ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LABORATORY

[As per Choice Based Credit System (CBCS) scheme]

SEMESTER – VII **Subject Code 18CSL76**

Course objectives:

This course will enable students to

1. Make use of Data sets in implementing the AI & machine learning algorithms
2. Implement the AI & machine learning concepts and algorithms in any suitable language of choice

Description (If any):

1. The programs can be implemented in either JAVA or Python.
2. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students.

Course outcomes:

The students should be able to:

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

Conduction of Practical Examination:

- All laboratory experiments are to be included for practical examination.
- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script
- Marks distribution: Procedure + Conduction + Viva.

Programs:

1) A* Algorithm.

```
def aStarAlgo(start_node,stop_node):
    open_set=set(start_node)
    closed_set=set()
    g={}
    parents={}
    g[start_node]=0
    parents[start_node]=start_node
    while len(open_set)> 0:
        n=None
        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):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m]=n
                    g[m]=g[n]+weight
                else:
                    if g[m]>g[n]+weight:
                        g[m]=g[n]+weight
                        parents[m]=n

                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n==None:
            print('Path does not exist!')
            return None
        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
            open_set.remove(n)
            closed_set.add(n)
        print('Path does not exist!')
        return None
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
def heuristic(n):
    H_dist={
        'A':11,
        'B':6,
        'C':99,
        'D':1,
        'E':7,
        'G':0,
    }
    return H_dist[n]
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']

2) AO* Algorithm.

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        self.graph=graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}
    def applyAOStar(self):
        self.aoStar(self.start,False)
```

```
    def getNeighbors(self,v):
        return self.graph.get(v,'')

    def getStatus(self,v):
        return self.status.get(v,0)

    def setStatus(self,v,val):
        self.status[v]=val

    def getHeuristicNodeValue(self,n):
        return self.H.get(n,0)

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

    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):
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True

        for nodeInfoTupleList in self.getNeighbors(v):
            cost=0
            nodeList=[]

            for c,weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)
            if flag==True:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
                flag=False
            else:
```

```
            if minimumCost>cost:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
        return minimumCost,costToChildNodeListDict[minimumCost]

    def aoStar(self,v,backTracking):
        print("HEURISTIC VALUES:",self.H)
        print("SOLUTION GHAPH:",self.solutionGraph)
        print("PROCESSING NODE:",v)
        print("__________________________________________")
        if self.getStatus(v)>=0:
            minimumCost,childNodeList=self.computeMinimumCostChildNodes(v)
            print(minimumCost,childNodeList)
            self.setHeuristicNodeValue(v,minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False
            if solved==True:
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList
            if v!=self.start:
                self.aoStar(self.parent[v],True)
            if backTracking==False:
                for childNode in childNodeList:
                    self.setStatus(childNode,0)
                    self.aoStar(childNode,False)

print("Graph -1")
h1={'A':1,'B':6,'C':2,'D':12,'E':2,'F':1,'G':5,'H':7,'I':7,'J':1}
graph1={
    'A':[[('B',1),('C',1)],[('D',1)]],
    'B':[[('G',1)],[('H',1)]],
    'C':[[('J',1)]],
    'D':[[('E',1),('F',1)]],
    'G':[[('I',1)]]
}

G1=Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

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

```
======================================================== RESTART:
C:\Users\VSC\Desktop\kit 2021-22\LAB\aostar.py
========================================================
Graph -1
HEURISTIC VALUES: {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: A
_______________________________________
10 ['B', 'C']
HEURISTIC VALUES: {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: B
_______________________________________
6 ['G']
HEURISTIC VALUES: {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: A
_______________________________________
10 ['B', 'C']
HEURISTIC VALUES: {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: G
_______________________________________
8 ['I']
HEURISTIC VALUES: {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: B
_______________________________________
8 ['H']
HEURISTIC VALUES: {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: A
_______________________________________
12 ['B', 'C']
HEURISTIC VALUES: {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GHAPH: {}
PROCESSING NODE: I
_______________________________________
0 []
```

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

---

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

---

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

---

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

---

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

---

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

---

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

---

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

---

5 ['B', 'C']
FOR GRAPH SOLUTION,TRAVERSE THE GRAPH FROM THE START NODE: A

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

3)CANDIDTE ELIMINATION ALGORITHM

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('2aabb.csv'))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    print("list of attributes")
    attributes = ['Sky','Temp','Humidity','Wind','Water','Forecast']
    print(attributes)
    num_attributes = len(attributes)
    specific_h = ['0'] * num_attributes
    print("Initial specific hypothesis\n",specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range (len(specific_h))]
    print("Initital General hypothesis\n",general_h)
    specific_h = concepts[0].copy()
    for i, h in enumerate(concepts):
        if target[i] == "YES":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "NO":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("steps of Candidate Elemination Algorithem",i+1)
        print("Instance",h)
        print("S",i+1,'=',specific_h)
        print("G",i+1,'=',general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific hypothesis", s_final, sep="\n")
print("Final General hypothesis", g_final, sep="\n")
```

