**Machine Learning Experiment 9**

**Title: Write a program to demonstrate the working of decision tree based CART algorithm. Build the decision tree and classify a new sample using suitable dataset. Compare the performance with that of ID, C4.5, and CART in terms of accuracy, recall, precision and sensitivity.**

**Objective:** The objective of this lab assignment is to implement the decision tree-based Classification and Regression Trees (CART) algorithm and compare its performance with other decision tree algorithms, namely ID3 and C4.5, in terms of accuracy, recall, precision, and sensitivity. The assignment includes building decision trees, classifying new samples, and evaluating the models using a suitable dataset.

**Tasks:**

**1. Implement the CART algorithm from scratch.**

**Code:**

import numpy as np

import pandas as pd

class TreeNode:

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

self.feature\_index = feature\_index

self.threshold = threshold

self.left = left

self.right = right

self.value = value

class CART:

def fit(self, X, y):

self.root = self.\_grow\_tree(X, y)

def \_grow\_tree(self, X, y):

if len(set(y)) == 1:

return TreeNode(value=y[0])

best\_gain = -float('inf')

best\_criteria = None

best\_sets = None

n\_samples, n\_features = X.shape

for feature\_index in range(n\_features):

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

for threshold in thresholds:

left\_indices = X[:, feature\_index] <= threshold

right\_indices = X[:, feature\_index] > threshold

if len(y[left\_indices]) == 0 or len(y[right\_indices]) == 0:

continue

gain = self.\_gini\_gain(y, left\_indices, right\_indices)

if gain > best\_gain:

best\_gain = gain

best\_criteria = (feature\_index, threshold)

best\_sets = (left\_indices, right\_indices)

if best\_gain == -float('inf'):

return TreeNode(value=np.bincount(y).argmax())

left\_node = self.\_grow\_tree(X[best\_sets[0]], y[best\_sets[0]])

right\_node = self.\_grow\_tree(X[best\_sets[1]], y[best\_sets[1]])

return TreeNode(feature\_index=best\_criteria[0], threshold=best\_criteria[1], left=left\_node, right=right\_node)

def \_gini\_gain(self, y, left\_indices, right\_indices):

parent\_impurity = self.\_gini\_impurity(y)

left\_impurity = self.\_gini\_impurity(y[left\_indices])

right\_impurity = self.\_gini\_impurity(y[right\_indices])

n = len(y)

n\_left = len(y[left\_indices])

n\_right = len(y[right\_indices])

weighted\_child\_impurity = (n\_left / n) \* left\_impurity + (n\_right / n) \* right\_impurity

return parent\_impurity - weighted\_child\_impurity

def \_gini\_impurity(self, y):

class\_counts = np.bincount(y)

return 1.0 - np.sum((class\_counts / len(y)) \*\* 2)

def predict(self, X):

return [self.\_predict(inputs) for inputs in X]

def \_predict(self, inputs):

node = self.root

while node.value is None:

if inputs[node.feature\_index] <= node.threshold:

node = node.left

else:

node = node.right

return node.value

**2. Build decision trees using the implemented CART, as well as the scikit-learn library's implementations for ID3 and C4.5.**

**Code:**

from sklearn.tree import DecisionTreeClassifier

def build\_id3(X\_train, y\_train):

model = DecisionTreeClassifier(criterion='entropy') # ID3 uses entropy

model.fit(X\_train, y\_train)

return model

def build\_c45(X\_train, y\_train):

model = DecisionTreeClassifier(criterion='gini') # C4.5 uses Gini impurity

model.fit(X\_train, y\_train)

return model

**3. Evaluate the performance of each decision tree model on the testing dataset.**

**Code:**

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, confusion\_matrix

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test) if hasattr(model, 'predict') else model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

# Confusion matrix

tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()

sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0 # Avoid division by zero

return accuracy, recall, precision, sensitivity

**4. Compare the performance of the CART, ID3, and C4.5 decision tree algorithms based on the evaluation metrics.**

**Code:**

from sklearn.model\_selection import train\_test\_split

def load\_titanic\_dataset():

url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"

data = pd.read\_csv(url)

# Preprocess the data: drop rows with missing target and fill missing values

data = data.drop(columns=["Name", "Ticket", "Cabin"]) # Drop non-feature columns

data['Embarked'] = data['Embarked'].fillna('S') # Fill missing Embarked

data['Age'] = data['Age'].fillna(data['Age'].median()) # Fill missing Age with median

data['Fare'] = data['Fare'].fillna(data['Fare'].median()) # Fill missing Fare with median

# Convert categorical variables to dummy variables

data = pd.get\_dummies(data, columns=["Sex", "Embarked"], drop\_first=True)

X = data.drop("Survived", axis=1) # Features

y = data["Survived"] # Target labels

return train\_test\_split(X, y, test\_size=0.2, random\_state=42)

def compare\_models():

X\_train, X\_test, y\_train, y\_test = load\_titanic\_dataset()

# Implement CART from scratch

cart\_model = CART()

cart\_model.fit(X\_train.to\_numpy(), y\_train.to\_numpy())

# Using scikit-learn for ID3 and C4.5

id3\_model = build\_id3(X\_train, y\_train)

c45\_model = build\_c45(X\_train, y\_train)

# Evaluate models

cart\_metrics = evaluate\_model(cart\_model, X\_test.to\_numpy(), y\_test.to\_numpy())

id3\_metrics = evaluate\_model(id3\_model, X\_test, y\_test)

c45\_metrics = evaluate\_model(c45\_model, X\_test, y\_test)

# Print comparison

print("CART Metrics (Accuracy, Recall, Precision, Sensitivity):", cart\_metrics)

print("ID3 Metrics (Accuracy, Recall, Precision, Sensitivity):", id3\_metrics)

print("C4.5 Metrics (Accuracy, Recall, Precision, Sensitivity):", c45\_metrics)

# Run the comparison

compare\_models()

**Output:**

****