**1. Write a program to implement Decision tree using Python**

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

# Importing the dataset using pandas module

dataset = pd.read\_csv('D:\mca pdf\Decision\_tree.csv')

# splitting the dataset into input and output datasets

X = dataset.iloc[:, [0,1]].values

y = dataset.iloc[:, 2].values

# splitting the dataaset into Training and Testing Data

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

# random state is 0 and test size if 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size=0.25, random\_state=0)

# importing standard scalling method from sklearn

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

# providing the inputs for the scalling purpose

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# importing decision tree algorithm

from sklearn.tree import DecisionTreeClassifier

# entropy means information gain

classifer=DecisionTreeClassifier(criterion='entropy', random\_state=0)

# providing the training dataset

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

y\_pred= classifer.predict(X\_test)

# creating confusion matrix

from sklearn.metrics import confusion\_matrix

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

from sklearn.metrics import accuracy\_score

print("Accuracy: ", accuracy\_score(y\_test,y\_pred))

**#OUTPUT:-**

Accuracy: 0.506

**2. Write program to calculate any one of the attribute selection measures (ASM) like Information Gain, Gain Ratio, and Gini Index etc. for decision tree.**

import matplotlib.pyplot as plt

import numpy as np

def gini(p):

return (p)\*(1 - (p)) + (1 - p)\*(1 - (1-p))

def entropy(p):

return - p\*np.log2(p) - (1 - p)\*np.log2((1 - p))

def classification\_error(p):

return 1 - np.max([p, 1 - p])

x = np.arange(0.0, 1.0, 0.01)

ent = [entropy(p) if p != 0 else None for p in x]

scaled\_ent = [e\*0.5 if e else None for e in ent]

c\_err = [classification\_error(i) for i in x]

fig = plt.figure()

ax = plt.subplot(111)

for j, lab, ls, c, in zip(

[ent, scaled\_ent, gini(x), c\_err],

['Entropy', 'Entropy (scaled)', 'Gini Impurity', 'Misclassification Error'],

['-', '-', '--', '-.'],

['lightgray', 'red', 'green', 'blue']):

line = ax.plot(x, j, label=lab, linestyle=ls, lw=1, color=c)

ax.legend(loc='upper left', bbox\_to\_anchor=(0.01, 0.85),

ncol=1, fancybox=True, shadow=False)

ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')

ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')

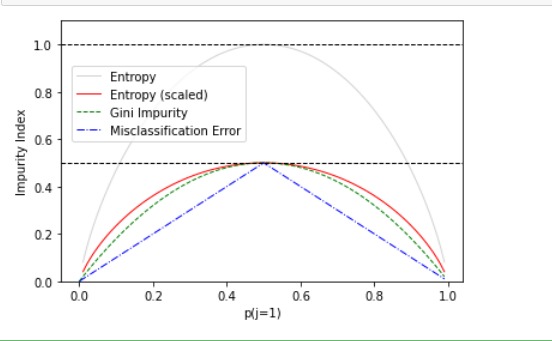
plt.ylim([0, 1.1])

plt.xlabel('p(j=1)')

plt.ylabel('Impurity Index')

plt.show()

Output :-



**3.Implement simple KNN using Euclidean distance in python.**

import math

import csv

with open(r'D:\iris.data') as csvfile:

lines = csv.reader(csvfile)

import random

def handleDataset(filename, split, trainingSet=[], testSet=[]):

with open(filename, 'r') as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

for x in range(len(dataset)-1):

for y in range(4):

dataset[x][y] = float(dataset[x][y])

if random.random() < split:

trainingSet.append(dataset[x])

else:

testSet.append(dataset[x])

trainingSet = []

testSet = []

handleDataset(r'D:\iris.data.', 0.66, trainingSet, testSet)

print('Train: ' + repr(len(trainingSet)))

print('Test: ' + repr(len(testSet)))

def euclideanDistance(instance1, instance2, length):

distance = 0

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2)

return math.sqrt(distance)

data1 = [2, 2, 2, 'a']

data2 = [4, 4, 4, 'b']

distance = euclideanDistance(data1, data2, 3)

print('Distance: ' + repr(distance))