DATABASE

sunny	warm	normal	strong	warm	same	YES
sunny	warm	high	strong	warm	same	YES
rainy	cold	high	strong	warm	change	NO
sunny	warm	high	strong	cool	change	YES

OUTPUT

list of attributes
['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']
Initial specific hypothesis
['0', '0', '0', '0', '0', '0']
Initital General hypothesis
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of Candidate Elemination Algorithem 1
Instance ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
S 1 = ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
G 1 = [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of Candidate Elemination Algorithem 2
Instance ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
S 2 = ['sunny' 'warm' '?' 'strong' 'warm' 'same']
G 2 = [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of Candidate Elemination Algorithem 3
Instance ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
S 3 = ['sunny' 'warm' '?' 'strong' 'warm' 'same']
G 3 = [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
steps of Candidate Elemination Algorithem 4
Instance ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
S 4 = ['sunny' 'warm' '?' 'strong' '?' '?']
G 4 = [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific hypothesis
['sunny' 'warm' '?' 'strong' '?' '?']
Final General hypothesis
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

4) ID3 Algorithm

```
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 = ""

def subtables(data, col, delete):
    dic = {}
    coldata = [ row[col] for row in data]
    attr = list(set(coldata))

    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:
        return 0

    counts = [0,0]
    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:
        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))
    node = Node(features[split])
    fea = features[:split]+features[split+1:]

    attr, dic = subtables(data, split, delete=True)
    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)
        return

    print(" "*level, node.attribute)
    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("3rddb.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("3rddb1.csv")
for xtest in testdata:
    print("The test instance : ",xtest)
    print("The predicted label : ", end="")
    classify(node,xtest,features)
```

DATABASE

Firstdatabase

Outlook	Temperature	Humidity	Wind	Target
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	yes
Rainy	Mild	High	Weak	yes
Rainy	Cool	Normal	Weak	yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	yes
Rainy	Mild	Normal	Weak	yes
Sunny	Mild	Normal	Strong	yes
Overcast	Mild	High	Strong	yes
Overcast	Hot	Normal	Weak	yes
Rainy	Mild	High	Strong	No

second database

Outlook	Temperature	Humidity	Wind
Rainy	Cool	Normal	Strong
Sunny	Mild	Normal	Strong

OUTPUT

The decision tree for the dataset using ID3 algorithm is
Outlook

```
  Sunny
   Humidity
    High
     No
    Normal
     yes
  Overcast
   yes
  Rainy
   Wind
    Strong
     No
    Weak
     yes
The test instance :  ['Rainy', 'Cool', 'Normal', 'Strong']
The predicted label : No
The test instance :  ['Sunny', 'Mild', 'Normal', 'Strong']
The predicted label : yes
```

5) Backpropogation algorithm

```
import numpy as np
X=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
X=X/np.amax(X,axis=0)
y=y/100
def sigmoid(x):
    return 1/(1+np.exp(-x))
def derivatives_sigmoid(x):
    return x*(1-x)
epoch=7000
learning_rate=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wo=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bo=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
    net_h=np.dot(X,wh)+bh
    sigma_h=sigmoid(net_h)
    net_o=np.dot(sigma_h,wo)+ bo
    output = sigmoid(net_o)
    deltaK =(y-output)*derivatives_sigmoid(output)
    deltaH = deltaK.dot(wo.T)*derivatives_sigmoid(sigma_h)
    wo = wo+sigma_h.T.dot(deltaK)*learning_rate
    wh = wh+X.T.dot(deltaH)*learning_rate
print("Input: \n"+str(X))
print("Actual Output: \n"+str(y))
print("Predicted Output: \n",output)
```

OUTPUT

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
```

```
 [1.        0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89504105]
 [0.88132429]
 [0.89368279]]
```

6) Naïve Bayesin Classifier Calculate accuracy,precision.

