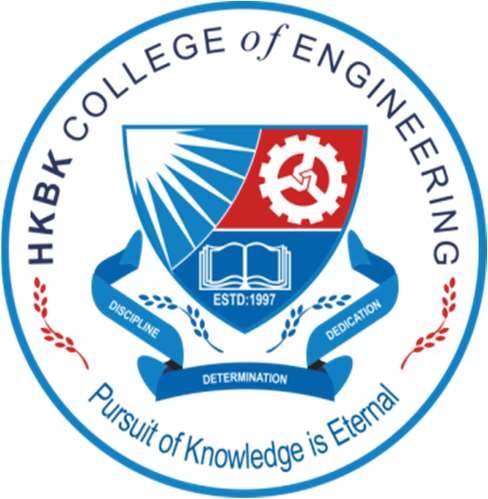
**HKBK COLLEGE OF ENGINEERING**

(Affiliated to VTU, Belgaum and Approved by AICTE)

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**NBA Accredited Programme**



**LABORATORY MANUAL**

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY**

18CSL76

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2018)



**PREPARED BY**

**1HK20CS409 Shubhanshu Kumar**

**HKBK COLLEGE OF ENGINEERING**

**Nagawara, Bangaluru -560045**

**www.hkbk.edu.in**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY**  (Effective from the academic year 2018 -2019)  SEMESTER VII | | | | |
| **Course Code** | | 18CSL76 | **CIE Marks** | 40 |
| **Number of Contact Hours/Week** | | 0:0:2 | **SEE Marks** | 60 |
| **Total Number of Lab Contact Hours** | | 36 | **Exam Hours** | 03 |
| Credits 2 | | | | |
| Course Learning Objectives: This course (18CSL76) will enable students to: | | | | |
| Implement and evaluate AI and ML algorithms in and Python programming language. | | | | |
| Descriptions (if any): | | | | |
| Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal. | | | | |
| Programs List: | | | | |
| 1. | Implement A\* Search algorithm. | | | |
| 2. | Implement AO\* Search algorithm. | | | |
| 3. | For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples. | | | |
| 4. | Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. | | | |
| 5. | Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. | | | |
| 6. | Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. | | | |
| 7. | Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program. | | | |
| 8. | Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem. | | | |
| 9. | Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points.  Select appropriate data set for your experiment and draw graphs | | | |
| Laboratory Outcomes: The student should be able to: | | | | |
| Implement and demonstrate AI and ML algorithms. Evaluate different algorithms. | | | | |
| Conduct of Practical Examination: | | | | |
| Experiment distribution o For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.  o For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.  Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.  Marks Distribution (Courseed to change in accoradance with university regulations)   1. For laboratories having only one part Procedure + Execution + Viva-Voce: 15+70+15 =   100 Marks   1. For laboratories having PART A and PART B   i. Part A Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks ii. Part B Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks | | | | |

1. **Implement A\* Search algorithm.**

class Graph:

def \_\_init\_\_(self,adjac\_lis):

self.adjac\_lis = adjac\_lis

def get\_neighbours(self,v):

return self.adjac\_lis[v]

def h(self,n):

H={'A':1,'B':1, 'C':1,'D':1}

return H[n]

def a\_star\_algorithm(self,start,stop):

open\_lst = set([start])

closed\_lst = set([])

dist ={}

dist[start] = 0

prenode ={}

prenode[start] =start

while len(open\_lst)>0:

n = None

for v in open\_lst:

if n==None or dist[v]+self.h(v)<dist[n]+self.h(n):

n=v;

if n==None:

print("path doesnot exist")

return None

if n==stop:

reconst\_path=[]

while prenode[n]!=n:

reconst\_path.append(n)

n = prenode[n]

reconst\_path.append(start)

reconst\_path.reverse()

print("path found:{}".format(reconst\_path))

return reconst\_path

for (m,weight) in self.get\_neighbours(n):

if m not in open\_lst and m not in closed\_lst:

open\_lst.add(m)

prenode[m] = n

dist[m] = dist[n]+weight

else:

if dist[m]>dist[n]+weight:

dist[m] = dist[n]+weight

prenode[m]=n

if m in closed\_lst:

closed\_lst.remove(m)

open\_lst.add(m)

open\_lst.remove(n)

closed\_lst.add(n)

print("Path doesnot exist")

return None

adjac\_lis ={'A':[('B',1),('C',3),('D',7)],'B':[('D',5)],'C':[('D',12)]}

graph1=Graph(adjac\_lis)

graph1.a\_star\_algorithm('A', 'D')

