VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. ROAD, V.V PURAM, BANGALORE – 560 004

DEPARTMENT OF
INFORMATION SCIENCE AND ENGINEERING

SUBJECT CODE: 18CSL76

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY MANUAL

As per Choice Based Credit System Scheme (CBCS)

FOR VII SEMESTER CSE/ISE AS PRESCRIBED BY VTU

Effective from the Academic year 2020-2021

Prepared By:
Prof. Shivakumar B R**,** M.Tech
Assistant Professor
Dept of ISE, BIT

1. PREREQUISITES:

- Creative thinking, sound mathematical insight and programming skills
- Data Structures and Applications & Lab (18CS32 & 18CSL38)
- Design and Analysis of Algorithms & Lab (18CS42 & 18CSL47)
- Object Oriented Concepts & Java Programming (18CS45)
- Application Development using Python (18CS55)

2. BASE COURSE:

- Artificial Intelligence and Machine Learning (18CS71)

3. COURSE OUTCOMES:

At the end of the course, the student will be able to:

CO1	**Apply** appropriate data sets to the Machine learning algorithms to predict the target.
CO2	**Analyze** the machine learning algorithms for different number of training examples, various numbers of epochs and hyper parameters.
CO3	**Evaluate** machine learning algorithms to select appropriate algorithm for a given problem for different contexts.
CO4	**Create** Python or Java program to implement A*, AO*, Find-S, candidate elimination, ID3, BPN, Naive Bayesian classifier, KNN, K-Means algorithm
CO5	Use the **modern tool** such as Windows/Linux operating system to develop and test machine learning program using Python/Java languages.

4. RESOURSES REQUIRED:

- Hardware resources
  - Desktop PC
  - Windows / Linux operating system
- Software resources
  - Python
  - Anaconda IDE with Spider
- Datasets from standard repositories (Ex: https://archive.ics.uci.edu/ml/datasets.html)

5. RELEVANCE OF THE COURSE:

Project work (18CSP77, 18CSP83)

6. GENERAL INSTRUCTIONS:

- Implement the program in Python editor like Spider or Jupyter and demonstrate the same.

- External practical examination.
  - All laboratory experiments are to be included
  - Students are allowed to pick one experiment from the lot.
  - Marks distribution:
    Procedure + Execution + Viva = 15 + 70 + 15 = 100 Marks
  - Change of experiment is allowed only once and marks allotted to the procedure part to be made zero

7. CONTENTS:

1. Implement and evaluate AI and ML algorithms in and Python programming language.
2. Data sets can be taken from standard repositories or constructed by the students.

Exp. No.	Title of the Experiments	RBT Level	CO
1	Implement A* Search algorithm.	L3	1,2,3,4
2	Implement AO* Search algorithm.	L3	1,2,3,4
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.	L3	1,2,3,4
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.	L3	1,2,3,4
5	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	L3	1,2,3,4
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.	L3	1,2,3,4
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.	L3	1,2,3,4
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.	L3	1,2,3,4
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	L3	1,2,3,4

Course outcomes: The students should be able to:

1. **Understand** the implementation procedures for the machine learning algorithms.
2. **Design** Python programs for various Learning algorithms.
3. **Apply** appropriate data sets to the Machine Learning algorithms.
4. **Identify** and apply Machine Learning algorithms to solve real world problems.

8. REFERENCE:

1. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education.
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning, 2nd edition, Springer series in statistics.
3. Ethem Alpaydın, Introduction to machine learning, second edition, MIT press.

C. EVALUATION SCHEME

For CBCS 2018 scheme:

Conduction of Practical Examination:
1. All laboratory experiments are to be included for practical examination.
2. Students are allowed to pick one experiment from the lot.
3. Strictly follow the instructions as printed on the cover page of answer script
4. Marks distribution: Procedure + Conduction + Viva:**15 + 70 +15 (100)**
Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

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 Contact Hours	36	**Exam Hours**	03
CREDITS – 02			
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
  - 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.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (Coursed to change in accordance with university regulations)
  a) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  b) For laboratories having PART A and PART B

  Procedure + Execution + Viva = 15 + 70 + 15 = 100 Marks

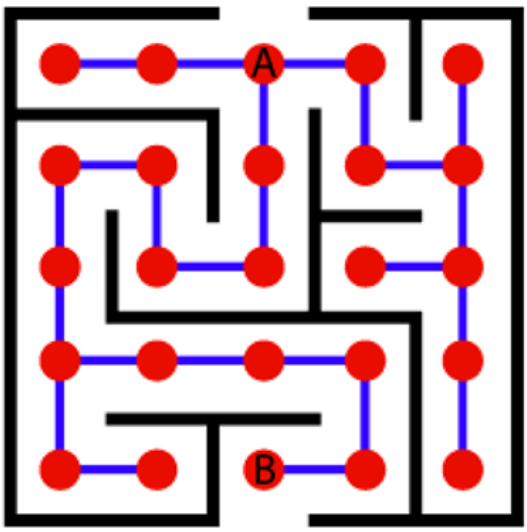
1. EXPERIMENT NO: 1
2. Implement A* Search algorithm
3. LEARNING OBJECTIVES:
   - What is A-Star (A*) Algorithm in Artificial Intelligence?
   - A* Algorithm Steps
   - Implementation with Python

4. AIM: Implement A* Search algorithm.

5. THEORY

What is A* Algorithm in Artificial Intelligence?

The most important advantage of A* search algorithm which separates it from other traversal techniques is that it has a brain. This makes A* very smart and pushes it much ahead of other conventional algorithms.
Consider the diagram below:



Let's try to understand Basic AI Concepts and to comprehend how does A* algorithm work. Imagine a huge maze, one that is too big that it takes hours to reach the endpoint manually. Once you complete it on foot, you need to go for another one. Which implies that you would end up investing a lot of time and effort to find the possible paths in this maze. Now, you want to make it less time-consuming. To make it easier, we will consider this maze as a search problem and will try to apply it to other possible mazes we might encounter in the due course, provided they follow the same structure and rules.
As the first step to convert this maze into a search problem, we need to define these six things.

