**Practical 8 A:**



**Decision Tree** is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

**Data-set Description :**

Title : Balance Scale Weight & Distance Database

Number of Instances : 625 (49 balanced, 288 left, 288 right)

Number of Attributes : 4 (numeric) + class name = 5

Attribute Information:

Class Name (Target variable): 3

L [balance scale tip to the left]

B [balance scale be balanced]

R [balance scale tip to the right]

Left-Weight: 5 (1, 2, 3, 4, 5)

Left-Distance: 5 (1, 2, 3, 4, 5)

Right-Weight: 5 (1, 2, 3, 4, 5)

Right-Distance: 5 (1, 2, 3, 4, 5)

**Missing Attribute Values: None**

Class Distribution:

46.08 percent are L

07.84 percent are B

46.08 percent are R

**Used Python Packages:**

**sklearn :**

In python, sklearn is a machine learning package which include a lot of ML algorithms.

Here, we are using some of its modules like train\_test\_split, DecisionTreeClassifier and accuracy\_score.

**NumPy :**

It is a numeric python module which provides fast maths functions for calculations.

It is used to read data in numpy arrays and for manipulation purpose.

**Pandas :**

Used to read and write different files.

Data manipulation can be done easily with dataframes.

**Installation of the packages :**

In Python, sklearn is the package which contains all the required packages to implement Machine learning algorithm. You can install the sklearn package by following the commands given below.

**using pip :**

pip install -U scikit-learn

**Before using the above command make sure you have scipy and numpy packages installed.**

If you don’t have pip. You can install it using

python get-pip.py

**using conda :**

conda install scikit-learn

**Assumptions we make while using Decision tree :**

At the beginning, we consider the whole training set as the root.

Attributes are assumed to be categorical for information gain and for gini index, attributes are assumed to be continuous.

On the basis of attribute values records are distributed recursively.

We use statistical methods for ordering attributes as root or internal node.

**Example for wether\_forecast**



import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

# Function importing Dataset

def importdata():

balance\_data = pd.read\_csv(

'<https://archive.ics.uci.edu/ml/machine-learning-'+>

'databases/balance-scale/balance-scale.data',

sep= ',', header = None)

# Printing the dataswet shape

print ("Dataset Length: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

# Printing the dataset obseravtions

print ("Dataset: ",balance\_data.head())

return balance\_data

# Function to split the dataset

def splitdataset(balance\_data):

# Separating the target variable

X = balance\_data.values[:, 1:5]

Y = balance\_data.values[:, 0]

# Splitting the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

# Function to perform training with giniIndex.

def train\_using\_gini(X\_train, X\_test, y\_train):

# Creating the classifier object

clf\_gini = DecisionTreeClassifier(criterion = "gini",

random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

# Performing training

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

# Function to perform training with entropy.

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeClassifier(

criterion = "entropy", random\_state = 100,

max\_depth = 3, min\_samples\_leaf = 5)

# Performing training

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

# Predicton on test with giniIndex

y\_pred = clf\_object.predict(X\_test)

print("Predicted values:")

print(y\_pred)

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

confusion\_matrix(y\_test, y\_pred))

print ("Accuracy : ",

accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ",

classification\_report(y\_test, y\_pred))

# Driver code

def main():

# Building Phase

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

# Operational Phase

print("Results Using Gini Index:")

# Prediction using gini

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

print("Results Using Entropy:")

# Prediction using entropy

y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

# Calling main function

if \_\_name\_\_=="\_\_main\_\_":

main()

**Data Information:**

Dataset Length: 625

Dataset Shape: (625, 5)

Dataset: 0 1 2 3 4

0 B 1 1 1 1

1 R 1 1 1 2

2 R 1 1 1 3

3 R 1 1 1 4

4 R 1 1 1 5

Results Using Gini Index:

**Predicted values:**

['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'

'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'

'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'

'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'

'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'

'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'

'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'

'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'

'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[ 0 6 7]

[ 0 67 18]

[ 0 19 71]]

Accuracy : 73.4042553191

Report :

precision recall f1-score support

B 0.00 0.00 0.00 13

L 0.73 0.79 0.76 85

R 0.74 0.79 0.76 90

avg/total 0.68 0.73 0.71 188

Results Using Entropy:

Predicted values:

['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'

'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'

'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'

'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'

'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'

'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'

'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'

'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[ 0 6 7]

[ 0 63 22]

[ 0 20 70]]

Accuracy : 70.7446808511

Report :

precision recall f1-score support

B 0.00 0.00 0.00 13

L 0.71 0.74 0.72 85

R 0.71 0.78 0.74 90

avg / total 0.66 0.71 0.68 188