**OUTPUT:**

Path found: ['A', 'F', 'G', 'I', 'J']

1. **Implement AO\* Search algorithm.**

class Graph:

def \_\_init\_\_(self, graph, heuristicNodeList, startNode): #instantiate graph object with graph topology, heuristic values, start node

self.graph = graph

self.H=heuristicNodeList

self.start=startNode

self.parent={}

self.status={}

self.solutionGraph={}

def applyAOStar(self): # starts a recursive AO\* algorithm

self.aoStar(self.start, False)

def getNeighbors(self, v): # gets the Neighbors of a given node

return self.graph.get(v,'')

def getStatus(self,v): # return the status of a given node

return self.status.get(v,0)

def setStatus(self,v, val): # set the status of a given node

self.status[v]=val

def getHeuristicNodeValue(self, n):

return self.H.get(n,0) # always return the heuristic value of a given node

def setHeuristicNodeValue(self, n, value):

self.H[n]=value # set the revised heuristic value of a given node

def printSolution(self):

print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE:",self.start)

print("------------------------------------------------------------")

print(self.solutionGraph)

print("------------------------------------------------------------")

def computeMinimumCostChildNodes(self, v): # Computes the Minimum Cost of child nodes of a given node v

minimumCost=0

costToChildNodeListDict={}

costToChildNodeListDict[minimumCost]=[]

flag=True

for nodeInfoTupleList in self.getNeighbors(v): # iterate over all the set of child node/s

cost=0

nodeList=[]

for c, weight in nodeInfoTupleList:

cost=cost+self.getHeuristicNodeValue(c)+weight

nodeList.append(c)

if flag==True: # initialize Minimum Cost with the cost of first set of child node/s

minimumCost=cost

costToChildNodeListDict[minimumCost]=nodeList # set the Minimum Cost child node/s

flag=False

else: # checking the Minimum Cost nodes with the current Minimum Cost

if minimumCost>cost:

minimumCost=cost

costToChildNodeListDict[minimumCost]=nodeList # set the Minimum Cost child node/s

return minimumCost, costToChildNodeListDict[minimumCost] # return Minimum Cost and Minimum Cost child node/s

def aoStar(self, v, backTracking): # AO\* algorithm for a start node and backTracking status flag

print("HEURISTIC VALUES :", self.H)

print("SOLUTION GRAPH :", self.solutionGraph)

print("PROCESSING NODE :", v)

print("-----------------------------------------------------------------------------------------")

if self.getStatus(v) >= 0: # if status node v >= 0, compute Minimum Cost nodes of v

minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)

self.setHeuristicNodeValue(v, minimumCost)

self.setStatus(v,len(childNodeList))

solved=True # check the Minimum Cost nodes of v are solved

for childNode in childNodeList:

self.parent[childNode]=v

if self.getStatus(childNode)!=-1:

solved=solved & False

if solved==True: # if the Minimum Cost nodes of v are solved, set the current node status as solved(-1)

self.setStatus(v,-1)

self.solutionGraph[v]=childNodeList # update the solution graph with the solved nodes which may be a part of solution

if v!=self.start: # check the current node is the start node for backtracking the current node value

self.aoStar(self.parent[v], True) # backtracking the current node value with backtracking status set to true

if backTracking==False: # check the current call is not for backtracking

for childNode in childNodeList: # for each Minimum Cost child node

self.setStatus(childNode,0) # set the status of child node to 0(needs exploration)

self.aoStar(childNode, False) # Minimum Cost child node is further explored with backtracking status as false

h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':1, 'T': 3}

graph1 = {

'A': [[('B', 1), ('C', 1)], [('D', 1)]],

'B': [[('G', 1)], [('H', 1)]],

'C': [[('J', 1)]],

'D': [[('E', 1), ('F', 1)]],

'G': [[('I', 1)]]}

G1= Graph(graph1, h1, 'A')

