CAMBRIDGE INSTITUTE OF TECHNOLOGY

KR PURAM, BENGALURU

AIRTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY 18CSL76

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

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 object-oriented, 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 modeling data. Some popular groups of models provided by scikit-learn include:

- Clustering: for grouping unlabeled 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 behavior.
- 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 multi-dimensional 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

<u>Best Python Machine Learning IDEs</u>

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.

<u>Features at a glance:</u>

- 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 numebering.
- 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:

Expt No.	Title of the Experiments	RBT	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

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 .

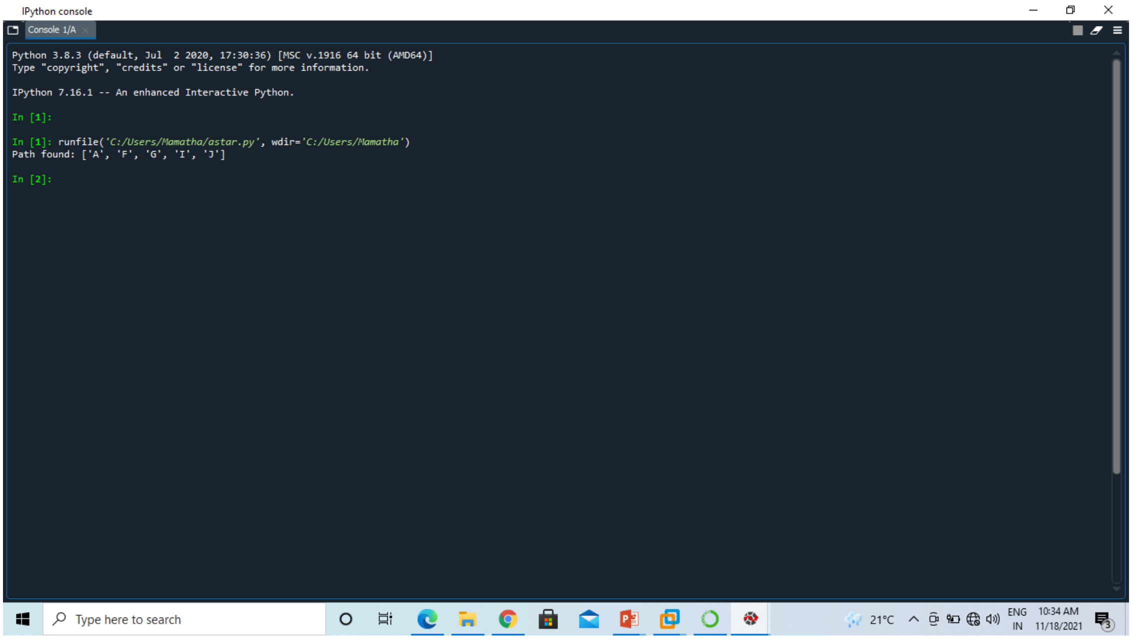
```
#!/usr/bin/env python
# coding: utf-8

# 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.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': 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']



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=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)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))

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

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


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

        if backTracking==False:     # check the current call is not for backtracking
            for childNode in childNodeList:   # for each Minimum Cost child node
                self.setStatus(childNode,0)   # set the status of child node to 0(needs exploration)
                self.aoStar(childNode, False) # Minimum Cost child node is further explored
with backtracking status as false



h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}  # Heuristic values of Nodes
graph2 = {                                  # Graph of Nodes and Edges
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],      # Neighbors of Node 'A', B, C & D with repective
weights
    'B': [[('G', 1)], [('H', 1)]],               # Neighbors are included in a list of lists
    'D': [[('E', 1), ('F', 1)]]                  # Each sublist indicate a "OR" node or "AND" nodes
}

G2 = Graph(graph2, h2, 'A')                    # Instantiate Graph object with graph, heuristic
values and start Node
G2.applyAOStar()                               # Run the AO* algorithm
G2.printSolution()                             # Print the solution graph as 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

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

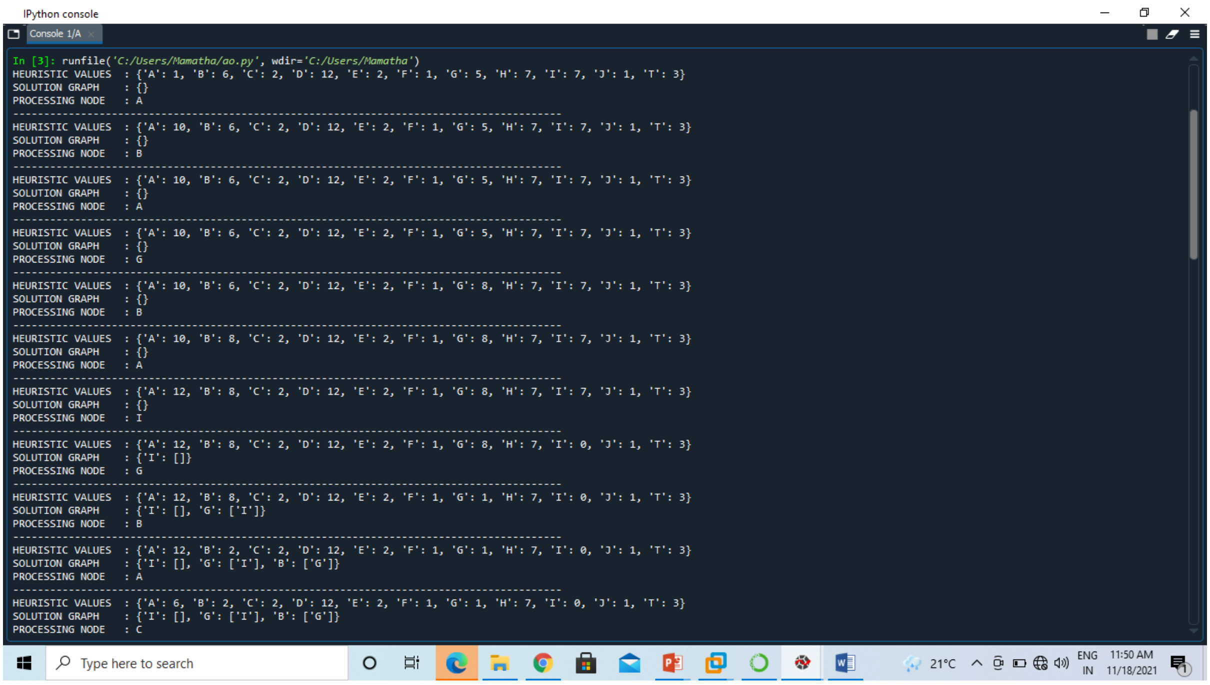
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : F

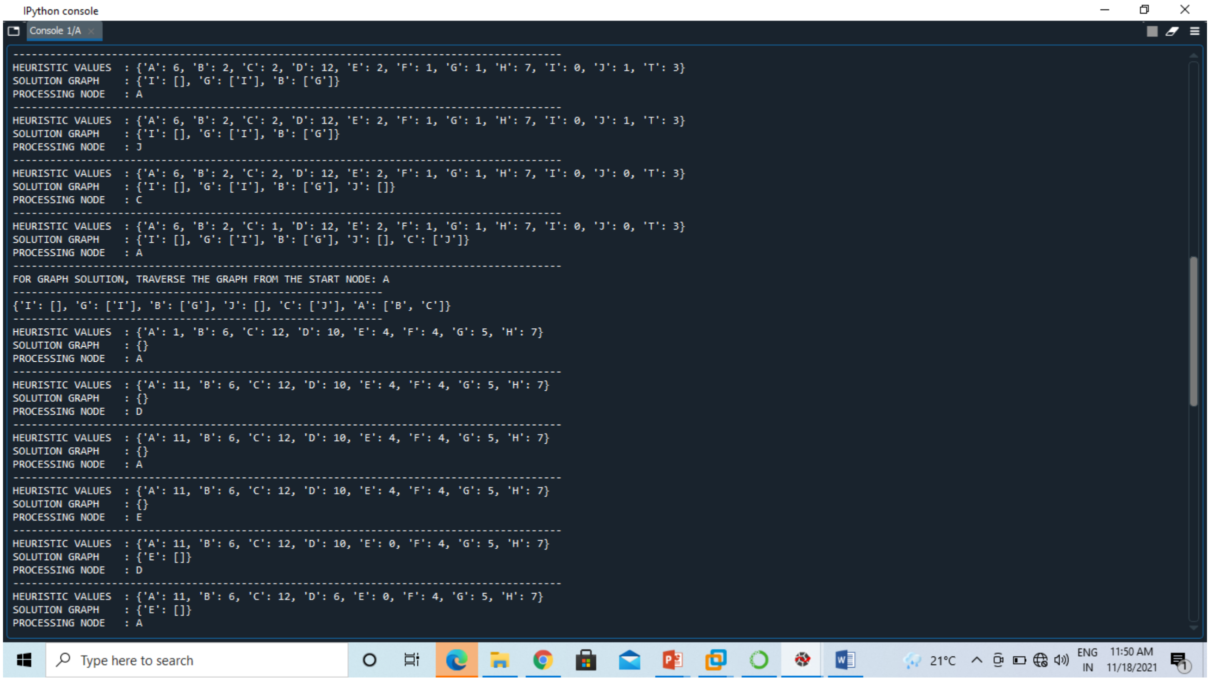
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D

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

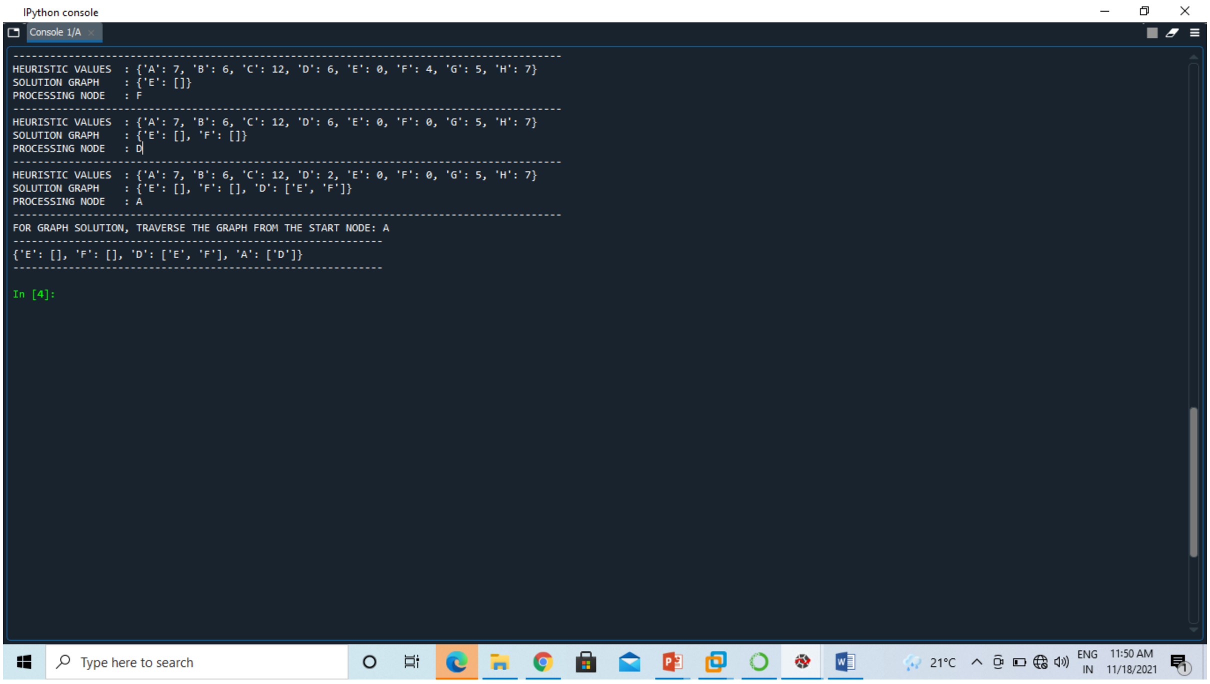
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

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





```
IPython console
Console 1/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   : 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
------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : A
```



```
IPython console
Console 1/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']}
------------------------------------------------------------

In [4]:
```

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 14 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.

```
import csv
def get_domains(examples):
d = [set() for i in examples[0]]
for x in examples:
for i, xi in enumerate(x):
d[i].add(xi)
return [list(sorted(x)) for x in d]
def more_general(h1, h2):
more_general_parts = []
for x, y in zip(h1, h2):
mg = x == "?" or (x != "0" and (x == y or y == "0"))
more_general_parts.append(mg)
return all(more_general_parts)
def fulfills(example, hypothesis):
# the implementation is the same as for hypotheses:
return more_general(hypothesis, example)
def min_generalizations(h, x):
h_new = list(h)
for i in range(len(h)):
if not fulfills(x[i:i+1], h[i:i+1]):
h_new[i] = '?' if h[i] != '0' else x[i]
return [tuple(h_new)]
def min_specializations(h, domains, x):
results = []
for i in range(len(h)):
if h[i] == "?":
for val in domains[i]:
if x[i] != val:
h_new = h[:i] + (val,) + h[i+1:]
results.append(h_new)
elif h[i] != "0":
h_new = h[:i] + ('0',) + h[i+1:]
```

```
results.append(h_new)
return results
def generalize_S(x, G, S):
S_prev = list(S)
for s in S_prev:
if s not in S:
continue
if not fulfills(x, s):
S.remove(s)
Splus = min_generalizations(s, x)
## keep only generalizations that have a counterpart in G
S.update([h for h in Splus if any([more_general(g,h)
for g in G])])
## remove hypotheses less specific than any other in S
S.difference_update([h for h in S if
any([more_general(h, h1)
for h1 in S if h != h1])])
return S
def specialize_G(x, domains, G, S):
G_prev = list(G)
for g in G_prev:
if g not in G:
continue
if fulfills(x, g):
G.remove(g)
Gminus = min_specializations(g, domains, x)
## keep only specializations that have a conuterpart in S
G.update([h for h in Gminus if any([more_general(h, s)
for s in S])])
## remove hypotheses less general than any other in G
G.difference_update([h for h in G if
any([more_general(g1, h)
for g1 in G if h != g1])])
return G
def candidate_elimination(examples):
domains = get_domains(examples)[:-1]
n = len(domains)
G = set([("?",)*n])
S = set([("0",)*n])
print("Maximally specific hypotheses - S ")
print("Maximally general hypotheses - G ")
i=0
print("\nS[0]:",str(S),"\nG[0]:",str(G))
for xcx in examples:
i=i+1
x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions
if cx=='Y': # x is positive example
G = {g for g in G if fulfills(x, g)}
S = generalize_S(x, G, S)
```

```
else: # x is negative example
S = {s for s in S if not fulfills(x, s)}
G = specialize_G(x, domains, G, S)
print("\nS[{0}]:".format(i),S)
print("G[{0}]:".format(i),G)
return
with open('data22_sports.csv') as csvFile:
examples = [tuple(line) for line in csv.reader(csvFile)]
candidate_elimination(examples)
```

Output:

Data: data21_sports.csv (Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport)

sunny,warm,normal,strong,warm,same,Y
sunny,warm,high,strong,warm,same,Y
rainy,cold,high,strong,warm,change,N
sunny,warm,high,strong,cool,change,Y

<u>Output:</u>

Maximally specific hypotheses - S
Maximally general hypotheses – G

S[0]: {('0', '0', '0', '0', '0', '0')}
G[0]: {('?', '?', '?', '?', '?', '?')}
S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}
G[1]: {('?', '?', '?', '?', '?', '?')}
S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[2]: {('?', '?', '?', '?', '?', '?')}
S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[3]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')}
S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}
G[4]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?')}

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.

```
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
for k in attr:
dic[k] = []
for y in range(len(data)):
key = data[y][col]
if delete:
del data[y][col]
dic[key].append(data[y])
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 = [compute_gain(data, col) for col in range(n) ]
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 : ", end="")
classify(node,xtest,features)
```

Output:

Training instances: data3.csv

Outlook,Temperature,Humidity,Wind,Target

sunny,hot,high,weak,no

sunny,hot,high,strong,no

overcast,hot,high,weak,yes

rain,mild,high,weak,yes

rain,cool,normal,weak,yes

rain,cool,normal,strong,no

overcast,cool,normal,strong,yes

sunny,mild,high,weak,no

sunny,cool,normal,weak,yes

rain,mild,normal,weak,yes

sunny,mild,normal,strong,yes

overcast,mild,high,strong,yes

overcast,hot,normal,weak,yes

rain,mild,high,strong,no

Testing instances: data3_test.csv

Outlook,Temperature,Humidity,Wind

rain,cool,normal,strong

sunny,mild,normal,strong

Output:

The decision tree for the dataset using ID3 algorithm is

Outlook

overcast

yes

rain

Wind

weak

yes

strong

no

sunny

Humidity

normal

yes

high

no

The test instance : ['rain', 'cool', 'normal', 'strong']

The predicted label : no

The test instance : ['sunny', 'mild', 'normal', 'strong']

The predicted label : yes

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.

```
import numpy as np
# X = (hours sleeping, hours studying), y = score on test
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
# scale units
X = X/np.amax(X, axis=0) # maximum of X array
y = y/100 # max test score is 100
class Neural_Network(object):
  def __init__(self):
    #parameters
    self.inputSize = 2
    self.outputSize = 1
    self.hiddenSize = 3

    #weights
    self.W1 = np.random.randn(self.inputSize, self.hiddenSize) # (3x2) weight matrix from
input to hidden layer
    self.W2 = np.random.randn(self.hiddenSize, self.outputSize) # (3x1) weight matrix from
hidden to output layer

  def forward(self, X):
    #forward propagation through our network
    self.z = np.dot(X, self.W1) # dot product of X (input) and first set of 3x2 weights
    self.z2 = self.sigmoid(self.z) # activation function
    self.z3 = np.dot(self.z2, self.W2) # dot product of hidden layer (z2) and second set of 3x1
weights
    o = self.sigmoid(self.z3) # final activation function
    return o

  def sigmoid(self, s):
    # activation function
    return 1/(1+np.exp(-s))
```

```
  def sigmoidPrime(self, s):
    #derivative of sigmoid
    return s * (1 - s)

  def backward(self, X, y, o):
    # backward propgate through the network
    self.o_error = y - o # error in output
    self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to error

    self.z2_error = self.o_delta.dot(self.W2.T) # z2 error: how much our hidden layer weights
contributed to output error
    self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid
to z2 error

    self.W1 += X.T.dot(self.z2_delta) # adjusting first set (input --> hidden) weights
    self.W2 += self.z2.T.dot(self.o_delta) # adjusting second set (hidden --> output) weights

  def train (self, X, y):
    o = self.forward(X)
    self.backward(X, y, o)