**OUTPUT** **:-**

Train: 108

Test: 42

Distance: 3.4641016151377544

**4. Write a program to implement k-Nearest Neighbour algorithm.**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

datasets = pd.read\_csv("D:\machine Learning\Csv files\iris.data",names = names)

datasets.head()

x = datasets.iloc[:, :-1].values

y = datasets.iloc[:, 4].values

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(x\_train)

x\_train = scaler.transform(x\_train)

x\_test = scaler.transform(x\_test)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train, y\_train)

y\_pred = classifier.predict(x\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

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

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

**OUTPUT :-**

[[9 0 0]

[0 9 3]

[0 1 8]]

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 9

Iris-versicolor 0.90 0.75 0.82 12

Iris-virginica 0.73 0.89 0.80 9

accuracy 0.87 30

macro avg 0.88 0.88 0.87 30

weighted avg 0.88 0.87 0.87 30

**5. Write a program to implement the naïve Bayesian classifier for a sample training dataset.**

from sklearn.datasets import load\_iris

iris = load\_iris()

# store the feature matrix (X) and response vector (y)

X = iris.data

y = iris.target

# splitting X and y into training and testing sets

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

# training the model on training set

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

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

# making predictions on the testing set

y\_pred = gnb.predict(X\_test)

# comparing actual response values (y\_test) with predicted response values (y\_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)

**OUTPUT :-**

Gaussian Naive Bayes model accuracy(in %): 95.0

**6 Program for Conffusion Matrix and calculate training dataset.**

import matplotlib.pyplot as plt

import numpy

from sklearn import metrics

actual = numpy.random.binomial(1,.9,size = 1000)

predicted = numpy.random.binomial(1,.9,size = 1000)

confusion\_matrix = metrics.confusion\_matrix(actual, predicted)

cm\_display = metrics.ConfusionMatrixDisplay(confusion\_matrix = confusion\_matrix, display\_labels = [False, True])

cm\_display.plot()

plt.show()

#precision recall and f1 measure

Precision = metrics.precision\_score(actual, predicted)

print(Precision)

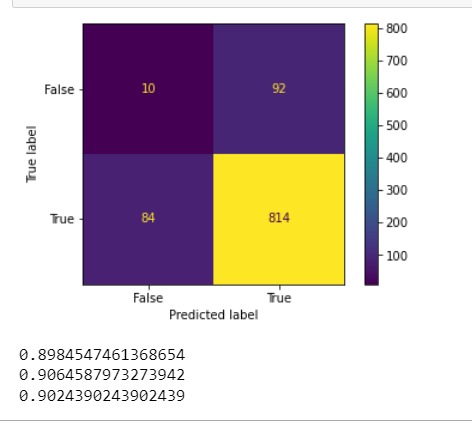
Sensitivity\_recall = metrics.recall\_score(actual, predicted)

print(Sensitivity\_recall)

F1\_measure = metrics.f1\_score(actual, predicted)

print(F1\_measure)

**OUTPUT** :-



**7. Write program for linear regression and find parameters like Sum of Squared Errors (SSE)**

import matplotlib

from matplotlib import style

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

#%matplotlib inline

dataset =pd.read\_csv("D:\iris.cvs")

dataset.shape

dataset.head()

dataset.describe()

x= dataset.iloc[:,:-1].values

y=dataset.iloc[:,1].values

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

X\_train, X\_test, y\_train, y\_test =train\_test\_split(x,y,test\_size=0.2, random\_state=0)

from sklearn.linear\_model import LinearRegression

regressor=LinearRegression()

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

print (regressor.intercept\_)

print(regressor.coef\_)

y\_pred=regressor.predict(X\_test)

df=pd.DataFrame({'Actual':y\_test, 'Predicted':y\_pred})

df

from sklearn import metrics

print('Mean Absolute Error:',metrics.mean\_absolute\_error(y\_test,y\_pred))

print('Mean Squared Error:',metrics.mean\_squared\_error(y\_test,y\_pred))

print('Root Mean Squared Error:',np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)))

**OUTPUT** **:-**

-4.440892098500626e-16

[-5.64989286e-17 1.00000000e+00 3.33066907e-16 -5.55111512e-17]

Mean Absolute Error: 9.917992353318065e-16

Mean Squared Error: 1.3607850615062454e-30

Root Mean Squared Error: 1.1665269227524264e-15

**8. Implement Agglomerative Clustering in python.**

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from sklearn.cluster import AgglomerativeClustering

import scipy.cluster.hierarchy as sch

dataset = pd.read\_csv('./data.csv')

