VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANA SANGAMA, BELGAVI-590018, KARNATAKA

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
LABORATORY 18CSL76

LAB MANUAL

Prepared By:
Mrs. Bhagyashri Shelke
Associate Professor Dept of CSE, RGIT

RAJIV GANDHI INSTITUTE OF TECHNOLOGY
CHOLA NAGAR, BENGALURU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Prerequisites

1. Programming experience in Python
2. Knowledge of basic Machine Learning Algorithms
3. Knowledge of common statistical methods and data analysis best practices.

Lab outcomes:

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

1. Implement and demonstrate AI and ML algorithms.
2. Evaluate different algorithms

Software Requirements

1. Python version 3.5 and above
2. Machine Learning packages
   - Scikit-Learn
   - Numpy - matrices and linear algebra
   - Scipy - many numerical routines
   - Matplotlib- creating plots of data
   - Pandas –facilitates structured/tabular data manipulation and visualisations
   - Pomegranate –for fast and flexible probabilistic models
3. An Integrated Development Environment (IDE) for Python Programming

Anaconda

It contains a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions. Anaconda is a new distribution of the Python and R data science package. It was formerly known as Continuum Analytics. Anaconda has more than 100 new packages. Anaconda is used for scientific computing, data science, statistical analysis, and machine learning

Operating System

Windows/Linux/Ubuntu

Anaconda Python distribution is compatible with Linux or windows.

Python

It is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including objectoriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is open source, multi-paradigm, Object-oriented and structured programming supported, Language. Python uses dynamic typing, and a

combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution, which binds method and variable names during program execution. Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions.

Python packages for machine learning

NumPy

NumPy (stands for Numerical Python) is one of the most famous and commonly used python package among data scientists and ML engineers. This is a part of Python's SciPy Stack, which is basically a collection of software specially designed for scientific computations. It provides several features to work with n-dimensional arrays and matrices in python. This library provides vectorization of mathematical operations on the NumPy array type which adds up to the performance of the execution.

Pandas

The Pandas library is too a well-known library in the world of Analytics and Data Sciences. This package is primarily designed to work with simple and relational data. This is one of the favorite libraries among the data scientists for easy data manipulation, visualization as well as aggregation.

If talking about the data structures, there are basically two prime data structures available in the library which are Series (one-dimensional) & Data Frames (two-dimensional) and we think these are not that significant to talk about as of now.

The basic functionalities that Pandas provides:

- We can very easily delete as well as add a columns from DataFrame Pandas can be used to convert the Data Structures in to DataFrame objects.
- If we have any redundancy in the dataset in the form of missing data represented as 'NaN', this is the perfect tool to remove that
- Can be used for grouping of the attributes based strictly on their functionality.

Libraries for Machine Learning

- Scikit-Learn: The library is focused on modelling data. Some popular groups of models provided by scikit-learn include:
- Clustering: for grouping unlabelled data such as KMeans.
- Cross Validation: for estimating the performance of supervised models on unseen data.

- Datasets: for test datasets and for generating datasets with specific properties for investigating model behaviour.
- Dimensionality Reduction: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- Ensemble methods: for combining the predictions of multiple supervised models. Feature extraction: for defining attributes in image and text data.
- Feature selection: for identifying meaningful attributes from which to create supervised models.
- Parameter Tuning: for getting the most out of supervised models.
- Manifold Learning: For summarizing and depicting complex multidimensional data.
- Supervised Models: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

TensorFlow

TensorFlow is an open source software library for numerical computation using dataflow graphs. Nodes in the graph represents mathematical operations, while graph edges represent multi dimensional data arrays (aka tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API

Best Python Machine Learning IDEs

Spyder is coming to our very first focus i.e. Spyder. This IDE got this short name from it's name itself: "Scientific Python Development Environment". Pierre Raybaut is the author of Spyder and it got officially released on October 18, 2009 and is written solely in Python.

Features at a glance:

- Very simple and light-weight IDE with detailed documentation and quite easy to install.
- This is an open source editor and supports code completion, introspection, goto definition as well as horizontal and vertical splitting.

- This editor comes with a Documentation Viewer where you can see the documentation related to classes or functions you gotta use.
- Like most of the IDEs, this also supports Variable Explorer which is a helpful tool to explore and edit the variables that we have created during file execution.
- It supports runtime debugging i.e. the errors will be seen on the screen as soon as you type them.

Geany is primarily a Python machine learning IDE authored by Enrico Troger and got officially released on October 19, 2005. It has been written in C & C++ and is a light-weight IDE. Despite of being a small IDE it is as capable as any other IDE present out there.

Features at a glance:

- Geany's editor supports highlighting of the Syntax and line numbering.
- It comes equipped with the features like code completion, auto closing of braces, auto HTML and XML tags closing.
- It also comes with code folding.
- This IDE supports code navigation.

Rodeo: This is special we got here. This is a Python IDE that primarily focuses and built for the purpose of machine learning and data science. This particular IDE use IPython kernel (you will know this later) and was authored by Yhat.

Features at a glance

- It is mainly famous due to its ability to let users explore, compare and interact with the data frames & plots. • Like Geany's editor this also comes with a editor that has capability of auto-completion, syntax highlighting.
- This also provides a support for IPython making the code writing fast.
- Also Rodeo comes with Python tutorials integrated within which makes it quite favourable for the users.
- This IDE is well known for the fact that for the data scientists and engineers who work on RStudio IDE can very easily adapt to it PyCharm is the IDE which is most famous in the professional world whether it is for data science or for conventional Python programming. This IDE is built by one of the big company out there that we all might have heard about: Jetbrains, company released the official version of PyCharm in October 2010.

