MACHINE LEARNING LABORATORY RECORD BOOK

Sahyadri
College of Engineering & Management
Mangaluru

Name:…………………………………………………………….....

USN:……………………………………………………………..

BE Semester:…………………………………………………..

Branch:…………………………………Section:…………..............

Subject Code: …………………………………………………

Empowering Young Minds

Vision

To be a premier institution in Technology and Management by fostering excellence in education, innovation, incubation and values to inspire and empower the young minds.

Mission

ME1: Creating an academic ambience to impart holistic education focusing on individual growth, integrity, ethical values and social responsibility.

ME2: Develop skill based learning through industry-institution interaction to enhance competency and promote entrepreneurship.

ME3: Fostering innovation and creativity through competitive environment with state-of-the-art infrastructure.

Goal

Good Governance & Administration Residential Campus & Township Establishment

- Attracting best talented students
- Best Placements
- Alumni Leveraging
- Attraction, Development and retention of best faculty
- Best teaching, learning and evaluation systems
- Establishing Research & Innovation Center
- Industry – Institute relationships & Consultancy
- MOUs with higher learning universities / institutions / R&D Centers
- Autonomous / University Status
- NBA – ABET / NAAC Accreditation
- Social extension / village

SAHYADRI

College of Engineering & Management
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

CERTIFICATE

This is to certify that

Ms./Mr. ……………………………………………………………

USN ……………………………………………………..

has satisfactorily completed the course of experiments in the practical class

Artificial Intelligence and Machine Learning Laboratory
(18CSL76) *as*

Prescribed by the VTU for ***7th*** *semester* ***B.E.***

for the year ***2021-2022***

Marks

Date:

Head of the Department

Signature of the Staff Member
In-Charge of the Batch

CONTENTS					
S.No.	Date	Experiments	Pages	Marks	
				Max	Awarded
1					
2					
3					
4					
5					
6					
7					
8					
9					

Experiment:1 Implement A* Search algorithm.

```
.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']
['A', 'F', 'G', 'I', 'J']
```

Experiment:2 Implement AO* Search algorithm.

```
def recAOStar(n):
   global finalPath
   print("Expanding Node:",n)
   and_nodes = []
   or_nodes =[]
   if(n in allNodes):
      if 'AND' in allNodes[n]:
         and_nodes = allNodes[n]['AND']
      if 'OR' in allNodes[n]:
         or_nodes = allNodes[n]['OR']
   if len(and_nodes)==0 and len(or_nodes)==0:
      return
      solvable = False
   marked ={}
      while not solvable:
      if len(marked)==len(and_nodes)+len(or_nodes):
         min_cost_least,min_cost_group_least = least_cost_group(and_nodes,or_nodes,{})
         solvable = True
         change_heuristic(n,min_cost_least)
         optimal_child_group[n] = min_cost_group_least
         continue
      min_cost,min_cost_group = least_cost_group(and_nodes,or_nodes,marked)
      is_expanded = False
      if len(min_cost_group)>1:
         if(min_cost_group[0] in allNodes):
            is_expanded = True
            recAOStar(min_cost_group[0])
         if(min_cost_group[1] in allNodes):
            is_expanded = True
            recAOStar(min_cost_group[1])
      else:
         if(min_cost_group in allNodes):
            is_expanded = True
            recAOStar(min_cost_group)
      if is_expanded:
         min_cost_verify, min_cost_group_verify = least_cost_group(and_nodes, or_nodes,
{})
         if min_cost_group == min_cost_group_verify:
            solvable = True
            change_heuristic(n, min_cost_verify)
            optimal_child_group[n] = min_cost_group
      else:
         solvable = True
         change_heuristic(n, min_cost)
```

```
            optimal_child_group[n] = min_cost_group
        marked[min_cost_group]=1
    return heuristic(n)
def least_cost_group(and_nodes, or_nodes, marked):
    node_wise_cost = {}
    for node_pair in and_nodes:
        if not node_pair[0] + node_pair[1] in marked:
            cost = 0
            cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
            node_wise_cost[node_pair[0] + node_pair[1]] = cost
    for node in or_nodes:
        if not node in marked:
            cost = 0
            cost = cost + heuristic(node) + 1
            node_wise_cost[node] = cost
    min_cost = 999999
    min_cost_group = None
    for costKey in node_wise_cost:
        if node_wise_cost[costKey] < min_cost:
            min_cost = node_wise_cost[costKey]
            min_cost_group = costKey
    return [min_cost, min_cost_group]
def heuristic(n):
    return H_dist[n]
def change_heuristic(n, cost):
    H_dist[n] = cost
    return
def print_path(node):
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]
    if len(node) > 1:
        if node[0] in optimal_child_group:
            print("->", end="")
            print_path(node[0])
        if node[1] in optimal_child_group:
            print("->", end="")
            print_path(node[1])
    else:
        if node in optimal_child_group:
            print("->", end="")
            print_path(node)
H_dist = {
 'A': -1,
 'B': 4,
 'C': 2,
 'D': 3,
```

```
 'E': 6,
 'F': 8,
 'G': 2,
 'H': 0,
 'I': 0,
 'J': 0
}
allNodes = {
 'A': {'AND': [('C', 'D')], 'OR': ['B']},
 'B': {'OR': ['E', 'F']},
 'C': {'OR': ['G'], 'AND': [('H', 'I')]},
 'D': {'OR': ['J']}
}
optimal_child_group = {}
optimal_cost = recAOStar('A')
print('Nodes which gives optimal cost are')
print_path('A')
print('\nOptimal Cost is :: ', optimal_cost)
```

