

**Indira Gandhi National**

**Open university**

# LABORATORY RECORD

Month &Year : …………….………………………………………………….

Name : …………….………………………………………………….

Study Center : 1402, SH College, Thevara, Kochi-13

Course : …………….………………………………………………….

Course Title : ………………………………………………………..…….…

Course Code : ………………………………………………………..…….…

Enrollment No. : ………………………………………………………..…….…

External Examiner Staff In-Charge

Experiment No: 01

Naive Baye’s Classification

Aim : Write a program to perform Naive Baye’s Classification.

Program :

# load the iris 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)

Result : Program is executed successfully and output verified.

Output :

Gaussian Naive Bayes model accuracy (in %) : 95.0

Experiment No: 02

Support Vector Machine

Aim : Write a program to perform support vector machine.

Program :

import pandas as pd

import numpy as np

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

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

from sklearn.metrics import accuracy\_score

from sklearn import datasets

# IRIS Data Set

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Creating training and test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1, stratify = y)

# Feature Scaling

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

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

# Training a SVM classifier using SVC class

svm = SVC(kernel= 'linear', random\_state=1, C=0.1)

svm.fit(X\_train\_std, y\_train)

# Mode performance

y\_pred = svm.predict(X\_test\_std)

print('Accuracy: %.3f' % accuracy\_score(y\_test, y\_pred))

Result : Program is executed successfully and output verified.

Output :

Accuracy : 0.978

Experiment No: 03

Polynomial Regression

Aim : Write a program to perform polynomial regression.

Program :

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

datas = pd.read\_csv('data.csv')

X = datas.iloc[:, 1:2].values

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

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 4)

X\_poly = poly.fit\_transform(X)

poly.fit(X\_poly, y)

# Fitting Linear Regression to the dataset

from sklearn.linear\_model import LinearRegression

lin2 = LinearRegression()

lin2.fit(X\_poly, y)

# Visualising the Polynomial Regression results

plt.scatter(X, y, color = 'blue')

plt.plot(X, lin2.predict(poly.fit\_transform(X)), color = 'red')

plt.title('Polynomial Regression')

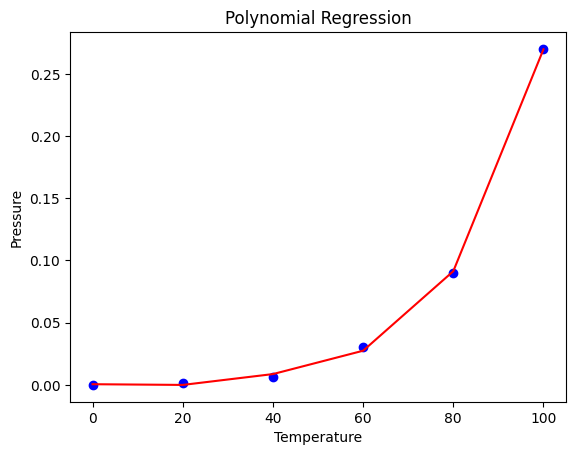
plt.xlabel('Temperature')

plt.ylabel('Pressure')

plt.show()

Result : Program is executed successfully and output verified.

Output :



Experiment No: 04

Linear Regression

Aim : Write a program to perform linear regression.

Program :

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

datas = pd.read\_csv('data.csv')

X = datas.iloc[:, 1:2].values

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

# Fitting Linear Regression to the dataset

from sklearn.linear\_model import LinearRegression

lin = LinearRegression()

lin.fit(X, y)

# Visualising the Linear Regression results

plt.scatter(X, y, color = 'blue')

plt.plot(X, lin.predict(X), color = 'red')

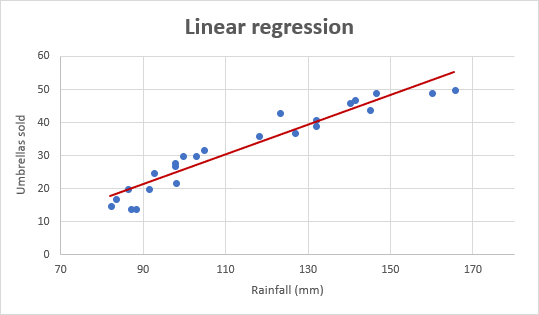
plt.title('Linear Regression')

plt.xlabel('Temperature')

plt.ylabel('Pressure')

plt.show()

Result : Program is executed successfully and output verified.

