|  |  |
| --- | --- |
| **Ex. No: 3**  **Date: 10.01.24** | **Decision Tree Classifier** |

**Name: S Sowmithaa Sri**

**Reg no: 22011101096**

**Aim:**

To implement a supervised Machine learning algorithm

1. Decision tree Classifier

In mathematical version and using the built-in function from sklearn.

**Dataset description:**

The dataset consists of observations on various weather conditions, each described by four features: 'Outlook' representing the current weather state (e.g., 'Sunny', 'Overcast', 'Rainy'), 'Temperature' denoting the temperature level ('Hot', 'Mild', 'Cool'), 'Humidity' indicating humidity status ('High', 'Normal'), and 'Wind' signifying wind strength ('Weak', 'Strong'). The target variable 'PlayTennis' signifies whether or not tennis was played under those conditions ('Yes' or 'No').

**Code:**

1. **DTC- Mathematical Version:**

import pandas as pd

import numpy as np

# Sample dataset

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes']

}

df = pd.DataFrame(data)

# Function to find the best split for a feature

def find\_best\_split(data, feature, target):

unique\_values = np.unique(data[feature])

best\_split\_value = None

best\_gini = float('inf')

for value in unique\_values:

left\_subset = data[data[feature] == value][target]

right\_subset = data[data[feature] != value][target]

gini = (left\_subset.shape[0] / data.shape[0]) \* calculate\_gini(left\_subset) \

+ (right\_subset.shape[0] / data.shape[0]) \* calculate\_gini(right\_subset)

if gini < best\_gini:

best\_gini = gini

best\_split\_value = value

return best\_split\_value

# Function to calculate Gini impurity

def calculate\_gini(target):

if target.empty:

return 0

total\_samples = target.shape[0]

# Calculate class probabilities

class\_probs = target.value\_counts() / total\_samples

# Calculate Gini impurity

gini = 1 - np.sum(np.square(class\_probs))

return gini

# Recursive function to build the decision tree

def build\_decision\_tree(data, features, target):

if len(np.unique(data[target])) == 1:

return np.unique(data[target])[0]

if len(features) == 0 or data.shape[0] == 0:

# Return the most frequent class in the parent node or a default value if necessary

return np.unique(data[target])[np.argmax(np.unique(data[target], return\_counts=True)[1])]

best\_feature = None

best\_split\_value = None

best\_gini = float('inf')

for feature in features:

split\_value = find\_best\_split(data, feature, target)

left\_data = data[data[feature] == split\_value]

right\_data = data[data[feature] != split\_value]

if left\_data.empty or right\_data.empty:

continue # Skip this split if either subset is empty

gini = (left\_data.shape[0] / data.shape[0]) \* calculate\_gini(left\_data[target]) \

+ (right\_data.shape[0] / data.shape[0]) \* calculate\_gini(right\_data[target])

if gini < best\_gini:

best\_gini = gini

best\_feature = feature

best\_split\_value = split\_value

if best\_gini == float('inf'):

return np.unique(data[target])[np.argmax(np.unique(data[target], return\_counts=True)[1])]

features.remove(best\_feature)

tree = {best\_feature: {}}

left\_subset = data[data[best\_feature] == best\_split\_value]

right\_subset = data[data[best\_feature] != best\_split\_value]

tree[best\_feature]['left'] = build\_decision\_tree(left\_subset, features.copy(), target)

tree[best\_feature]['right'] = build\_decision\_tree(right\_subset, features.copy(), target)

return tree

# Example usage

features = ['Outlook', 'Temperature', 'Humidity', 'Wind']

target = 'PlayTennis'

decision\_tree = build\_decision\_tree(df, features, target)

print(decision\_tree)

**Result:**

{'Outlook': {'left': {'Temperature': {'left': 'Yes', 'right': 'No'}}, 'right': {'Wind': {'left': 'No', 'right': 'Yes'}}}}

1. **DTC- Built-in Version**

**Code:**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import OrdinalEncoder

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

from sklearn.metrics import accuracy\_score

# Sample dataset

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes']

}

df = pd.DataFrame(data)

# Encode categorical features

encoder = OrdinalEncoder()

X\_encoded = encoder.fit\_transform(df[['Outlook', 'Temperature', 'Humidity', 'Wind']])

y = df['PlayTennis']

# Split the dataset into training and testing sets

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

# Create and train Decision Tree classifier

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred\_dt = dt\_model.predict(X\_test)

# Evaluate the accuracy

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

print(f"Decision Tree Accuracy: {accuracy\_dt}")

**Result:**

Decision Tree Accuracy: 0.5

**Result Description:**

For the decision tree classifier implemented using a mathematical approach, the code recursively builds a tree by selecting the best feature to split the data at each node based on the Gini impurity. The tree is constructed until a stopping criterion is met, and the leaf nodes represent the predicted class. The implementation handles scenarios where a subset of data is empty by returning the most frequent class in the parent node.

In the version using the built-in library (scikit-learn), the code encodes categorical features, splits the dataset into training and testing sets, and trains a Decision Tree classifier. The trained model is used to predict the test set, and accuracy is evaluated.

**Conclusion:**

Thus the supervised ML algorithm-Decision Tree Classifier is successfully implemented and executed.