**ML-LAB**

**10. Q LEARNING**

import numpy as np

# R matrix

R = np.matrix([ [-1,-1,-1,-1,0,-1],

[-1,-1,-1,0,-1,100],

[-1,-1,-1,0,-1,-1],

[-1,0,0,-1,0,-1],

[-1,0,0,-1,-1,100],

[-1,0,-1,-1,0,100] ])

# Q matrix

Q = np.matrix(np.zeros([6,6]))

gamma = 0.8

initial\_state = 1

def available\_actions(state):

 current\_state\_row = R[state,]

 av\_act = np.where(current\_state\_row >= 0)[1]

 return av\_act

available\_act = available\_actions(initial\_state)

def sample\_next\_action(available\_actions\_range):

 next\_action = int(np.random.choice(available\_act,1))

 return next\_action

action = sample\_next\_action(available\_act)

def update(current\_state, action, gamma):

 max\_index = np.where(Q[action,] == np.max(Q[action,]))[1]

 if max\_index.shape[0] > 1:

  max\_index = int(np.random.choice(max\_index, size = 1))

 else:

  max\_index = int(max\_index)

 max\_value = Q[action, max\_index]

 # Q learning formula

 Q[current\_state, action] = R[current\_state, action] + gamma \* max\_value

update(initial\_state,action,gamma)

for i in range(10000):

 current\_state = np.random.randint(0, int(Q.shape[0]))

 available\_act = available\_actions(current\_state)

 action = sample\_next\_action(available\_act)

 update(current\_state,action,gamma)

# Normalize the "trained" Q matrix

print("Trained Q matrix:")

print(Q/np.max(Q)\*100)

#-------------------------------------------------------------------------------

# Testing

# Goal state = 5

# Best sequence path starting from 2 -> 2, 3, 1, 5

current\_state = 2

steps = [current\_state]

while current\_state != 5:

 next\_step\_index = np.where(Q[current\_state,] == np.max(Q[current\_state,]))[1]

 if next\_step\_index.shape[0] > 1:

  next\_step\_index = int(np.random.choice(next\_step\_index, size = 1))

 else:

  next\_step\_index = int(next\_step\_index)

 steps.append(next\_step\_index)

 current\_state = next\_step\_index

# Print selected sequence of steps

print("Selected path:")

print(steps)

**OUTPUT:**

Trained Q matrix:

[[ 0. 0. 0. 0. 80. 0. ]

[ 0. 0. 0. 64. 0. 100. ]

[ 0. 0. 0. 64. 0. 0. ]

[ 0. 80. 51.2 0. 80. 0. ]

[ 0. 80. 51.2 0. 0. 100. ]

[ 0. 80. 0. 0. 80. 100. ]]

Selected path:

[2, 3, 4, 5]

**9. GENETIC OPERATORS**

**#single-point-crossover.py**

import random

def crossover(l, q):

    l = list(l)

    q = list(q)

    k = random.randint(0, 15)

    print("Crossover point :", k)

    for j in range(k, len(s)):

        l[j], q[j] = q[j], l[j]

    l = ''.join(l)

    q = ''.join(q)

    print(l)

    print(q, "\n\n")

    return l, q

s = '1100110110110011'

p = '1000110011011111'

print("Parents")

print("P1 :", s)

print("P2 :", p, "\n")

for i in range(3):

    print("Generation ", i + 1, "Childrens :")

    s, p = crossover(s, p)

**OUTPUT:**



Parents

P1 : 1100110110110011

P2 : 1000110011011111

Generation 1 Childrens :

Crossover point : 2

1100110011011111

1000110110110011

Generation 2 Childrens :

Crossover point : 3

1100110110110011

1000110011011111

Generation 3 Childrens :

Crossover point : 7

1100110011011111

1000110110110011

**#double-point-crossover.py**

import numpy as np

import random

p1 = np.array([1,1,0,1,1,0,0,1,0,0,1,1,0,1,1,0])

p2 = np.array([1,0,1,0,1,1,1,0,0,0,0,1,1,1,1,0])

print(f"Parent 1 - {p1}")

print(f"Parent 2 - {p2}")

n = len(p1)

c1 = np.zeros(n)

c2 = np.zeros(n)

co1 = random.randint(0, 4)

co2 = random.randint(6, 9)

print(f"Crossover Point 1 - {co1}")

print(f"Crossover Point 2 - {co2}")

for i in range(n):

    if co1 <= i <= co2:

        c1[i], c2[i] = p2[i], p1[i]

    else:

        c1[i], c2[i] = p1[i], p2[i]

print(f"Child-1 : {c1}")

print(f"Child-2 : {c2}")

**OUTPUT:**

Parent 1 - [1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0]

Parent 2 - [1 0 1 0 1 1 1 0 0 0 0 1 1 1 1 0]

Crossover Point 1 - 4

Crossover Point 2 - 8

Child-1 : [1. 1. 0. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 0.]

Child-2 : [1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 0.]

**#uniform-crossover.py**

import numpy as np

p1 = np.array([1,1,0,1,1,0,0,1,0,0,1,1,0,1,1,0])

p2 = np.array([1,0,1,0,1,1,1,0,0,0,0,1,1,1,1,0])

m = np.array([1,0,1,1,0,0,1,0,1,0,1,1,0,0,1,0])

print(f"Parent 1 - {p1}")

print(f"Parent 2 - {p2}")

print(f"M value  - {m}")

n = len(p1)

c1 = np.zeros(n)

c2 = np.zeros(n)

for i in range(n):

    if m[i] == 1:

        c1[i], c2[i] = p2[i], p1[i]

    else:

        c1[i], c2[i] = p1[i], p2[i]

print(f"Child-1 : {c1}")

print(f"Child-2 : {c2}")

**OUTPUT:**

Parent 1 - [1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0]

Parent 2 - [1 0 1 0 1 1 1 0 0 0 0 1 1 1 1 0]

M value - [1 0 1 1 0 0 1 0 1 0 1 1 0 0 1 0]

Child-1 : [1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0.]

Child-2 : [1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0.]

**#mutation-crossover.py**

import numpy as np

import random

p = np.array([1,1,0,1,1,0,0,1,0,0,1,1,0,1,1,0])

b = random.randint(0, 15)

print(f"Parent - {p}")

print(f"Crossover Point - {b}")

p[b] = not p[b]

print(f"Child : {p}")

**OUTPUT:**

Parent - [1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0]

Crossover Point - 6

Child : [1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0]

**8. K-MEANS**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

df = pd.read\_csv('iriskmean.csv')

print(df)

x = df.iloc[:, [0,1]].values

print(x)

kmeans2 = KMeans(n\_clusters=2)

y\_kmeans2 = kmeans2.fit\_predict(x)

print(y\_kmeans2)

print("Cluster centers are:")

print(kmeans2.cluster\_centers\_)

plt.scatter(x[:,0],x[:,1],c=y\_kmeans2,cmap='viridis')

plt.show()

**OUTPUT:**

sepal.length sepal.width variety

0 5.1 3.5 Iris setosa

1 4.9 3.0 Iris setosa

2 4.7 3.2 Iris setosa

3 7.0 3.2 Iris versicolor

4 6.4 3.2 Iris versicolor

5 6.9 3.2 Iris versicolor

[[5.1 3.5]

[4.9 3. ]

[4.7 3.2]

[7. 3.2]

[6.4 3.2]

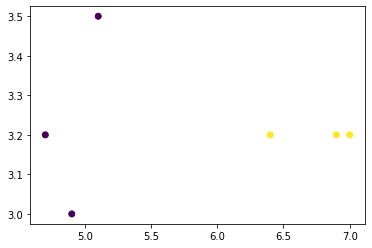
[6.9 3.2]]

[0 0 0 1 1 1]

Cluster centers are:

[[4.9 3.23333333]

[6.76666667 3.2 ]]

****

**7.** **KNN**

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

df = pd.read\_csv("acid.csv")

x = df.drop('Classification',axis=1).values

y = df['Classification'].values

print("Entire dataset is\n",df)

print("\n X array is\n",x)

print("\n Y array is\n",y)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(x, y)

z=knn.predict([[3,7]])

print("The predicted Output for (3,7) is",z)

**OUTPUT:**

Entire dataset is

ACID PRABABILITY(X1) STRENGTH(X2) Classification

0 7 7 BAD

1 7 4 BAD

2 3 4 GOOD

3 1 4 GOOD

4 5 6 GOOD

5 8 2 GOOD

6 9 1 GOOD

7 12 5 GOOD

8 45 10 GOOD

X array is

[[ 7 7]

[ 7 4]

[ 3 4]

[ 1 4]

[ 5 6]

[ 8 2]

[ 9 1]

[12 5]

[45 10]]

Y array is

['BAD' 'BAD' 'GOOD' 'GOOD' 'GOOD' 'GOOD' ' GOOD' 'GOOD' 'GOOD']

The predicted Output for (3,7) is ['GOOD']

**6.** **NAVIE BAYES**

weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',

'Rainy','Sunny','Overcast','Overcast','Rainy']

temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

weather\_encoded=le.fit\_transform(weather)

print ("Weather:",weather\_encoded)

temp\_encoded=le.fit\_transform(temp)

label=le.fit\_transform(play)

print ("Temp:",temp\_encoded)

print ("Play:",label)

features=zip(weather\_encoded,temp\_encoded)

features=list(features)

print (features)

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(features,label)

predicted= model.predict([[0,2]])

x=print ("Predicted Value:", predicted)

**OUTPUT:**

Weather: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]

Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]

Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]

[(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]

Predicted Value: [1]

**5.** **PERCEPTRON**

import numpy as np

#np.random.seed(0)

def sigmoid (x):

  return 1/(1 + np.exp(-x))

def sigmoid\_derivative(x):

  return x \* (1 - x)

#Input datasets

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

expected\_output = np.array([[0],[1],[1],[0]])

epochs = 10000

lr = 1

inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1

#Random weights and bias initialization

hidden\_weights =np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))

hidden\_bias =np.random.uniform(size=(1,hiddenLayerNeurons))

output\_weights =np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))

output\_bias = np.random.uniform(size=(1,outputLayerNeurons))

print("Initial hidden weights: ",end='')

print(\*hidden\_weights)

print("Initial hidden biases: ",end='')

print(\*hidden\_bias)

print("Initial output weights: ",end='')

print(\*output\_weights)

print("Initial output biases: ",end='')

print(\*output\_bias)

#Training algorithm

for \_ in range(epochs):

  #Forward Propagation

  hidden\_layer\_activation = np.dot(inputs,hidden\_weights)

  hidden\_layer\_activation += hidden\_bias

  hidden\_layer\_output = sigmoid(hidden\_layer\_activation)

  output\_layer\_activation = np.dot(hidden\_layer\_output,output\_weights)

  output\_layer\_activation += output\_bias

  predicted\_output = sigmoid(output\_layer\_activation)

  #Backpropagation

  error = expected\_output - predicted\_output

  d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

  error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)

  d\_hidden\_layer = error\_hidden\_layer \*sigmoid\_derivative(hidden\_layer\_output)

#Updating Weights and Biases

  output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* lr

  output\_bias += np.sum(d\_predicted\_output,axis=0,keepdims=True) \* lr

  hidden\_weights += inputs.T.dot(d\_hidden\_layer) \* lr

  hidden\_bias += np.sum(d\_hidden\_layer,axis=0,keepdims=True) \* lr

print("Final hidden weights: ",end='')

print(\*hidden\_weights)

print("Final hidden bias: ",end='')

print(\*hidden\_bias)

print("Final output weights: ",end='')

print(\*output\_weights)

print("Final output bias: ",end='')

print(\*output\_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')

print(\*predicted\_output)

**OUTPUT:**

Initial hidden weights: [0.90420777 0.56996783] [0.1116005 0.04363249]

Initial hidden biases: [0.92395537 0.67630182]