G1.applyAOStar()

G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7} # Heuristic values of Nodes

graph2 = { # Graph of Nodes and Edges

'A': [[('B', 1), ('C', 1)], [('D', 1)]], # Neighbors of Node 'A', B, C & D with repective weights

'B': [[('G', 1)], [('H', 1)]], # Neighbors are included in a list of lists

'D': [[('E', 1), ('F', 1)]] # Each sublist indicate a "OR" node or "AND" nodes}

G2 = Graph(graph2, h2, 'A') # Instantiate Graph object with graph, heuristic values and start Node

G2.applyAOStar() # Run the AO\* algorithm

G2.printSolution() # print the solution graph as AO\* Algorithm search

**OUTPUT:-**

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}

SOLUTION GRAPH : {}

PROCESSING NODE : I

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A

------------------------------------------------------------

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

------------------------------------------------------------

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : D

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : E

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': []}

PROCESSING NODE : D

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': []}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': []}

PROCESSING NODE : F

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': [], 'F': []}

PROCESSING NODE : D

-----------------------------------------------------------------------------------------

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A

------------------------------------------------------------

{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}

------------------------------------------------------------

1. **For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

import csv

with open("trainingexamples.csv") as f:

csv\_file = csv.reader(f)

data = list(csv\_file)

specific = data[0][:-1]

general = [['?' for i in range(len(specific))] for j in range(len(specific))]

for i in data:

if i[-1] == "Yes":

for j in range(len(specific)):

if i[j] != specific[j]:

specific[j] = "?"

general[j][j] = "?"

elif i[-1] == "No":

for j in range(len(specific)):

if i[j] != specific[j]:

general[j][j] = specific[j]

else:

general[j][j] = "?"

print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")

print(specific)

print(general)

gh = [] # gh = general Hypothesis

for i in general:

for j in i:

if j != '?':

gh.append(i)

break

print("\nFinal Specific hypothesis:\n", specific)

print("\nFinal General hypothesis:\n", gh)

**OUTPUT:-**

Step 1 of Candidate Elimination Algorithm

['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm

['?', '?', '?', '?', '?', '?']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm

['?', '?', '?', '?', '?', '?']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm

['?', '?', '?', '?', '?', '?']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 5 of Candidate Elimination Algorithm

['?', '?', '?', '?', '?', '?']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:

['?', '?', '?', '?', '?', '?']

Final General hypothesis:

[]

1. **Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.**

import pandas as pd

from pprint import pprint

from sklearn.feature\_selection import mutual\_info\_classif

from collections import Counter

def id3(df, target\_attribute, attribute\_names, default\_class=None):

cnt=Counter(x for x in df[target\_attribute])

if len(cnt)==1:

return next(iter(cnt))

elif df.empty or (not attribute\_names):

return default\_class

else:

gainz = mutual\_info\_classif(df[attribute\_names],df[target\_attribute],discrete\_features=True)

index\_of\_max=gainz.tolist().index(max(gainz))

best\_attr=attribute\_names[index\_of\_max]

tree={best\_attr:{}}

remaining\_attribute\_names=[i for i in attribute\_names if i!=best\_attr]

for attr\_val, data\_subset in df.groupby(best\_attr):

subtree=id3(data\_subset, target\_attribute, remaining\_attribute\_names,default\_class)

tree[best\_attr][attr\_val]=subtree

return tree

df=pd.read\_csv("ptennis.csv")

attribute\_names=df.columns.tolist()

print("List of attribut name")

attribute\_names.remove("PlayTennis")

for colname in df.select\_dtypes("object"):

df[colname], \_ = df[colname].factorize()

print(df)

tree= id3(df,"PlayTennis", attribute\_names)

print("The tree structure")

pprint(tree)

**OUTPUT:-**

List of attribut name

Outlook Temperature Humidity Windy PlayTennis

0 0 0 0 False 0

1 0 0 0 True 0

2 1 0 0 False 1

3 2 1 0 False 1

4 2 2 1 False 1

5 2 2 1 True 0

6 1 2 1 True 1

7 0 1 0 False 0

8 0 2 1 False 1

9 2 1 1 False 1

10 0 1 1 True 1

11 1 1 0 True 1

12 1 0 1 False 1

13 2 1 0 True 0

The tree structure

{'Outlook': {0: {'Humidity': {0: 0, 1: 1}},

1: 1,

2: {'Windy': {False: 1, True: 0}}}}

1. **Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5000 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+ bout

output = sigmoid(outinp)

#Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

#how much hidden layer wts contributed to error

hiddengrad = derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop

wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**OUTPUT:-**

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89613915]

[0.878037 ]

[0.89523334]]

1. **Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

# import necessary libraries

import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

# Load Data from CSV

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

print("The first 5 Values of data is :\n", data.head())

# obtain train data and train output

X = data.iloc[:, :-1]

print("\nThe First 5 values of the train data is\n", X.head())

y = data.iloc[:, -1]

print("\nThe First 5 values of train output is\n", y.head())

# convert them in numbers

le\_outlook = LabelEncoder()

X.Outlook = le\_outlook.fit\_transform(X.Outlook)

le\_Temperature = LabelEncoder()

X.Temperature = le\_Temperature.fit\_transform(X.Temperature)

le\_Humidity = LabelEncoder()

X.Humidity = le\_Humidity.fit\_transform(X.Humidity)

le\_Windy = LabelEncoder()

X.Windy = le\_Windy.fit\_transform(X.Windy)

print("\nNow the Train output is\n", X.head())

le\_PlayTennis = LabelEncoder()

y = le\_PlayTennis.fit\_transform(y)

print("\nNow the Train output is\n",y)

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)

classifier = GaussianNB()

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

from sklearn.metrics import accuracy\_score

print("Accuracy is:", accuracy\_score(classifier.predict(X\_test), y\_test))

**OUTPUT:-**

The first 5 Values of data is :

Outlook Temperature Humidity Windy PlayTennis

0 Sunny Hot High False No

1 Sunny Hot High True No

2 Overcast Hot High False Yes

3 Rainy Mild High False Yes

4 Rainy Cool Normal False Yes

The First 5 values of the train data is

Outlook Temperature Humidity Windy

0 Sunny Hot High False

1 Sunny Hot High True

2 Overcast Hot High False

3 Rainy Mild High False

4 Rainy Cool Normal False

The First 5 values of train output is

0 No

1 No

2 Yes

3 Yes

4 Yes

Name: PlayTennis, dtype: object

Now the Train output is

Outlook Temperature Humidity Windy

0 2 1 0 0

1 2 1 0 1

2 0 1 0 0

3 1 2 0 0

4 1 0 1 0

Now the Train output is

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

Accuracy is: 1.0

1. **Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

from sklearn import datasets

from sklearn import metrics

from sklearn.cluster import KMeans

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

iris = datasets.load\_iris()

print(iris)

X\_train,X\_test,y\_train,y\_test = train\_test\_split(iris.data,iris.target)

model =KMeans(n\_clusters=3)

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

model.score

print('K-Mean: ',metrics.accuracy\_score(y\_test,model.predict(X\_test)))

#-------Expectation and Maximization----------

from sklearn.mixture import GaussianMixture

model2 = GaussianMixture(n\_components=3)

model2.fit(X\_train,y\_train)

model2.score

print('EM Algorithm:',metrics.accuracy\_score(y\_test,model2.predict(X\_test)))