Output

```
Expanding Node: A
Expanding Node: B
Expanding Node: C
Expanding Node: D
Nodes which gives optimal cost are
CD->HI->J
Optimal Cost is ::  5
```

Experiment 3: Candidate-Elimination Algorithm.

Aim: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
from pandas import DataFrame
data=DataFrame.from_csv('EnjoySport.csv')
concepts=data.values[:,:-1]
target=data.values[:,-1]

def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [['?' for i in range(len(specific_h))] for i in range(len(specific_h))]
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            #print(target[i])
            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] = '?'
    indices = [i for i,val in enumerate(general_h) if val==['?' for i in range(len(specific_h))]]
    for i in indices:
        general_h.remove(['?' for i in range(len(specific_h))])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final S:", s_final)
print("Final G:", g_final)
```

Output:

```
Final S: ['sunny' 'warm' '?' 'strong' '?' '?']
Final G: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Dataset:

```
          sky airTemp humidity    wind water forecast enjoySport
Sl.No
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
```

Experiment 4: Decision Tree based ID3 Algorithm.

Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
def infoGain(P, N):
    import math
    return -P / (P + N) * math.log2(P / ( P + N)) - N / (P + N) * math.log2(N / (P + N))

def insertNode(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertNode(v, addTo, Node)
    if addTo in tree:
        if isinstance(tree[addTo], dict):
            tree[addTo][Node] = 'None'
        else:
            tree[addTo] = {Node:'None'}
    return tree

def insertConcept(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertConcept(v, addTo, Node)
    if addTo in tree:
        tree[addTo] = Node
    return tree

def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):
    Total = data.shape[0]
    if Total == 0:
        return tree

    countC = {}
    for cVal in conceptVals:
        dataCC = data[data[concept] = = cVal]
        countC[cVal] = dataCC.shape[0]

    if countC[conceptVals[0]] = = 0:
        tree = insertConcept(tree, addTo, conceptVals[1])
        return tree

    if countC[conceptVals[1]] = = 0:
        tree = insertConcept(tree, addTo, conceptVals[0])
        return tree
```

