Recursively build the tree"""

if len(np.unique(y)) == 1 or depth >= max\_depth:

leaf\_value = np.bincount(y).argmax()

return Node(value=leaf\_value)

feature, threshold = best\_split(X, y)

if feature is None:

leaf\_value = np.bincount(y).argmax()

return Node(value=leaf\_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)

def predict\_single(node, x):

"""Predict class for a single sample"""

if node.value is not None:

return node.value

if x[node.feature] <= node.threshold:

return predict\_single(node.left, x)

else:

return predict\_single(node.right, x)

def predict(tree, X):

return [predict\_single(tree, x) for x in X]

**RESULT:**

The Decision Tree Classifier was successfully implemented.