NN = Neural_Network()
for i in range(10): # trains the NN 1,000 times
  print(i)
  print ("Input: \n" + str(X))
  print ("Actual Output: \n" + str(y))
  print ("Predicted Output: \n" + str(NN.forward(X)))
  print ("Loss: \n" + str(np.mean(np.square(y - NN.forward(X)))) )# mean sum squared loss
  print ("\n")
  NN.train(X, y)
```

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.8930243 ]
 [0.86917478]
 [0.90947008]]
Loss:
0.0003969829742387784
```

Input:
[[0.66666667 1.   ]
 [0.33333333 0.55555556]
 [1.   0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.89325922]
 [0.86906258]
 [0.90922387]]
Loss:
0.00038891893394468784

###output in the last epoch…….

Input:

[[0.66666667 1.   ]

 [0.33333333 0.55555556]

 [1.   0.66666667]]

Actual Output:

[[0.92]

 [0.86]

 [0.89]]

Predicted Output:

[[0.89870965]

 [0.86523337]

 [0.90596392]]

Loss:

0.00024517132775894306

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	Pima Indian Diabetes dataset stored as .CSV .The attributes are Number of times pregnant, Plasma glucose concentration, Blood Pressure, Triceps skin fold thickness, serum insulin, Body mass index, Diabetes pedigree function, Age
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 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 (correct/float(len(testSet))) * 100.0

def main():
	filename = 'naivedata.csv'
	splitRatio = 0.67
	dataset = loadCsv(filename)
	trainingSet, testSet = splitDataset(dataset, splitRatio)
	print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingSet), len(testSet)))
	# prepare model
	summaries = summarizeByClass(trainingSet)
	# test model
	predictions = getPredictions(summaries, testSet)
	accuracy = getAccuracy(testSet, predictions)
	print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

OUTPUT:

Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is : 74.80314960629921%

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.

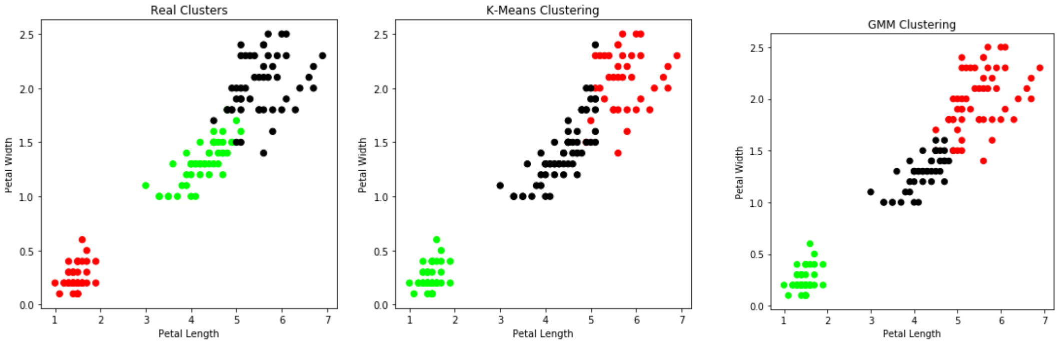
Objective	To group a set of unlabelled data into similar classes/clusters and label them and to compare the quality of algorithm.
Dataset	Delivery fleet driver dataset Data set in .csv file with features “Driver_ID”, “distance_feature”,”speeding_feature” having more than 20 instances
ML algorithm	EM algorithm, K means algorithm – Unsupervised clustering
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
# 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.fit(X) # model.labels_ : Gives cluster no for which samples belongs to
# # 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')
```

```
# 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')
# 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')
print('Observation: The GMM using EM algorithm based clustering matched the true labels
more closely than the Kmeans.')
```

Output:



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.

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….

```
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_iris()
iris_data=iris.data
iris_labels=iris.target
#print(iris_data)
#print(iris_labels)
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,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_test,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

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,xmat, 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*ymat.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.csv')
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,mbill.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: