# ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

**(Effective from the academic year 2022 -2023)**

**SEMESTER – VII** **Course Code 18CSL76**

**CIE Marks** 40 **SEE Marks** 60

**Number of Contact Hours/Week** 0:0:2 **Total Number of Lab Contact Hours** 36

**Exam Hours** 03 **Credits – 2**

**Course Learning Objectives:** This course (18CSL76) will enable students to:
- Implement and evaluate AI and ML algorithms in Python programming language.

**Descriptions (if any):**
Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

## Programs List:

1. Implement A* Search algorithm.
2. Implement AO* Search algorithm.
3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an Appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the Same using appropriate data sets.
6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
8. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print Both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

**Laboratory Course Outcomes**: The student should be able to:

- Implement and demonstrate AI and ML algorithms.
- Evaluate different algorithms.

**Conduct of Practical Examination:**

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

- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Coursed to change in accordance with university regulations)*

- For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
- For laboratories having PART A and PART B

i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks

ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

## Anaconda

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. This is advantageous as when you are working on a data science project, you will find that you need many different packages (numpy, scikit-learn, scipy, pandas to name a few), which an installation of Anaconda comes preinstalled with. If you need additional packages after installing Anaconda, you can use Anaconda's package manager, conda, or pip to install those packages. This is highly advantageous as you don't have to manage dependencies between multiple packages yourself. Conda even makes it easy to switch between Python 2 and 3 (you can learn more here). In fact, an installation of Anaconda is also the recommended way to install Jupyter Notebooks which you can learn more about here on the DataCamp community.

## IDLE

Integrated Development and Learning Environment, IDLE can be used to execute a single statement and create, modify, and execute Python scripts. IDLE provides a fully-featured text editor to create Python scripts that include features like syntax highlighting, autocompletion, and smart indent.

## PROGRAM NUMBER - 1

## Implement A* Search algorithm.

```
def aStarAlgo(start_node, stop_node):              #{A}, len{open_set}=1
        open_set = set(start_node)
        closed_set = set()
        g = {}                      #store distance from starting node
        parents = {}                #parents contains an adjacency map of all nodes
        g[start_node] = 0           #distance of starting node from itself is zero
#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            #parents['A']='A'


        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:
```

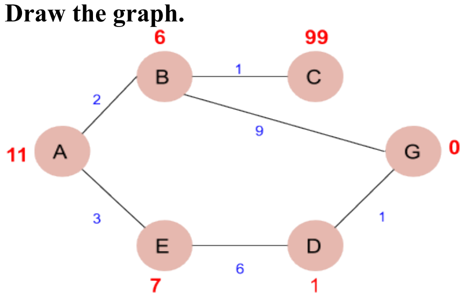
```
                    for (m, weight) in get_neighbors(n):

#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)
#m=A weight=1 {'S','A','G'} len{open_set}=2
                            parents[m] = n          #parents={'A':S, 'G':S} len{parent}=2
                            g[m] = g[n] + weight #g={'S':0,'A':1,'G':10} len{g}=2

#for each node m,compare its distance from start i.e g(m) to the
#from start through n node
                        else:
                            if g[m] > g[n] + weight:
#update g(m)
                                g[m] = g[n] + weight
#change parent of m to n
                                parents[m] = n
#if m in closed set,remove and add to open
                                if m in closed_set:
                                    closed_set.remove(m)
                                    open_set.add(m)
                if n == None:
                    print('Path does not exist!')
                    return None
 # if the current node is the stop_node
 # then we begin reconstructin the path from it to the start_node


                if n == stop_node:
                    path = []
                    while parents[n] != n:
                        path.append(n)
                        n = parents[n]
                    path.append(start_node)
                    path.reverse()
                    print('Path found: {}'.format(path))

                    return path

 # remove n from the open_list, and add it to closed_list
 # because all of his neighbors were inspected
                open_set.remove(n)
                closed_set.add(n)
            print('Path does not exist!')
            return None

#define fuction to return neighbor and its distance
#from the passed node


def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None


#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes


def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]

#Describe your graph here
```

```
Graph_nodes = {
        'A': [('B', 2), ('E', 3)],
        'B': [('C', 1),('G', 9),]
        'C': None,
        'E': [('D', 6)],
        'D': [('G', 1)],
        }
aStarAlgo('A', 'G')
```

