**Experiment-5**

**Aim-** Write a program to perform decision tree classification.

**Description-**

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:**

1. **Class Name (Target variable)**: 3
   * L [balance scale tip to the left]
   * B [balance scale be balanced]
   * R [balance scale tip to the right]
2. **Left-Weight**: 5 (1, 2, 3, 4, 5)
3. **Left-Distance**: 5 (1, 2, 3, 4, 5)
4. **Right-Weight**: 5 (1, 2, 3, 4, 5)
5. **Right-Distance**: 5 (1, 2, 3, 4, 5)
6. **Missing Attribute Values**: None
7. **Class Distribution:**
   * 46.08 percent are L
   * 07.84 percent are B
   * 46.08 percent are R

**Code-**

!python get.pip.py

!pip install -U scikit-learn

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

def importdata():

balance\_data = pd.read\_csv(

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

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

sep= ',', header = None)

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

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

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

return balance\_data

def splitdataset(balance\_data):

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

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

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

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

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

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

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

return clf\_gini

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

clf\_entropy = DecisionTreeClassifier(

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

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

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

return clf\_entropy

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

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

print("Predicted values:")

print(y\_pred)

return y\_pred

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))

def main():

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)

print("Results Using Gini Index:")

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

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

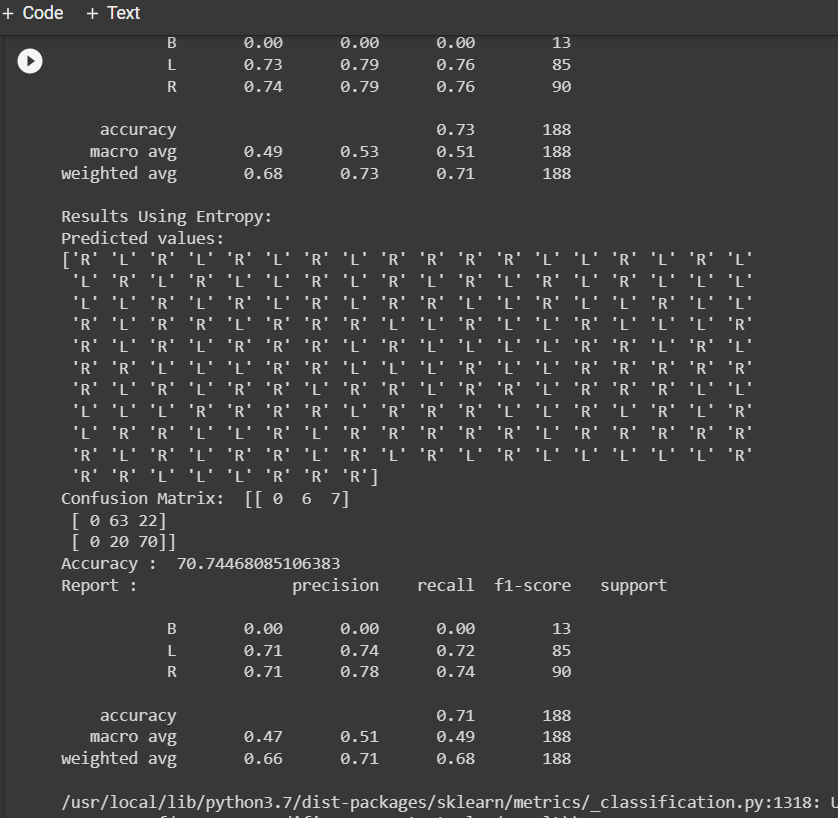
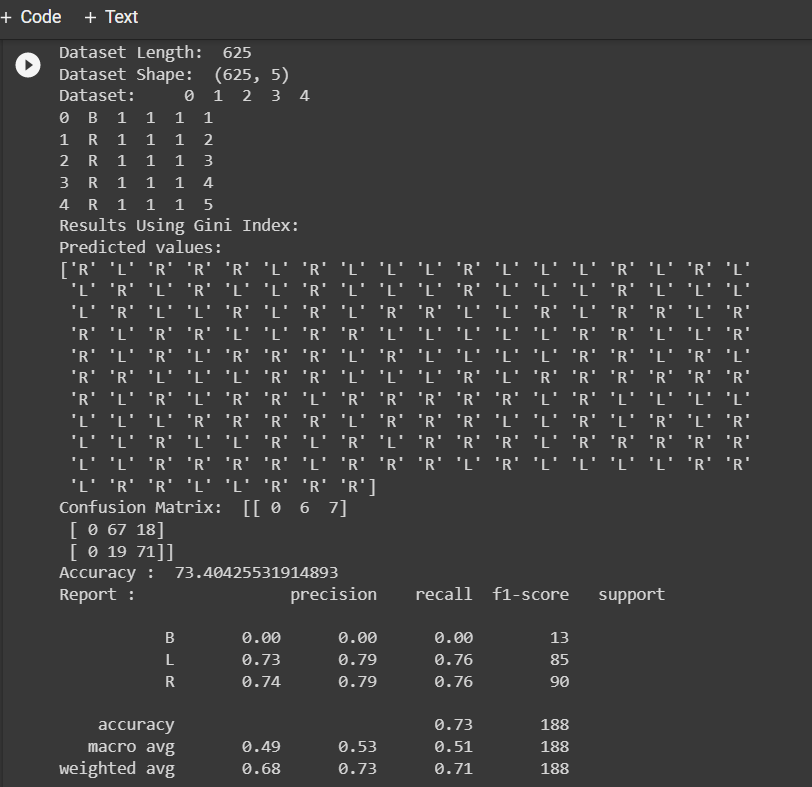
print("Results Using Entropy:")

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

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

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

main()

****