Rashtreeya Sikshana Samithi Trust

R V Institute of Technology and Management®

(Affiliated to VTU, Belagavi)

JP Nagar, Bengaluru - 560076

Department of Information Science and Engineering

Course Name: ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LABORATORY

Course Code: 18CSL76

VII Semester

2018 Scheme

Prepared By:

Dr. A. Ajina
Assistant Professor,
Department of Information Science and Engineering
RVITM, Bengaluru - 560076
Email: ajinaa.rvitm@rvei.edu.in

Compiled By:

Mrs.Shilpa C
Instructor,
Department of Information Science and Engineering
RVITM, Bengaluru - 560076
Email: shilpac.rvitm@rvei.edu.in

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LABORATORY

(Effective from the academic year 2018 -2019)

SEMESTER – VII

Subject Code	**18CSL76**	IA Marks	40
Number of Contact Hours/Week	**0:0:3**	Exam Marks	60
Total Number of Contact Hours	**3 Hours/Week**	Exam Hours	3

CREDITS – 02

Laboratory Objectives: This course (18CSL76) will enable students to:

Implement and evaluate AI and ML algorithms in and Python programming language.

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 Back propagation 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 Neighbor 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.

Course Learning Outcomes:

After completing the course, the students will be able to select and implement AI and ML techniques and computing environment that are suitable for the applications under consideration

Select the implementation procedures for the artificial intelligence and machine learning algorithms.

Identify suitable data sets to implement AI and ML algorithms.

Design the Python programs for various AI and ML algorithms.

Find different approaches to improve the accuracy of the AI and ML model.

Apply artificial intelligence and machine learning algorithms to solve problems

Procedure to Conduct Practical Examination

Experiment distribution as per VTU Syllabus

- For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
- 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.

1. **Change of experiment** is allowed only once and marks allotted for procedure to be made zero of the changed part only.
2. **Marks Distribution** (Coursed to change in accordance with university regulations)

For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks.

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

Program 1

1. Implement A* Algorithm

Find the most cost effective path to read from start state A to final state J using A* Algorithm.

Step 1: Place the starting node into OPEN and find its f (n) value.

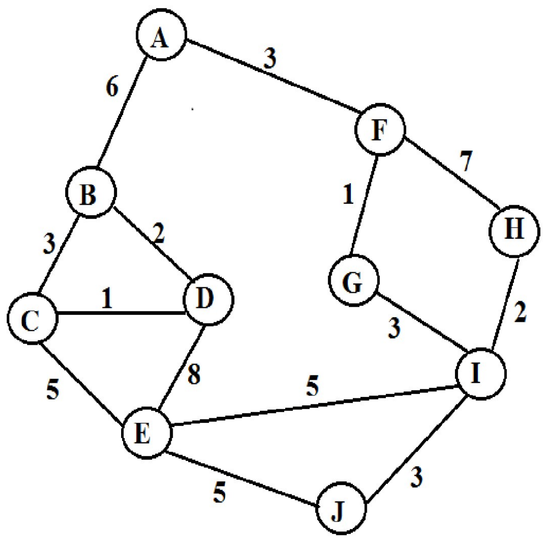
Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

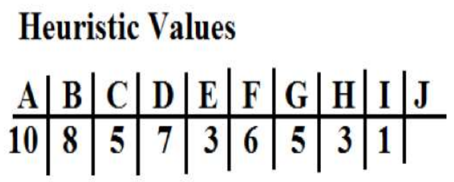
Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2

Step 6: Exit.





Heuristic Values

A	B	C	D	E	F	G	H	I	J
10	8	5	7	3	6	5	3	1	

Program:

```
from collections import deque
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis
    def get_neighbors(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([])
        poo = {}
        poo[start] = 0
        par = {}
        par[start] = start
        while len(open_lst) > 0:
            n = None
            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    n = v;
            if n == None:
                print('Path does not exist!')
                return None
            if n == stop:
                reconst_path = []
                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]
                reconst_path.append(start)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path
            for (m, weight) in self.get_neighbors(n):
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = poo[n] + weight
                else:
                    if poo[m] > poo[n] + weight:
                        poo[m] = poo[n] + weight
                        par[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 does not exist!')
        return None
```

Input 1:

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

```
Path found: ['A', 'B', 'D']
```

Input 2:

```
H = {
    'A': 10,
    'B': 8,
    'C': 5,
    'D': 7,
    'E': 3,
    'F': 6,
    'G': 5,
    'H': 3,
    'I': 1,
    'J': 0
}

adjac_lis = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],


    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'J')
```

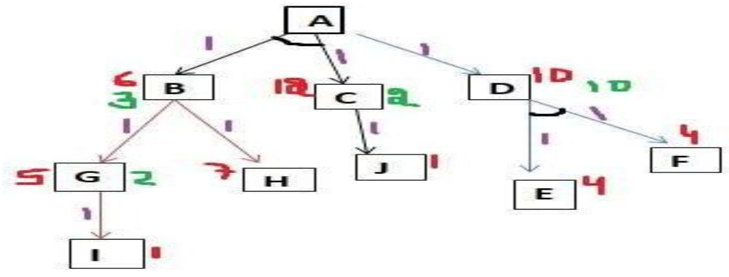
```
Path found: ['A', 'F', 'G', 'I', 'J']
```

Program 2

Implement AO* Algorithm

Algorithm

1. It is an informed search and works as Best First Search.
2. AO* algorithm is based on problem decomposition.
3. It represents an AND-OR graph algorithm that is used to find more than one solution.
4. It is an efficient method to explore a solution path.



The algorithm always moves towards a lower cost value.

Basically, we will calculate the cost function here (F(n)= G (n) + H (n))

H: heuristic/ estimated value of the nodes. and G: actual cost or edge value (here unit value).

Here we have taken the edges value 1, meaning we have to focus solely on the heuristic value.

1. The Purple color values are edge values (here all are same that is one).

2. The Red color values are Heuristic values for nodes.

3. The Green color values are New Heuristic values for nodes.

Procedure:

1. In the above diagram we have two ways from A to D or A to B-C (because of and condition). calculate cost to select a path

2. F(A-D) = 1+10 = 11 and F(A-BC) = 1 + 1 + 6 +12 = 20

3. As we can see F(A-D) is less than F(A-BC) then the algorithm choose the path F(A-D).

4. Form D we have one choice that is F-E.

5. F(A-D-FE) = 1+1+ 4 +4 =10

6. Basically 10 is the cost of reaching FE from D. And Heuristic value of node D also denote the cost of reaching FE from D. So, the new Heuristic value of D is 10.

7. And the Cost from A-D remain same that is 11.

Suppose we have searched this path and we have got the Goal State, then we will
never explore the other path. (this is what AO* says but here we are going to explore
other path as well to see what happen)

Program 2:

```
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis
        self.H1 = {
            'A': 1,
            'B': 6,
            'C': 2,
            'D': 2,
            'E': 2,
            'F': 1,
            'G': 5,
            'H': 7,
            'I': 7,
            'J': 1,
            'T': 3
        }
        self.H = {
            'A': 1,
            'B': 6,
            'C': 12,
            'D': 10,
            'E': 4,
            'F': 4,
            'G': 5,
            'H': 7,
        }
        self.parent={}
        self.openList=set()
        self.hasRevised=[]
        self.solutionGraph={}
        self.solvedNodeList=set()

    def get_neighbors(self, v):
        return self.adjac_lis.get(v,'')

    def updateNode(self, v):

        if v in self.solvedNodeList:
            return

        feasibleChildNodeList=[]
        minimumCost=None
```

```
minimumCostFeasibleChildNodesDict={}

print("CURRENT PROCESSING NODE:", v)
print("___________________________")

for (c, weight) in self.get_neighbors(v):

    feasibleChildNodeList=[]

    cost= self.getHeuristicNodeValue(c) + 1
    feasibleChildNodeList.append(c)
    andNodesList=self.getAndNodes(v)

    for nodeTuple in andNodesList:
        if c in nodeTuple:
            for andNode in nodeTuple:
                if andNode!=c:
                    feasibleChildNodeList.append(andNode)
                    cost=cost+self.getHeuristicNodeValue(andNode) + 1
    if minimumCost==None:
        minimumCost=cost
        for child in feasibleChildNodeList:
            self.parent[child]=v
        minimumCostFeasibleChildNodesDict[minimumCost]=feasibleChildNodeList

    else:
        if minimumCost>cost:
            minimumCost=cost
            for child in feasibleChildNodeList:
                self.parent[child]=v
            minimumCostFeasibleChildNodesDict[minimumCost]=feasibleChildNodeList

if minimumCost==None:
    minimumCost=self.getHeuristicNodeValue(v)
    self.solvedNodeList.add(v)

else:
    self.setHeuristicNodeValue(v,minimumCost)
    for child in minimumCostFeasibleChildNodesDict[minimumCost]:
        if child not in self.solvedNodeList:
            self.openList.add(child)
            self.solutionGraph[v]= minimumCostFeasibleChildNodesDict[minimumCost]

solved=True
```

```
        for c in self.solutionGraph.get(v,''):
            if c not in self.solvedNodeList:
                solved=solved & False

        if solved == True:
            self.solvedNodeList.add(v)

        print("HEURISTIC VALUES  :", self.H)
        print("OPEN LIST         :", list(self.openList))
        print("MINIMUM COST NODES:", minimumCostFeasibleChildNodesDict.get(minimumCost,"[ ]"))
        print("SOLVED NODE LIST  :", list(self.solvedNodeList))
        print("-----------------------------------------------------------------------------------------")

    def getAndNodes(self,v):
        andNodes={
            'A':[('B','C')],
            'D':[('E','F')]
        }
        return andNodes.get(v, '')
    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value

    def ao_star_algorithm(self, start):

        self.openList = set([start])

        while len(self.openList) > 0:

            v = self.openList.pop()
            self.updateNode(v)

            while v!=start and self.parent[v] not in self.solvedNodeList:

                parent=self.parent[v];
                self.updateNode(parent)
                v=parent

            print("TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION
GRAPH")
            print("-------------------------------------------------------------")
            print("SOLUTION GRAPH:",self.solutionGraph)
            print("\n")
```

```
nodeList1 = {
    'A': [('B', 1), ('C', 1), ('D', 1)],
    'B': [('G', 1), ('H', 1)],
    'C': [('J', 1)],
    'D': [('E', 1), ('F', 1)],
    'G': [('I', 1)]
}
nodeList = {
    'A': [('B', 1), ('C', 1), ('D', 1)],
    'B': [('G', 1), ('H', 1)],
    'D': [('E', 1), ('F', 1)]
}
graph = Graph(nodeList)
graph.ao_star_algorithm('A')
```

Output:

```
______________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D']
MINIMUM COST NODES: ['D']
SOLVED NODE LIST  : []
------------------------------------------------------------------------------------------
TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION GRAPH
------------------------------------------------------------------
SOLUTION GRAPH: {'A': ['D']}


CURRENT PROCESSING NODE: D
______________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['F', 'E']
MINIMUM COST NODES: ['E', 'F']
SOLVED NODE LIST  : []
------------------------------------------------------------------------------------------
CURRENT PROCESSING NODE: A
______________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['E', 'D', 'F']
MINIMUM COST NODES: ['D']
SOLVED NODE LIST  : []
------------------------------------------------------------------------------------------
TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION GRAPH
```

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


CURRENT PROCESSING NODE: E
______________________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D', 'F']
MINIMUM COST NODES: [ ]
SOLVED NODE LIST  : ['E']
--------------------------------------------------------------------------------------
CURRENT PROCESSING NODE: D
______________________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D', 'F']
MINIMUM COST NODES: ['E', 'F']
SOLVED NODE LIST  : ['E']
--------------------------------------------------------------------------------------
CURRENT PROCESSING NODE: A
______________________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D', 'F']
MINIMUM COST NODES: ['D']
SOLVED NODE LIST  : ['E']
--------------------------------------------------------------------------------------
TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION GRAPH
------------------------------------------------------------
SOLUTION GRAPH: {'A': ['D'], 'D': ['E', 'F']}


CURRENT PROCESSING NODE: D
______________________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['F']
MINIMUM COST NODES: ['E', 'F']
SOLVED NODE LIST  : ['E']
--------------------------------------------------------------------------------------
CURRENT PROCESSING NODE: A
______________________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D', 'F']
MINIMUM COST NODES: ['D']
SOLVED NODE LIST  : ['E']
--------------------------------------------------------------------------------------
```

```
TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION GRAPH
------------------------------------------------------------
SOLUTION GRAPH: {'A': ['D'], 'D': ['E', 'F']}


CURRENT PROCESSING NODE: F
______________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D']
MINIMUM COST NODES: [ ]
SOLVED NODE LIST  : ['F', 'E']
------------------------------------------------------------------------------------
CURRENT PROCESSING NODE: D
______________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D']
MINIMUM COST NODES: ['E', 'F']
SOLVED NODE LIST  : ['D', 'F', 'E']
------------------------------------------------------------------------------------
CURRENT PROCESSING NODE: A
______________________________
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
OPEN LIST         : ['D']
MINIMUM COST NODES: ['D']
SOLVED NODE LIST  : ['A', 'D', 'F', 'E']
------------------------------------------------------------------------------------
TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION GRAPH
------------------------------------------------------------
SOLUTION GRAPH: {'A': ['D'], 'D': ['E', 'F']}


TRAVERSE SOLUTION FROM ROOT TO COMPUTE THE FINAL SOLUTION GRAPH
------------------------------------------------------------
SOLUTION GRAPH: {'A': ['D'], 'D': ['E', 'F']}
```

Program 3

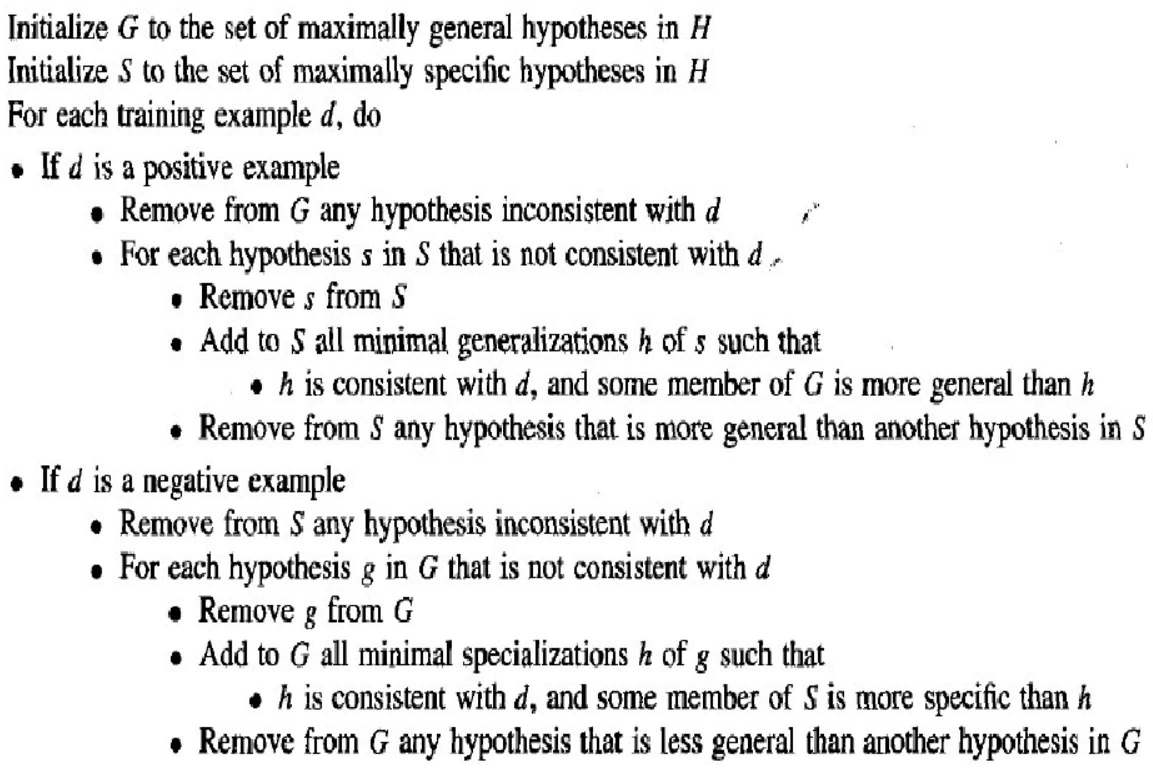
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Task: The CANDIDATE-ELIMINATION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Dataset: Enjoy Sports Training Examples:

Example	Sky	Air Temp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Candidate Elimination Algorithm:



```
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do
• If d is a positive example
    • Remove from G any hypothesis inconsistent with d
    • For each hypothesis s in S that is not consistent with d
        • Remove s from S
        • Add to S all minimal generalizations h of s such that
            • h is consistent with d, and some member of G is more general than h
        • Remove from S any hypothesis that is more general than another hypothesis in S
• If d is a negative example
    • Remove from S any hypothesis inconsistent with d
    • For each hypothesis g in G that is not consistent with d
        • Remove g from G
        • Add to G all minimal specializations h of g such that
            • h is consistent with d, and some member of S is more specific than h
        • Remove from G any hypothesis that is less general than another hypothesis in G
```

Candidate Elimination Program:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('finds.csv'))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
     specific_h = concepts[0].copy()
     general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    for i, h in enumerate(concepts):

        if target[i] == "Yes":
            for x in range(len(specific_h)):

                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
                if target[i] == "No":
            for x in range(len(specific_h)):

                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
            indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
                general_h.remove(['?', '?', '?', '?', '?', '?'])
            return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")
```

Input 1:

```
Sky       AirTemp   Humidity  Wind      Water     Forecast  EnjoySport
Sunny     Warm      Normal    Strong    Warm      Same      Yes
Sunny     Warm      High      Strong    Warm      Same      Yes
Rainy     Cold      High      Strong    Warm      Change    No
Sunny     Warm      High      Strong    Cool      Change    Yes
```

Output 1:

```
Final S:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final G:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Program 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.

Task: ID3 determines the information gain for each candidate attribute, then selects the one with highest information gain as the root node of the tree. The information gain values for all four attributes are calculated using the following formula:

$$\text{Entropy}(S)=\sum - P(I).\log 2P(I)$$

$$\text{Gain}(S,A)=\text{Entropy}(S)-\sum[P(S/A).\text{Entropy}(S/A)]$$

Dataset: pima-indians-diabetes.csv

<u>ID3 Algorithm:</u>

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

```
• Create a Root node for the tree
• If all Examples are positive, Return the single-node tree Root, with label = +
• If all Examples are negative, Return the single-node tree Root, with label = -
• If Attributes is empty, Return the single-node tree Root, with label = most common value
  of Target_attribute in Examples
• Otherwise Begin
    ❖ A ← the attribute from Attributes that best* classifies Examples
    ❖ The decision attribute for Root ←A
    ❖ For each possible value, υi, of A,
  • Add a new tree branch below Root, corresponding to the test A = υi
  • Let Examplesυi ,be the subset of Examples that have value υi for A
  • If Examples υi, is empty
        ❖ Then below this new branch add a leaf node with label=most common value of
          Target_attribute in Examples
        ❖ Else below this new branch add the subtree
```

ID3 Program:

```
import pandas as pd
import math
import numpy as np
data = pd.read_csv("tennis.csv")
features = [feat for feat in data]
features.remove("answer")
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
```

```
            #print ("\n",subdata)
            sub_e = entropy(subdata)
            gain -= (float(len(subdata)) / float(len(examples))) * sub_e
            #print ("\n",gain)
        return gain
    def ID3(examples, attrs):
        root = Node()
        max_gain = 0
        max_feat = ""
        for feature in attrs:
            #print ("\n",examples)
            gain = info_gain(examples, feature)
            if gain > max_gain:
                max_gain = gain
                max_feat = feature
        root.value = max_feat
        #print ("\nMax feature attr",max_feat)
        uniq = np.unique(examples[max_feat])
        #print ("\n",uniq)
        for u in uniq:
            #print ("\n",u)
            subdata = examples[examples[max_feat] == u]
            #print ("\n",subdata)
            if entropy(subdata) == 0.0:
                newNode = Node()
                newNode.isLeaf = True
                newNode.value = u
                newNode.pred = np.unique(subdata["answer"])
                root.children.append(newNode)
            else:
                dummyNode = Node()
                dummyNode.value = u
                new_attrs = attrs.copy()
                new_attrs.remove(max_feat)
                child = ID3(subdata, new_attrs)
                dummyNode.children.append(child)
                root.children.append(dummyNode)
        return root
    def printTree(root: Node, depth=0):
        for i in range(depth):
            print("\t", end="")
        print(root.value, end="")
        if root.isLeaf:
            print(" -> ", root.pred)
        print()
        for child in root.children:
            printTree(child, depth + 1)
    def classify(root: Node, new):
        for child in root.children:
            if child.value == new[root.value]:
                if child.isLeaf:
                    print ("Predicted Label for new example", new," is:", child.pred)
                    exit
                else:
                    classify (child.children[0], new)
    root = ID3(data, features)
    print("Decision Tree is:")
    printTree(root)
    print ("------------------")
    new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal",
"wind":"strong"}
    classify (root, new)
```

Output:

```
Decision Tree is:
outlook
        overcast ->  ['yes']
        rain
```

```
                wind
                        strong ->  ['no']
                        weak ->  ['yes']
        sunny
                humidity
                        high ->  ['no']
                        normal ->  ['yes']
------------------------------------------------------------------------------------------------------------------------
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind':
'strong'}  is: ['yes']
```

Program 5:

Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data set

The stochastic gradient descent version of the BACKPROPAGATION algorithm for feed forward networks containing two layers of sigmoid units.

Step 1: begins by constructing a network with the desired number of hidden and output units and initializing all network weights to small random values. . For each training example, it applies the network to the example, calculates the error of the network output for this example, computes the gradient with respect to the error on this example, then updates all weights in the network. This gradient descent step is iterated (often thousands of times, using the same training examples multiple times) until the network performs acceptably well.

Step 2: The gradient descent weight-update rule is similar to the delta training rule The only difference is that the error (t - o) in the delta rule is replaced by a more complex error term aj.

Step 3: updates weights incrementally, following the Presentation of each training example. This corresponds to a stochastic approximation to gradient descent. To obtain the true gradient of E one would sum the Sj, xji values over all training examples before altering weight values.

Step 4: The weight-update loop in BACKPROPAGATION may be iterated thousands of times in a typical application. A variety of termination conditions can be used to halt the procedure. One may choose to halt after a fixed number of iterations through the loop, or once the error on the training examples falls below some threshold.

Program 5: Back Propagation

```
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)
y = y/100
def sigmoid (x):
 return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x):
 return x * (1 - x)
epoch=7000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
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))
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)
   hiddengrad = derivatives_sigmoid(hlayer_act)
```

```
    d_hiddenlayer = EH * hiddengrad
    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:

```
 [[0.89632124]
 [0.87734683]
 [0.89627697]]
```

Program 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.

Task: It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Dataset: Pima-indians-diabetes.csv

It is a classification technique based on Bayes" Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as „Naive".

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

1) **Handling Of Data:**
    - Load the data from the CSV file and split in to training and test data set.
    - Training data set can be used to by Naïve Bayes to make predictions.
    - And Test data set can be used to evaluate the accuracy of the model.
2) **Summarize Data:**

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value.

- These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.

- Summary data can be break down into the following sub-tasks:

  - **Separate Data by Class**: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.

  - **Calculate Mean**: We need to calculate the mean of each attribute for a class value. The mean is the central middle or central tendency of the data, and we will use it as the middle of our Gaussian distribution when calculating probabilities.

  - **Calculate Standard Deviation**: We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.

  - **Summarize Dataset**: For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.

  - The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.
  - **Summarize Attributes By Class**: We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

3) **Make Predictions:**
   - ❖ Making predictions involves calculating the probability that a given data instance belongs to each class,
   - ❖ then selecting the class with the largest probability as the prediction.
   - ❖ Finally, estimation of the accuracy of the model by making predictions for each data instance in the test dataset.

4) **Evaluate Accuracy**: The predictions can be compared to the class values in the test dataset and a classification\ accuracy can be calculated as an accuracy ratio between 0& and 100%.

Naïve Bayes Program:

```
import numpy as np
import pandas as pd
mush = pd.read_csv("mushroom.csv")
mush.replace('?',np.nan,inplace=True)

print(len(mush.columns),"columns, after dropping NA,",len(mush.dropna(axis=1).columns))
mush.dropna(axis=1,inplace=True)
target = 'class'
features = mush.columns[mush.columns != target]
classes = mush[target].unique()
test = mush.sample(frac=0.3)
mush = mush.drop(test.index)
probs = {}
probcl = {}
for x in classes:
    mushcl = mush[mush[target]==x][features]
    clsp = {}
    tot = len(mushcl)
    for col in mushcl.columns:
        colp = {}
        for val,cnt in mushcl[col].value_counts().iteritems():
            pr = cnt/tot
            colp[val] = pr
            clsp[col] = colp
       probs[x] = clsp
    probcl[x] = len(mushcl)/len(mush)
def probabs(x):
    if not isinstance(x,pd.Series):
        raise IOError("Arg must of type Series")
    probab = {}
      for cl in classes:
        pr = probcl[cl]
        for col,val in x.iteritems():
            try:
                pr *= probs[cl][col][val]
            except KeyError:
                pr = 0
        probab[cl] = pr
    return probab
def classify(x):
    probab = probabs(x)
    mx = 0
    mxcl = ''
    for cl,pr in probab.items():
        if pr > mx:
            mx = pr
            mxcl = cl
    return mxcl
b = []
for i in mush.index:
       b.append(classify(mush.loc[i,features]) == mush.loc[i,target])
print(sum(b),"correct of",len(mush))
print("Accuracy:", sum(b)/len(mush))
```

#Test data

```
b = []
for i in test.index:
        b.append(classify(test.loc[i,features]) == test.loc[i,target])
print(sum(b),"correct of",len(test))
print("Accuracy:",sum(b)/len(test))
```

Output

```
20 columns, after dropping NA, 19
45 correct of 45
Accuracy: 1.0
19 correct of 20
Accuracy: 0.95
```

Program 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.

Introduction to Expectation-Maximization (EM)

The EM algorithm tends to get stuck less than K-means algorithm. The idea is to assign data points partially to different clusters instead of assigning to only one cluster. To do this partial assignment, we model each cluster using a probabilistic distribution So a data point associates with a cluster with certain probability and it belongs to the cluster with the highest probability in the final assignment

Expectation-Maximization (EM) algorithm

Step 1: An initial guess is made for the model's parameters and a probability distribution is created. This is sometimes called the "E-Step" for the "Expected" distribution.

Step 2: Newly observed data is fed into the model.

Step 3: The probability distribution from the E-step is drawn to include the new data. This is sometimes called the "M-step."

Step 4: Steps 2 through 4 are repeated until stability.

EM algorithm Programs:

```
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
data = pd.read_csv('ex.csv')
print("Input Data and Shape")
print(data.shape)
data.head()
print(data.head())
f1 = data['V1'].values
```

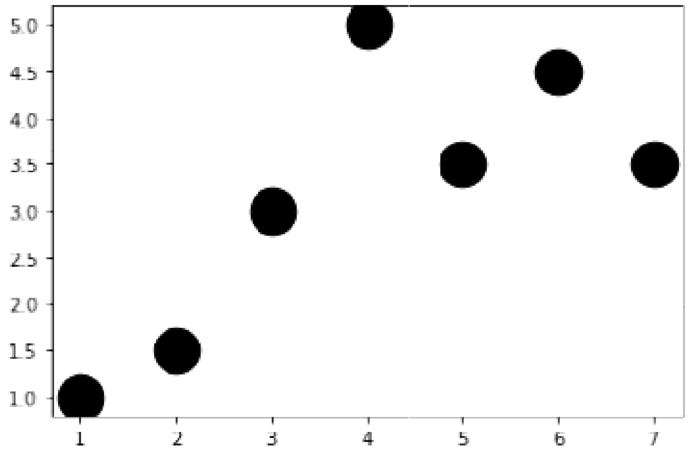
```
print("f1")
print(f1)
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))
print("x")
print(X)
print('Graph for whole dataset')
plt.scatter(f1, f2, c='black', s=600)
plt.show()
kmeans = KMeans(2, random_state=0)
labels = kmeans.fit(X).predict(X)
print("labels")
print(labels)
centroids = kmeans.cluster_centers_
print("centroids")
print(centroids)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40);
print('Graph using Kmeans Algorithm')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='#050505')
plt.show()
gmm = GaussianMixture(n_components=2).fit(X)
labels = gmm.predict(X)
print("lLABELS GMM")
print(labels)
probs = gmm.predict_proba(X)
size = 10 * probs.max(1) ** 3
print('Graph using EM Algorithm')
plt.scatter(X[:, 0], X[:, 1], c=labels, s=size, cmap='viridis');
plt.show()
```

Output

```
Input Data and Shape
(7, 3)
  V1   V2  Unnamed: 2
```

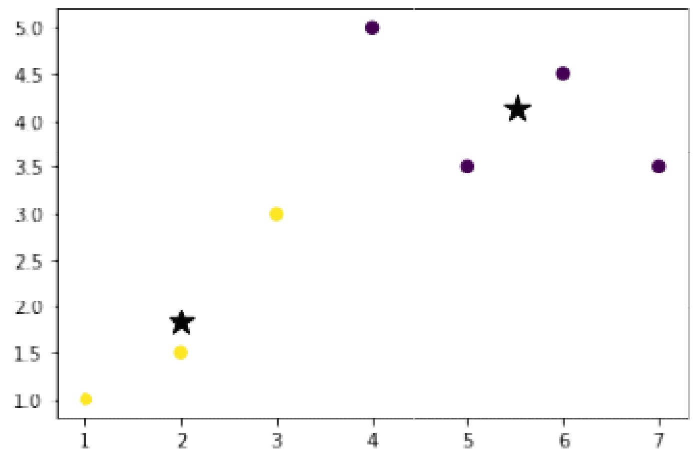
```
0   1  1.0       1.0
1   2  1.5       2.0
2   3  3.0       4.0
3   4  5.0       7.0
4   5  3.5       5.0
f1
[1 2 3 4 5 6 7]
x
[[1.  1. ]
 [2.  1.5]
 [3.  3. ]
 [4.  5. ]
 [5.  3.5]
 [6.  4.5]
 [7.  3.5]]
```

Graph for whole dataset



```
labels
[1 1 1 0 0 0 0]
centroids
[[5.5        4.125     ]
 [2.         1.83333333]]
```

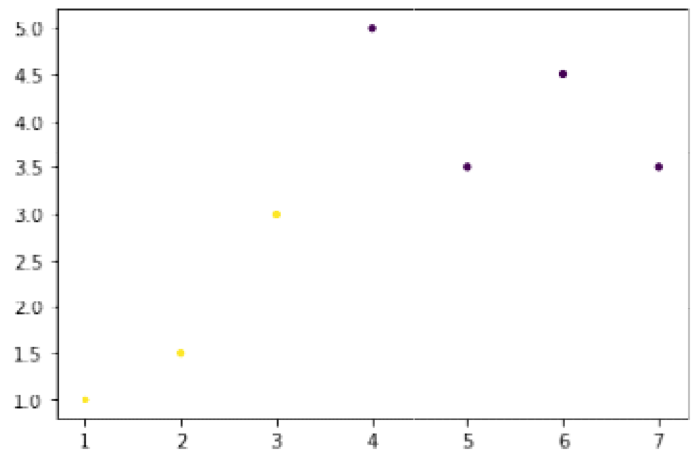
Graph using Kmeans Algorithm



lLABELS GMM

[1 1 1 0 0 0 0]

Graph using EM Algorithm



Program 8

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

TASK: The task of this program is to classify the IRIS data set examples by using the k-Nearest Neighbour algorithm. The new instance has to be classified based on its k nearest neighbors.

Dataset: iris.csv

ALGORITHM

Let m be the number of training data samples. Let p be an unknown point.

```
1. Store the training samples in an array of data points arr[]. This means each element of this
   array represents a tuple (x, y).
2. for i=0 to m:
     Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an
   already classified data point.
4. Return the majority label among S.
```

KNN Program

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()
print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
   print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))
print("\n IRIS DATA :\n",iris_dataset["data"])
 X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"],
random_state=0)
```

```
print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)
 for i in range(len(X_test)):
    x = X_test[i]
    x_new = np.array([x])
    prediction = kn.predict(x_new)
    print("\n Actual : {0} {1}, Predicted :{2}{3}".format(y_test[i],iris_dataset["target_names"]
[y_test[i]] , prediction,iris_dataset["target_names"][prediction]))
print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

OUTPUT

```
IRIS FEATURES \ TARGET NAMES:
['setosa' 'versicolor' 'virginica']
[0]:[setosa]
[1]:[versicolor]
[2]:[virginica]
 IRIS DATA :
 [[5.1 3.5 1.4 0.2]
 ….
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
 Target :
[ 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 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 2 2 ]
```

X TRAIN

```
[[5.9 3.  4.2 1.5]
[5.8 2.6 4.  1.2]
[6.8 3.  5.5 2.1]
[7.7 3.8 6.7 2.2]
[4.6 3.2 1.4 0.2]]
```

X TEST

```
[[5.8 2.8 5.1 2.4]
[5.7 3.8 1.7 0.3]
[6.  2.7 5.1 1.6]]
```

Y TRAIN

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

Y TEST

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

```
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 1 versicolor, Predicted :[2]['virginica']
```

TEST SCORE [ACCURACY]: 0.97

Program 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.

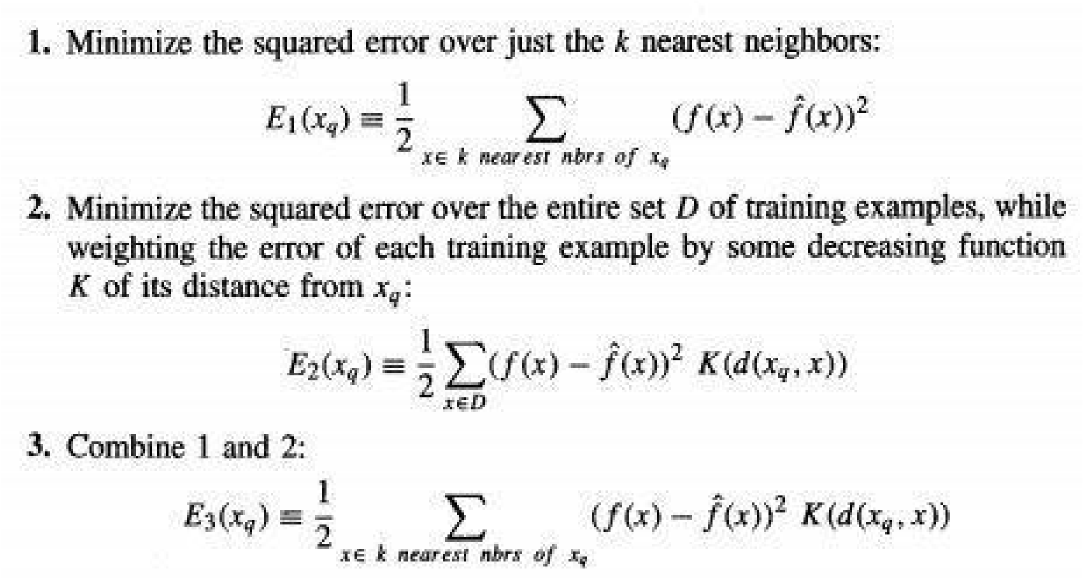
Locally Weighted Regression –

1. Nonparametric regression is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data(training examples).
2. Nonparametric regression requires larger sample sizes than regression based on parametric models. Because larger the data available, accuracy will be high.

Locally Weighted Linear Regression –

1. Locally weighted regression is called local because the function is approximated based a only on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point.
2. Query point is nothing but the point nearer to the target function, which will help in finding the actual position of the target function.

Let us consider the case of locally weighted regression in which the target function f is approximated near x, using a linear function of the form



1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \; k \; nearest \; nbrs \; of \; x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

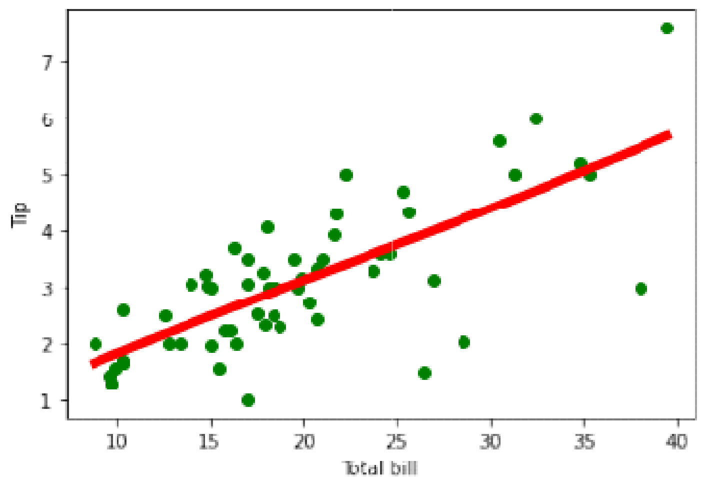
$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \; k \; nearest \; nbrs \; of \; x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

Program

```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from numpy.linalg import *
def kernel(point,xmat, k):
    m,n = shape(xmat)
    weights = mat(eye((m)))
    for j in range(m):
        diff = point - X[j]
          weights[j,j] = exp(diff*diff.T/(-2.0*k**2))
           return weights
 def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
       W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
     def localWeightRegression(xmat,ymat,k):
    m,n = shape(xmat)
        ypred = zeros(m)
    for i in range(m):
                ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
data = pd.read_csv('tips.csv')
bill = array(data.total_bill)
tip = array(data.tip)
mbill = mat(bill)
mtip = mat(tip)
m= shape(mbill)[1]
one = mat(ones(m))
X= hstack((one.T,mbill.T))
```

```
#set k here
ypred = localWeightRegression(X,mtip,10)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
    fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output



Viva Questions

1. Is A* better than Dijkstra?

A* is usually considered better than Dijkstra as it performs informed and not uninformed searches. It expands more promising vertices.

2. What is A* algorithm?

A* search algorithm is an algorithm that separates it from other traversal techniques. This makes A* smart and pushes it much ahead of conventional algorithms.

3. How overestimation is handled in the A* algorithm?

Overestimation happens when the estimate of the heuristic is more than the actual cost of the final path.

4. Why is A* optimal?

A* Algorithms are optimal. It relies on an open and closed list to find a path that is optimal and complete towards the goal.

5. Why is the A* algorithm popular?

A* Algorithm is popular because it is a technique that is used for finding path and graph traversals. Many web-based maps and games use this algorithm

6. Differentiate between A* and Ao* algorithm?

An A* is an OR graph algorithm used to find a single solution, while AO* Algorithm is an AND -OR graph algorithm used to find many solutions by ANDing over more than one branch.

7. Which algorithm guarantees an Optimal Solution A* or AO* ?

Advantages of Ao*

It is an optimal algorithm.If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages of Ao*

Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

8. What is an informed search?

Any search where the goal state as well as the path to reach the goal is defined is known as informed search. Generally makes use of heuristics to make the path defined.

9. What makes the AND-OR graph efficient?

If the cost function of first branch of AND-graph is more than the calculated cost of OR graph , the Algorithm directly picks the path of the OR graph and continues to traverse, this method is also known as short circuiting

10. What is a decision tree algorithm?

A Decision Tree is a supervised machine learning algorithm that can be used for both Regression and Classification problem statements. It divides the complete dataset into smaller subsets while at the same time an associated Decision Tree is incrementally developed. The final output of the Decision Trees is a Tree having Decision nodes and leaf nodes. A Decision Tree can operate on both categorical and numerical data.

11. Explain entropy in the decision tree.

Entropy is the measurement of disorder or impurities in the information processed in machine learning. It determines how a decision tree chooses to split data

12. What is Back Propagation?

Backpropagation can be written as a function of the neural network. Backpropagation algorithms are a set of methods used to efficiently train artificial neural networks following a gradient descent approach which exploits the chain rule.

13. Back Propagation will comes in which classification of machine learning algorithm?

Specified algorithms

14. What are the types of back propagation technique?

Two Types of Backpropagation Networks are:Static Back-propagation. Recurrent Backpropagation

15. Which rule is used in backpropagation algorithm?

The backpropagation algorithm is based on generalizing the Widrow-Hoff learning rule.

16. What is Naive Bayes classifier in machine learning?

Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions

17. What are the applications of naive bayes learning method?

Face recognition

Medical diagnosis

News recognition

Weather Prediction

18. What is EM algorithm in machine learning?

The Expectation-Maximization (EM) algorithm is defined as the combination of various unsupervised machine learning algorithms, which is used to determine the local maximum likelihood estimates (MLE) or maximum a posteriori estimates (MAP) for unobservable variables in statistical models.

19. Why KNN is best for machine learning?

The KNN algorithm can compete with the most accurate models because it makes highly accurate predictions

20. What is the accuracy of KNN?

Based on the result of the test, the highest average value of accuracy is SVM because the accuracy value is higher, it is 92.40% at linear kernel. The average value of KNN accuracy is only 71.28% at K=7