```
    ClassEntropy = infoGain(countC[conceptVals[1]],countC[conceptVals[0]])
    Attr = {}
    for a in AttributeList:
        Attr[a] = list(set(data[a]))
    AttrCount = {}
    EntropyAttr = {}
    for att in Attr:
        for vals in Attr [att]:
            for c in conceptVals:
                iData = data[data[att] = = vals]
                dataAtt = iData[iData[concept] = = c]
                AttrCount[c] = dataAtt.shape[0]
            TotalInfo = AttrCount[conceptVals[1]] + AttrCount[conceptVals[0]]
            if AttrCount[conceptVals[1]] = = 0 or AttrCount[conceptVals[0]] = = 0:
                InfoGain=0
            else:
                InfoGain = infoGain(AttrCount[conceptVals[1]], AttrCount[conceptVals[0]])

            if att not in EntropyAttr:
                EntropyAttr[att] = ( TotalInfo / Total ) * InfoGain
            else:
                EntropyAttr[att] = EntropyAttr[att] + ( TotalInfo / Total ) * InfoGain

    Gain = {}
    for g in EntropyAttr:
        Gain[g] = ClassEntropy - EntropyAttr[g]

    Node = max(Gain, key = Gain.get)
    tree = insertNode(tree, addTo, Node)
    for nD in Attr[Node]:
        tree = insertNode(tree, Node, nD)
        newData = data[data[Node] = = nD].drop(Node, axis = 1)
        AttributeList=list(newData)[:-1]   #New Attribute List
        tree = getNextNode(newData, AttributeList, concept, conceptVals, tree, nD)
    return tree

def main():
    from pandas import DataFrame
    data = DataFrame.from_csv('PlayTennis.csv')
    print(data)
    AttributeList = list(data)[:-1]
    concept = str(list(data)[-1])
    conceptVals = list(set(data[concept]))
    tree = getNextNode(data, AttributeList, concept, conceptVals, {'root':'None'}, 'root')
    print(tree)
```

main()

Output:

```
{'root': {'Outlook': {'Sunny': {'Humidity': {'Normal': 'Yes', 'High': 'No'}},
 'Rain': {'Wind': {'Strong': 'No', 'Weak':'Yes'}}, 'Overcast': 'Yes'}}}
```

Dataset:

```
       Outlook Temperature Humidity    Wind PlayTennis
slno
0        Sunny         Hot     High    Weak         No
1        Sunny         Hot     High  Strong         No
2     Overcast         Hot     High    Weak        Yes
3         Rain        Mild     High    Weak        Yes
4         Rain        Cool   Normal    Weak        Yes
5         Rain        Cool   Normal  Strong         No
6     Overcast        Cool   Normal  Strong        Yes
7        Sunny        Mild     High    Weak         No
8        Sunny        Cool   Normal    Weak        Yes
9         Rain        Mild   Normal    Weak        Yes
10       Sunny        Mild   Normal  Strong        Yes
11    Overcast        Mild     High  Strong        Yes
12    Overcast         Hot   Normal    Weak        Yes
13        Rain        Mild     High  Strong         No
```

Experiment 5: Artificial Neural Network using Back propagation Algorithm.

Aim: Build an Artificial Neural Network by implementing the Backpropagation 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)
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:
 [[0.89454606]
 [0.88145161]
 [0.89393293]]
```

Experiment 6: Naïve Bayes Classifier.

Aim: Write a program to implement the Naïve Bayes classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
def probAttr(data,attr,val):
    Total=data.shape[0]
    from collections import Counter
    cnt=Counter(x for x in data[attr])
    return cnt[val],cnt[val]/Total

def probAttrConcept(data,attr,val,concept,cVal,countConcept):
    count={}
    prob={}
    C = data[concept]
    A = data[attr]
    for a in range(len(data[attr])):
        for v in val:
            if(A[a]==v and C[a]==cVal):
                if v not in count:
                    count[v]=1
                else:
                    count[v]=count[v]+1
    for a in count:
        prob[a] = count[a]/countConcept
    return prob

def train(data,AttributeList,concept):
    Attr={}
    probability_list={}
    #Get attribute values
    for a in AttributeList:
        Attr[a] = list(set(data[a]))

    conceptVals = list(set(data[concept]))
    conceptProbs = {}
    countConcept={}
    AttrConcept = {}
    for cVal in conceptVals:
        countConcept[cVal],conceptProbs[cVal] = probAttr(data,concept,cVal)
    for val in Attr:
        probability_list[val]={}
        AttrConcept[val] = {}
        for v in Attr[val]:
            a,probability_list[val][v]=probAttr(data,val,v)
        for cVal in conceptVals:
```