PyCharm comes in two different editions: Community Edition which we all can have access to essentially for free and second one is the Professional Edition for which you will need to pay some bucks.

Features at a glance

- It includes code completion, auto-indentation and code formatting.

- This also comes with runtime debugger i.e. will display the errors as soon as you type them.
- It contains PEP-8 that enables writing neat codes.
- It consist of debugger for Javascript and Python with a GUI.
- It has one of the most advanced documentation viewer along with video tutorials.
- PyCharm being accepted widely among big companies for the purpose of Machine Learning is due to its ability to provide support for important libraries like Matplotlib, NumPy and Pandas.

JuPyter Notebook or IPython Notebook

It is simple and this became a sensational IDE among the data enthusiasts as it is the descendant of IPython. Best thing about JuPyter is that there you can very easily switch between the different versions of python (or any other language) according to your preference.

Features at a glance

- It's an open source platform
- It can support up to 40 different languages to work on including languages beneficial for data sciences like R, Python, Julia, etc.
- It supports sharing live codes, and even documents with equations and visualizations.
- In JuPyter you can produce outputs in the form of images, videos and even LaTex with the help of several useful widgets.

Contents:

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

Program 1

Implement A* Search algorithm.

Objective	Finds the shortest path through a search space to goal state using heuristic function.
Dataset	Heuristic values
Description	Finds the shortest path through a search space to goal state using heuristic function. It requires the heuristic function to evaluate the cost of path that passes through the particular state .

```
#!/usr/bin/env
python #
coding: utf-8

# In[1]:

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.rem
        ove(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(sart _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':
       10,
       'B': 8,
       'C': 5,
       'D': 7,
       'E': 3,
       'F': 6,
```

```
            'G': 5,
            'H': 3,
            'I': 1,
            'J': 0
        }
        return H_dist[n]

#Describe your
graph here
Graph_nodes = {
'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)],

}
aStarAlgo('A', 'J')
```

Output:

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

Viva Questions

1. What are or graphs?
2. Give advantage of A* algorithm
3. Define heuristic function?
4. Disadvantages
5. What are open and closed lists?
6. Explain working of A*.

Program 2

Implement AO* Search algorithm

Objective	Finds the shortest path through a search space to goal state using heuristic function.
Dataset	Heuristic values
Description	It is an informed search and works as best first search. AO* is based on problem decomposition. It represents and AND-OR graph algorithm that is used to find more than one solution. It is an efficient methos to explore the solution path.

```
class Graph:   def init (self, graph,
  heuristicNodeList,
startNode):  #instantiate graph object with graph
topology, heuristic  values, start node

    self.graph =   graph
    self.H=heuristicN
    odeList
    self.start=startN
    ode   self.parent={}
    self.status={}
    self.solutionGrap
    h={}

  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[minimu
    mCost]=[] 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.getHeuristicNodeVa
        lue(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]=nodeLi
st # set  the Minimum Cost child node/s
        flag=False
      else: # checking the Minimum Cost nodes
with the current Minimum
        Cost   if
        minimumCost>c  ost:
        minimumCost=c
          ost
          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   self.solutionGraph)
    print("PROCESSING NODE :", v)
    print("HEURISTIC VALUES :", self.H)
    print("SOLUTION GRAPH :",
  print("_")
```

```
    if self.getStatus(v) >= 0: # if status node v
      >= 0,  compute Minimum Cost nodes of
      v minimumCost,   childNodeList =
      self.computeMinimumCostChildNodes(v)
      self.setHeuristicNodeValue(v, minimumCost)
      self.setStatus(v,len(childNodeList))

      solved=True  #  check  the  Minimum  Cost
      nodes  of v are solved for childNode in
      childNodeList:  self.parent[childNode]=v
        if
          self.getStatus(child
          Node)!=-1:
          solved=solved &   False

      if solved==True: # if the Minimum Cost nodes
of v are  solved, set the current node status as
solved(-1)                      self.setStatus(v,-1)
self.solutionGraph[v]=childNodeList # update
the  solution graph with the solved nodes which may
be a part of  solution

      if v!=self.start: # check the current node is
the  start node for backtracking the current node
value  self.aoStar(self.parent[v], True) #
backtracking the  current node value with
backtracking status set to true

      if backTracking==False: # check the
        current  call is not for backtracking
        for childNode in
        childNodeList: # for each Minimum Cost child
        node
          self.setStatus(childNode,0) # set the status
of child  node to 0(needs exploration)
self.aoStar(childNode,  False) # Minimum Cost child
node is further explored  with backtracking status as
false
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 5,  'H': 7, 'I': 7, 'J': 1, 'T': 3}
graph1 = {
  'A': [[('B', 1), ('C', 1)], [('D', 1)]],
  'B': [[('G', 1)], [('H', 1)]],
  'C': [[('J', 1)]],
  'D': [[('E', 1), ('F', 1)]],
  'G': [[('I', 1)]]
}
G1= Graph(graph1,   h1,
'A')   G1.applyAOStar()
G1.printSolution()
```

```
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F':
4, 'G': 5,  'H': 7} # Heuristic values of Nodes
graph2 = { # Graph of Nodes and Edges  'A':
[[('B', 1), ('C', 1)], [('D', 1)]], # Neighbors
of  Node 'A', B, C & D with repective weights
  'B': [[('G', 1)], [('H', 1)]], # Neighbors are included
  in a list  of lists
   'D': [[('E', 1), ('F', 1)]] # Each sublist indicate a
  "OR"  node or "AND" nodes
 }
G2 = Graph(graph2, h2, 'A') # Instantiate Graph
object with  graph, heuristic values and start Node
G2.applyAOStar() # Run the AO* algorithm
G2.printSolution() # Print the solution graph as
output of the AO* algorithm search
```