1. A set of prospective states we might be in
2. A beginning and end state
3. A way to decide if we've reached the endpoint
4. A set of actions in case of possible direction/path changes
5. A function that advises us about the result of an action
6. A set of costs incurring in different states/paths of movement

To solve the problem, we need to map the intersections to the nodes (denoted by the red dots) and all the possible ways we can make movements towards the edges (denoted by the blue lines).

A denotes the starting point and B denotes the endpoint. We define the starting and endpoint at the nodes A and B respectively.

If we use an uninformed search algorithm, it would be like finding a path that is blind, while an informed algorithm for a search problem would take the path that brings you closer to your destination. For instance, consider Rubik's cube; it has many prospective states that you can be in and this makes the solution very difficult. This calls for the use of a guided search algorithm to find a solution. This explains the importance of A*.

Unlike other algorithms, A* decides to take up a step only if it is convincingly sensible and reasonable as per its functions. Which means, it never considers any non-optimal steps. This is why A* is a popular choice for AI systems that replicate the real world – like video games and machine learning.

A* Algorithm Steps

1. Firstly, add the beginning node to the open list
2. Then repeat the following step

– In the open list, find the square with the lowest F cost – and this denotes the current square.
– Now we move to the closed square.
– Consider 8 squares adjacent to the current square and

- Ignore it if it is on the closed list, or if it is not workable. Do the following if it is workable
- Check if it is on the open list; if not, add it. You need to make the current square as this square's a parent. You will now record the different costs of the square like the F, G and H costs.
- If it is on the open list, use G cost to measure the better path. Lower the G cost, the better the path. If this path is better, make the current square as the parent square. Now you need to recalculate the other scores – the G and F scores of this square.

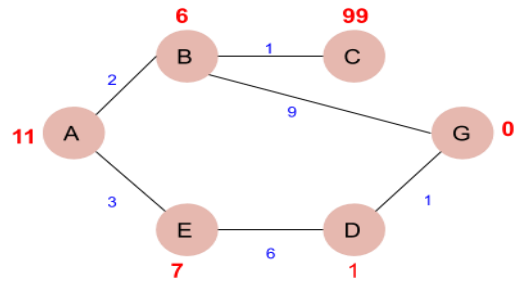
– You'll stop:

- If you find the path, you need to check the closed list and add the target square to it.
- There is no path if the open list is empty and you could not find the target square.

3. Now you can save the path and work backwards starting from the target square, going to the parent square from each square you go, till it takes you to the starting square. You've found your path now.

Implementation with Python

In this section, we are going to find out how A* algorithm can be used to find the most cost-effective path in a graph. Consider the following graph below



The numbers written on edges represent the distance between the nodes while the numbers written on nodes represent the heuristic values. Let us find the most cost-effective path to reach from start state A to final state G using A* Algorithm.

Let's start with node A.Since A is a starting node, therefore, the value of g(x) for A is zero and from the graph, we get the heuristic value of A is 11, therefore

- g(x) + h(x) = f(x)
- 0+ 11 =11
- Thus for A, we can write
- A=11

Now from A, we can go to point B or point E, so we compute f(x) for each of them

- A → B = 2 + 6 = 8
- A → E = 3 + 6 = 9

Since the cost for A → B is less, we move forward with this path and compute the f(x) for the children nodes of B

Since there is no path between C and G, the heuristic cost is set infinity or a very high value

- A → B → C = (2 + 1) + 99= 102
- A → B → G = (2 + 9) + 0 = 11

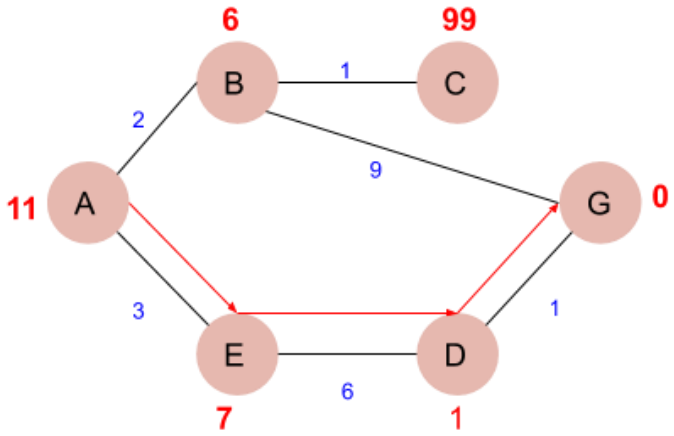
Here the path A → B → G has the least cost but it is still more than the cost of A → E, thus we explore this path further

A → E → D = (3 + 6) + 1 = 10

Comparing the cost of A → E → D with all the paths we got so far and as this cost is least of all we move forward with this path. And compute the f(x) for the children of D

A → E → D → G = (3 + 6 + 1) +0 =**10**

Now comparing all the paths that lead us to the goal, we conclude that A → E → D → G is the most cost-effective path to get from A to G.



Next, we write a program in Python that can find the most cost-effective path by using the a-star algorithm.

First, we create two sets, viz- open, and close. The open contains the nodes that have been visited but their neighbors are yet to be explored. On the other hand, close contains nodes that along with their neighbors have been visited.

6. **PROCEDURE / PROGRAMME :**

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)
```

```
closed_set = set()
g = {} #store distance from starting node
parents = {}
    # parents contains an adjacency map of all #nodes

#ditance of starting node from itself is zero
g[start_node] = 0
#start_node is root node i.e it has no parent nodes
#so start_node is set to its own parent node
parents[start_node] = start_node


while len(open_set) > 0:
   n = None
    #node with lowest f() is found
   for v in open_set:
      if n == None or g[v]+heuristic(v)< g[n] + heuristic(n):
         n = v


   if n == stop_node or Graph_nodes[n] == None:
      pass
   else:
      for (m, weight) in get_neighbors(n):
      #nodes 'm' not in first and last set are added to first
         #n is set its parent
         if m not in open_set and m not in closed_set:
            open_set.add(m)
            parents[m] = n
            g[m] = g[n] + weight
#for each node m,compare its distance from start i.e g(m) to the
#from start through n node
         else:
            if g[m] > g[n] + weight:
               #update g(m)
               g[m] = g[n] + weight
               #change parent of m to n
```

```
                parents[m] = n

                #if m in closed set,remove and add to open
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)

        if n == None:
            print('Path does not exist!')
            return None
         # if the current node is the stop_node then we
        # begin reconstructin the path from it to the start_node
        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]

            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)

    print('Path does not exist!')
    return None
#define fuction to return neighbor and its distance from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
```

```
#for simplicity we ll consider heuristic distances given and this function returns heuristic
#distance for all nodes
def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,
        }
        return H_dist[n]
 #Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],
}
aStarAlgo('A', 'G')
```

OUTPUT:

```
Path found: ['A', 'E', 'D', 'G']
```

1. EXPERIMENT NO: 2
2. TITLE: **AO* SEARCH ALGORITHM**
3. LEARNING OBJECTIVES
4. AIM: **Implement AO* Search algorithm.**
5. THEORY

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

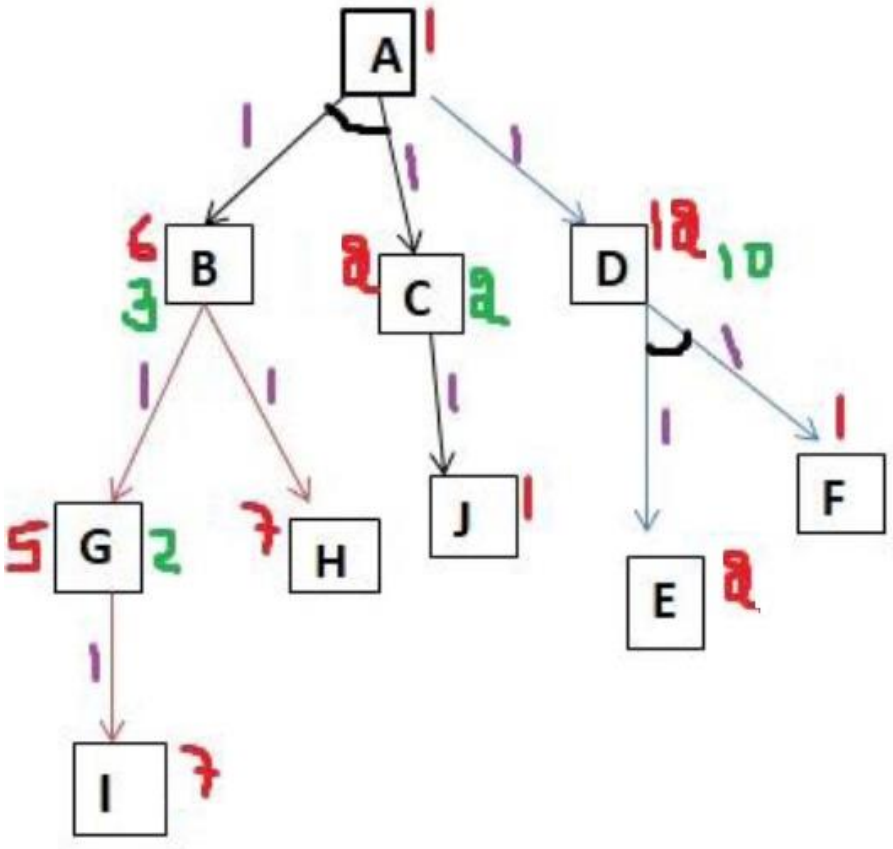
CLOSE:

It contains the nodes that have already been processed.

The distance from current node to goal node.

Implementation:

Graph – 1 as Input to AO Star Search Algorithm




```
#for simplicity we will consider heuristic distances given
print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
'H': 7, 'I': 7, 'J': 1}
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()
```

Output of AO Star Search Algorithm

```
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : A
```

```
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : B

6 ['G']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : A

10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : G

8 ['I']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : B

8 ['H']
HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : A

12 ['B', 'C']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : I

0 []
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G
```

```
1 ['I']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

2 ['G']
HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2,
'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

2 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

0 []
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

1 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C':
['J']}

PROCESSING NODE : A
```

```
5 ['B', 'C']

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

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

6. PROCEDURE/PROGRAMME

```
# Program to Implement recursive AO* 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 START NODE:",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)
        print(minimumCost, childNodeList)
        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}
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()
```

OUTPUT:

```
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
------------------------------------------------------------------------------------
6 ['G']
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : G
------------------------------------------------------------------------------------
8 ['I']
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
------------------------------------------------------------------------------------
8 ['H']
HEURISTIC VALUES  : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
```

```
12 ['B', 'C']
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH    : {}
PROCESSING NODE   : I
------------------------------------------------------------------------------------
0 []
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH    : {'I': []}
PROCESSING NODE   : G
------------------------------------------------------------------------------------
1 ['I']
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH    : {'I': [], 'G': ['I']}
PROCESSING NODE   : B
------------------------------------------------------------------------------------
2 ['G']
HEURISTIC VALUES  : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : C
------------------------------------------------------------------------------------
2 ['J']
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : J
------------------------------------------------------------------------------------
0 []
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE   : C
------------------------------------------------------------------------------------
1 ['J']
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE   : A
------------------------------------------------------------------------------------
5 ['B', 'C']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
```

1. EXPERIMENT NO: 3
2. TITLE: **CANDIDATE-ELIMINATION ALGORITHM**
3. LEARNING OBJECTIVES:
   a. Make use of Data sets in implementing the machine learning algorithms.
   b. Implement ML concepts and algorithms in Python

4. AIM: 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.

5. THEORY:
   a. The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.
   b. It computes the description of this set without explicitly enumerating all of its members.
   c. This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.
   d. The algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.
   e. A version space can be represented with its general and specific boundary sets.
   f. The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.
   g. Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses. Every member of the version space lies between these boundaries

Algorithm

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H

3. For each training example d, do
    3.1. 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, hypothesis that is more general than another hypothesis in S
    3.2. 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

6. **PROCEDURE / PROGRAMME :**

```
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('finds.csv'))
# Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])
# Isolating target into a separate DataFrame
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h=[0,0,0,0,0,0,0]
    print ('s0',specific_h)
    specific_h = concepts[0].copy()
    print('s1',specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print('g0',general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                # Change values in S & G only if values change
```

```
            if h[x] != specific_h[x]:
                specific_h[x] = '?'
                general_h[x][x] = '?'
                print(f"s{x}",specific_h)
                print(f"g{x}",general_h)

    if target[i] == "No":
        for x in range(len(specific_h)):

    # For negative hyposthesis change values only  in G
            if h[x] != specific_h[x]:
                general_h[x][x] = specific_h[x]
    else:
        general_h[x][x] = '?'
    # find indices where we have empty rows, meaning those that are unchanged
    indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?', '?']]
    for i in indices:
      # remove those rows from general_h
      general_h.remove(['?', '?', '?', '?', '?', '?', '?'])
      print('i',indices)
    # Return final values
  return specific_h,general_h
s_final, g_final = learn(concepts, target)
```

Data sets Used for this code:

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:

```
s0 [0, 0, 0, 0, 0, 0, 0]
s1 ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
g0 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
g2 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
g2 [['?', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
s4 ['Sunny' 'Warm' '?' 'Strong' '?' 'Same']
g4 [['?', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
s5 ['Sunny' 'Warm' '?' 'Strong' '?' '?']
g5 [['?', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

7. LEARNING OUTCOMES :
   - The students will be able to apply candidate elimination algorithm and output a description of the set of all hypotheses consistent with the training examples

8. APPLICATION AREAS:
   - Classification based problems.

1. EXPERIMENT NO: 4
2. TITLE: **ID3 ALGORITHM**
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python
4. AIM:
   - **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. THEORY:
   - ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?".
   - To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.
   - A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute).
   - The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
   - A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described below.

Algorithm: ID3(Examples, TargetAttribute, Attributes) Input: Examples are the training examples. Targetattribute 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.

Output: Returns a decision tree that correctly classiJies the given Examples Method:

1. Create a Root node for the tree

2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,
   Return the single-node tree Root, with label = most common value of TargetAttribute in Examples
   Else
   A ← the attribute from Attributes that best classifies Examples The decision attribute for Root ←A
   For each possible value, vi, of A,
   Add a new tree branch below Root, corresponding to the test A = vi Let Examplesvi be the subset of Examples that have value vi for A
   If Examplesvi is empty Then below this new branch add a leaf node with label = most common value of TargetAttribute in Examples
   Else
   below this new branch add the subtree ID3(Examplesvi, TargetAttribute, Attributes–{A})
   End

5. Return Root

6. PROCEDURE / PROGRAMME:

Dataset set used – playtennis.csv

outlook	temperature	humidity	wind	play
Sunny	Hot	high	Weak	no
Sunny	Hot	high	strong	no
Overcast	Hot	high	Weak	yes
rainy	mild	high	Weak	yes
rainy	Cool	Normal	Weak	yes
rainy	Cool	Normal	strong	no
Overcast	Cool	Normal	strong	yes
Sunny	mild	high	Weak	no
Sunny	Cool	Normal	Weak	yes
rainy	mild	Normal	Weak	yes
Sunny	mild	Normal	strong	yes
Overcast	mild	high	strong	yes
Overcast	Hot	Normal	Weak	yes
rainy	mild	high	strong	no

```
import csv
import math
import random

# Class Node which will be used while classify a test-instance using the tree which was built
#earlier
```

```
class Node():
    value = ""
    children = []

    def __init__(self, val, dictionary):
        self.value = val
        if (isinstance(dictionary, dict)):
            self.children = dictionary.keys()

 # Majority Function which tells which class has more entries in given data-set
def majorClass(attributes, data, target):

    freq = {}
    index = attributes.index(target)
    for tuple in data:
        if tuple[index] in freq:
            freq[tuple[index]] += 1
        else:
            freq[tuple[index]] = 1



    max = 0
    major = ""
    for key in freq.keys():
        if freq[key]>max:
            max = freq[key]
            major = key
    return major

 # Calculates the entropy of the data given the target attribute
def entropy(attributes, data, targetAttr):

    freq = {}
    dataEntropy = 0.0
    i = 0
    for entry in attributes:
        if (targetAttr == entry):
            break
        i = i + 1
    i = i - 1
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]]  = 1.0
```

```
    for freq in freq.values():
        dataEntropy += (-freq/len(data)) * math.log(freq/len(data), 2)

    return dataEntropy

# Calculates the information gain (reduction in entropy) in the data when a particular #attribute is chosen for splitting the data.
def info_gain(attributes, data, attr, targetAttr):

    freq = {}
    subsetEntropy = 0.0
    i = attributes.index(attr)

    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]]  = 1.0

    for val in freq.keys():
        valProb        = freq[val] / sum(freq.values())
        dataSubset     = [entry for entry in data if entry[i] == val]

        subsetEntropy += valProb * entropy(attributes, dataSubset, targetAttr)

    return (entropy(attributes, data, targetAttr) - subsetEntropy)

# This function chooses the attribute among the remaining attributes which has the #maximum #information gain.
def attr_choose(data, attributes, target):

    best = attributes[0]
    maxGain = 0;

    for attr in attributes:
        newGain = info_gain(attributes, data, attr, target)
        if newGain>maxGain:
            maxGain = newGain
            best = attr

    return best

# This function will get unique values for that particular attribute from the given data
def get_values(data, attributes, attr):
```

```
    index = attributes.index(attr)
    values = []

    for entry in data:
        if entry[index] not in values:
            values.append(entry[index])

    return values

# This function will get all the rows of the data where the chosen "best" attribute has a #value "val"
def get_data(data, attributes, best, val):

    new_data = [[]]
    index = attributes.index(best)

    for entry in data:
        if (entry[index] == val):
            newEntry = []
            for i in range(0,len(entry)):
                if(i != index):
                    newEntry.append(entry[i])
            new_data.append(newEntry)

    new_data.remove([])
    return new_data

# This function is used to build the decision tree using the given data, attributes and the #target
attributes. It returns the decision tree in the end.
def build_tree(data, attributes, target):

    data = data[:]
    vals = [record[attributes.index(target)] for record in data]
    default = majorClass(attributes, data, target)

    if not data or (len(attributes) - 1) <= 0:
        return default
    elif vals.count(vals[0]) == len(vals):
        return vals[0]
    else:
        best = attr_choose(data, attributes, target)
        tree = {best:{}}
        for val in get_values(data, attributes, best):
            new_data = get_data(data, attributes, best, val)
            newAttr = attributes[:]
            newAttr.remove(best)
```

```
            subtree = build_tree(new_data, newAttr, target)
            tree[best][val] = subtree

    return tree

#Main function
def execute_decision_tree():

    data = []

    #load file
    with open("playtennis.csv") as tsv:
        for line in csv.reader(tsv):
            data.append(tuple(line))
        print("Number of records:",len(data))

        #set attributes
        attributes=['outlook','temperature','humidity','wind','play']
        target = attributes[-1]

        #set training data
        acc = []
        training_set = [x for i, x in enumerate(data)]
        tree = build_tree( training_set, attributes, target )

        #execute algorithm on test data
        results = []
        test_set = [('rainy','mild','high','strong')]
        for entry in test_set:
            tempDict = tree.copy()
            result = ""

            while(isinstance(tempDict, dict)):
                root = Node(next(iter(tempDict)), tempDict[next(iter(tempDict))])
                tempDict = tempDict[next(iter(tempDict))]
                index = attributes.index(root.value)
                value = entry[index]
                if(value in tempDict.keys()):
                    child = Node(value, tempDict[value])
                    result = tempDict[value]
                    tempDict = tempDict[value]
                else:
                    result = "Null"
                    break
            if result != "Null":
```

```
            results.append(result == entry[-1])
        print(result)


if __name__ == "__main__":
    execute_decision_tree()
```

OUTPUT:

INPUT	OUTPUT
for the input Rain | Mild | High | Strong	**Output 1:** **Number of records: 15** **No**
for the input Rain | Cool | Normal | Weak	Output 2: Number of records: 15 Yes
for the input Overcast | Hot | Normal | strong	Output 3: Number of records: 15 Null

7. LEARNING OUTCOMES :

- The student will be able 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.

8. APPLICATION AREAS:

- Classification related problem areas

1. EXPERIMENT NO: 5
2. TITLE: **BACKPROPAGATION ALGORITHM**
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python
4. AIM:
   - Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
5. THEORY:
   - Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples.
   - Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs.
   - ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies.

Backpropogation algorithm

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do
   For each training example <(x1,…xn),t>, do
   // Propagate the input forward through the network:
   a. Input the instance (x1, ..,xn) to the n/w & compute the n/w outputs ok for every unit // Propagate the errors backward through the network:
   b. For each output unit k, calculate its error term k ; k = ok(1-ok)(tk-ok)
   c. For each hidden unit h, calculate its error term h; h=oh(1-oh) k wh,k kj xi,j

6. PROCEDURE / PROGRAMME :

```
import matplotlib.pyplot as plt
import numpy as np
import time

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

start = time.time()  # start time

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

# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Variable initialization
epoch = 700  # 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)
    hiddengrad = derivatives_sigmoid(hlayer_act)
# how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) * lr  # dotproduct of nextlayererror and currentlayerop

    # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) * lr
    # bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
    end = time.time()

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n", output)
print("Elapsed time is {}".format(end - start))
```

OUTPUT:

```
Input:

[[0.66666667 1.       ]

 [0.33333333 0.55555556]

 [1.         0.66666667]]

Actual Output:

[[0.92]

 [0.86]

 [0.89]]

Predicted Output:

 [[0.89434998]

 [0.87931195]

 [0.89600298]]
```

7. LEARNING OUTCOMES :

- The student will be able to build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

8. APPLICATION AREAS:

- Speech recognition, Character recognition, Human Face recognition

1. EXPERIMENT NO: 6

2. TITLE: **NAIVE BAYESIAN CLASSIFIER**

3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:

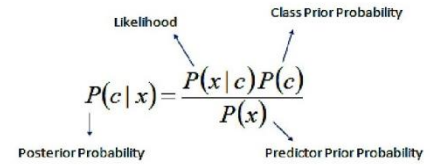
- Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

5. THEORY:

Naive Bayes algorithm: Naive Bayes algorithm 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'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

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:



$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

where

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes). P(c) is the prior probability of class.

P(x|c) is the likelihood which is the probability of predictor given class. P(x) is the prior probability of predictor.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (a1, a2, ... ,an). The learner is asked to predict the target value, or classification, for this new instance.

6. PROCEDURE / PROGRAMME :

Pima Indians Diabetes Database

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies(P): Number of times pregnant
- Glucose(G): Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure(BP): Diastolic blood pressure (mm Hg)
- SkinThickness(ST): Triceps skin fold thickness (mm)
- Insulin(I): 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction(DPF): Diabetes pedigree function
- Age(A): Age (years)
- Outcome(O): Class variable (0 or 1)

Pregnancies (P)	Glucose (G)	Blood Pressure (BP)	Skin Thickness (ST)	Insulin (I)	BMI	Diabetes Pedigree Function (DPF)	Age (A)	Outcome (O)
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1

10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0

```
# Example of Naive Bayes implemented from Scratch in Python
# Online Resource: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

import csv
import random
import math

def loadCsv(filename):
	lines = csv.reader(open(filename, "r"))
	dataset = list(lines)
	for i in range(len(dataset)):
		dataset[i] = [float(x) for x in dataset[i]]
	return dataset

def splitDataset(dataset, splitRatio):
	trainSize = int(len(dataset) * splitRatio)
	trainSet = []
	copy = list(dataset)
	while len(trainSet) < trainSize:
		index = random.randrange(len(copy))
		trainSet.append(copy.pop(index))
	return [trainSet, copy]

def separateByClass(dataset):
	separated = {}
	for i in range(len(dataset)):
		vector = dataset[i]
		if (vector[-1] not in separated):
			separated[vector[-1]] = []
		separated[vector[-1]].append(vector)
	return separated

def mean(numbers):
	return sum(numbers)/float(len(numbers))

def stdev(numbers):
	avg = mean(numbers)
	variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
	return math.sqrt(variance)
```

```
def summarize(dataset):
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
        del summaries[-1]
        return summaries

def summarizeByClass(dataset):
        separated = separateByClass(dataset)
        summaries = {}
        for classValue, instances in separated.items():
                summaries[classValue] = summarize(instances)
        return summaries

def calculateProbability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
        probabilities = {}
        for classValue, classSummaries in summaries.items():
                probabilities[classValue] = 1
                for i in range(len(classSummaries)):
                        mean, stdev = classSummaries[i]
                        x = inputVector[i]
                        probabilities[classValue] *= calculateProbability(x, mean, stdev)
        return probabilities

def predict(summaries, inputVector):
        probabilities = calculateClassProbabilities(summaries, inputVector)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
                if bestLabel is None or probability > bestProb:
                        bestProb = probability
                        bestLabel = classValue
        return bestLabel

def getPredictions(summaries, testSet):
        predictions = []
        for i in range(len(testSet)):
                result = predict(summaries, testSet[i])
                predictions.append(result)
        return predictions

def getAccuracy(testSet, predictions):
        correct = 0
        for i in range(len(testSet)):
```

```
            if testSet[i][-1] == predictions[i]:
                correct += 1
    return (float(correct)/float(len(testSet))) * 100.0

def main():
    filename = 'NaiveBayesDiabetes.csv'
    dataset = loadCsv(filename)
    trainingSet=dataset
    testSet=loadCsv('NaiveBayesDiabetes1.csv')
    print('Records in training data={1} and test data={2} rows'.format(len(dataset),
len(trainingSet), len(testSet)))
    # prepare model
    summaries = summarizeByClass(trainingSet)
    # test model
    predictions = getPredictions(summaries, testSet)
    print(predictions)
    accuracy = getAccuracy(testSet, predictions)
    print("Accuracy:",accuracy,"%")

main()
```

OUTPUT:

```
Records in training data=768 and test data=11 rows
[1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0]
Accuracy: 72.72727272727273 %
```

7. LEARNING OUTCOMES :

- The student will be able to apply naive Bayesian classifier for the relevant problem and analyze the results.

8. APPLICATION AREAS:

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

1. EXPERIMENT NO: 7
2. TITLE: **CLUSTERING BASED ON EM ALGORITHM AND K-MEANS**
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

4. AIM: 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.

5. THEORY:

Expectation Maximization algorithm

- The basic approach and logic of this clustering method is as follows.
- Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution.
- The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution).
- Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering.
- The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities.

- In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.

K means Clustering

- The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.
- The algorithm works as follows:
    1. First we initialize k points, called means, randomly.
    2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
    3. We repeat the process for a given number of iterations and at the end, we have our clusters.
- The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).

- Pseudocode:
    1. Initialize k means with random values
    2. For a given number of iterations:
       Iterate through items:

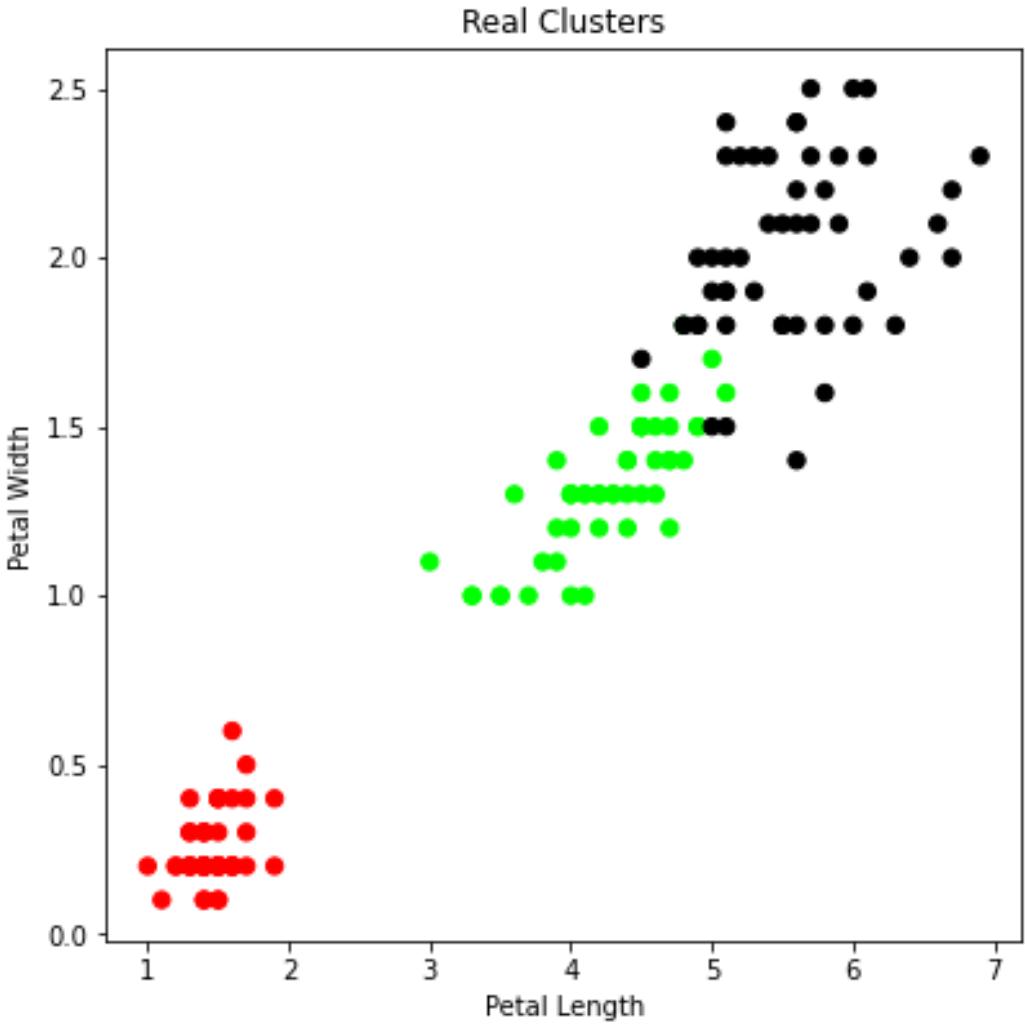
    Find the mean closest to the item Assign item to mean
    Update mean

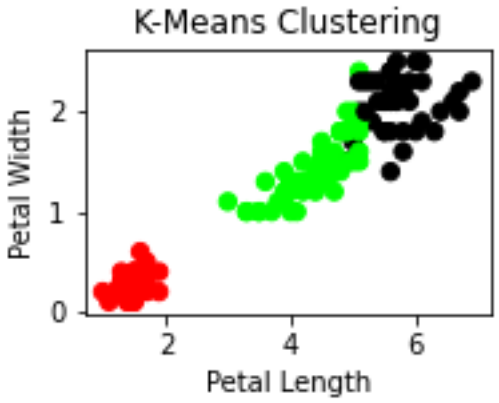
6. PROCEDURE / PROGRAMME :

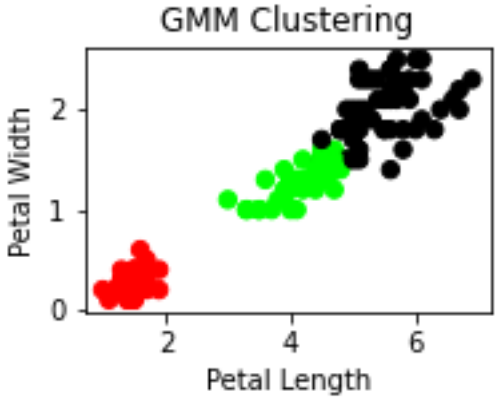
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
# model.labels_ : Gives cluster no for which samples belongs to
```

```
model.fit(X)

# Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()

# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()



# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
print('Observation: The GMM using EM algorithm based clustering matched the true labels more
closely than the Kmeans.')
```

OUTPUT:


Real Clusters
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


K-Means Clustering
Petal Width
Petal Length
0
1
2
2
4
6


GMM Clustering
Petal Width
Petal Length
0
1
2
2
4
6

```
Observation: The GMM using EM algorithm based clustering matched the true
labels more closely than the Kmeans.
```

7. LEARNING OUTCOMES :
   - The students will be apble to apply EM algorithm and k-Means algorithm for clustering and anayse the results.

8. APPLICATION AREAS:
   - Text mining
   - Pattern recognition
   - Image analysis
   - Web cluster engines

1. EXPERIMENT NO: 8
2. TITLE: **K-NEAREST NEIGHBOUR**
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

4. AIM:
   - 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.

5. THEORY:
   - K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
   - It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.
   - Algorithm
     Input: Let m be the number of training data samples. Let p be an unknown point.
     Method:
     1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).

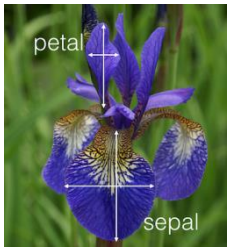
2. for i=0 to m
   Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
4. Return the majority label among S.

6. PROCEDURE / PROGRAMME :

Dataset: (150 Rows)

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
1	5.1	3.5	1.4	0.2	*I. setosa*
2	4.9	3.0	1.4	0.2	*I. setosa*
3	4.7	3.2	1.3	0.2	*I. setosa*
98	6.2	2.9	4.3	1.3	*I. versicolor*
99	5.1	2.5	3.0	1.1	*I. versicolor*
100	5.7	2.8	4.1	1.3	*I. versicolor*
101	6.3	3.3	6.0	2.5	*I. virginica*
102	5.8	2.7	5.1	1.9	*I. virginica*
103	7.1	3.0	5.9	2.1	*I. virginica*

Iris flower sample image




```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")

# Split the data into train and test samples
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)

# Prints Label no. and their names
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))
# Create object of KNN classifier
classifier = KNeighborsClassifier(n_neighbors=1)

# Perform Training
classifier.fit(x_train, y_train)

# Perform testing
```

```
y_pred=classifier.predict(x_test)

# Display the results
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));

from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT:

Iris Data set loaded...
Dataset is split into training and testing...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 – virginica

Results of Classification using K-nn with K=1

```
Sample: [5.7  2.9  4.2  1.3]  Actual-label:  1  Predicted-label:  1
Sample: [6.4  2.7  5.3  1.9]  Actual-label:  2  Predicted-label:  2
Sample: [6.1  2.9  4.7  1.4]  Actual-label:  1  Predicted-label:  1
Sample: [5.2  3.4  1.4  0.2]  Actual-label:  0  Predicted-label:  0
Sample: [5.   3.4  1.5  0.2]  Actual-label:  0  Predicted-label:  0
Sample: [6.2  2.2  4.5  1.5]  Actual-label:  1  Predicted-label:  1
Sample: [5.   2.3  3.3  1  ]  Actual-label:  1  Predicted-label:  1
Sample: [5.   3.2  1.2  0.2]  Actual-label:  0  Predicted-label:  0
Sample: [6.3  3.3  6    2.5]  Actual-label:  2  Predicted-label:  2
Sample: [6.7  2.5  5.8  1.8]  Actual-label:  2  Predicted-label:  2
Sample: [5.9  3.2  4.8  1.8]  Actual-label:  1  Predicted-label:  2
Sample: [5.7  2.8  4.1  1.3]  Actual-label:  1  Predicted-label:  1
Sample: [5.1  3.8  1.5  0.3]  Actual-label:  0  Predicted-label:  0
Sample: [5.1  2.5  3    1.1]  Actual-label:  1  Predicted-label:  1
Sample: [6.8  3.2  5.9  2.3]  Actual-label:  2  Predicted-label:  2
```

Classification Accuracy : 0.9333333333333333
Confusion Matrix
[[4 0 0]

```
[0 6 1]
[0 0 4]]
```

Accuracy Metrics

	precision	recall	f1-score	support
0	1	1	1	4
1	1	0.86	0.92	7
2	0.8	1	0.89	4
accuracy			0.93	15
macro avg	0.93	0.95	0.94	15
weighted avg	0.95	0.93	0.93	15

7. LEARNING OUTCOMES :
   - The student will be able to implement k-Nearest Neighbour algorithm to classify the iris data set and Print both correct and wrong predictions.

8. APPLICATION AREAS:
   - Recommender systems
   - Classification problems

1. EXPERIMENT NO: 9
2. TITLE: **LOCALLY WEIGHTED REGRESSION ALGORITHM**
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

4. AIM:
   - Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

5. THEORY:
   - Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations.
   - Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead.
   - Alternate approach is given by locally weighted regression.
   - Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors.
   - The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel.
   - The solution is very similar to Normal equations, we only need to insert diagonal

weight matrix W.

6. PROCEDURE / PROGRAMME : Dataset: (244 Rows)

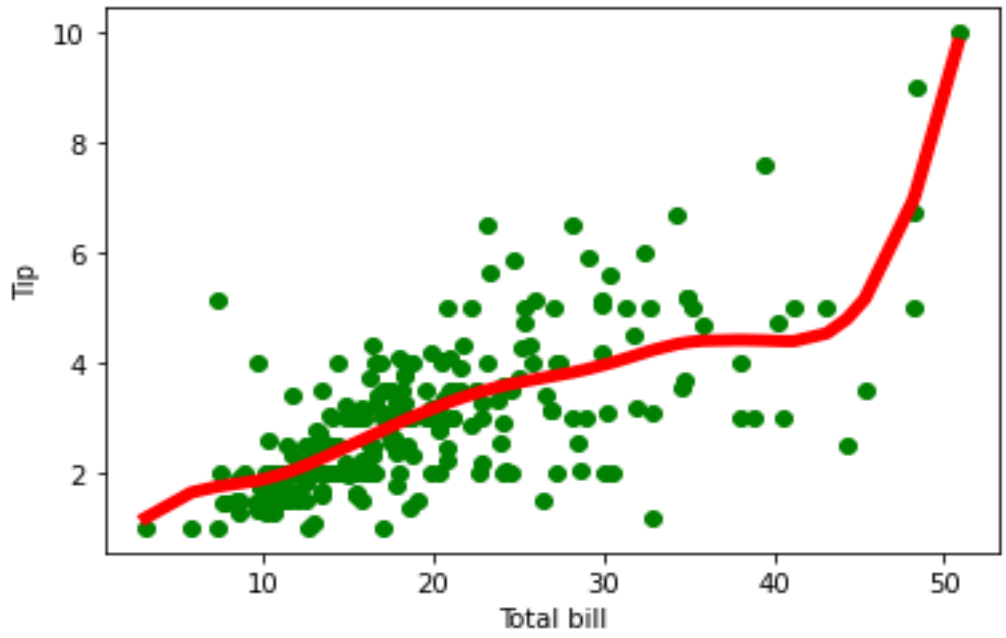
total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.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 = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
```

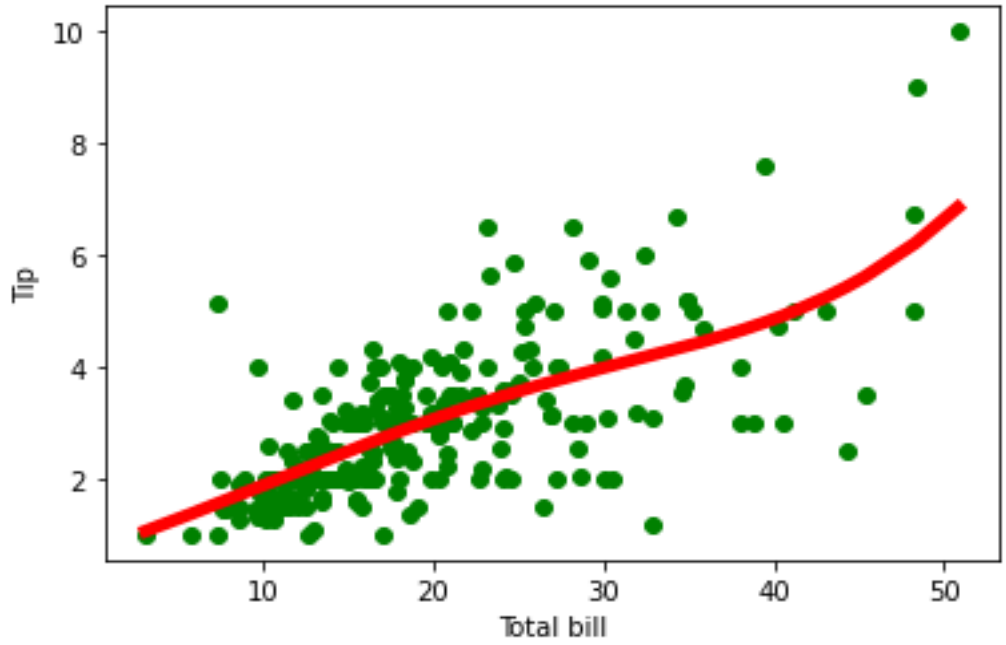
```
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    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();


# load data points
data = pd.read_csv('data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)
mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols
# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

OUTPUT:

Regression with parameter k = 3



Regression with parameter k = 9



7. LEARNING OUTCOMES :
   - To understand and implement linear regression and analyse the results with change in the parameters
8. APPLICATION AREAS:
   - Demand anaysis in business
   - Optimization of business processes
   - Forecasting

ML viva Questions

1. What's the trade-off between bias and variance?
2. What is the difference between supervised and unsupervised machine learning?
3. How is KNN different from k-means clustering?
4. Define precision and recall.
5. What is Bayes' Theorem? How is it useful in a machine learning context?
6. Why is "Naive" Bayes naive?
7. What's the difference between probability and likelihood?
8. How is a decision tree pruned?
9. What's the F1 score? How would you use it?
10. When should you use classification over regression?

11. How do you ensure you're not over fitting with a model?
12. How would you evaluate a logistic regression model?
13. How do you handle missing or corrupted data in a dataset?
14. **How is True Positive Rate and Recall related? Write the equation.**
15. **What is the difference between supervised and unsupervised machine learning?**
16. Comparison between Machine Learning and Big Data
17. **What is deep learning?**
18. Compare kmeans and EM algorithm
19. Compare find s and candidate elimination algorithm
20. Compare candidate elimination algorithm with decision tree
21. Define numpy
22. Define pandas
23. What is matplotlib? Why it is used?
24. What is tensor flow
25. Define data frame
26. What is hypothesis
27. What is perceptron
28. What is back propagation
29. What is ANN
30. What is feed forward network
31. Define multilayer perceptron
32. Define concept learning.