## OUTPUT :

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

**Note:**
**Draw the graph.**



# PROGRAM NUMBER – 2

## Implement AO* Search algorithm.

**AO* Algorithm**
AO* Algorithm basically based on problem decomposition (Breakdown problem into small pieces) When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, **AND-OR graphs** or **AND - OR trees** are used for representing the solution.
The decomposition of the problem or problem reduction generates AND arcs.

**AND-OR Graph**

**The figure shows an AND-OR graph**

1. To pass any exam, we have two options, either cheating or hard work.
2. In this graph we are given two choices, first do cheating **or (The red line)** work hard and **(The arc)** pass.
3. When we have more than one choice and we have to pick one, we apply **OR condition** to choose one. (That's what we did here).
   - Basically, the **ARC** here denote **AND condition**.
   - Here we have replicated the arc between the work hard and the pass because by doing the hard work the possibility of passing an exam is more than cheating.

**A* Vs AO***

1. Both are part of informed search technique and use heuristic values to solve the problem.
2. The solution is guaranteed in both algorithms.
3. A* **always** gives an **optimal solution** (shortest path with low cost) But it is not guaranteed to that **AO*** always provide **an optimal solution**.
4. **Reason:** Because AO* does not explore all the solution path once it has solution.

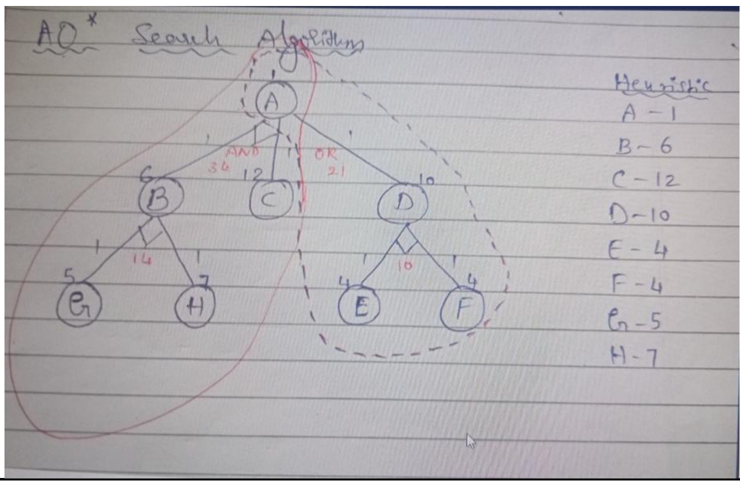
**Program code**

```
class Graph:
        def __init__(self, graph, heuristicNodeList, startNode):
#instantiate graph object with graph topology, heuristic values, start node
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}


  def applyAOStar(self):            # starts a recursive AO* algorithm
    self.aoStar(self.start, False)


  def getNeighbors(self, v):         # gets the Neighbors of a given node
    return self.graph.get(v,'')


  def getStatus(self,v):             # return the status of a given node
    return self.status.get(v,0)


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


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


  def setHeuristicNodeValue(self, n, value):
    self.H[n]=value
# set the revised heuristic value of a given node
```

```
    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE:",self.start)
        print("------------------------------------------------------------")
        print(self.solutionGraph)
        print("------------------------------------------------------------")

#Computes the Minimum Cost of child nodes of a given node v

    def computeMinimumCostChildNodes(self, 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

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

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

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

## Graph:




## Output:

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

```
HEURISTIC VALUES: {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH: {}
PROCESSING NODE: D
-----------------------------------------------------------------------------------------
10 ['E', 'F']
HEURISTIC VALUES: {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH: {}
PROCESSING NODE: A
-----------------------------------------------------------------------------------------
11 ['D']
HEURISTIC VALUES: {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH: {}
PROCESSING NODE: E
-----------------------------------------------------------------------------------------
0 []
HEURISTIC VALUES: {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH: {'E': []}
PROCESSING NODE: D
-----------------------------------------------------------------------------------------
6 ['E', 'F']
HEURISTIC VALUES: {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH: {'E': []}
PROCESSING NODE: A
-----------------------------------------------------------------------------------------
7 ['D']
HEURISTIC VALUES: {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH: {'E': []}
PROCESSING NODE: F
-----------------------------------------------------------------------------------------
0 []
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
-----------------------------------------------------------------------------------------
2 ['E', 'F']
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
-----------------------------------------------------------------------------------------
3 ['D']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------
```





## PROGRAM NUMBER – 3

### (Candidate Elimination Algorithm)

**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.**

NumPy, short for Numerical Python, is one such library. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

**How to install numpy:** https://youtu.be/Dq-GratRgwA?si=EoPm3Vz1HHyNj4Ni

1. Idle should be installed with pip (Check the box -Add python.exe to PATH) ,
2. CMD – pip –version
   pip 22.3.1 from C:\Program Files\Python311\Lib\site-packages\pip (python 3.11)
3. CMD- pip install numpy
   Collecting numpy
   Downloading numpy-1.25.2-cp311-cp311-win_amd64.whl (15.5 MB)
   ---------------------------------------- 15.5/15.5 MB 8.8 MB/s eta 0:00:00
   Installing collected packages: numpy
   Successfully installed numpy-1.25.2
4. IDLE –
   Import numpy as np
   print("numpy version:" +np.__version__)
   Numpy version: 1.24.0
5. CMD – pip install pandas ( https://youtu.be/VZec3iow_2M?si=saNZigMqVSlUbHCi )

**Program code:**

```
import numpy as np
import pandas as pd
data=pd.DataFrame(data=pd.read_csv('C:\Sonia\AI & ML\enjoysport.csv'))
print(data)
concepts=np.array(data.iloc[:,0:-1])          # Except the last column - Separating concept features from Target
print(concepts)
target=np.array(data.iloc[:,-1])          # Isolating target into a separate DataFrame - Requires last column
print(target)

def learn(concepts,target):
''' learn() function implements the learning method of the Candidate elimination algorithm. Arguments: concepts - a data frame with all the features target - a data frame with corresponding output values '''
# Initialise S0 with the first instance from concepts
# .copy() makes sure a new list is created instead of just pointing to the same memory location
  specific_h=concepts[0].copy()                  # copying 1st row into specific
  print("initialization of specific_h and general_h")
  print(specific_h)
  general_h=[["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
  print(general_h)

  # The learning iterations
  for i, h in enumerate(concepts):
# Checking if the hypothesis has a positive target
    if target[i]=="yes":                          #Positive example
      for x in range(len(specific_h)):
# Change values in S & G only if values change
        if h[x]!=specific_h[x]:                   #not equal attributes
          specific_h[x]='?'                      #then place ?
          general_h[x][x]='?'
```

```
# Checking if the hypothesis has a positive target
    if target[i]=="no":
      for x in range(len(specific_h)):
# For negative hyposthesis change values only in G
        if h[x]!=specific_h[x]:
          general_h[x][x]=specific_h[x]
        else:
          general_h[x][x]='?'

    print("steps of candidate Elimination Algorithm",i+1)
    print(specific_h)
    print(general_h)

# find indices where we have empty rows, meaning those that are unchanged
  indices=[i for i,val in enumerate(general_h)if val==['?','?','?','?','?','?']]

for i in indices:
# remove those rows from general_h
    general_h.remove(['?','?','?','?','?','?'])
# Return final values
  return specific_h,general_h

s_final,g_final=learn(concepts,target)
print("\nFinal Specific_h:",s_final,sep="\n")
print("/nFinal General_h:",g_final,sep="\n")
```

**Note: Use enjoysport.csv file**

| 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 | Cool | Change | Yes |

**OUTPUT:**

```
     Sky          airtemp      humidity     wind         water
forecast     enjoysport
0      sunny        warm             normal       strong       warm         same
       yes
1      sunny        warm        high             strong       warm         same
yes
2      rainy        cold             high             strong       warm
change           no
3      sunny        warm        high             strong       cool              change
       yes

[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
['rainy' 'cold' 'high' 'strong' 'warm' 'change']
```

['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
steps of candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
/nFinal General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

## PROGRAM NUMBER – 4 (Decision trees)

**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.**

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("C:\Sonia\AI & ML\Id3.csv")
features = [feat for feat in data]
features.remove("answer")

#Create a class named Node with four members children, value, isLeaf and pred.
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

```
#Define a function called entropy to find the entropy oof the dataset.
def entropy(examples):
  pos = 0.0
  neg = 0.0
  for _, row in examples.iterrows():
    if row["answer"] == "yes":
      pos += 1
    else:
      neg += 1
  if pos == 0.0 or neg == 0.0:
    return 0.0
  else:
    p = pos / (pos + neg)
    n = neg / (pos + neg)
  return -(p * math.log(p, 2) + n * math.log(n, 2))

#Define a function named info_gain to find the gain of the attribute
def info_gain(examples, attr):
  uniq = np.unique(examples[attr])
  #print ("\n",uniq)
  gain = entropy(examples)
  #print ("\n",gain)
  for u in uniq:
    subdata = examples[examples[attr] == u]
    #print ("\n",subdata)
    sub_e = entropy(subdata)
    gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    #print ("\n",gain)
  return gain

#Define a function named ID3 to get the decision tree for the given dataset
def ID3(examples,attrs):
  root = Node()
  max_gain = 0
  max_feat = ""
  for feature in attrs:
    #print ("\n",examples)
    gain=info_gain(examples, feature)
    if gain > max_gain:
      max_gain = gain
      max_feat = feature
  root.value = max_feat
  #print ("\nMax feature attr",max_feat)
  uniq = np.unique(examples[max_feat])
  #print ("\n",uniq)
  for u in uniq:
    #print ("\n",u)
    subdata = examples[examples[max_feat] == u]
    #print ("\n",subdata)
    if entropy(subdata) == 0.0:
      newNode = Node()
      newNode.isLeaf = True
      newNode.value = u
      newNode.pred = np.unique(subdata["answer"])
      root.children.append(newNode)
    else:
      dummyNode = Node()
      dummyNode.value = u
      new_attrs = attrs.copy()
      new_attrs.remove(max_feat)
      child = ID3(subdata, new_attrs)
      dummyNode.children.append(child)
```

```
            root.children.append(dummyNode)
    return root

#Define a function named printTree to draw the decision tree
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

#Define a function named classify to classify the new example
def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)

#Finally, call the ID3, printTree and classify functions

root=ID3(data,features)
print("Decision Tree is:")
printTree(root)
print ("------------------")
```

**Note: Use tennis.csv as 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 :

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

        rain
                wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                humidity
                        high -> ['no']

                        normal -> ['yes']

------------------
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'}
 is: ['yes']
```

## PROGRAM NUMBER – 5 (Back propagation Algorithm)

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

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([.92], [.86], [.89]), dtype=float)
X = X/np.amax(X, axis=0)                    #maximum of X array longitudinally

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

#Derivative of Sigmoid function
def der_sigmoid(x):
  return x * (1 - x)

#Variable initialization
epoch = 5000                        # setting training iterations
lr = 0.01                           # setting learning rate
neurons_i = 2                       #number of features in data set-input
neurons_h = 3                       #number of hidden layers neurons
neurons_o = 1                       #number of neurons at output layer

#weight and bias initialization
weight_h = np.random.uniform(size=(neurons_i, neurons_h))
bias_h = np.random.uniform(size=(1, neurons_h))
weight_o = np.random.uniform(size=(neurons_h, neurons_o))
bias_o = np.random.uniform(size=(1, neurons_o))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
  inp_h = np.dot(X, weight_h) + bias_h
  out_h = sigmoid(inp_h)

  inp_o = np.dot(out_h, weight_o) + bias_o
  out_o = sigmoid(inp_o)

#backpropagation
  err_o = y - out_o
  grad_o = der_sigmoid(out_o)
  delta_o = err_o * grad_o

  err_h = delta_o.dot(weight_o.T)                    #to resolve errors
  grad_h = der_sigmoid(out_h)
  #how much hidden layer wts contributed to error
  delta_h = err_h * grad_h

  weight_o += out_h.T.dot(delta_o) * lr         #dot product of next lauer and currentlayer learning rate
  weight_h += X.T.dot(delta_h) * lr

print('Input: ', X)
print('Actual: ', y)
```

```
print('Predicted: ', out_o)
```

**OUTPUT:**

```
Input: [[ 0.66666667 1.        ]
 [ 0.33333333 0.55555556]
 [ 1.         0.66666667]]
Actual: [[ 0.92]
 [ 0.86]
 [ 0.89]]
Predicted: [[ 0.89371021]
 [ 0.87852765]
 [ 0.89052431]]
```





## PROGRAM NUMBER - 6 (Naive bayes Classifier)

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

```
import 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]]          #converting strings into numbers for processing
  return dataset

def splitdataset(dataset, splitratio):                    #67% training size
```

```
    trainsize = int(len(dataset) * splitratio);
    trainset = []
    copy = list(dataset);

    while len(trainset) <trainsize:
#generate indices for the dataset list randomly to pick ele for training data
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separatebyclass(dataset):
    separated = {}                          #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are the instances belonging to each class

    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):                     #creates a dictionary of classes
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
    del summaries[-1]                       #excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset); #print(separated)
    summaries = {}

    for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
        summaries[classvalue] = summarize(instances)        #summarize is used to cal to mean and std
    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 = {}                      # probabilities contains the all prob of all class of test data
    for classvalue, classsummaries in summaries.items():
#class and attribute information as mean and sd
        probabilities[classvalue] = 1

        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperately
            x = inputvector[i]              #testvector's first attribute
            probabilities[classvalue] *= calculateprobability(x, mean, stdev);
```

```
 #use normal dist
  return probabilities


def predict(summaries, inputvector):                           #training and test data is passed
  probabilities = calculateclassprobabilities(summaries, inputvector)
  bestLabel, bestProb = None, -1

  for classvalue, probability in probabilities.items():      #assigns that class which has the highest prob
      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 = "Naïve-dataset.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); #print(summaries)
# test model
  predictions = getpredictions(summaries, testset)       #find the predictions of test data with the training data
  accuracy = getaccuracy(testset, predictions)
  print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

**NOTE: Use Naïve-dataset.csv as dataset**

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabetic Pedigree Function | Age | Outcome |
|---|---|---|---|---|---|---|---|---|
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |
| 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 10 | 115 | 0 | 0 | 0 | 35.3 | 0.134 | 29 | 0 |
| 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 8 | 125 | 96 | 0 | 0 | 0 | 0.232 | 54 | 1 |

**OUTPUT :**

Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is: 74.01574803149606%
# training set will be
higher then test set

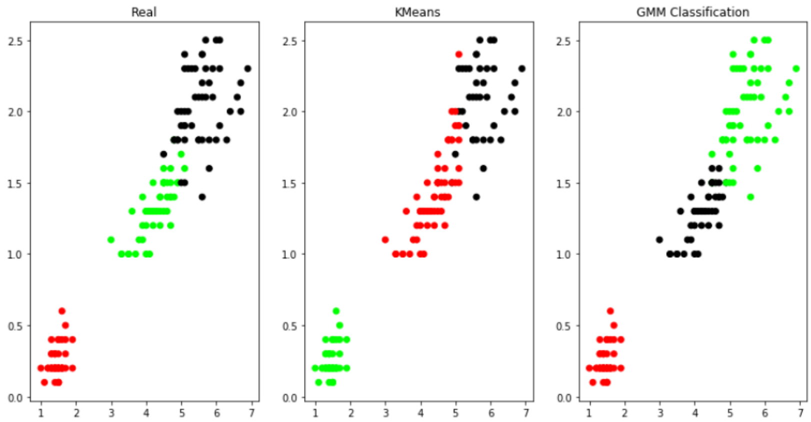
**PROGRAM NUMBER - 7 (K-Means and EM algorithm)**

**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.**

```
CMD- pip install scikit-learn
pip install matplotlib

from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("EM-dataset.csv", names=names)

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

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

**NOTE: Use EM-dataset.csv as dataset**

**OUTPUT:**




The accuracy score of K-Mean: 0.24

The Confusion matrixof K-Mean:

[[ 0 50 0]

[48 0 2]

[14 0 36]]

The accuracy score of EM: 0.3333333333333333

The Confusion matrix of EM:

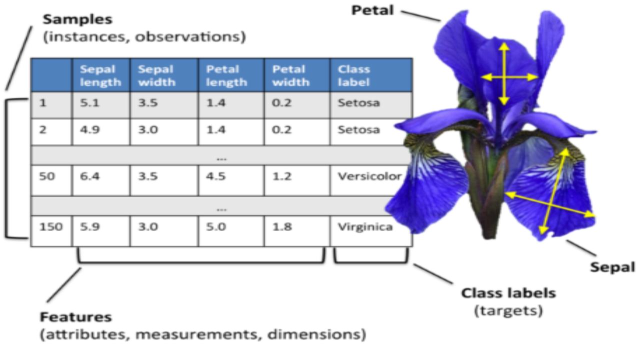
[[ 50 0 0]

[0 45 5]

[ 0 50 0]]

## PROGRAM NUMBER – 8 (KNN)

**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.**


```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("Knearest-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
  print ('%-25s %-25s' % (label, ypred[i]), end="")
  if (label == ypred[i]):
     print (' %-25s' % ('Correct'))
  else:
     print (' %-25s' % ('Wrong'))
  i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-------------------------------------------------------------------------")
```

**NOTE: Use Knearest-dataset.csv as dataset**

**OUTPUT:**

| | sepal-length | sepal-width | petal-length | petal-width |
|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 |

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

| Original Label | Predicted Label | Correct/Wrong |
|---|---|---|
| Iris-versicolor | Iris-versicolor | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-virginica | Iris-versicolor | Wrong |
| Iris-virginica | Iris-virginica | Correct |
| Iris-versicolor | Iris-versicolor | Correct |
| Iris-virginica | Iris-virginica | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-versicolor | Iris-versicolor | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-versicolor | Iris-versicolor | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-setosa | Iris-setosa | Correct |
| Iris-virginica | Iris-virginica | Correct |

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

Confusion Matrix:
[[7 0 0]
[0 4 0]
[0 1 3]]

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

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Iris-setosa | 1.00 | 1.00 | 1.00 | 7 |
| Iris-versicolor | 0.80 | 1.00 | 0.89 | 4 |
| Iris-virginica | 1.00 | 0.75 | 0.86 | 4 |
| accuracy | | 0.93 | 15 | |
| macro avg | 0.93 | 0.92 | 0.92 | 15 |
| weighted avg | 0.95 | 0.93 | 0.93 | 15 |

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

Accuracy of the classifer is 0.93

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

## PROGRAM NUMBER – 9 (Locally Weighted Regression Algorithm)

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

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

def kernel(point, xmat, k):
  m,n = np.shape(xmat)
  weights = np.mat(np.eye((m)))          # eye creates identity using formula=matrix w(x,x0)

  for j in range(m):
     diff = point - X[j]                              #matrix formula using (x-x0)2
     weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))    #matrix formula w(x,x0)denominator calculation
  return weights

def localWeight(point, xmat, ymat, k):
  wei = kernel(point,xmat,k)                                # w(x,xo)
   W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
   # beta parameter: to reduce sq error return W # returns to local weight
  return W

def localWeightRegression(xmat, ymat, k):
  m,n = np.shape(xmat)
  ypred = np.zeros(m)
  for i in range(m):
     ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
  return ypred

# load data points
data = pd.read_csv('C:\\Sonia\\AI & ML\\LWR-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
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();
```

**NOTE: LWR-dataset.csv as dataset**

**OUTPUT:**