Output:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE

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

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

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

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

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

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

PROCESSING NODE : D

HEUISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : A

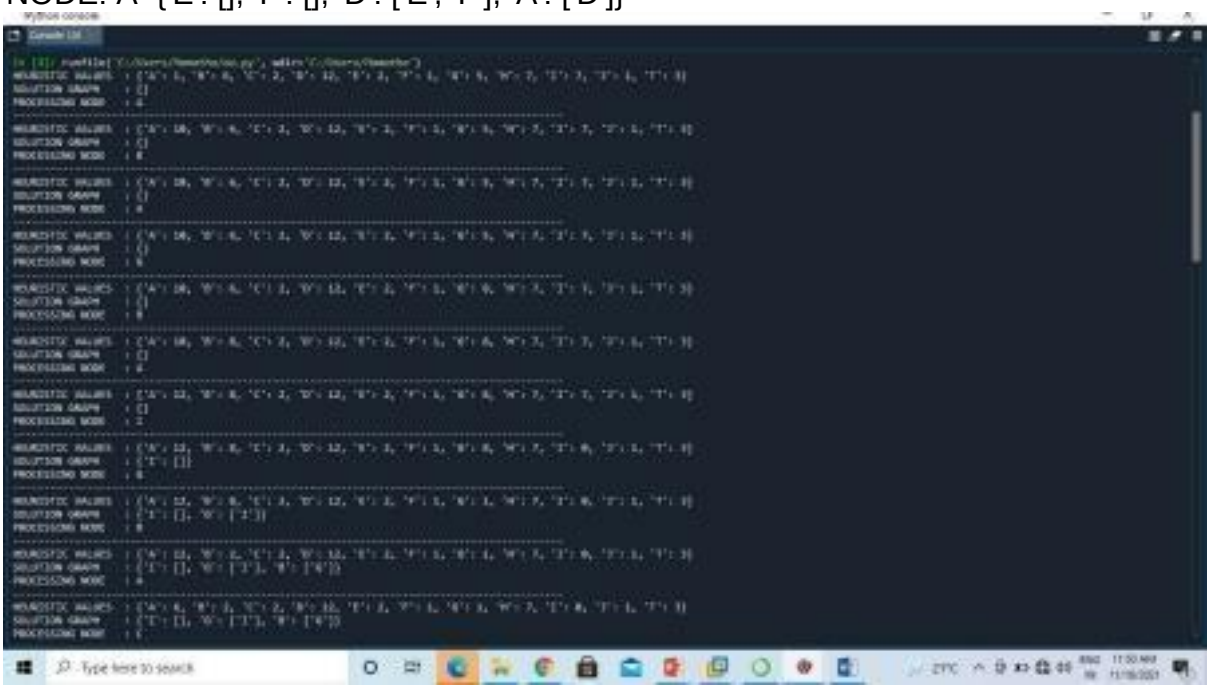
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : F

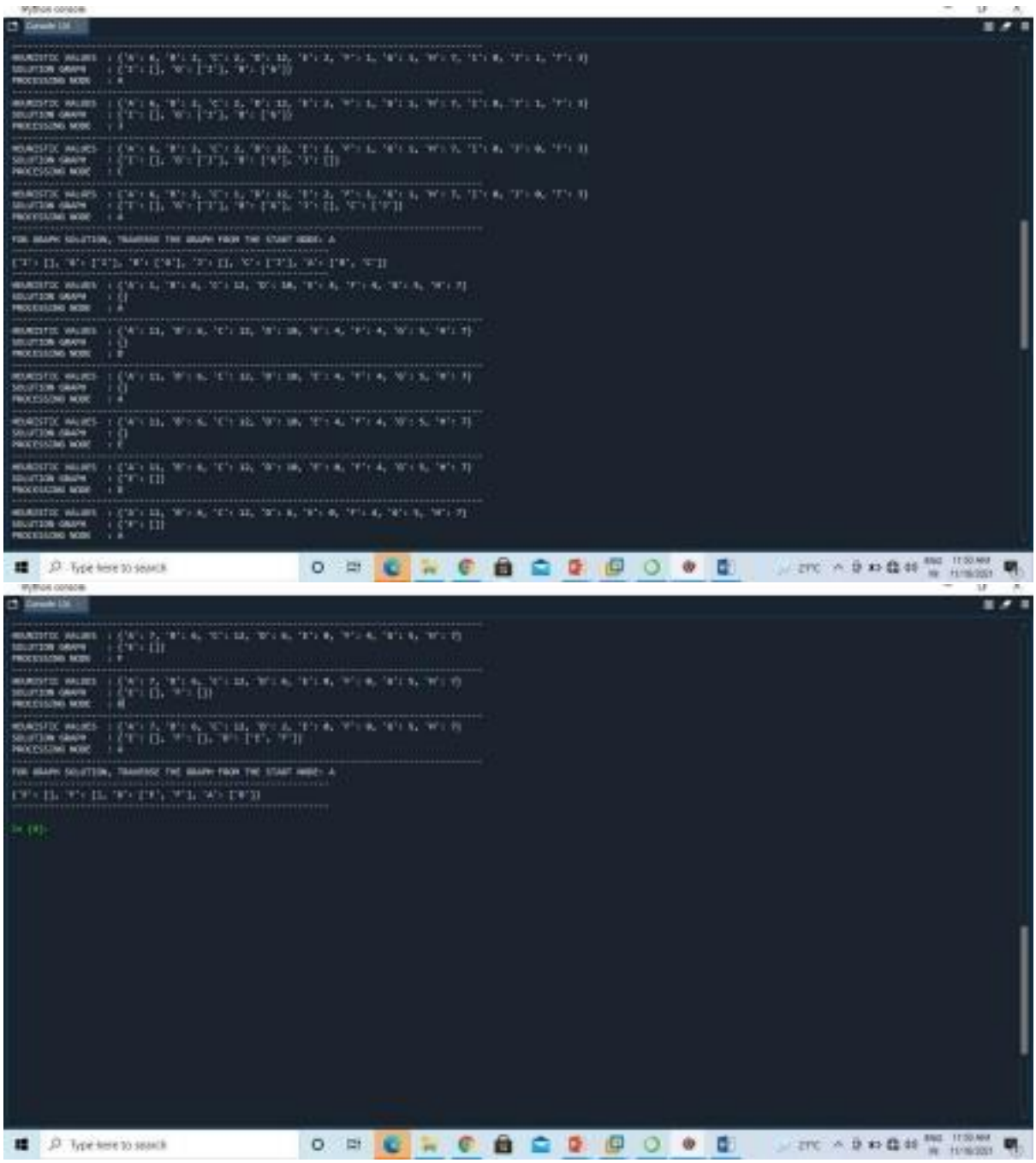
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE : A