X = dataset.iloc[:, [3, 4]].values

dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))

model = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

model.fit(X)

labels = model.labels\_

plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o', color='red')

plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o', color='blue')

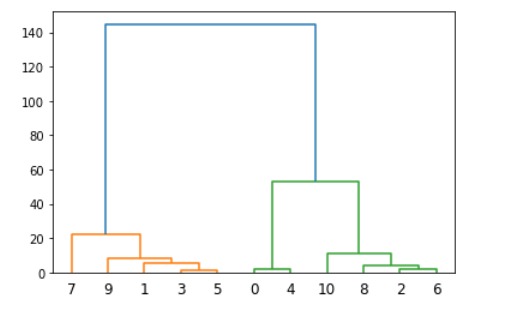
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o', color='green')

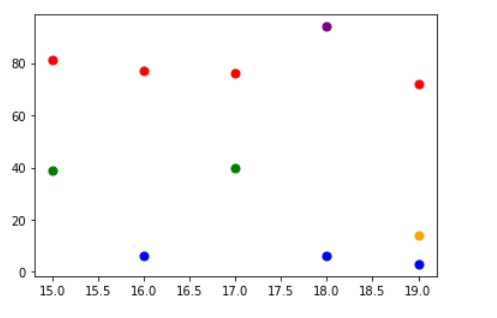
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o', color='purple')

plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o', color='orange')

plt.show()

**OUTPUT :-**

****

****

PROGRAM 8 OUTPUT

**9. Write a Program to implement SVM.**

import pandas as pd

import numpy as np

dataset =pd.read\_csv("D:\iris.data")

x= dataset.iloc[:,[2,3]].values

y=dataset.iloc[:,4].values

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

X\_train, X\_test, y\_train, y\_test =train\_test\_split(x,y,test\_size=0.20, random\_state=0)

from sklearn.preprocessing import StandardScaler

sc\_x= StandardScaler()

X\_train = sc\_x.fit\_transform(X\_train)

X\_test = sc\_x.transform(X\_test)

from sklearn.svm import SVC

classifier = SVC(kernel="linear",random\_state=0)

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

y\_pred = classifier.predict(X\_test)

y\_pred

from sklearn.metrics import confusion\_matrix

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

from sklearn.metrics import accuracy\_score

print("Accuracy: ",accuracy\_score(y\_test,y\_pred))

**OUTPUT :-**

Accuracy: 0.9

**10. Implement Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

import matplotlib.pyplot as plt

from scipy import stats

x = [1,2,5,6,8,11,14,15,16,17]

y = [11000,12000,15000,19000,22000,32000,38000,40000,50000,52000]

slope, intercept, r, p, std\_err = stats.linregress(x, y)

def myfunc(x):

return slope \* x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)

plt.plot(x, mymodel)

plt.show()

#future prediction at 20 and 26 experience

def myfunc(x):

return slope \* x + intercept

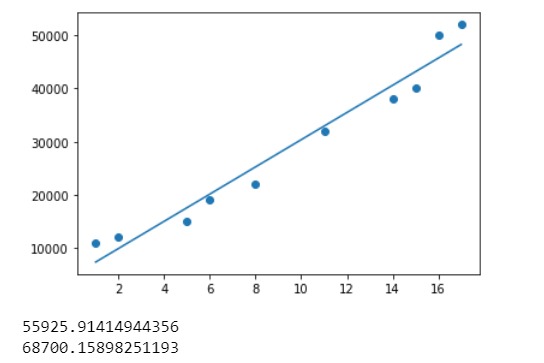
speed = myfunc(20)

print(speed)

speed = myfunc(25)

print(speed)

**OUTPUT** :-



PROGRAM NO 1O OUTPUT

**12. Implement K-means Clustering in python.**

import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]

y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)

plt.show()

from sklearn.cluster import KMeans

data = list(zip(x, y))

inertias = []

for i in range(1,11):

kmeans = KMeans(n\_clusters=i)

kmeans.fit(data)

inertias.append(kmeans.inertia\_)

plt.plot(range(1,11), inertias, marker='o')

plt.title('Elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

PROGRAM 12 OUT PUT

**OUTPUT :-**