```
import pandas as pd
import pdb
msg=pd.read_csv('6thdb.csv',names=['message','label']) #names-> name of the cols
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
Y=msg.labelnum

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()

xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
#pdb.set_trace()
```

DATABASE

I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg

He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

OUTPUT

```
Accuracy metrics
Accuracy of the classifer is 0.8
Confusion matrix
[[2 0]
 [1 2]]
Recall and Precison
0.6666666666666666
1.0
```

Program 7

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

A Bayesian belief network describes the probability distribution over a set of variables.

Probability

P(A) is used to denote the probability of A. For example if A is discrete with states {True, False} then P(A) might equal [0.2, 0.8]. I.e. 20% chance of being True, 80% chance of being False.

Joint probability

A joint probability refers to the probability of more than one variable occurring together, such as the probability of A and B, denoted P(A,B).

Conditional probability

Conditional probability is the probability of a variable (or set of variables) given another variable (or set of variables), denoted P(A|B).For example, the probability of Windy being True, given that Raining is True might equal 50%.This would be denoted P(Windy = True | Raining = True) = 50%.

Once the structure has been defined (i.e. nodes and links), a Bayesian network requires a probability distribution to be assigned to each node.Each node X in a Bayesian network requires a probability distribution P(X | pa(X)).Note that if a node X has no parents pa(X) is empty, and the required distribution is just P(X) sometimes referred to as the prior.This is the probability of itself given its parent nodes.

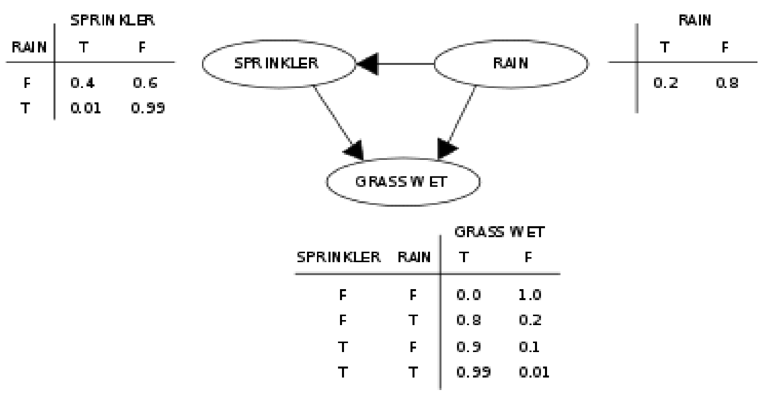
If U = $\{A_1,...,A_n\}$ is the universe of variables (all the variables) in a Bayesian network, and pa(A_i) are the parents of A_i then the joint probability distribution P(U) is the simply the product of all the probability distributions (prior and conditional) in the network, as shown in the equation below.This equation is known as the chain rule.

$$P(\mathbf{X},\mathrm{e}) = \sum_{\mathbf{U}\backslash\mathbf{X}} P(\mathbf{U},\mathrm{e}) = \sum_{\mathbf{U}\backslash\mathbf{X}} \prod_i P(\mathbf{U}_i|pa(\mathbf{U}_i))\mathrm{e}$$

From the joint distribution over U we can in turn calculate any query we are interested in (with or without evidence set).

Suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a Bayesian network (shown to the right). All three variables have two possible values, T (for true) and F (for false).

The joint probability function is: $\Pr(G,S,R) = \Pr(G|S,R)\Pr(S|R)\Pr(R)$



$$\Pr(R=T|G=T) = \frac{\Pr(G=T,R=T)}{\Pr(G=T)} = \frac{\sum_{S\in\{T,F\}}\Pr(G=T,S,R=T)}{\sum_{S,R\in\{T,F\}}\Pr(G=T,S,R)}$$

$$\begin{aligned}\Pr(G=T,S=T,R=T) &= \Pr(G=T|S=T,R=T)\Pr(S=T|R=T)\Pr(R=T)\\ &= 0.99\times 0.01\times 0.2\\ &= 0.00198.\end{aligned}$$

BBn in python:

```
import bayespy as bp

import numpy as np

import csv
```

```
from colorama import init
from colorama import Fore, Back, Style
init()
ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3, 'Teen':4}
genderEnum = {'Male':0, 'Female':1}
familyHistoryEnum = {'Yes':0, 'No':1}
dietEnum = {'High':0, 'Medium':1, 'Low':2}
lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}
cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}
heartDiseaseEnum = {'Yes':0, 'No':1}
with open('heart_disease_data.csv') as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
    for x in dataset:
        data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x[3]],lifeStyleEnum[x[4]],
                cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])
data = np.array(data)
N = len(data)
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:,0])
p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])
p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:,2])
p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
```

```
diet = bp.nodes.Categorical(p_diet, plates=(N,))

diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))

lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))

lifestyle.observe(data[:,4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))

cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))

cholesterol.observe(data[:,5])

p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))

heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical,
p_heartdisease)

heartdisease.observe(data[:,6])

p_heartdisease.update()


m = 0

while m == 0:

   print("\n")

   res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' +
str(genderEnum))),

                               int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' +
str(dietEnum))),

                               int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' +
str(cholesterolEnum)))],

                              bp.nodes.Categorical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]

   print("Probability(HeartDisease) = " +  str(res))

   m = int(input("Enter for Continue:0, Exit :1  "))
```

Output:

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen':
4}1
Enter Gender: {'Male': 0, 'Female': 1}1
Enter FamilyHistory: {'Yes': 0, 'No': 1}1
```

```
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}1
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1  0


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen':
4}0
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}0
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}3
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}0
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1
```

8)EM algorithm and k-means algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import pdb
df1 = pd.read_csv("8thdb.csv")
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values

X = np.matrix(list(zip(f1,f2)))
plt.plot(1)
plt.subplot(511)
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
plt.scatter(f1,f2)

colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
# create new plot and data for K- means algorithm
plt.plot(2)
ax=plt.subplot(513)
kmeans_model = KMeans(n_clusters=3).fit(X)

for i, l in enumerate(kmeans_model.labels_):
   plt.plot(f1[i], f2[i], color=colors[l],marker=markers[l])

plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('K- Means')
plt.ylabel('speeding_feature')
```

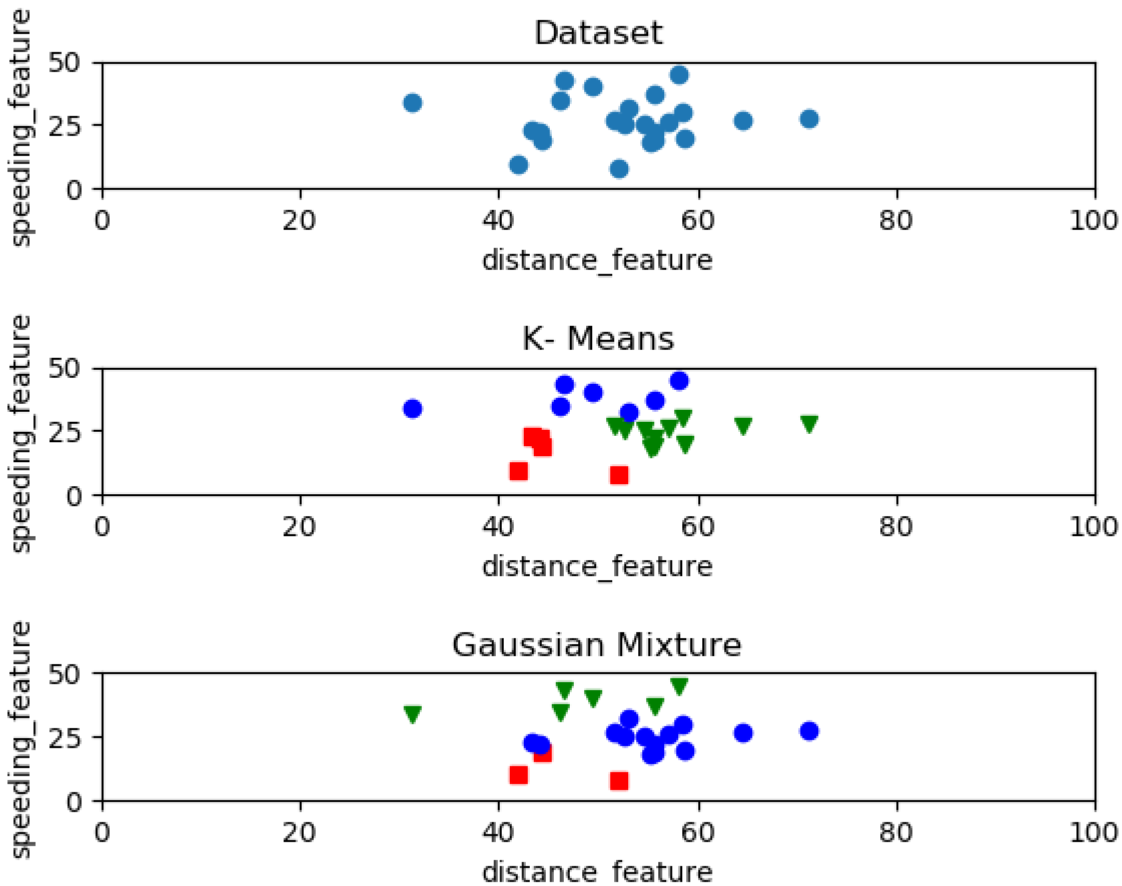
```
plt.xlabel('distance_feature')


# create new plot and data for gaussian mixture
plt.plot(3)
plt.subplot(515)
gmm=GaussianMixture(n_components=3).fit(X)
labels= gmm.predict(X)

for i, l in enumerate(labels):
    plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l])

plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Gaussian Mixture')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')


plt.show()
pdb.set_trace()
```

DATABASE

Driver_ID	Distance_Feature	Speeding_Feature
3.42E+09	71.24	28
3.42E+09	52.53	25
3.42E+09	64.54	27
3.42E+09	55.69	22
3.42E+09	54.58	25
3.42E+09	41.91	10
3.42E+09	58.64	20
3.42E+09	52.02	8
3.42E+09	31.25	34
3.42E+09	44.31	19
3.42E+09	49.35	40
3.42E+09	58.07	45
3.42E+09	44.22	22
3.42E+09	55.73	19
3.42E+09	46.63	43
3.42E+09	52.97	32
3.42E+09	46.25	35
3.42E+09	51.55	27
3.42E+09	57.05	26
3.42E+09	58.45	30
3.42E+09	43.42	23
3.42E+09	55.68	37
3.42E+09	55.15	18

OUTPUT

	Driver_ID	Distance_Feature	Speeding_Feature
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22
4	3423310999	54.58	25
5	3423313857	41.91	10
6	3423312432	58.64	20
7	3423311434	52.02	8
8	3423311328	31.25	34
9	3423312488	44.31	19
10	3423311254	49.35	40
11	3423312943	58.07	45
12	3423312536	44.22	22
13	3423311542	55.73	19
14	3423312176	46.63	43
15	3423314176	52.97	32
16	3423314202	46.25	35
17	3423311346	51.55	27
18	3423310666	57.05	26
19	3423313527	58.45	30
20	3423312182	43.42	23
21	3423313590	55.68	37
22	3423312268	55.15	18



8) K Nearest Neighnour Algorithm

```
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.30)

from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)

from sklearn.metrics import classification_report,confusion_matrix
print('Confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Matrics')
print(classification_report(y_test,y_pred))
```

OUTPUT

```
Confusion matrix is as follows
[[13  0  0]
 [ 0 16  0]
 [ 0  0 16]]
Accuracy Matrics
            precision    recall  f1-score   support

         0       1.00      1.00      1.00        13
         1       1.00      1.00      1.00        16
         2       1.00      1.00      1.00        16

avg / total       1.00      1.00      1.00        45
```

9) LOCALLY WEIGHTED REGRESSION ALGORITHM

```
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
import pdb
def kernel(point, xmat, k):
	m,n=np1.shape(xmat)  #size of matrix m
	weights=np1.mat(np1.eye(m)) #np.eye returns mat with 1 in the diagonal
	for j in range(m):
		diff=point-xmat[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)
	W=(xmat.T*(wei*xmat)).I*(xmat.T*(wei*ymat.T))
```

```
        return W

def localWeightRegression(xmat,ymat,k):
        row,col=np1.shape(xmat) #return 244 rows and  2 columns
        ypred=np1.zeros(row)
        for i in range(row):
                ypred[i]=xmat[i]*localWeight(xmat[i],xmat,ymat,k)
        return ypred

data=pd.read_csv('10thdb.csv')
bill=np1.array(data.total_bill)
tip=np1.array(data.tip)

mbill=np1.mat(bill)
mtip=np1.mat(tip)

mbillMatCol=np1.shape(mbill)[1] # 1 for vertical i.e columns
onesArray=np1.mat(np1.ones(mbillMatCol))
xmat=np1.hstack((onesArray.T,mbill.T)) #hstack concate horizontal lists it takes one value from the fist and one from the second
print(xmat)

ypred=localWeightRegression(xmat,mtip,2)
SortIndex=xmat[ :,1].argsort(0) #argsort take the index of each and sort them according to the orginal value
xsort=xmat[SortIndex][:,0]

fig= plt.figure()
ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='blue')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=1)
plt.xlabel('Total bill')
plt.ylabel('tip')
plt.show();
pdb.set_trace()
```

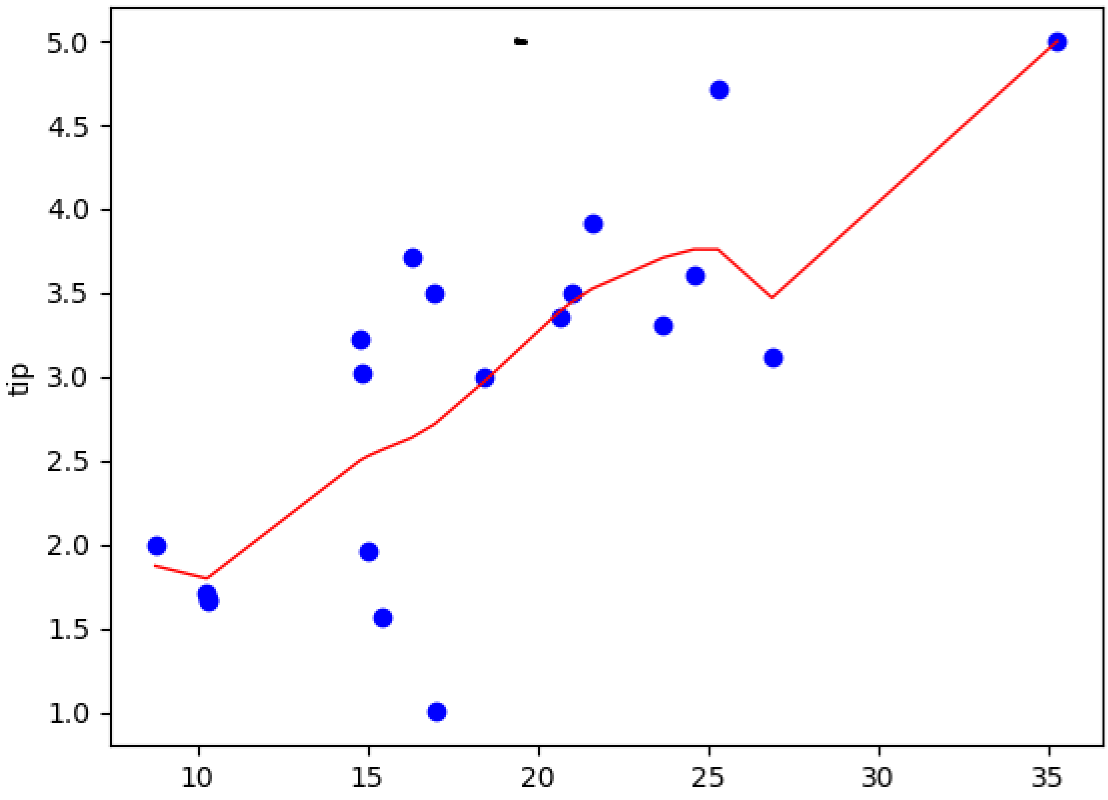
DATABASE

total_bill	tip
16.99	1.01
10.34	1.66
21.01	3.5
23.68	3.31
24.59	3.61
25.29	4.71
8.77	2
26.88	3.12
15.04	1.96
14.78	3.23
10.27	1.71
35.26	5
15.42	1.57
18.43	3

14.83	3.02
21.58	3.92
10.33	1.67
16.29	3.71
16.97	3.5
20.65	3.35

OUTPUT

```
[[ 1.   16.99]
 [ 1.   10.34]
 [ 1.   21.01]
 [ 1.   23.68]
 [ 1.   24.59]
 [ 1.   25.29]
 [ 1.    8.77]
 [ 1.   26.88]
 [ 1.   15.04]
 [ 1.   14.78]
 [ 1.   10.27]
 [ 1.   35.26]
 [ 1.   15.42]
 [ 1.   18.43]
 [ 1.   14.83]
 [ 1.   21.58]
 [ 1.   10.33]
 [ 1.   16.29]
 [ 1.   16.97]
 [ 1.   20.65]]
```


5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
tip
10
15
20
25
30
35