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START

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





Viva Questions

1. Define AND-OR graphs.
2. Give advantages of AO* algorithm.
3. What are disadvantages of AO* algorithm.
4. Explain working of AO*.
5. Mention applications of it.

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.

Objective	Incrementally builds the version space given a hypothesis space H and a set E of examples.
Dataset	Tennis data set: This data set contain the set of example days on which playing of tennis is possible or not. Based on attribute Sky, AirTemp, Humidity, Wind, Water and Forecast.The dataset has 4 instances.
ML algorithm	Supervised Learning-Candidate elimination algorithm
Description	The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. ... The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

Program

```
import csv   a
= []
print("\n The Given Training Data Set \n")

with open('ws.csv', 'r') as csvFile:
 reader = csv.reader(csvFile)
for row in reader:
 a.append (row)
print(row)
num_attributes = len(a[0])-1

print("\n The initial value of hypothesis: ")
S = ['0'] * num_attributes
G = ['?'] * num_attributes   print ("\n The
most specific hypothesis S0 :
[0,0,0,0,0,0]\n")  print (" \n The most
general hypothesis G0 : [?,?,?,?,?,?]\n")

# Comparing with First Training Example:wq
for j in range(0,num_attributes):    S[j]
= a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

print("\n Candidate Elimination algorithm Hypotheses Version
Space Computation\n")  temp=[]

for i in range(0,len(a)):    print("----------------------------------
----------------------------
```

```
-------------- ----")    if

a[i][num_attributes]=='Yes':

for j in

range(0,num_attributes):    if

a[i][j]!=S[j]:    S[j]='?'


 for j in range(0,num_attributes):    for
k in range(1,len(temp)):    if
temp[k][j]!= '?' and temp[k][j] !=S[j]:
del temp[k]
```



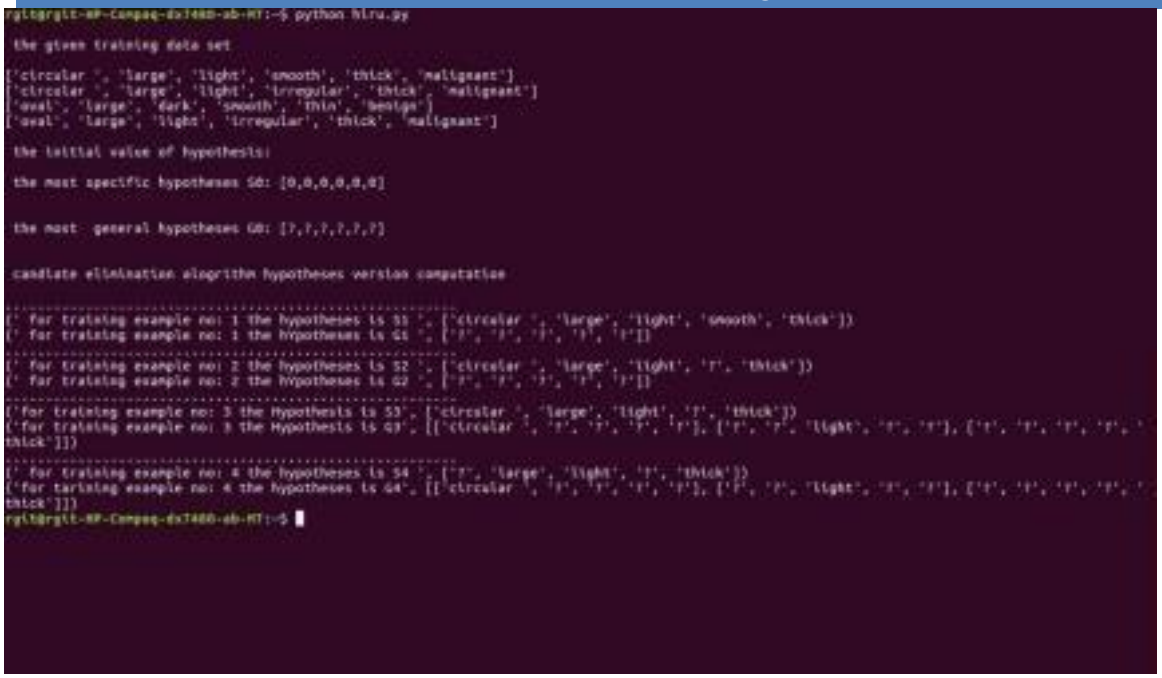
```
  print(" For Training Example
No :{0} the hypothesis is S{0}
".format(i+1),S)   if (len(temp)==0):    print(" For
Training Example No :{0} the hypothesis is G{0}
".format(i+1),G)   else:    print(" For Training
Example No :{0} the hypothesis is
G{0}".format(i+1),temp)

 if a[i][num_attributes]=='No':
for j in range(0,num_attributes):
if S[j] != a[i][j] and S[j]!= '?':
G[j]=S[j]    temp.append(G)
 G = ['?'] * num_attributes

 print(" For Training Example No :{0} the hypothesis is S{0}
".format(i+1),S)   print(" For Training Example No :{0} the
hypothesis is G{0}".format(i+1),temp)
```