**OUTPUT:-**

{'data': array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

[5.4, 3.9, 1.7, 0.4],

[4.6, 3.4, 1.4, 0.3],

[5. , 3.4, 1.5, 0.2],

[4.4, 2.9, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.1],

[5.4, 3.7, 1.5, 0.2],

[4.8, 3.4, 1.6, 0.2],

[4.8, 3. , 1.4, 0.1],

[4.3, 3. , 1.1, 0.1],

[5.8, 4. , 1.2, 0.2],

[5.7, 4.4, 1.5, 0.4],

[5.4, 3.9, 1.3, 0.4],

[5.1, 3.5, 1.4, 0.3],

[5.7, 3.8, 1.7, 0.3],

[5.1, 3.8, 1.5, 0.3],

[5.4, 3.4, 1.7, 0.2],

[5.1, 3.7, 1.5, 0.4],

[4.6, 3.6, 1. , 0.2],

[5.1, 3.3, 1.7, 0.5],

[4.8, 3.4, 1.9, 0.2],

[5. , 3. , 1.6, 0.2],

[5. , 3.4, 1.6, 0.4],

[5.2, 3.5, 1.5, 0.2],

[5.2, 3.4, 1.4, 0.2],

[4.7, 3.2, 1.6, 0.2],

[4.8, 3.1, 1.6, 0.2],

[5.4, 3.4, 1.5, 0.4],

[5.2, 4.1, 1.5, 0.1],

[5.5, 4.2, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.2],

[5. , 3.2, 1.2, 0.2],

[5.5, 3.5, 1.3, 0.2],

[4.9, 3.6, 1.4, 0.1],

[4.4, 3. , 1.3, 0.2],

[5.1, 3.4, 1.5, 0.2],

[5. , 3.5, 1.3, 0.3],

[4.5, 2.3, 1.3, 0.3],

[4.4, 3.2, 1.3, 0.2],

[5. , 3.5, 1.6, 0.6],

[5.1, 3.8, 1.9, 0.4],

[4.8, 3. , 1.4, 0.3],

[5.1, 3.8, 1.6, 0.2],

[4.6, 3.2, 1.4, 0.2],

[5.3, 3.7, 1.5, 0.2],

[5. , 3.3, 1.4, 0.2],

[7. , 3.2, 4.7, 1.4],

[6.4, 3.2, 4.5, 1.5],

[6.9, 3.1, 4.9, 1.5],

[5.5, 2.3, 4. , 1.3],

[6.5, 2.8, 4.6, 1.5],

[5.7, 2.8, 4.5, 1.3],

[6.3, 3.3, 4.7, 1.6],

[4.9, 2.4, 3.3, 1. ],

[6.6, 2.9, 4.6, 1.3],

[5.2, 2.7, 3.9, 1.4],

[5. , 2. , 3.5, 1. ],

[5.9, 3. , 4.2, 1.5],

[6. , 2.2, 4. , 1. ],

[6.1, 2.9, 4.7, 1.4],

[5.6, 2.9, 3.6, 1.3],

[6.7, 3.1, 4.4, 1.4],

[5.6, 3. , 4.5, 1.5],

[5.8, 2.7, 4.1, 1. ],

[6.2, 2.2, 4.5, 1.5],

[5.6, 2.5, 3.9, 1.1],

[5.9, 3.2, 4.8, 1.8],

[6.1, 2.8, 4. , 1.3],

[6.3, 2.5, 4.9, 1.5],

[6.1, 2.8, 4.7, 1.2],

[6.4, 2.9, 4.3, 1.3],

[6.6, 3. , 4.4, 1.4],

[6.8, 2.8, 4.8, 1.4],

[6.7, 3. , 5. , 1.7],

[6. , 2.9, 4.5, 1.5],

[5.7, 2.6, 3.5, 1. ],

[5.5, 2.4, 3.8, 1.1],

[5.5, 2.4, 3.7, 1. ],

[5.8, 2.7, 3.9, 1.2],

[6. , 2.7, 5.1, 1.6],

[5.4, 3. , 4.5, 1.5],

[6. , 3.4, 4.5, 1.6],

[6.7, 3.1, 4.7, 1.5],

[6.3, 2.3, 4.4, 1.3],

[5.6, 3. , 4.1, 1.3],

[5.5, 2.5, 4. , 1.3],

[5.5, 2.6, 4.4, 1.2],

[6.1, 3. , 4.6, 1.4],

[5.8, 2.6, 4. , 1.2],

[5. , 2.3, 3.3, 1. ],

[5.6, 2.7, 4.2, 1.3],

[5.7, 3. , 4.2, 1.2],

[5.7, 2.9, 4.2, 1.3],

[6.2, 2.9, 4.3, 1.3],

[5.1, 2.5, 3. , 1.1],

[5.7, 2.8, 4.1, 1.3],

[6.3, 3.3, 6. , 2.5],

[5.8, 2.7, 5.1, 1.9],

[7.1, 3. , 5.9, 2.1],

[6.3, 2.9, 5.6, 1.8],

[6.5, 3. , 5.8, 2.2],

[7.6, 3. , 6.6, 2.1],

[4.9, 2.5, 4.5, 1.7],

[7.3, 2.9, 6.3, 1.8],

[6.7, 2.5, 5.8, 1.8],

[7.2, 3.6, 6.1, 2.5],

[6.5, 3.2, 5.1, 2. ],

[6.4, 2.7, 5.3, 1.9],

[6.8, 3. , 5.5, 2.1],

[5.7, 2.5, 5. , 2. ],

[5.8, 2.8, 5.1, 2.4],

[6.4, 3.2, 5.3, 2.3],

[6.5, 3. , 5.5, 1.8],

[7.7, 3.8, 6.7, 2.2],

[7.7, 2.6, 6.9, 2.3],

[6. , 2.2, 5. , 1.5],

[6.9, 3.2, 5.7, 2.3],

[5.6, 2.8, 4.9, 2. ],

[7.7, 2.8, 6.7, 2. ],

[6.3, 2.7, 4.9, 1.8],

[6.7, 3.3, 5.7, 2.1],

[7.2, 3.2, 6. , 1.8],

[6.2, 2.8, 4.8, 1.8],

[6.1, 3. , 4.9, 1.8],

[6.4, 2.8, 5.6, 2.1],

[7.2, 3. , 5.8, 1.6],

[7.4, 2.8, 6.1, 1.9],

[7.9, 3.8, 6.4, 2. ],

[6.4, 2.8, 5.6, 2.2],

[6.3, 2.8, 5.1, 1.5],

[6.1, 2.6, 5.6, 1.4],

[7.7, 3. , 6.1, 2.3],

[6.3, 3.4, 5.6, 2.4],

[6.4, 3.1, 5.5, 1.8],

[6. , 3. , 4.8, 1.8],

[6.9, 3.1, 5.4, 2.1],

[6.7, 3.1, 5.6, 2.4],

[6.9, 3.1, 5.1, 2.3],

[5.8, 2.7, 5.1, 1.9],

[6.8, 3.2, 5.9, 2.3],

[6.7, 3.3, 5.7, 2.5],

[6.7, 3. , 5.2, 2.3],

[6.3, 2.5, 5. , 1.9],

[6.5, 3. , 5.2, 2. ],

[6.2, 3.4, 5.4, 2.3],

[5.9, 3. , 5.1, 1.8]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'frame': None, 'target\_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '.. \_iris\_dataset:\n\nIris plants dataset\n--------------------\n\n\*\*Data Set Characteristics:\*\*\n\n :Number of Instances: 150 (50 in each of three classes)\n :Number of Attributes: 4 numeric, predictive attributes and the class\n :Attribute Information:\n - sepal length in cm\n - sepal width in cm\n - petal length in cm\n - petal width in cm\n - class:\n - Iris-Setosa\n - Iris-Versicolour\n - Iris-Virginica\n \n :Summary Statistics:\n\n ============== ==== ==== ======= ===== ====================\n Min Max Mean SD Class Correlation\n ============== ==== ==== ======= ===== ====================\n sepal length: 4.3 7.9 5.84 0.83 0.7826\n sepal width: 2.0 4.4 3.05 0.43 -0.4194\n petal length: 1.0 6.9 3.76 1.76 0.9490 (high!)\n petal width: 0.1 2.5 1.20 0.76 0.9565 (high!)\n ============== ==== ==== ======= ===== ====================\n\n :Missing Attribute Values: None\n :Class Distribution: 33.3% for each of 3 classes.\n :Creator: R.A. Fisher\n :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature. Fisher\'s paper is a classic in the field and\nis referenced frequently to this day. (See Duda & Hart, for example.) The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant. One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n Mathematical Statistics" (John Wiley, NY, 1950).\n - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n Structure and Classification Rule for Recognition in Partially Exposed\n Environments". IEEE Transactions on Pattern Analysis and Machine\n Intelligence, Vol. PAMI-2, No. 1, 67-71.\n - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions\n on Information Theory, May 1972, 431-433.\n - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II\n conceptual clustering system finds 3 classes in the data.\n - Many, many more ...', 'feature\_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename': 'iris.csv', 'data\_module': 'sklearn.datasets.data'}

K-Mean: 0.3157894736842105

EM Algorithm: 0.9473684210526315

1. **Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.**

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

iris=datasets.load\_iris()

print("Iris Data set loaded...")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris.data,iris.target,test\_size=0.1)

#random\_state=0

for i in range(len(iris.target\_names)):

print("Label", i , "-",str(iris.target\_names[i]))

classifier = KNeighborsClassifier(n\_neighbors=2)

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

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

print("Results of Classification using K-nn with K=1 ")

for r in range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-label:", str(y\_test[r])," Predicted-label:", str(y\_pred[r]))

print("Classification Accuracy :" , classifier.score(x\_test,y\_test));

**OUTPUT:-**

Iris Data set loaded...

Label 0 - setosa

Label 1 - versicolor

Label 2 - virginica

Results of Classification using K-nn with K=1

Sample: [5.1 3.8 1.5 0.3] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [6.1 2.6 5.6 1.4] Actual-label: 2 Predicted-label: 1

Classification Accuracy : 0.8666666666666667

Sample: [4.4 3. 1.3 0.2] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [6. 2.2 5. 1.5] Actual-label: 2 Predicted-label: 1

Classification Accuracy : 0.8666666666666667

Sample: [5.8 2.8 5.1 2.4] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.8666666666666667

Sample: [5.5 2.5 4. 1.3] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.8666666666666667

Sample: [4.4 3.2 1.3 0.2] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [5.8 2.6 4. 1.2] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.8666666666666667

Sample: [4.8 3. 1.4 0.1] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [4.8 3.4 1.9 0.2] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [5.6 2.7 4.2 1.3] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.8666666666666667

Sample: [4.6 3.6 1. 0.2] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [6.7 3.1 5.6 2.4] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.8666666666666667

Sample: [4.5 2.3 1.3 0.3] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.8666666666666667

Sample: [4.9 2.4 3.3 1. ] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.8666666666666667

1. **Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points.**

import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 1000)

y = np.log(np.abs((x \*\* 2) - 1) + 0.5)

x = x + np.random.normal(scale=0.05, size=1000)

plt.scatter(x, y, alpha=0.3)

def local\_regression(x0, x, y, tau):

x0 = np.r\_[1, x0]

x = np.c\_[np.ones(len(x)), x]

xw =x.T \* radial\_kernel(x0, x, tau)

beta = np.linalg.pinv(xw @ x) @ xw @ y

return x0 @ beta

def radial\_kernel(x0, x, tau):

return np.exp(np.sum((x - x0) \*\* 2, axis=1) / (-2 \* tau \*\* 2))

def plot\_lr(tau):

domain = np.linspace(-5, 5, num=500)

pred = [local\_regression(x0, x, y, tau) for x0 in domain]

plt.scatter(x, y, alpha=0.3)

plt.plot(domain, pred, color="red")

return plt

plot\_lr(1).show()

**OUTPUT: -**