Output :   
  


Experiment No: 05

Logistic Regression

Aim : Write a program to perform logistic regression.

Program :

import numpy

from sklearn import linear\_model

import matplotlib.pyplot as plt

#Reshaped for Logistic function.

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear\_model.LogisticRegression()

logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm:

predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))

print(predicted)

# Visualising the Logistic Regression results

plt.scatter(X, y, color = 'blue')

plt.plot(X, logr.predict(X), color = 'red')

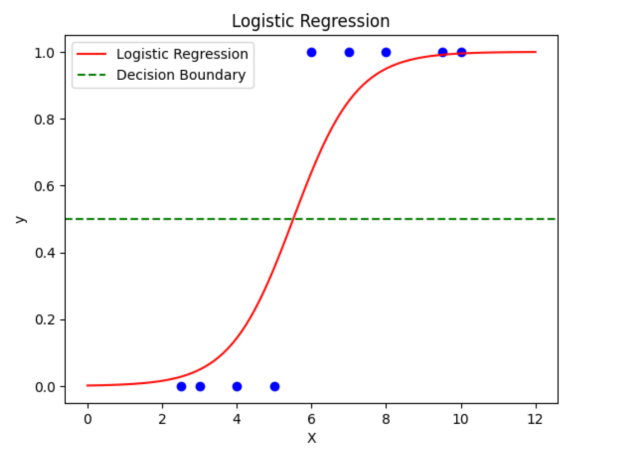
plt.title('Logistic Regression')

plt.xlabel('X')

plt.ylabel('Y')

plt.show()

Result : Program is executed successfully and output verified.

Output :   


Experiment No: 06

Depth First Search

Aim : Write a program to perform Depth First Search.

Program :

# Using a Python dictionary to act as an adjacency list

graph = {

  '5' : ['3','7'],

  '3' : ['2', '4'],

  '7' : ['8'],

  '2' : [],

  '4' : ['8'],

  '8' : []

}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs

    if node not in visited:

        print (node)

        visited.add(node)

        for neighbour in graph[node]:

            dfs(visited, graph, neighbour)

# Driver Code

print("Following is the Depth-First Search")

dfs(visited, graph, '5')

Result : Program is executed successfully and output verified.

Output :

Following is the Depth-First Search

5

3

2

4

8

7

Experiment No: 07

Breadth First Search

Aim : Write a program to perform Breadth First Search.

Program :

graph = {

  '5' : ['3','7'],

  '3' : ['2', '4'],

  '7' : ['8'],

  '2' : [],

  '4' : ['8'],

  '8' : []

}

visited = [] # List for visited nodes.

queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS

  visited.append(node)

  queue.append(node)

  while queue:          # Creating loop to visit each node

    m = queue.pop(0)

    print (m, end = " ")

    for neighbour in graph[m]:

      if neighbour not in visited:

        visited.append(neighbour)

        queue.append(neighbour)

# Driver Code

print("Following is the Breadth-First Search")

bfs(visited, graph, '5')    # function calling

Result : Program is executed successfully and output verified.

Output :

Following is the Breadth-First search

5 3 7 2 4 8

Experiment No: 08

1. Means Algorithm

Aim : Write a program to perform K-Means Algorithm

Program :

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

X, y = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)

plt.scatter(X[:,0], X[:,1])

kmeans = KMeans(n\_clusters=4, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

pred\_y = kmeans.fit\_predict(X)

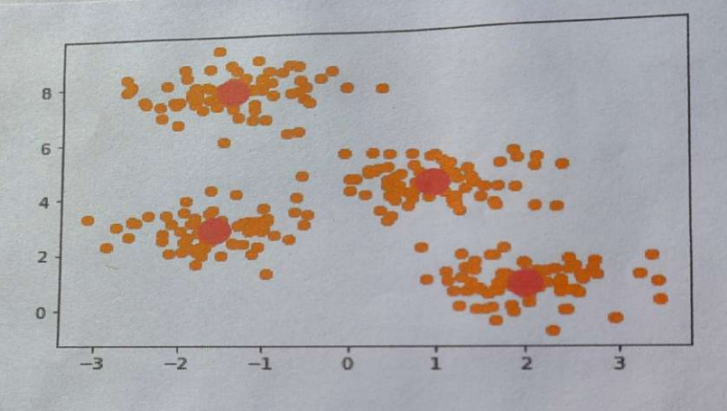
plt.scatter(X[:,0], X[:,1])

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red')

plt.show()

Result : Program is executed successfully and output verified.