Displaying 2.py. DATASET

Enjoy Sport Training Examples

Example	Sky	AirTemp	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	Warm	Change	YES

OUTPUT



Viva Questions

1. Define Hypothesis.
2. What is training data?
3. Give advantages of Candidate elimination.
4. Give disadvantages of Candidate elimination.
5. Explain the CE.

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.

Objective	To demonstrate the working of the decision tree based ID3 algorithm.
Dataset	Tennis data set: This data set contain the set of example days on which playing of tennis is possible or not. Based on attribute Sky, AirTemp, Humidity, Wind, Water and Forecast.
ML algorithm	Supervised Learning-Decision Tree algorithm
Description	Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

```
Program:
import math   import csv   def
load_csv(filename):    lines =
csv.reader(open(filename, "r"));
dataset = list(lines)    headers =
dataset.pop(0)    return dataset,
headers   class Node:    def
__init__(self, attribute):
 self.attribute = attribute
self.children = []
self.answer = ""
 # NULL indicates children exists.    #
Not Null indicates this is a Leaf Node
def subtables(data, col, delete):
 dic = {}
 coldata = [ row[col] for row in data]    attr =
list(set(coldata)) # All values of attribute
retrived   counts = [0]*len(attr)    r = len(data)
c = len(data[0])    for x in range(len(attr)):
 for y in range(r):    if
data[y][col] == attr[x]:
 counts[x] += 1    for x in range(len(attr)):    dic[attr[x]] =
[[0 for i in range(c)] for j in range(counts[x])]   pos = 0
for y in range(r):    if data[y][col] == attr[x]:    if delete:
 del data[y][col]
dic[attr[x]][pos] = data[y]   pos +=
1    return attr, dic
def entropy(S):
 attr = list(set(S))
 if len(attr) == 1: #if all are +ve/-ve then entropy =
```

```
0   return 0
 counts = [0,0] # Only two values possible 'yes' or 'no'
for i in range(2):
 counts[i] = sum( [1 for x in S if attr[i] == x] ) / (len(S) *
1.0)    sums = 0
for cnt in counts:
 sums += -1 * cnt * math.log(cnt, 2)
return sums
def compute_gain(data, col):
 attValues, dic = subtables(data, col, delete=False)
total_entropy = entropy([row[-1] for row in data])    for x
in range(len(attValues)):    ratio = len(dic[attValues[x]]) /
( len(data) * 1.0)   entro = entropy([row[-1] for row in
dic[attValues[x]]])   total_entropy -= ratio*entro    return
total_entropy
def build_tree(data, features):
 lastcol = [row[-1] for row in data]
 if (len(set(lastcol))) == 1: # If all samples have same labels
return  that label
 node=Node("")
node.answer = lastcol[0]
return node    n =
len(data[0])-1    gains
= [0]*n    for col in
range(n):
 gains[col] = compute_gain(data, col)
 split = gains.index(max(gains)) # Find max gains and returns
index   node = Node(features[split]) # 'node' stores
attribute selected   #del (features[split])    fea =
features[:split]+features[split+1:]
 attr, dic = subtables(data, split, delete=True) # Data will be
spilt in  subtables
 for x in range(len(attr)):
 child = build_tree(dic[attr[x]], fea)
node.children.append((attr[x], child))    return node
def print_tree(node, level):    if node.answer != "":
print(" "*level, node.answer) # Displays leaf node yes/no
return
 print(" "*level, node.attribute) # Displays attribute
Name   for value, n in node.children:
 print(" "*(level+1), value)
print_tree(n, level + 2)   def
classify(node,x_test,features):
if node.answer != "":
print(node.answer)    return
 pos = features.index(node.attribute)    for
value, n in node.children:    if
x_test[pos]==value:
classify(n,x_test,features)   ''' Main program
'''   dataset, features = load_csv("data3.csv") #
```

```
Read Tennis data  node = build_tree(dataset,
features)
# Build decision tree   print("The decision tree
for the dataset using ID3 algorithm is ")
print_tree(node, 0)   testdata, features =
load_csv("data3_test.csv")   for xtest in
testdata:
 print("The test instance : ",xtest)
print("The predicted label : ")

   classify(node,xtest,features)
```

DATASET

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

OUTPUT

```
rgit@rgit-HP-Compaq-dx7480-ab-MT:~$ python c3.py
The decision tree for the dataset using ID3 algorithm is
('', 'Outlook')
(' ', 'overcast')
('  ', 'yes')
(' ', 'sunny')
('  ', 'Humidity')
('   ', 'high')
('    ', 'no')
('   ', 'normal')
('    ', 'yes')
(' ', 'rain')
('  ', 'Wind')
('   ', 'strong')
('    ', 'no')
('   ', 'weak')
('    ', 'yes')
('The test instance : ', ['sunny', 'cool', 'normal', 'weak'])
The predicted label :
yes
('The test instance : ', ['rain', 'hot', 'normal', 'strong'])
The predicted label :
no
rgit@rgit-HP-Compaq-dx7480-ab-MT:~$
```


Viva Questions

1. What are decision trees?
2. What is the use of entropy in DT
3. Define Information gain.
4. State its applications.
5. Explain the DTL algorithm..

Program 5

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

Objective	To build an artificial neural network using the backpropagation algorithm.
Dataset	Data stored as a list having two features- number of hours slept, number of hours studied with the test score being the class label
ML algorithm	Supervised Learning –Backpropagation algorithm
Description	The neural network using back propagation will model a single hidden layer with three inputs and one output. The network will be predicting the score of an exam based on the inputs of number of hours studied and the number of hours slept the day before. The test score is the output.

```
Program:
 import numpy as
np
X = np.array(([2, 9], [1, 5], [3, 6]),
dtype=float)    y = np.array(([92], [86], [89]),
dtype=float)    y = y/100    #Sigmoid Function    def
sigmoid (x):
 return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
return x * (1 - x)    #Variable
initialization
epoch=10000 #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_n
eurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_ne
urons))     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
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

OUTPUT:

```
rgit@rgit-HP-Compaq-dx7480-ab-MT:~$ python c4.py
Input:
[[2. 9.]
 [1. 5.]
 [3. 6.]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
('Predicted Output:\n', array([[0.91615704],
       [0.8773431 ],
       [0.894075  ]]))
rgit@rgit-HP-Compaq-dx7480-ab-MT:~$
```

Viva Questions

1. What are ANNs?
2. What is perceptron?
3. Why are sigmoid functions used?
4. State applications.
5. Explain the Backpropagation algotithm..

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.

Objective	To implement a classification model for a sample training dataset and computing the accuracy of the classifier for test data.
Dataset	Tennis data set: This data set contain the set of example days on which playing of tennis is possible or not. Based on attribute Sky, AirTemp, Humidity, Wind, Water and Forecast.
ML algorithm	Supervised Learning -Naïve Bayes Algorithm
Description	The Naïve Bayes classifier is a probabilistic classifier that is based on Bayes Theorem. The algorithm builds a model assuming that the attributes in the dataset are independent of each other.

Program

```
import numpy as np   import math
import csv   def
read_data(filename):    with
open(filename, 'r') as csvfile:
 datareader =
csv.reader(csvfile)    metadata
= next(datareader)
traindata=[]    for row in
datareader:
traindata.append(row)    return
(metadata, traindata)

def splitDataset(dataset, splitRatio):
 #splits dataset to training set and test set based on
split ratio   trainSize = int(len(dataset) * splitRatio)
trainSet = []    testset = list(dataset)    i=0       while
len(trainSet) < trainSize:
trainSet.append(testset.pop(i))

 return [trainSet, testset]

def classify(data,test):    total_size
= data.shape[0]    print("training data
size=",total_size)    print("test data
sixe=",test.shape[0])
target=np.unique(data[:,-1])
```

```
 count = np.zeros((target.shape[0]), dtype=np.int32)
prob = np.zeros((target.shape[0]), dtype=np.float32)
print("target count probability")


for y in range(target.shape[0]):    for
x in range(data.shape[0]):    if
data[x,data.shape[1]-1] == target[y]:
```



```
count[y] += 1
 prob[y]=count[y]/total_size
 print(target[y], count[y], prob[y]) # comptes the probability of
target
 prob0 = np.zeros((test.shape[1]-1), dtype=np.float32)
prob1 = np.zeros((test.shape[1]-1), dtype=np.float32)
accuracy=0
 print("Instance prediction target")    for t in
range(test.shape[0]):    for k in range(test.shape[1]-1): # for
each attribute in column   count1=count0=0    for j in
range(data.shape[0]):    if test[t,k]== data[j,k] and
data[j,data.shape[1]-1]== target[0]:
 count0+=1
 elif test[t,k]== data[j,k] and
data[j,data.shape[1]-1]== target[1]:
 count1+=1
 prob0[k]= count0/count[0] #Find no probability of each  attribute
prob1[k]= count1/count[1] #Find yes probability of each  attribute

 probno=prob[0]
probyes=prob[1]
    for i in
range(test.shape[1]-1):
 probno=probno*prob0[i]
probyes=probyes*prob1[i]

 if probno>probyes: #
prediction    predict='no'
else:
 predict='yes'

 print(t+1, predict, test[t,test.shape[1]-1])

 if predict== test[t,test.shape[1]-1]:
 accuracy+=1 # computing accuracy

 final_accuracy=(accuracy/test.shape[0])*100
print("accuracy",final_accuracy,"%")
```

```
return
metadata, traindata = read_data("tennis.csv")
splitRatio = 0.6
trainingset, testset = splitDataset(traindata, splitRatio)
training=np.array(trainingset)   testing=np.array(testset)
print("------------------Training Data-------------
--
--
-")  print(trainingset)
print("-------------------Test Data-------------------")
print(testset)
classify(training,testing)
```

DATASET

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

OUTPUT

```
rgit@rgit-HP-Compaq-dx7400-ab-MT:~$ python c5.py
___________Training Data_____________
[['sunny', 'hot', 'high', 'weak', 'no'], ['sunny', 'hot', 'high', 'strong', 'no'], ['overcast', 'hot', 'high', 'weak', 'yes'], ['rainy', 'mil
d', 'high', 'weak', 'yes'], ['rainy', 'cool', 'normal', 'weak', 'yes'], ['rainy', 'cool', 'normal', 'strong', 'no'], ['overcast', 'cool', 'no
rmal', 'strong', 'yes'], ['sunny', 'mild', 'high', 'weak', 'no']]
___________Test Data_____________
[['sunny', 'cool', 'normal', 'weak', 'yes'], ['rainy', 'mild', 'normal', 'weak', 'yes'], ['sunny', 'mild', 'normal', 'strong', 'yes'], ['over
cast', 'mild', 'high', 'strong', 'yes'], ['overcast', 'hot', 'normal', 'weak', 'yes'], ['rainy', 'mild', 'high', 'strong', 'no']]
('training data size=', 8)
('test data size=', 6)
target count probability
('no', 4, 0.0)
('yes', 4, 0.0)
INSTANCE PREDICTION TARGET
(1, 'yes', 'yes')
(2, 'yes', 'yes')
(3, 'yes', 'yes')
(4, 'yes', 'yes')
(5, 'yes', 'yes')
(6, 'yes', 'no')
('accuracy', 83.33333333333334, '%')
rgit@rgit-HP-Compaq-dx7400-ab-MT:~$
```

Viva Questions

1. What is Bayes Theorem?
2. What is test data?
3. What is posteriori hypothesis?
4. Difference between posteriori and priori hypothesis.
5. Explain the this program

Program 7 :

Apply EM algorithm to cluster a set of data stored in a .csv file. Use the same data set for clustering 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.

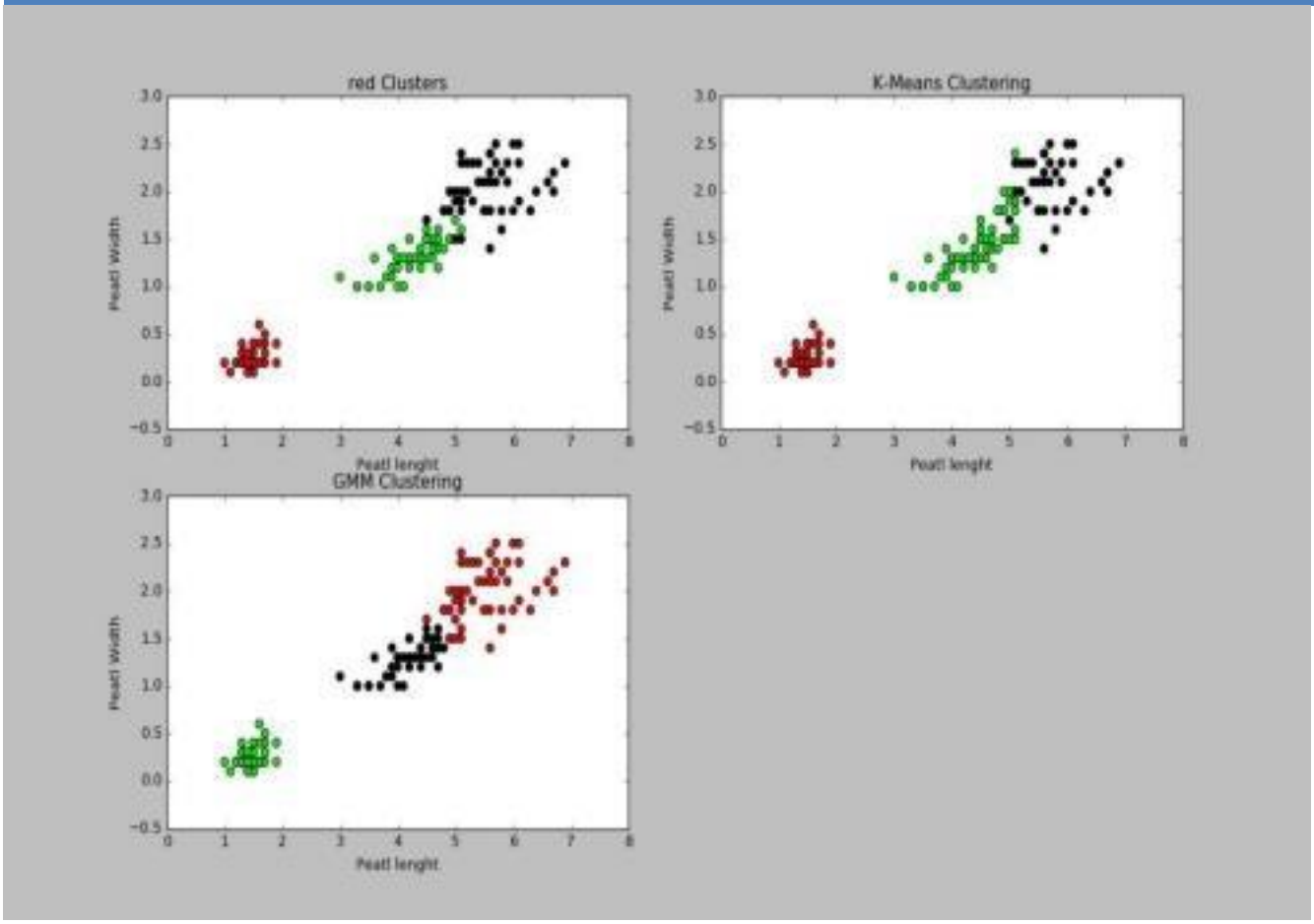
Objective	To group a set of unlabelled data into similar classes/clusters and label them and to compare the quality of algorithm.
Dataset	Iris data present in sklearn package
ML algorithm	EM algorithm, K means algorithm – Unsupervised clustering, GM algorithm
Packages	Scikit learn(sklearn), pandas
Description	EM algorithm – soft clustering - can be used for variable whose value is never directly observed, provided the general probability distribution governing these varaiable is known. EM algorithm can be used to train Bayesian belief networks as well as radial basis function network. K-Means – Hard Clustering - to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

Program

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

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']
model = KMeans(n_clusters=3)
model.fit(X)

plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])

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.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')
from sklearn import preprocessing

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')  print(' true labels more closely than the Kmeans.')
```

OUTPUT:



Viva Questions

1. Define k-means
2. What is unsupervised learning?
3. What are BBN?
4. What is conditional dependency?
5. What is clustering?