```
AttrConcept[val][cVal]=probAttrConcept(data,val,Attr[val],concept,cVal,countConcept[cVal
])

    print("P(A) : ",conceptProbs,"\n")
    print("P(X/A) : ",AttrConcept,"\n")
    print("P(X) : ",probability_list,"\n")
    return conceptProbs,AttrConcept,probability_list
def
test(examples,AttributeList,conceptProbs,AttrConcept,probability_list,data,concept_list,Total
):
    misclassification_count=0
    Total1 = len(examples)-1
    for ex in range(1,len(examples)):
        px={}
        for c in concept_list:
            for x in range(1,len(examples[ex])-1):
                for a in AttributeList:
                    if examples[ex][x] in AttrConcept[a][c]:
                        if c not in px:
                            px[c] =
1*AttrConcept[a][c][examples[ex][x]]/probability_list[a][examples[ex][x]]
                        else:
                            px[c] =
px[c]*AttrConcept[a][c][examples[ex][x]]/probability_list[a][examples[ex][x]]
            px[c] = px[c]*conceptProbs[c]
        print(px)
        classification = max(px,key=px.get)
        print("Classification :",classification,"Expected :",examples[ex][-1])
        if(classification!=examples[ex][-1]):
            misclassification_count+=1
    misclassification_rate=misclassification_count*100/Total1
    accuracy=100-misclassification_rate
    print("Misclassification Count={}".format(misclassification_count))
    print("Misclassification Rate={}%".format(misclassification_rate))
    print("Accuracy={}%".format(accuracy))

def main():
    import pandas as pd
    from pandas import DataFrame
    from collections import Counter
    data = DataFrame.from_csv('PlayTennis_train1.csv')
    AttributeList=list(data)[:-1]
    num_of_attributes=len(AttributeList)
    concept=str(list(data)[-1])
    Total=data.shape[0]
```

```
    conceptProbs,AttrConcept,probability_list = train(data,AttributeList,concept)

    import csv
    with open('PlayTennis_test1.csv')  as csvFile:
        examples = [list(line) for line in csv.reader(csvFile)]
    concept_list =list(Counter(x for x in data[concept]))

test(examples,AttributeList,conceptProbs,AttrConcept,probability_list,data,concept_list,Total
)

main()
```

Output:

```
P(A) :  {'No': 0.35714285714285715, 'Yes': 0.6428571428571429}

P(X/A) :{
'Outlook': {'Rain': {'No': 0.4, 'Yes': 0.3333333333333333}, 'Overcast': {'No': 0.0, 'Yes': 0.4444444444444444},
                     'Sunny': {'No': 0.6, 'Yes': 0.2222222222222222}},
 'Temperature': {'Cool': {'No': 0.2, 'Yes': 0.3333333333333333}, 'Hot': {'No': 0.4, 'Yes': 0.2222222222222222},
                         'Mild': {'No': 0.4, 'Yes': 0.4444444444444444}},
 'Humidity': {'Normal': {'No': 0.2, 'Yes': 0.6666666666666666}, 'High': {'No': 0.8, 'Yes': 0.3333333333333333}},
     'Wind': {'Strong': {'No': 0.6, 'Yes': 0.3333333333333333}, 'Weak': {'No': 0.4, 'Yes': 0.6666666666666666}}}


P(X) :  {'Outlook': {'Rain': 0.35714285714285715, 'Overcast': 0.2857142857142857, 'Sunny': 0.35714285714285715},
         'Temperature': {'Cool': 0.2857142857142857, 'Hot': 0.2857142857142857, 'Mild': 0.42857142857142855},
         'Humidity': {'Normal': 0.5, 'High': 0.5},
         'Wind': {'Strong': 0.42857142857142855, 'Weak': 0.5714285714285714}}

