Lab 6

1. Write a program to implement k-means clustering (Example of unsupervised algorithm).
2. Write program for implementing Neural Networks for realization of AND, OR gates.
3. Write program for implementing Back propagation Learning.
4. Write program for implementing Naive Bayes with any dataset from KAGGLE.

Theory:

K-means:

K-mean is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

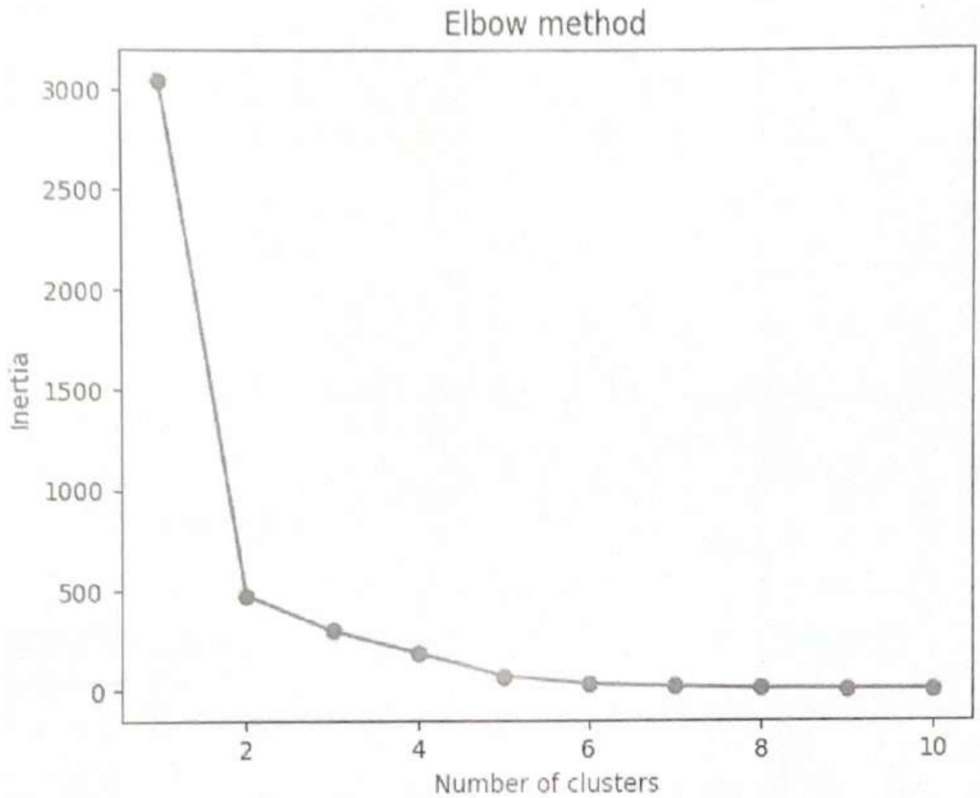
Working mechanism:

First, each data point a randomly assigned to one of the K clusters. Then, we compute the centroid (functionally the center) of each cluster and reassign each data point to the cluster with the closest centriod. We repeat this process until the cluster assignments for each data point are no longer ~~o~~ changing.

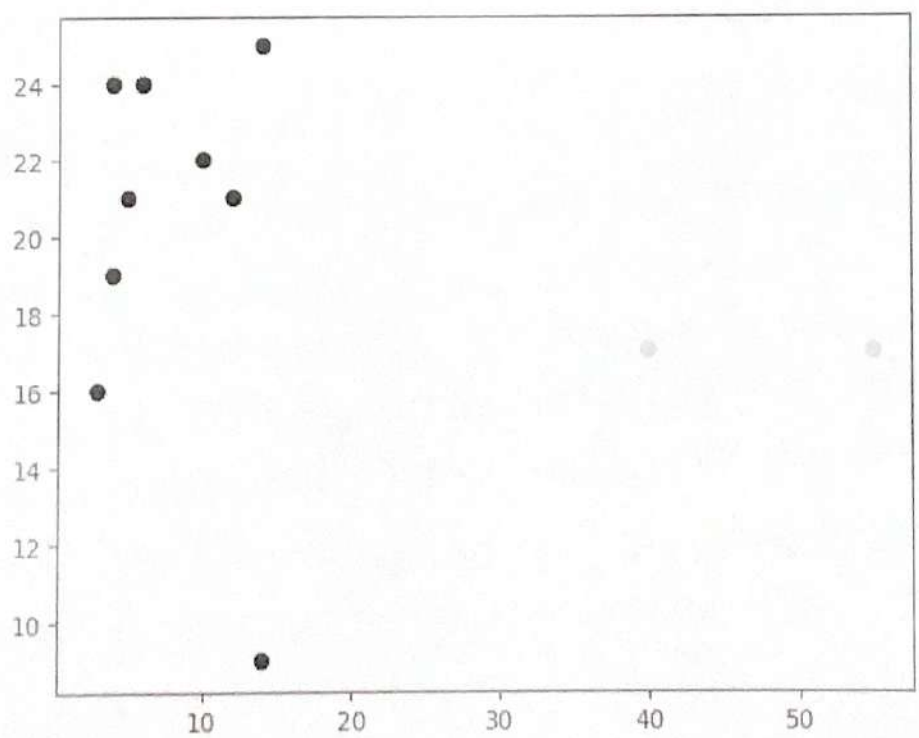
Program to implement K-means clustering

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x=[5,4,4,40,3,14,6,10,12,14,55]
y=[21,19,24,17,16,25,24,22,21,9,17]
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()
Kmeans= KMean(n_clusters=2)
Kmeans.fit(data)
plt.scatter(x,y,c=kmeans.labels_)
plt.show()
```

OUTPUT



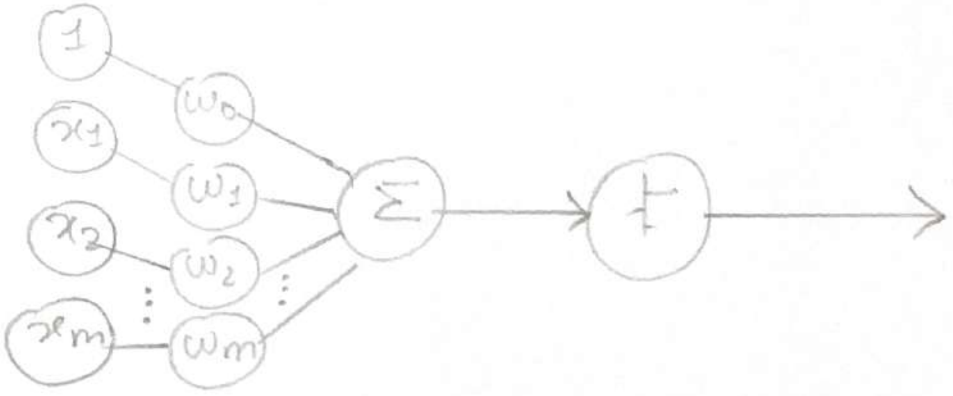
Elbow method
3000
2500
2000
1500
1000
500
0
Inertia
2
4
6
8
10
Number of clusters



24
22
20
18
16
14
12
10
10
20
30
40
50

Theory:

Neural networks using preceptron:



Input layer:

This is the primary component of perceptron which accepts the initial data into the system for further processing.

Weight and Bias:

Weight parameter represents the strength of the connection between units. This is another most important parameter of perceptron components. weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation

Activation function:

This component determines whether the neuron will fire or not.

Program for implementing neural network for realization of basic gates.

```
import numpy as np
def activation_function (v):
  if v <=0
    return 0
  else
    return 1

def perceptron (x, w, b):
  v= np.dot (w, x) +b
  y= activation_function (v)
  return y

def logicFunction (logic, x):
  if logic == "OR":
    b=-0.5
    w= np.array ([1,1])
  elif logic == "AND":
    b= -1.5
    w= np.array ([1,1])
  elif logic == "NOT"
    b=1
    w= np.array ([-1)]
return perceptron (x, w, b)
```

```
test1=np.array([1,1])
test2= np.array ([1,0])
test3 = np.array([0,1])
tes4 = np.array([0,0])
test5= np.array ([0])
test6. np.array ([1])

print ("FOR OR logic")
print ("OR ({},{})= {}".format(1,1, logicFunction
                                  ("OR", test1)))
  :
  :

print ("\n For AND logic")
print("AND({},{})= {}".format (1,1,
                               logicfunction
                               ("AND", test1)))
  :
  :

print£ ("\n For Not logic")
print ("NOT({})={}".format (0, logic Function
                                ("NOT", test 5)))
```

OUTPUTS

```
For OR logic
OR(1, 1) = 1
OR(1, 0) = 1
OR(0, 1) = 1
OR(0, 0) = 0

For AND logic
AND(1, 1) = 1
AND(1, 0) = 0
AND(0, 1) = 0
AND(0, 0) = 0

For NOT logic
NOT(0) = 1
NOT(1) = 0
```

Theory:

Back propagation:

It is an algorithm that is designed to test for errors working back from output node to input node.

Program for implementing Back propagation Learning.

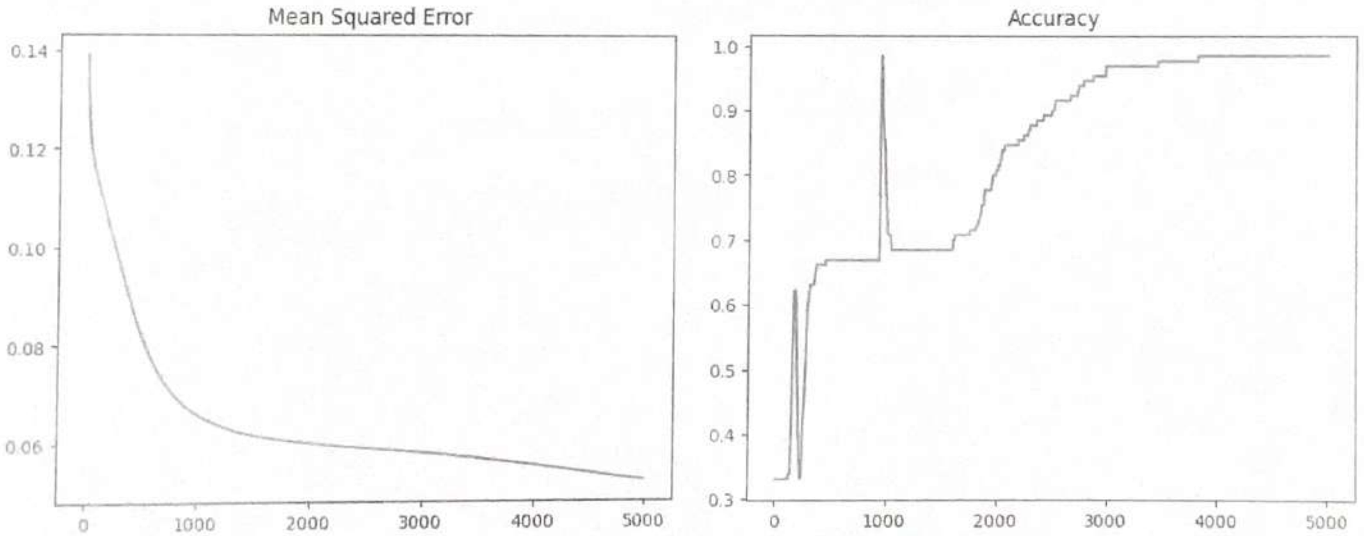
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import *

data= load_iris()
x= data.data
y=data.target

x_train, x_test, y_train, y_test = train_test_split
                                   (x, y, test_size=20,
                                    random_state=4)

learning rate = 0.1
iterations = 5000
N= y_train.size
input_size =4
hidden_size =2
output_size = 3
```

```
np.random.seed(10)
w1= np.randoom.normal(scale=0.5, size=(input.size,
                                   hidden size))
w2= np.random.normal(scale=0.5, size=(hidden-size,
                                   output size))

def sigmoid(x):
  return 1/(1+np.exp(-x))

def mean-squared_error(y-pred,y-true):
  y-true_one_hot= np.eye(output-size)[y-true]
  y-true_reshaped = y-true_one-hot.reshape(y-pred.
                                           shape)
  error= ((y-pred - y-true_reshape)**2).sum()
                        /(2*y-pred.size)
  return error

def accuracy (y-pred,y-true):
  acc = y-pred.argmax(axis=1) == y-true.argmax
                                        (axis=1)
  return acc.mean()

results = pd.DataFrame(columns=["mse","accuracy"])
```

```
for itr in range(iterations):
    Z1 = np.dot (x_train, W1)
    A1 = sigmoid (Z1)
    Z2 = np.dot (A1, W2)
    A2 = sigmoid (Z2)

    mse = mean_square_error (A2, y_train)
    acc = accuracy (np.eye (output_size) [y_train], A2)
    new_row = pd.DataFrame ({"mse": [mse],
                             {"accuracy": [acc]})

    results = pd.concat ([results, new_row],
                          ignore_index = True)

    E1 = A2 - np.eye (output_size) [y_train]
    dW1 = E1 * A2 * (1 - A2)
    E2 = np.dot (dW1, W2.T)
    dW2 = Ed * A1 * (1 - A1)

    W2_update = np.dot (A1.T, dW1) / N
    W1_update = np.dot (x_train, T, dW2) / N
    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update
results.mse.plot (title = "mean squared error")
plt.show ()
results.accuracy.plot (title = "Accuracy")
plot.show ()
```

```
Z1 = np.dot(x_test, W1)
A1 = sigmoid(z1)
Z2 = np.dot(A1.W2)
A2 = sigmoid(z2)
test_acc = accuracy(np.eye(output_size)
                    [y_test], A2)
print("Test accuracy: {}".format(test_acc))
```

OUTPUT



Test accuracy: 0.95

Theory:

The Naïve Bayes classifier is a supervised ~~mec~~ machine learing algorithm, which is used for classification tasks, like text classification.

It works on the principle of conditional probabability

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

where,

$P(A|B)$ = conditional probability of A given B

$P(B|A)$ = conditional probability of A given B

$P(A)$ = probability of event A

$P(B)$ = probability of event B

Program to implement Naive Bayes algorithm

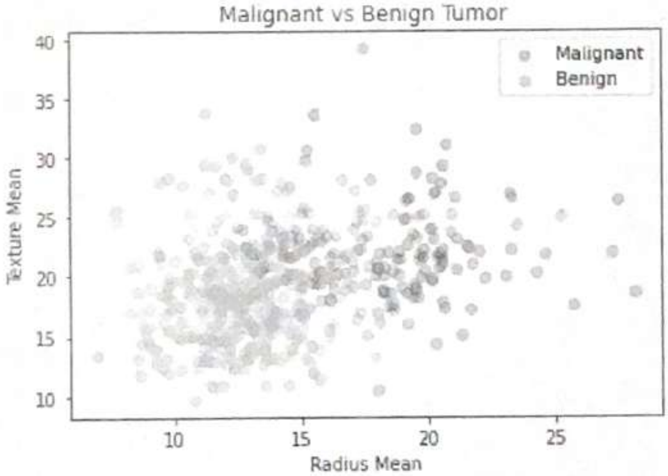
The dataset used for the program was downloaded from kaglle. "cancer.csv"

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

dataset= pd.read_csv("lab6/cancer.csv")
dataset.info()
dataset= dataset.drop(["id"], axis=1)
dataset= dataset.drop(["unnamed:32"]}, axis=1)
M= dataset[dataset.diagnosis=="M"]
B = dataset[dataset.diagnosis=="B"]

plt.title("Malignant vs Benign Tumor")
plt.xlabel("Radius Mean")
plt.ylabel("Texture mean")
plt.scatter(M.radius_mean, M.texture_mean,
            color="red", label="malignant")

plt.scatter(B.radius_mean, B.texture_mean,
            color="lime", label="Benign")
```

```
plt.legend()
plt.show()

dataset.diagnosis = [1 if i=="M" else 0 for i in
                        dataset.diagnosis]

x= dataset.drop(["diagnosis"], axis=1)
y= dataset.diagnosis.value

x= (x-np.min(x))/(np.max(x)-np.min(x))
x_train, x_test, y_train, y_test = train_test_split
            (x,y, test_size =0.3, random_state =42)

nb= GaussianNB()
nb.fit(x_train, y train)
print ("Naive Bayes score :", nb.score (x_train,
                        y_test))
```

OUTPUT




```
Naive Bayes score:  0.935672514619883
```