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.

Objective	To implement a classification model that classifies a set of documents and calculates the accuracy, precision and recall for the dataset
Dataset	IRIS data set with features “petal_length”, “petal_width”, “sepal_length”, “sepal_width” having more than 150 instances
ML algorithm	Supervised Learning – Lazy learning algorithm
Packages	Scikit learn(sklearn),pandas
Description	When we get training set/instances, machine won’t learn or a model can’t be built. Instead instances/examples will be just stored in memory.Test instance is given, attempt to find the closest instance/most neighboring instances in the instance space….

Program

```
from sklearn.model_selection import
train_test_split from
sklearn.neighbors import
KNeighborsClassifier   from
sklearn.metrics import
classification_report, confusion_matrix #from
sklearn import metrics   #import pandas   as
pd #import   numpy as np   #import
matplotlib.pyplot as   plt from sklearn
import datasets
iris=datasets.load_ir
is()
iris_data=iris.data
iris_labels=iris.targ  et
#print(iris_data)
#print(iris_labels)
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_lab
els,test_size=0.30)
```

```
classifier=KNeighborsClassifier( n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test ) print('Confusion
matrix is as  follows')
print(confusion_matrix(y_test,y_ pred)) print('Accuracy
Metrics')  print(classification_report(y_te st,y_pred))
```

OUTPUT:

Confusion matrix is as follows

[[11 0 0]

[0 9 1]

[0 1 8]]

Accuracy Metrics precision recall f1-

score support

0 1.00 1.00 1.00 11

1 0.90 0.90 0.90 10

2 0.89 0.89 0.89 9 avg / total 0.93

0.93 0.93 30

Viva Questions

1. Define lazy learning
2. State meaning of k-neighbors.
3. What is Euclidean distance?
4. State advantages of it.
5. Explain the working of this.

Program 9

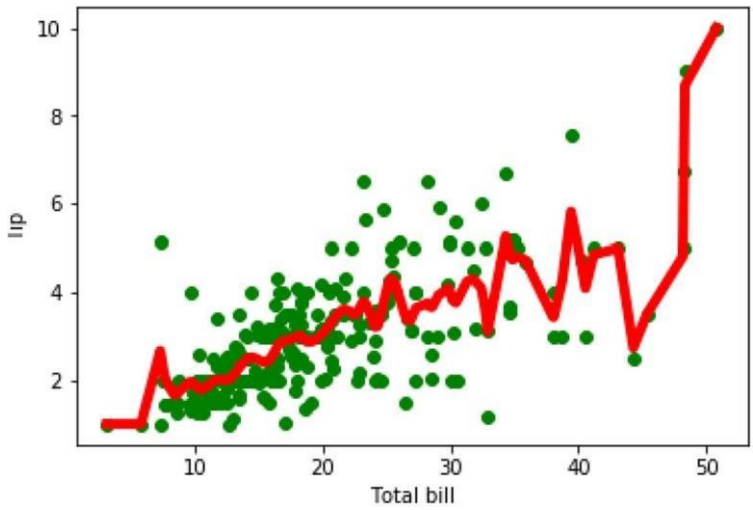
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Objective	To implement Regression algorithm to fit the given data
Dataset	The dataset contains billing information based on the attributes total_bill,tip,sex,smoker,day,time,size
ML algorithm	Locally Weighted Regression Algorithm – Instance Based learning
Description	Regression means approximating a real valued target function.Given a new query instance X_q, the general approach is to construct an approximation function F that fits the training example in the neighbourhood surrounding X_q.This approximation is then used to estimate the target value $F(X_{q)}$

Program

```
 import
matplotlib.pyplot as
plt import pandas as
pd   import numpy as
np1   def
  kernel(point,xm  at,
  k): m,n =
  np1.shape(xmat)
  weights =
  np1.mat(np1.eye((m)))
  for j in range(m):
  diff = point - X[j]
  weights[j,j] =
  np1.exp(diff*diff.T/(-2.0*k**2))
  return weights

def
  localWeight(point,xmat
  ,ymat,k): wei =
  kernel(point,xmat,k)
  B =
   (X.T*(wei*X)).I*(X.T*(wei*ym  at.T))
  return B

def
  localWeightRegression(xmat
  ,ymat,k): m,n =
  np1.shape(xmat)    ypred =
```

```
  np1.zeros(m)   for i in
  range(m):
    ypred[i] =
  xmat[i]*localWeight(xmat[i],xmat,ymat
  ,k) return ypred

# load data points   data
=   pd.read_csv('tips.cs
v') bill =
np1.array(data.total
_bill) tip =
np1.array(data.tip)

#preparing and add   1
in bill mbill =
np1.mat(bill)


mtip = np1.mat(tip)

m=   np1.shape(mbill)
[1] print(m)   one =
np1.mat(np1.ones(m))
X=
np1.hstack((one.T,mbi
ll.T))   #set k here
ypred =
localWeightRegression(X,mtip,
0.3) 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. Give Meaning of regression.
2. Meaning of locally weighted.
3. What is error term in this?
4. Explain the working of this