Output :



Experiment No: 09

Decision Tree Algorithm

Aim : Write a program to perform Decision Tree Algorithm .

Program :

#%%

from sklearn.datasets import make\_classification

from sklearn import tree

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

X, t = make\_classification(100, 5, n\_classes = 2, shuffle = True, random\_state= 10)

X\_train, X\_test, t\_train, t\_test = train\_test\_split(X, t, test\_size=0.3, shuffle = True, random\_state=1)

#%%

model = tree.DecisionTreeClassifier()

model = model.fit(X\_train, t\_train)

#%%

predicted\_value = model.predict(X\_test)

print(predicted\_value)

#%%

tree.plot\_tree(model)

#%%

zeroes = 0

ones = 0

for i in range(0,len(t\_train)):

    if t\_train[i] == 0:

        zeroes +=1

    else:

        ones +=1

#%%

print(zeroes)

print(ones)

#%%

val = 1 - ((zeroes/70)\*2 + (ones/70)\*2)

print("Gini :",val)

match = 0

UnMatch = 0

for i in range(30):

    if predicted\_value[i] == t\_test[i]:

        match += 1

    else:

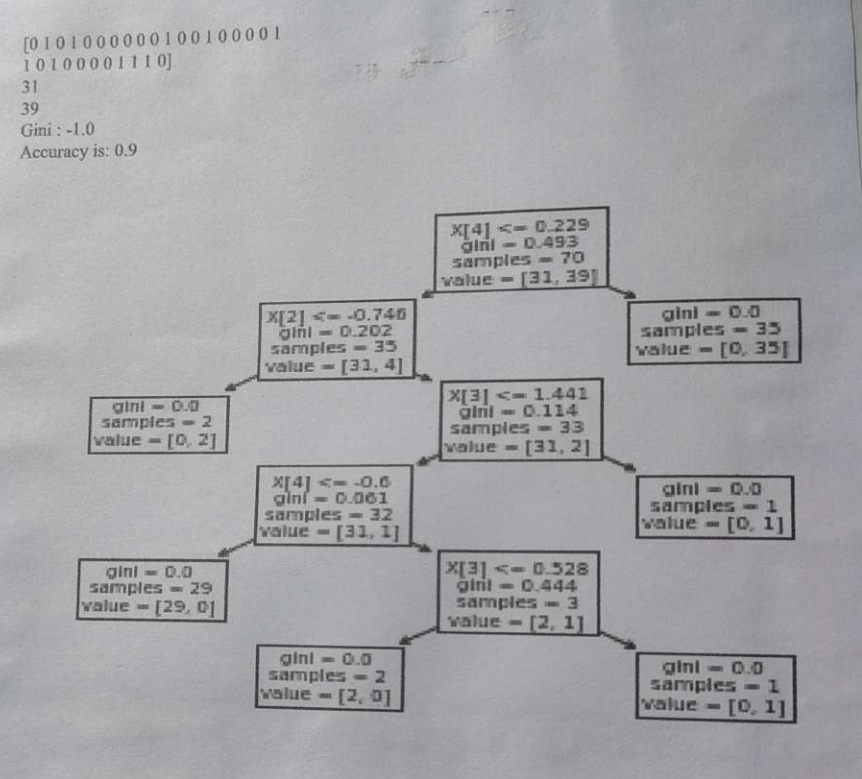
        UnMatch += 1

accuracy = match/30

print("Accuracy is: ",accuracy)

Result : Program is executed successfully and output verified.

Output :



Experiment No: 10

KNN Classifier

Aim : Write a program to perform KNN Classifier.

Program :

# Import necessary modules

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

# Loading data

irisData = load\_iris()

# Create feature and target arrays

X = irisData.data

y = irisData.target

# Split into training and test set

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

            X, y, test\_size = 0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=7)

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

# Predict on dataset which model has not seen before

print(knn.predict(X\_test))

Result : Program is executed successfully and output verified.

Output :

[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0]