Initial output weights: [0.56911765] [0.42171878]

Initial output biases: [0.74523818]

Final hidden weights: [9.87799206 9.66351285] [ 4.94486545 -3.20787633]

Final hidden bias: [-0.98081738 2.19632066]

Final output weights: [7.14943903] [-6.22135978]

Final output bias: [-0.92791372]

Output from neural network after 10,000 epochs: [0.01018392] [0.98821614] [0.49980793] [0.50031148]

**4. DECISION TREE**

import pandas as pd

from pandas import DataFrame

df\_tennis = pd.read\_csv('tennis-ise.csv')

print( df\_tennis)

def entropy(probs):

    import math

    return sum([-prob\*math.log(prob, 2) for prob in probs])

def entropy\_of\_list(a\_list):

  from collections import Counter

  cnt = Counter(x for x in a\_list) #Count the positive and negative ex

  num\_instances = len(a\_list)

  probs = [x / num\_instances for x in cnt.values()]

  return entropy(probs)

def information\_gain(df, split\_attribute\_name, target\_attribute\_name, trace=0):

  print("Information Gain Calculation of ",split\_attribute\_name)

  print("target\_attribute\_name",target\_attribute\_name)

  df\_split = df.groupby(split\_attribute\_name)

  for name,group in df\_split:

      print("Name: ",name)

      print("Group: ",group)

  nobs = len(df.index) \* 1.0

  print("NOBS",nobs)

  df\_agg\_ent = df\_split.agg({target\_attribute\_name : [entropy\_of\_list, lambda x: len(x)/nobs] })[target\_attribute\_name]

# Calculate Information Gain

  df\_agg\_ent.columns=['Entropy','Prob1']

  avg\_info = sum( df\_agg\_ent['Entropy'] \* df\_agg\_ent['Prob1'] )

  old\_entropy = entropy\_of\_list(df[target\_attribute\_name])

  return old\_entropy - avg\_info

print('Info-Gain for Outlook is :'+str(information\_gain(df\_tennis, 'Outlook','PlayTennis')),"\n")

**OUTPUT:**

Unnamed: 0 PlayTennis Outlook Temperature Humidity Wind

0 0 No Sunny Hot High Weak

1 1 No Sunny Hot High Strong

2 2 Yes Overcast Hot High Weak

3 3 Yes Rain Mild High Weak

4 4 Yes Rain Cool Normal Weak

5 5 No Rain Cool Normal Strong

6 6 Yes Overcast Cool Normal Strong

7 7 No Sunny Mild High Weak

8 8 Yes Sunny Cool Normal Weak

9 9 Yes Rain Mild Normal Weak

10 10 Yes Sunny Mild Normal Strong

11 11 Yes Overcast Mild High Strong

12 12 Yes Overcast Hot Normal Weak

13 13 No Rain Mild High Strong

Information Gain Calculation of Outlook

target\_attribute\_name PlayTennis

Name: Overcast

Group: Unnamed: 0 PlayTennis Outlook Temperature Humidity Wind

2 2 Yes Overcast Hot High Weak

6 6 Yes Overcast Cool Normal Strong

11 11 Yes Overcast Mild High Strong

12 12 Yes Overcast Hot Normal Weak

Name: Rain

Group: Unnamed: 0 PlayTennis Outlook Temperature Humidity Wind

3 3 Yes Rain Mild High Weak

4 4 Yes Rain Cool Normal Weak

5 5 No Rain Cool Normal Strong

9 9 Yes Rain Mild Normal Weak

13 13 No Rain Mild High Strong

Name: Sunny

Group: Unnamed: 0 PlayTennis Outlook Temperature Humidity Wind

0 0 No Sunny Hot High Weak

1 1 No Sunny Hot High Strong

7 7 No Sunny Mild High Weak

8 8 Yes Sunny Cool Normal Weak

10 10 Yes Sunny Mild Normal Strong

NOBS 14.0

Info-gain for Outlook is :0.2467498197744391

**3. LINEAR REGRESSION**

**a)Using user defined methods**

import numpy as np

import matplotlib.pyplot as plt

#X=[3.5,3,3.2,3.1,3.6,3.9,3.4,3.5,2.9]

#Y=[5.1,4.9,4.7,4.6,5,5.4,4.6,5,4.4]

X=[95,85,80,70,60]

Y=[85,95,70,65,70]

x\_p=np.array([95,85,80,70,60])

y\_p=np.array([85,95,70,65,70])

x\_mean=np.mean(X)

y\_mean=np.mean(Y)

print("Mean of X ",x\_mean)

print("Mean of Y ",y\_mean)

n=len(Y)

num=den=0

for i in range(n):

num+=(X[i]-x\_mean)\*(Y[i]-y\_mean)

den+=(X[i]-x\_mean)\*\*2

m=num/den

intercept=y\_mean-(m\*x\_mean)

print("Slope = ",m)

print("Intercept (c) = ",intercept)

print("The equation is y = ",round(m,2),"x + ",round(intercept))

print("Enter the x value.. ",end='')

x1=float(input())

y1=m\*x1+intercept

print("For the given X1 = ",x1," the Y1 = ",round(y1))

plt.scatter(x\_p, y\_p)

plt.plot(x\_p, y\_p)

plt.show()

**OUTPUT:**

Mean of X 78.0

Mean of Y 77.0

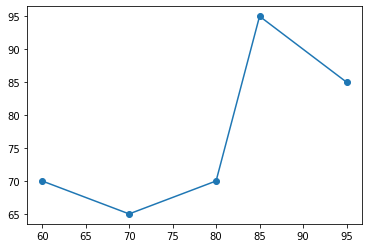
Slope = 0.6438356164383562

Intercept (c) = 26.78082191780822

The equation is y = 0.64 x + 27

Enter the x value.. 80

For the given X1 = 80.0 the Y1 = 78

****

**b) Using scipy statistics built-in methods**

import numpy as np

import matplotlib.pyplot as plt

#plt.rcParams['figure.figsize'] = (20.0, 10.0)

from scipy import stats

x = [3.5, 3, 3.2, 3.1, 3.6, 3.9, 3.4, 3.5, 2.9]

y = [5.1, 4.9, 4.7, 4.6, 5, 5.4, 4.6, 5, 4.4]

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

def myfunc(x):

return slope \* x + intercept

mymodel = list(map(myfunc, x))

print("Y=",mymodel)

print("Slope: ", slope)

print("Intercept", intercept)

print("Enter the x value for which y is to be found…")

x1=float(input())

y1=slope \* x1 + intercept

print("y value for the given x is ", y1)

plt.scatter(x, y)

plt.scatter(x, mymodel)

plt.plot(x, mymodel)

plt.show()

**OUTPUT:**

Y= [4.9799999999999995, 4.58, 4.74, 4.66, 5.06, 5.299999999999999, 4.8999999999999995, 4.9799999999999995, 4.5]

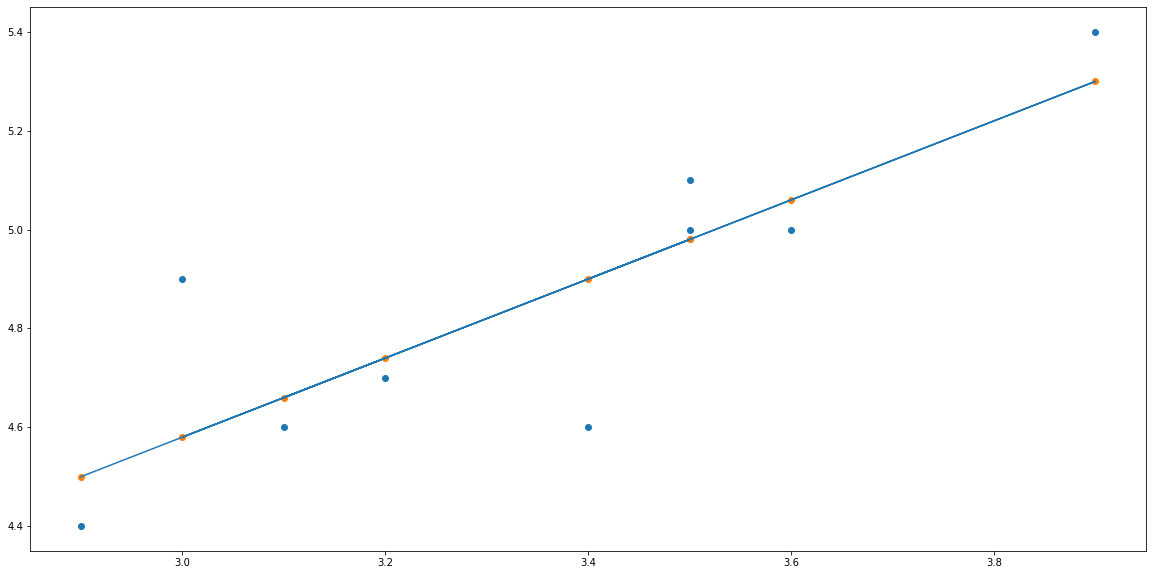
Slope: 0.7999999999999998

Intercept 2.18

Enter the x value for which y is to be found…

3.6

y value for the given x is 5.06

****

**c) Using Training and Test data split up**

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

data\_set= pd.read\_csv('E:\Mukilan Assignments\Infotainment\ML\LAB\simple-Linear-Regression-master\Salary\_Data.csv')

x= data\_set.iloc[:, :-1].values

y= data\_set.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= 1/3, random\_state=0)

x\_train,x\_test,y\_train,y\_test

from sklearn.linear\_model import LinearRegression

regressor= LinearRegression()

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

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

x\_pred= regressor.predict(x\_train)

mtp.scatter(x\_train, y\_train, color="green")

mtp.plot(x\_train, x\_pred, color="red")

mtp.title("Salary vs Experience (Training Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()

mtp.scatter(x\_test, y\_test, color="blue")

mtp.plot(x\_train, x\_pred, color="red")

mtp.title("Salary vs Experience (Test Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()

**OUTPUT:**

[[ 1.1]

[ 1.3]

[ 1.5]

[ 2. ]

[ 2.2]

[ 2.9]

[ 3. ]

[ 3.2]

[ 3.2]

[ 3.7]

[ 3.9]

[ 4. ]

[ 4. ]

[ 4.1]

[ 4.5]

[ 4.9]

[ 5.1]

[ 5.3]

[ 5.9]

[ 6. ]

[ 6.8]

[ 7.1]

[ 7.9]

[ 8.2]

[ 8.7]

[ 9. ]

[ 9.5]

[ 9.6]

[10.3]

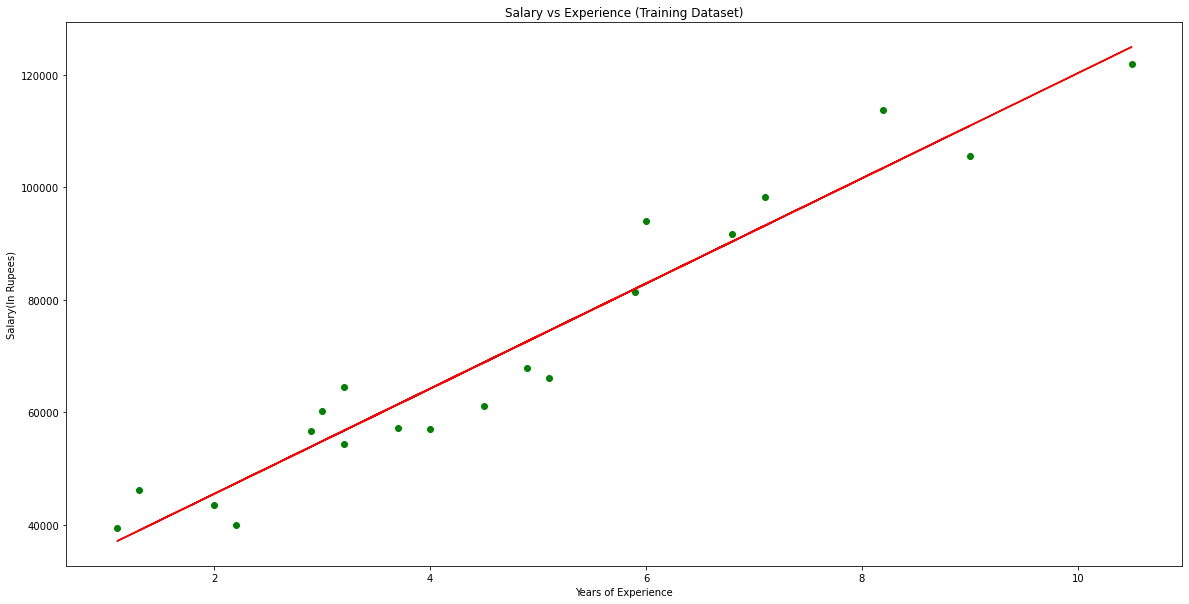
[10.5]]

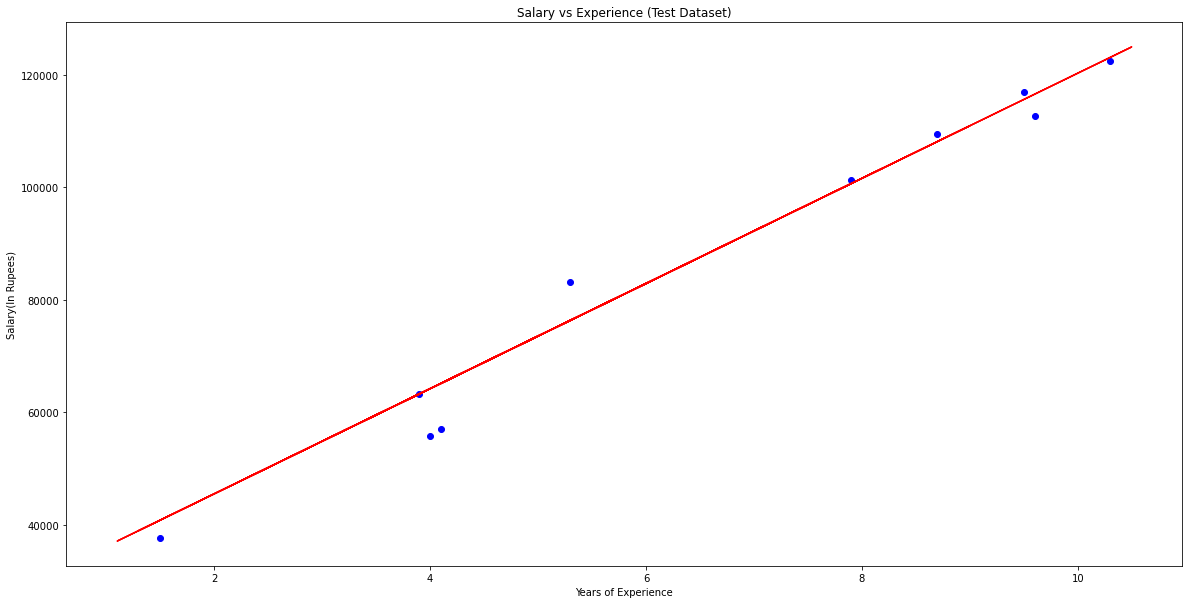
[ 39343. 46205. 37731. 43525. 39891. 56642. 60150. 54445. 64445.

57189. 63218. 55794. 56957. 57081. 61111. 67938. 66029. 83088.

81363. 93940. 91738. 98273. 101302. 113812. 109431. 105582. 116969.

112635. 122391. 121872.]

****

****

**EXPECTED QUESTIONS**

**Linear regression**

**Decision tree**

**Perceptron**

**KNN**

**Kmeans**

**Q learning**

**Navie Bayes**

**Genetic algorithm**