{'No': 0.9408000000000001, 'Yes': 0.24197530864197522}
Classification : No Expected : No
Misclassification Count=0
Misclassification Rate=0.0%
Accuracy=100.0%
```

Dataset:

Training Set

slno	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

Testing example

slno	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Cool	High	Strong	No

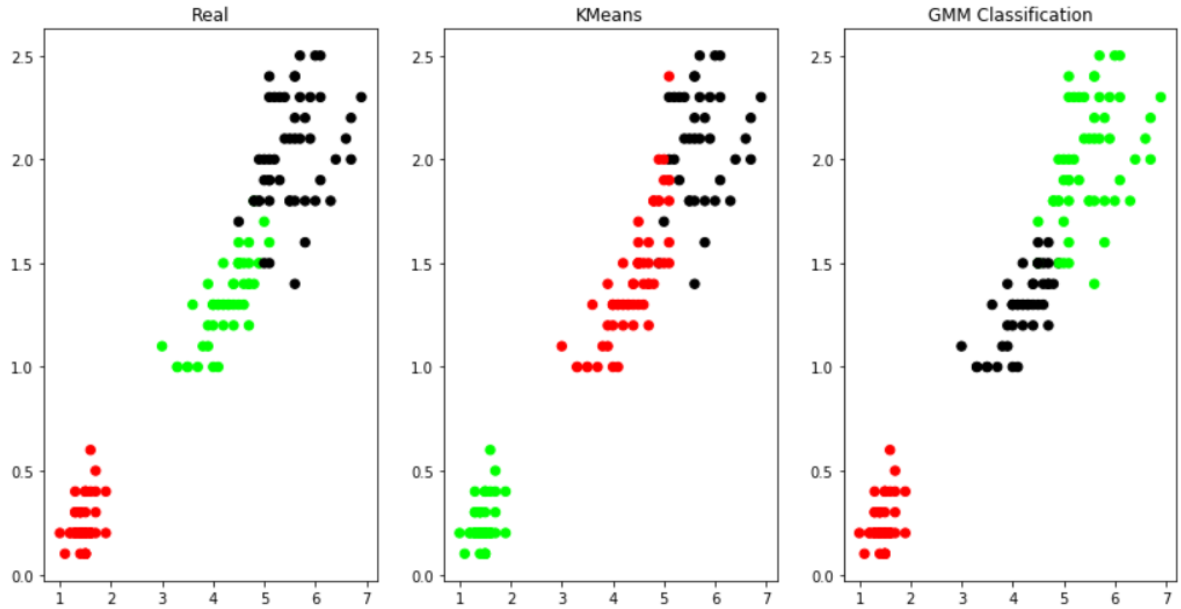
Experiment 7: Clustering using EM Algorithm & k-Means Algorithm.

Aim: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

#REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

#KMeans -PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

#GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
```

```
plt.title('GMM Classification')
```

Output



Dataset:

	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
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2
15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
136	6.3	3.4	5.6	2.4
137	6.4	3.1	5.5	1.8
138	6.0	3.0	4.8	1.8
139	6.9	3.1	5.4	2.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Experiment 8: K-Nearest Neighbour Algorithm.

Aim: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
dataset=load_iris()
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=
0)
clf=KNeighborsClassifier(n_neighbors=1)
clf.fit(X_train,y_train)
for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=clf.predict(x_new)

print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,da
taset["target_names"][prediction])
print(clf.score(X_test,y_test))
```

Output:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
```

```
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
```

0.973684210526

Dataset:

	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
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2
15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
136	6.3	3.4	5.6	2.4
137	6.4	3.1	5.5	1.8
138	6.0	3.0	4.8	1.8
139	6.9	3.1	5.4	2.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Experiment 9: Locally Weighted Regression Algorithm.

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

```
from math import ceil
import numpy as np
from scipy import linalg

def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x),
np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2

    return yest

def main():
    import math
    n = 100
    x = np.linspace(0, 2 * math.pi, n)
    y = np.sin(x) + 0.3 * np.random.randn(n)
    f =0.25
    iterations=3
    yest = lowess(x, y, f, iterations)

    import matplotlib.pyplot as plt
    plt.plot(x,y,"r.")
    plt.plot(x,yest,"b-")

main()
```

Output: