MAHA BARATHI ENGINEERING COLLEGE

NH-79, SALEM-CHENNAI HIGHWAY, A.VASUDEVANUR, CHINNASALEM (TK), KALLAKURICHI (DT) 606 201.

Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai

Accredited by NAAC and Recognized under section 2(f) & 12B of UGC, New Delhi.

www.mbec.ac.in | Ph: 04151-256333, 257333 | E-mail: mbec123@gmail.com

Department of Computer Science and Engineering

Lab Manual

III Year/VI Semester B.E ECE

Regulation 2021

(As Per Anna University, Chennai syllabus

PREPARED BY,

A.ANISHA DARATHY(AP/CSE)

APPROVED BY

Mr. N. KHADIRKUMAR (HOD / CSE)

CONTENTS

Subject Code : CS3491
Subject Name : ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
Year & Sem : III / VI

EXP. NO.	LIST OF EXPERIMENTS/ PROGRAMS	PAGE NO.
1(A)	IMPLEMENTATION OF UNINFORMED SEARCH ALGORITHMS(BFS)	1
1(B)	IMPLEMENTATION OF UNINFORMED SEARCH ALGORITHMS(DFS)	4
2(A)	IMPLEMENTATION OF INFORMED SEARCH ALGORITHMS (A*)	7
2(B)	IMPLEMENTATION OF INFORMED SEARCH ALGORITHMS (MEMORY-BOUNDED AO*)	11
3	IMPLEMENT NAÏVE BAYES MODELS	18
4	IMPLEMENT BAYESIAN NETWORKS	21
5(A)	BUILD REGRESSION MODEL ((LINEAR REGRESSION)	24
5(B)	BUILD REGRESSION MODEL (LOGISTICS REGRESSION)	27
6(A)	BUILD DECISION TREE	32
6(B)	BUILD RANDOM FORESTS TREE	40
7	BUILD SVM MODELS	50
8	IMPLEMENT ENSEMBLING TECHNIQUES	61
9	IMPLEMENT CLUSTERING ALGORITHMS	63
10	IMPLEMENT EM FOR BAYESIANNETWORKS	71
11	BUILD SIMPLE NN MODELS	79
12	BUILD DEEP LEARNING NN MODEL.	93

CS3491 / ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

LIST OF EXPERIMENTS

COURSE OBJECTIVES:

- Study about uninformed and Heuristic search techniques
- Learn techniques for reasoning under uncertainty
- Introduce Machine Learning and supervised learning algorithms
- Study about ensembling and unsupervised learning algorithms
- Learn the basics of deep learning using neural networks

LIST OF EXPERIMENTS

1. Implementation of Uninformed search algorithms(BFS,DFS)
2. Implementation of Informed search algorithms(A*,memory-boundedA*)
3. Implement naïve Bayes models
4. Implement Bayesian Networks
5. Build Regression models
6. Build decision trees and random forests
7. Build SVM models
8. Implement ensembling techniques
9. Implement clustering algorithms
10. Implement EM for Bayesian networks
11. Build simple NN models
12. Build deep learning NN models

TOTAL: 30 PERIODS

COURSE OUTCOMES:

At the end of the course, the students will be able to:

CO1*:* Use appropriate search algorithms for problem solving

CO2: Apply reasoning under uncertainty

CO3: Build supervised learning models

CO4: Build ensembling and unsupervised models

CO5: Build deep learning neural network models

LIST OF EXPERIMENTS
FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

Anaconda – Jupyter Notebook

HARDWARE:

Standalone desktops - 30 Nos.

PLATFORM NEEDED: Python for windows

Ex.No:1(a)	Implementation of Uniformed Search Algorithms -BFS
Date:	

Aim:

To implements the simple uniformed search algorithm breadth first search methods using python.

Algorithm:

Step 1: Initialize an empty list called 'visited' to keep track of the nodes visited during the traversal.
Step 2:Initialize an empty queue called 'queue' to keep track of the nodes to be traversedin the future.
Step 3:Add the starting node to the 'visited' list and the 'queue'.
Step 4:While the 'queue' is not empty, do the following:
a. Dequeue the first node from the 'queue' and store it in a variable called 'current'.
b. Print 'current'.
c. For each of the neighbours of 'current' that have not been visited yet, do the following:
i. Mark the neighbour as visited and add it to the 'queue'.
Step 5:When all the nodes reachable from the starting node have been visited, terminate the algorithm.

Program :

```
graph = {
 '5' : ['3','7'],
 '3' : ['2', '4'],
 '7' : ['8'],
 '2' : [],
 '4' : ['8'],
 '8' : []
}
visited = []
queue = []
def bfs(visited, graph, node):
 visited.append(node)
 queue.append(node)
 while queue:
  m = queue.pop(0)
  print (m, end = " ")
  for neighbour in graph[m]:
   if neighbour not in visited:
    visited.append(neighbour)
    queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

Output:

Following is the Breadth-First Search
5 3 7 2 4 8

Result:

Thus, the program for breadth first search was executed and output is verified.

Ex.No:1(b)	Implementation of uniformed search algorithms-DFS
Date:	

Aim:

To implements the simple uniformed search algorithm depth first search methods using python.

Algorithm:

Step 1: Initialize an empty set called 'visited' to keep track of the nodes visited during thetraversal.

Step 2: Define a DFS function that takes the current node, the graph, and the 'visited' setas input.

Step 3: If the current node is not in the 'visited' set, do the following:

a. Print the current node.

b. Add the current node to the 'visited' set.

c. For each of the neighbours of the current node, call the DFS function recursively with the neighbour as the current node.

Step 4: When all the nodes reachable from the starting node have been visited, terminate the algorithm.

Program:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set()
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

Output:

Following is the Depth-First Search
5
3
2
4
8
7

Result:

Thus the uninformed search algorithms DFS have beenexecuted successfully and the output got verified.

Ex.No:2(a)	Implementation of Informed search algorithms - A*
Date:	

Aim:

To implement the informed search algorithm A*.

Algorithm:

1. Initialize the distances dictionary with float('inf') for all vertices in the graphexcept for the start vertex which is set to 0.
2. Initialize the parent dictionary with None for all vertices in the graph.
3. Initialize an empty set for visited vertices.
4. Initialize a priority queue (pq) with a tuple containing the sum of the heuristic value and the distance from start to the current vertex, the distance from start to the current vertex, and the current vertex.
5. While pq is not empty, do the following:
   a. Dequeue the vertex with the smallest f-distance (sum of the heuristic value and the distance from start to the current vertex).
   b. If the current vertex is the destination vertex, return distances and parent.
   c. If the current vertex has not been visited, add it to the visited set.
   d. For each neighbor of the current vertex, do the following:
   i. Calculate the distance from start to the neighbor (g) as the sum of the distance from start to the current vertex and the edge weight between the current vertex and the neighbor.
   ii. Calculate the f-distance (f = g + h) for the neighbor.
   iii. If the f-distance for the neighbor is less than its current distance in the distances dictionary, update the distances dictionary with the new distance and the parent dictionary with the current vertex as the parent of the neighbor.
   iv. Enqueue the neighbor with its f-distance, distance from start to neighbor, and the neighbor itself into the priority queue.
6. Return distances and parent.

Program :

```
from collections import deque

class Graph:
  # example of adjacency list (or rather map)
  # adjacency_list = {
  # 'A': [('B', 1), ('C', 3), ('D', 7)],
  # 'B': [('D', 5)],
  # 'C': [('D', 12)]
  # }

  def __init__(self, adjacency_list):
    self.adjacency_list = adjacency_list

  def get_neighbors(self, v):
    return self.adjacency_list[v]
```

```
# heuristic function with equal values for all nodes
 def h(self, n):
    H = {
       'A': 1,
       'B': 1,
       'C': 1,
       'D': 1
    }

    return H[n]

 def a_star_algorithm(self, start_node, stop_node):
    # open_list is a list of nodes which have been visited, but who's neighbors
    # haven't all been inspected, starts off with the start node
    # closed_list is a list of nodes which have been visited
    # and who's neighbors have been inspected
    open_list = set([start_node])
    closed_list = set([])

    # g contains current distances from start_node to all other nodes
    # the default value (if it's not found in the map) is +infinity
    g = {}

    g[start_node] = 0

    # parents contains an adjacency map of all nodes
    parents = {}
    parents[start_node] = start_node

    while len(open_list) > 0:
       n = None

       # find a node with the lowest value of f() - evaluation function
       for v in open_list:
          if n == None or g[v] + self.h(v) < g[n] + self.h(n):
             n = v;

       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:
          reconst_path = []

          while parents[n] != n:
             reconst_path.append(n)
             n = parents[n]

          reconst_path.append(start_node)

          reconst_path.reverse()

          print('Path found: {}'.format(reconst_path))
          return reconst_path
```

```
        # for all neighbors of the current node do
        for (m, weight) in self.get_neighbors(n):
            # if the current node isn't in both open_list and closed_list
            # add it to open_list and note n as it's parent
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                parents[m] = n
                g[m] = g[n] + weight

            # otherwise, check if it's quicker to first visit n, then m
            # and if it is, update parent data and g data
            # and if the node was in the closed_list, move it to open_list
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n

                    if m in closed_list:
                        closed_list.remove(m)
                        open_list.add(m)

        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_list.remove(n)
        closed_list.add(n)

    print('Path does not exist!')
    return None
adjacency_list = {
 'A': [('B', 1), ('C', 3), ('D', 7)],
 'B': [('D', 5)],
 'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

Output:

```
Path found: ['A', 'B', 'D']
['A', 'B', 'D']
```

Result:

Thus the program to implement informed search A* algorithm have been executed successfully and output got verified.

Ex.No:2(b)	Implementation of Informed search algorithms - Memory bounded A*
Date:	

Aim:

To implement the informed search algorithm memory bounded A*-IDA*.

Algorithm:

1. The limit = h'(root node), current node = root node
2. Path = root node, Best node = root node
3. Pick up leftmost child of current node (if in a graph, one can pick up any directly connected node at random), and make it current
4. If it is goal node quit
5. Find g value of the child and apply h' to it.
6. If g + h' > limit
   a. update parent node estimate based on the best path and update it till the root node
   b. pick up another node as if reached to dead end (usually the right child of the parent)
   c. change path variable accordingly
7. otherwise
   a. If no node left in the tree
   b. change limit value to limit = g (best node) + h' (best node)
   c. Go to 2.
8. Otherwise,
   a. add this node to path
   b. if this node is better than best node = current node
   c. explore this node, go to 3

Program:

```
#node           current node
#g              the cost to reach current node
#f              estimated cost of the cheapest path (root..node..goal)
#h(node)        estimated cost of the cheapest path (node..goal)
#cost(node, succ)  step cost function
#is_goal(node)     goal test
#successors(node)  node expanding function
V={}
E={}
V=({'A':7,'B':9,'C':6,'D':5,'E':6,'F':4.5,'H':4,'I':2,'J':3,'K':3.5,'G':0})
E=({('B','D'):2,('A','B'):4,('A','C'):4,('A','D'):7,('D','E'):6,('E','F'):5,('D','F'):8,('D','H'):5,('H','I'):3,('I','J'):3
,('J','K'):3,('K','H'):3,('F','G'):5})
INFINITY=10000000
cameFrom={}
def h(node):
   return V[node]
def cost(node, succ):
   return E[node,succ]

def successors(node):
   neighbours=[]
   for item in E:
      if node==item[0][0]:
         neighbours.append(item[1][0])
   return neighbours
```

```
def reconstruct_path(cameFrom, current):
    total_path = [current]
    while current in cameFrom:
        current = cameFrom[current]
        total_path.append(current)
    return total_path

def ida_star(root,goal):
    global cameFrom
    def search(node, g, bound):
        min_node=None
        global cameFrom
        f = g + h(node)
        if f > bound:return f
        if node==goal:return "FOUND"
        minn = INFINITY
        for succ in successors(node):
            t = search(succ, g + cost(node, succ), bound)
            if t == "FOUND":return "FOUND"
            if t < minn:
                minn = t
                min_node=succ
        cameFrom[min_node]=node
        return minn
    bound= h(root)
    count =1
    while 1:
        print("itertion"+str(count))
        count+=1
        t = search(root, 0, bound)

        if t == "FOUND":
            print(reconstruct_path(cameFrom, goal))
            return bound
        if t == INFINITY:return "NOT_FOUND"
        bound = t
print(ida_star('A','G'))
```

Output:

```
itertion1
 itertion2
 itertion3
itertion4
itertion5
 itertion6
 itertion7
 itertion8
 itertion9
itertion10
 ['G', 'F', 'E', 'D', 'B', 'A']
19
```

Result:

Thus the program to implement memory bounded A* -IDA* algorithm have been executed successfully and output got verified.

Ex.No:3	Implementation of Naive Bayes Models
Date:	

Aim:

To diagnose the climate dataset with Naïve Bayes Classifier Algorithm.

Procedure:

Step 1 - Import basic libraries.
Step 2 - Importing the dataset.
Step 3 - Data preprocessing.
Step 4 - Training the model.
Step 5 - Testing and evaluation of the model. Step 6 - Visualizing the model.

Program:

```
# import necessary libraries import pandas as pd
 from sklearn import tree
from sklearn.preprocessing import LabelEncoder
 from sklearn.naive_bayes import GaussianNB

# Load Data from CSV
 data=pd.read_csv('tennisdata.csv')
print("The first 5 Values of data is :\n", data.head())
```

```
The first 5 Values of data is :
Outlook Temperature Humidity Windy PlayTennis
Sunny    Hot    High    Weak    No

Sunny    Hot    High Strong    No

Overcast Hot    High    Weak    Yes

Rain     Mild   High    Weak    Yes

Rain     Cool Normal    Weak    Yes
```

```
# obtain train data and train output
 X=data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())
```

```
The First 5 values of the train data is
Outlook Temperature   Humidity    Windy
Sunny          Hot     High       Weak
Sunny          Hot     High       Strong
Overcast       Hot     High       Weak
Rain           Mild    High       Weak
Rain           Cool   Normal      Weak
```

```
y=data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())
```

The First 5 values of train output is
No
No
Yes
Yes
Yes

Name: PlayTennis, dtype: object

```
# convert them in numbers
 le_outlook=LabelEncoder()
X.Outlook=le_outlook.fit_transform(X.Outlook)

le_Temperature=LabelEncoder()
X.Temperature=le_Temperature.fit_transform(X.Temperature)

le_Humidity=LabelEncoder()
X.Humidity=le_Humidity.fit_transform(X.Humidity)

le_Windy=LabelEncoder()
X.Windy=le_Windy.fit_transform(X.Windy)

print("\nNow the Train output is\n", X.head())
```

```
Now the Train output is
Outlook Temperature Humidity Windy
                    2          1
                    2          1
                    0          1
                    1          2
                    1          0
```

```
le_PlayTennis=LabelEncoder()
 y=le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
Now the Train output is [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
fromsklearn.model_selectionimporttrain_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.20)

classifier=GaussianNB() classifier.fit(X_train, y_train)

fromsklearn.metricsimportaccuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

Output:

Accuracy is: 0.3333333333333333

Result:

Thus the program with Naïve Bayes Classifier Algorithm have been executed successfully and output got verified

Ex.No:4	**Implementation of Bayesian Networks**
Date:	

Aim:

To implement a Bayesian network.

Algorithm:

Step 1: Import required modules
Step 2: Define network structure
Step 3: Define the parameters using CPT
Step 4: Associate the parameters with the model structure
Step 5: Check if the cpds are valid for the model
Step 6: View nodes and edges of the model
Step 7: Check independencies of a node
Step 8: List all Independencies

Program:

```
from pgmpy.models
 import BayesianNetwork from pgmpy.inference
import VariableElimination
# Defining network structure
alarm_model = BayesianNetwork(
   [
      ("Burglary", "Alarm"),
      ("Earthquake", "Alarm"),
      ("Alarm", "JohnCalls"),
      ("Alarm", "MaryCalls"),
   ]
)

# Defining the parameters using CPT
from pgmpy.factors.discrete import TabularCPD

cpd_burglary = TabularCPD(
   variable="Burglary", variable_card=2, values=[[0.999], [0.001]]
)
cpd_earthquake = TabularCPD(
   variable="Earthquake", variable_card=2, values=[[0.998], [0.002]]
)
cpd_alarm = TabularCPD(
   variable="Alarm",
   variable_card=2,
   values=[[0.999, 0.71, 0.06, 0.05], [0.001, 0.29, 0.94, 0.95]],
   evidence=["Burglary", "Earthquake"],
   evidence_card=[2, 2],
)
cpd_johncalls = TabularCPD(
   variable="JohnCalls",
variable_card=2,
```

```
    values=[[0.95, 0.1], [0.05, 0.9]],
    evidence=["Alarm"],
    evidence_card=[2],
)
cpd_marycalls = TabularCPD(
    variable="MaryCalls",
    variable_card=2, values=[[0.1,
    0.7], [0.9, 0.3]],
    evidence=["Alarm"],
    evidence_card=[2],
)

# Associating the parameters with the model structurealarm_model.add_cpds(
    cpd_burglary, cpd_earthquake, cpd_alarm, cpd_johncalls, cpd_marycalls
)
# Checking if the cpds are valid
    for the model
    alarm_model.check_model()
# Viewing nodes of the model
    alarm_model.nodes()
# Viewing edges of the model
    alarm_model.edges()
# Checking independcies of a
    node
    alarm_model.local_independe
    ncies("Burglary")
  # Listing all Independencies
  alarm_model.get_independencies(
```

Output:

True

NodeView(('Burglary', 'Alarm', 'Earthquake', 'JohnCalls', 'MaryCalls'))
OutEdgeView([('Burglary', 'Alarm'), ('Alarm', 'JohnCalls'), ('Alarm', 'MaryCalls'), ('Earthquake', 'Alarm')])
(Burglary ⊥ Earthquake)
(MaryCalls ⊥ Earthquake, Burglary, JohnCalls | Alarm) (MaryCalls ⊥ Burglary, JohnCalls | Earthquake, Alarm)
(MaryCalls ⊥ Earthquake, JohnCalls | Burglary, Alarm)
(MaryCalls ⊥ Earthquake, Burglary | JohnCalls, Alarm)
(MaryCalls ⊥ JohnCalls | Earthquake, Burglary, Alarm)
(MaryCalls ⊥ Burglary | Earthquake, JohnCalls, Alarm)
(MaryCalls ⊥ Earthquake | Burglary, JohnCalls, Alarm)
(JohnCalls ⊥ Earthquake, Burglary, MaryCalls | Alarm)
(JohnCalls ⊥ Burglary, MaryCalls | Earthquake, Alarm)
(JohnCalls ⊥ Earthquake, MaryCalls | Burglary, Alarm)
(JohnCalls ⊥ Earthquake, Burglary | MaryCalls, Alarm)
(JohnCalls ⊥ MaryCalls | Earthquake, Burglary, Alarm)
(JohnCalls ⊥ Burglary | Earthquake, MaryCalls, Alarm)
(JohnCalls ⊥ Earthquake | Burglary, MaryCalls, Alarm)
(Earthquake ⊥ Burglary)
(Earthquake ⊥ MaryCalls, JohnCalls | Alarm)
(Earthquake ⊥ MaryCalls, JohnCalls | Burglary, Alarm)
(Earthquake ⊥ JohnCalls | MaryCalls, Alarm)
(Earthquake ⊥ MaryCalls | JohnCalls, Alarm)
(Earthquake ⊥ JohnCalls | Burglary, MaryCalls, Alarm)
(Earthquake ⊥ MaryCalls | Burglary, JohnCalls, Alarm)
(Burglary ⊥ Earthquake)
(Burglary ⊥ MaryCalls, JohnCalls | Alarm)
(Burglary ⊥ MaryCalls, JohnCalls | Earthquake, Alarm)
(Burglary ⊥ JohnCalls | MaryCalls, Alarm)
(Burglary ⊥ MaryCalls | JohnCalls, Alarm)
(Burglary ⊥ JohnCalls | Earthquake, MaryCalls, Alarm)
(Burglary ⊥ MaryCalls | Earthquake, JohnCalls, Alarm)

Result:

Thus the program to implement a bayesian networks have been executedsuccessfully and the output got verified.

Ex.No:5	Regression Models
Date:	

Aim:

To build regression models such as locally weighted linear regression and plotthe necessary graphs.

Algorithm:

1. Read the Given data Sample to X and the curve (linear or non-linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

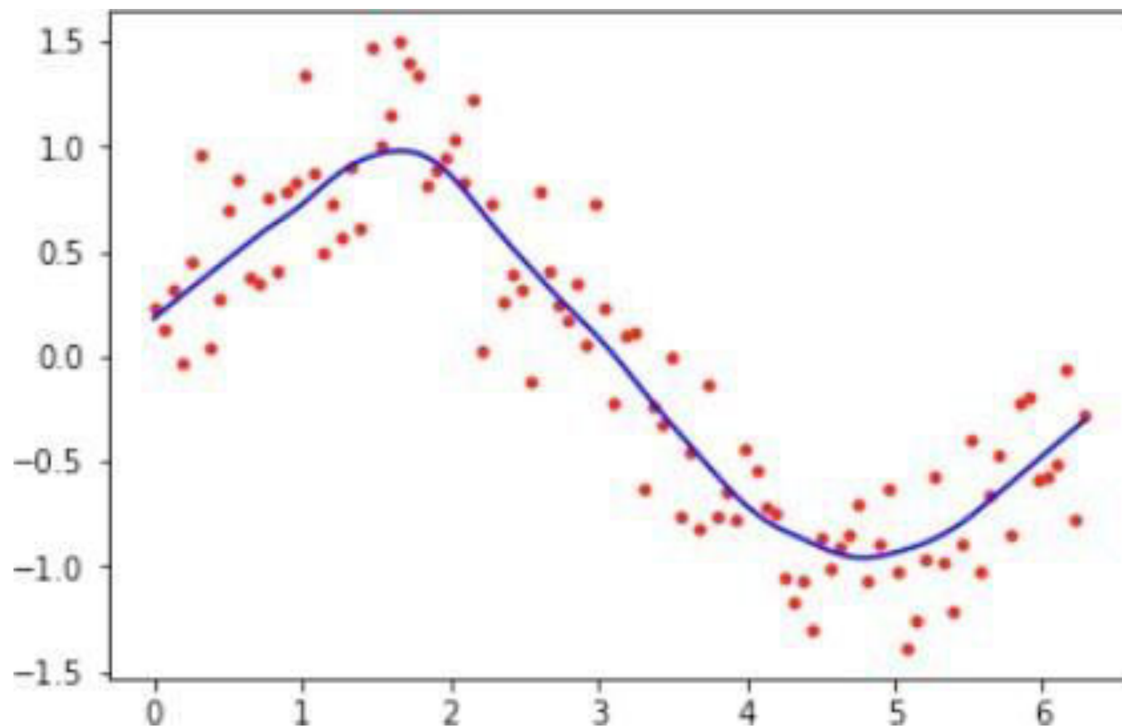
6. Prediction = x0*β.

Program:

```
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) **
   3yest = 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

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-")
```

Output:



Result:

Thus the program to implement non-parametric Locally Weighted Regression algorithm in order to fit data points with a graph visualization have been executed successfully.

Ex.No:6(a)	Decision Trees
Date:	

Aim:

To implement the concept of decision trees .

Algorithm:

Step 1: Import the required libraries

Step 2: Load the iris dataset using the datasets.load_iris() function and split it into training and testing sets using **train_test_split()** function.
Step 3: create a Random Forest classifier using **RandomForestClassifier()** constructor
Step 4: and specify the number of trees (**n_estimators**) in the forest and setting **random_state** for reproducibility
Step 5: Train the classifier using the **fit()** method with the training data
Step 6: Make predictions on the testing data using the **predict()** method.
Step 7: Evaluate the performance of the classifier by calculating the accuracy of the predictions using **accuracy_score()** function.
Step 8: visualize the decision trees in a Random Forest using the **plot_tree** function from the **sklearn.tree** module.
Step 9: Display the plot using plt.show()

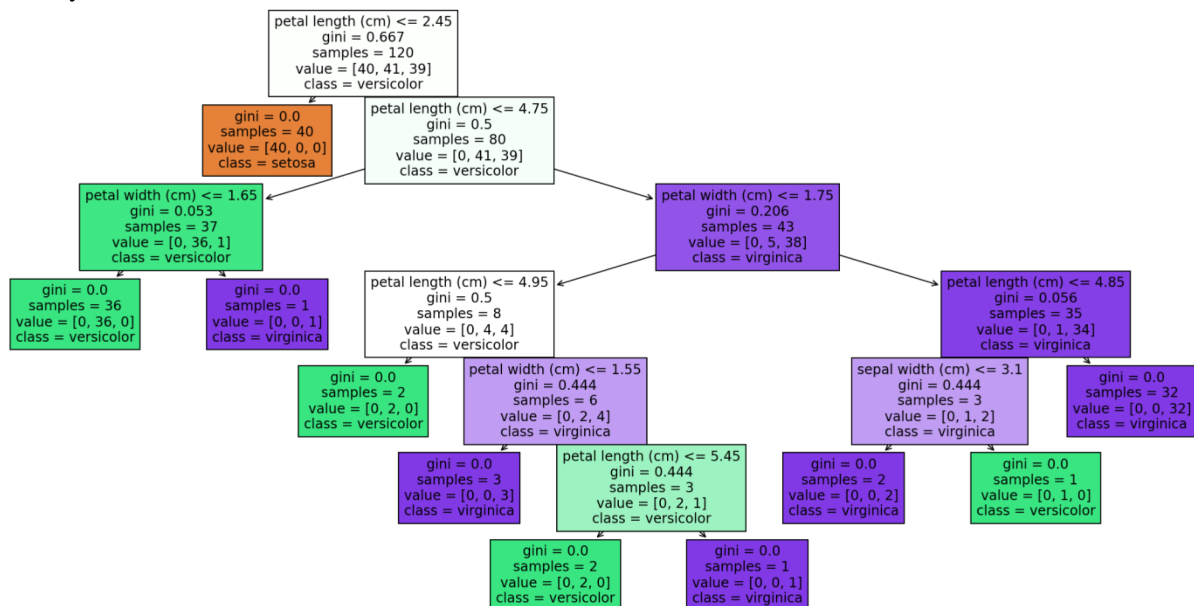
Program:

```
# Importing necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
dt_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = dt_classifier.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Visualize the Decision Tree
plt.figure(figsize=(20,10))
plot_tree(dt_classifier,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True)
plt.show()
```

Output:

Accuracy: 1.0



Result:

Thus the program to implement the concept of decision trees with suitable dataset have been executed successfully.

Ex.No:6(b)	Random Forest Regression
Date:	

Aim:

To implement Random Forest Regression.

Algorithm:

Step 1: Import the required libraries

Step 2: Load the iris dataset using the datasets.load_iris() function and split it into training and testing sets using **train_test_split()** function.
Step 3: create a Random Forest classifier using **RandomForestClassifier()** constructor
Step 4: and specify the number of trees (**n_estimators**) in the forest and setting **random_state** for reproducibility
Step 5: Train the classifier using the **fit()** method with the training data
Step 6: Make predictions on the testing data using the **predict()** method.
Step 7: Evaluate the performance of the classifier by calculating the accuracy of the predictions using **accuracy_score()** function.
Step 8: visualize the decision trees in a Random Forest using the **plot_tree** function from the **sklearn.tree** module.
Step 9: Display the plot using plt.show()

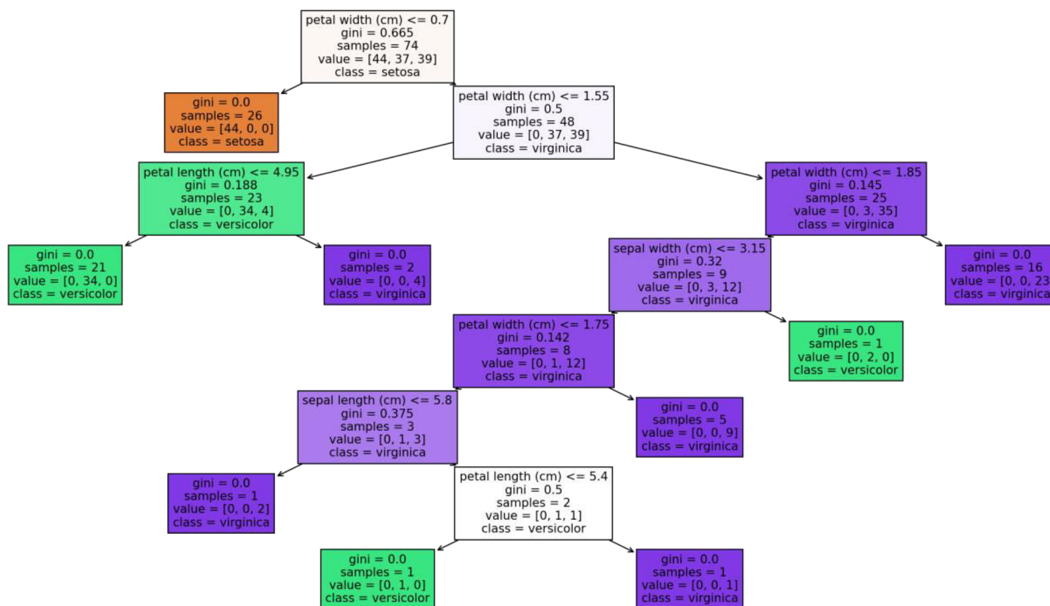
Program:

```
# Importing necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf_classifier.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Visualize one of the decision trees in the Random Forest
plt.figure(figsize=(20,10))
plot_tree(rf_classifier.estimators_[0],
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True)
plt.show()
```

Output:

Accuracy:1.0



Result:

Thus the program to implement random forest regression has been executed successfully and the output is verified.

Ex.No:7	Support Vector Machine algorithm
Date:	

Aim:

To create a machine learning model using Support Vector Machine algorithm.

Algorithm:

Step 1: Import the required libraries

Step 2: Load the iris dataset using the datasets.load_iris() function and store the data andtarget values in variables X and y respectively.

Step 3: Create a pandas dataframe from the iris data using iris_data.data[:, [2, 3]] and column names as iris_data.feature_names[2:].

Step 4: Split the data into training and test sets using train_test_split

Step 5: Print the number of samples in the training and test sets

Step 6: Define the markers, colors, and colormap to be used for plotting the data.

Step 7: Plot the data using a scatter plot by iterating through the unique labels andplotting the points with the corresponding color and marker.

Step 8: Standardize the training and test data
Store the transformed data in X_train_standard and X_test_standard respectively.

Step 9: Train the SVM model using the standardized training data and the SVM() functionStore the trained model in SVM.

Step 10: Print the accuracy of the SVM model on the training and test data using SVM.score(X_train_standard, y_train) and SVM.score(X_test_standard, y_test) respectively.

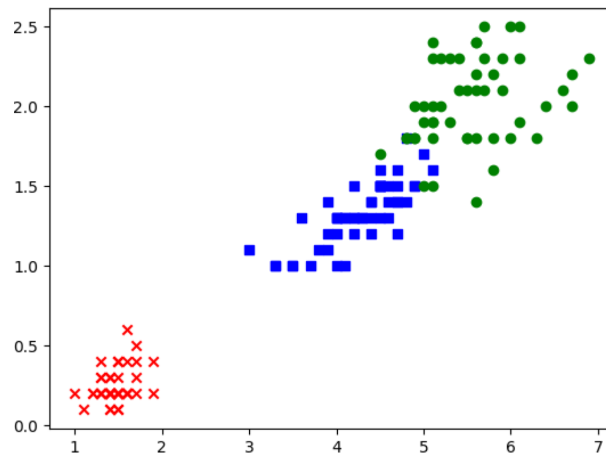
Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

iris_data = datasets.load_iris()
X = iris_data.data[:, [2, 3]]
y = iris_data.target
iris_dataframe = pd.DataFrame(iris_data.data[:, [2, 3]],columns=iris_data.feature_names[2:])
print(iris_dataframe.head())
```

```
print('\n' + 'Unique Labels contained in this data are ' + str(np.unique(y)))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print('The training set contains {} samples and the test set contains {}
samples'.format(X_train.shape[0], X_test.shape[0]))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print('The training set contains {} samples and the test set contains {}
samples'.format(X_train.shape[0], X_test.shape[0]))
markers = ('x', 's', 'o')
colors = ('red', 'blue', 'green')
cmap = ListedColormap(colors[:len(np.unique(y_test))])

for idx, cl in enumerate(np.unique(y)):
   plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
        c=cmap(idx), marker=markers[idx], label=cl)
standard_scaler = StandardScaler()
standard_scaler.fit(X_train)
X_train_standard = standard_scaler.transform(X_train)
X_test_standard = standard_scaler.transform(X_test)
print('The first five rows after standardisation look like this:\n')
print(pd.DataFrame(X_train_standard, columns=iris_dataframe.columns).head())SVM =
SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0) SVM.fit(X_train_standard,
y_train)
print('Accuracy of our SVM model on the training data is',(SVM.score(X_train_standard,
y_train)))
print('Accuracy of our SVM model on the test data is',(SVM.score(X_test_standard, y_test)))
```

Output:

```
 petal length (cm)  petal width (cm)
0             1.4             0.2
1             1.4             0.2
2             1.3             0.2
3             1.5             0.2
4             1.4             0.2

Unique Labels contained in this data are [0 1 2]
The training set contains 105 samples and the test set contains 45 samples
The training set contains 105 samples and the test set contains 45 samples
The first five rows after standardisation look like this:

   petal length (cm)  petal width (cm)0
         -0.182950       -0.293181
1          0.930661        0.737246
2          1.042022        1.638870
3          0.652258        0.350836
4          1.097702        0.737246
Accuracy of our SVM model on the training data is 0.9523809523809523
Accuracy of our SVM model on the test data is 0.9777777777777777
```



Result:

Thus the machine learning model was created using Support Vector Machinealgorithm.

Ex.No:8	Implementation Of Ensembling Techniques
Date:	

Aim:

To implement the ensembling technique of Blending with the given Alcohol QCM Dataset.

Algorithm:

1. Split the training dataset into train, test and validation dataset.
2. Fit all the base models using train dataset.
3. Make predictions on validation and test dataset.
4. These predictions are used as features to build a second level model
5. This model is used to make predictions on test and meta-features.

Program:

```
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import
RandomForestRegressor import xgboost as xgb
from sklearn.linear_model
import LinearRegression f
rom sklearn.model_selection import train_test_split
df = pd.read_csv("train_data.csv")
target = df["target"] train = df.drop("target")
X_train, X_test, y_train, y_test = train_test_split(train, target, test_size=0.20)
 train_ratio = 0.70
validation_ratio = 0.20
test_ratio = 0.10
x_train, x_test, y_train, y_test = train_test_split( train, target, test_size=1 - train_ratio)
x_val, x_test, y_val, y_test = train_test_split(
    x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))
model_1 = LinearRegression()
model_2 = xgb.XGBRegressor()
model_3 =RandomForestRegressor()
model_1.fit(x_train, y_train)
val_pred_1 = model_1.predict(x_val)
test_pred_1 = model_1.predict(x_test)
val_pred_1 = pd.DataFrame(val_pred_1)
test_pred_1 = pd.DataFrame(test_pred_1)
model_2.fit(x_train, y_train)
val_pred_2 = model_2.predict(x_val)
test_pred_2 = model_2.predict(x_test)
val_pred_2 = pd.DataFrame(val_pred_2)
test_pred_2 = pd.DataFrame(test_pred_2)
model_3.fit(x_train, y_train)
val_pred_3 =
model_1.predict(x_val)
```

```
test_pred_3 = model_1.predict(x_test)
val_pred_3 = pd.DataFrame(val_pred_3) test_pred_3
= pd.DataFrame(test_pred_3)
df_val = pd.concat([x_val, val_pred_1, val_pred_2, val_pred_3], axis=1)
df_test = pd.concat([x_test, test_pred_1, test_pred_2, test_pred_3], axis=1)
final_model = LinearRegression()
final_model.fit(df_val, y_val)
final_pred = final_model.predict(df_test)
print(mean_squared_error(y_test, pred_final))
```

Output:

4790

Result:

Thus the program to implement ensembling technique of Blending with the given Alcohol QCM Dataset have been executed successfully and the output gotverfied.

Ex.No:9	Implementation Of Clustering Algorithms
Date:	

Aim:

To implment k-Nearest Neighbour algorithm to classify the Iris Dataset.

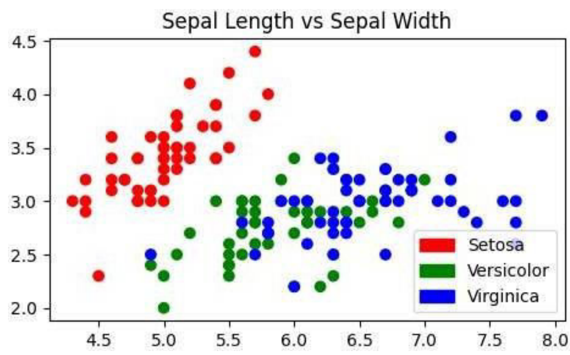
Algorithm:

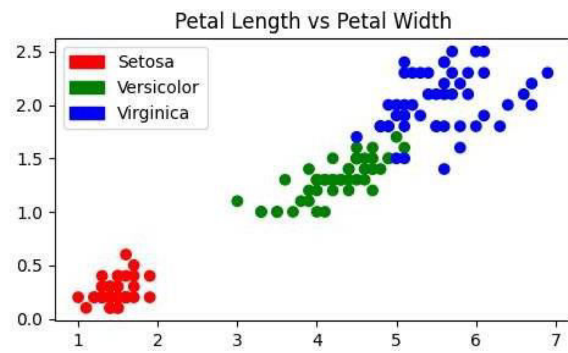
Step 1: Import necessary libraries
Step 2: Load iris dataset using datasets.load_iris() function and create a pandasdataframe named 'x' to store the data.
Step 3: Create a new pandas dataframe named 'y' to store the target variable.Step 4: Create a scatter plot with two subplots
Step 5: Create a KMeans model named 'iris_k_mean_model' with 3 clusters using KMeans(n_clusters=3)
Step 6: Fit the KMeans model on the iris data using iris_k_mean_model.fit(x)
Step 7: Print the cluster centers of the fitted KMeans model using iris_k_mean_model.cluster_centers_

Program:

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import sklearn.metrics as sm
iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns=['Sepal Length', 'Sepal Width', 'Petal Length', 'PetalWidth'])
y = pd.DataFrame(iris.target, columns=['Target'])
plt.figure(figsize=(12,3))
colors = np.array(['red', 'green', 'blue']) iris_targets_legend
= np.array(iris.target_names) red_patch =
mpatches.Patch(color='red', label='Setosa')
green_patch = mpatches.Patch(color='green', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')
plt.subplot(1, 2, 1)
plt.scatter(x['Sepal Length'], x['Sepal Width'], c=colors[y['Target']])
plt.title('Sepal Length vs Sepal Width') plt.legend(handles=[red_patch,
green_patch, blue_patch]) plt.subplot(1,2,2)
plt.scatter(x['Petal Length'], x['Petal Width'], c= colors[y['Target']])
plt.title('Petal Length vs Petal Width') plt.legend(handles=[red_patch,
green_patch, blue_patch]) iris_k_mean_model =
KMeans(n_clusters=3) iris_k_mean_model.fit(x)
print (iris_k_mean_model.cluster_centers_)
```

Output:





Result:

Thus the program to implement k-Nearest Neighbour Algorithm for clustering Iris dataset have been executed successfully and output gotverified.

Ex.No:10	Implement EM for Bayesian networks
Date:	

Aim:

To implement the EM algorithm for clustering networks using the given dataset.

Algorithm:

Step 1: Initialize θ randomly Repeat until convergence:
Step 2: E-step:
Compute q(h) = P(H = h | E = e; θ) for each h (probabilistic inference)
Create fully-observed weighted examples: (h, e) with weight q(h) Step
3: M-step:
Maximum likelihood (count and normalize) on weighted examples to get θ

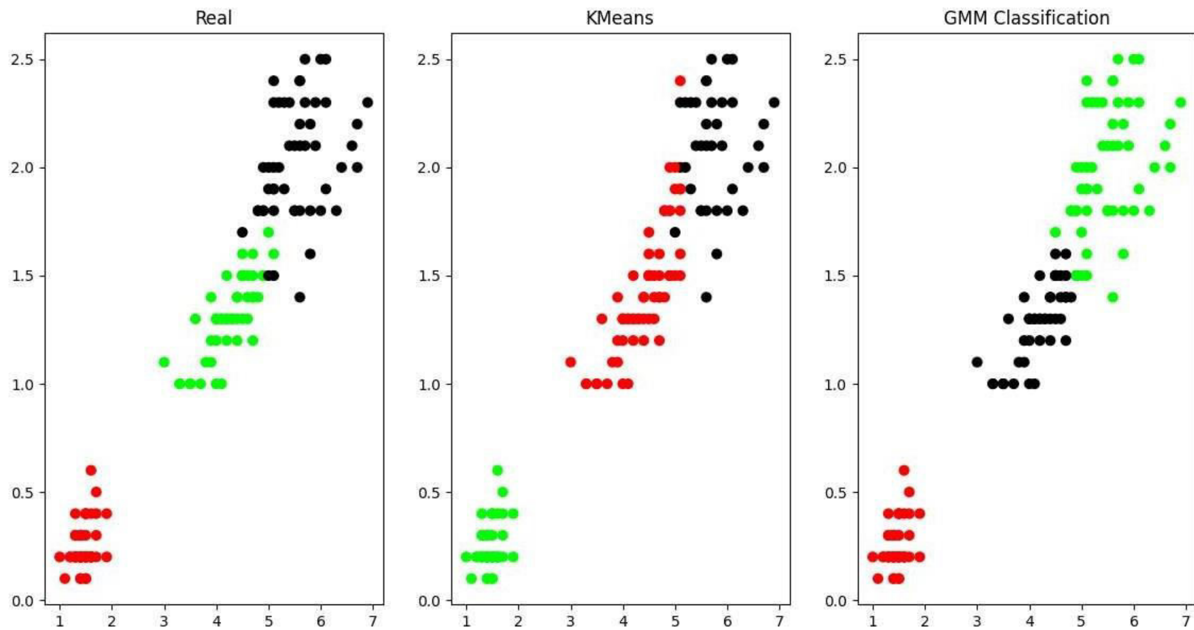
Program:

```
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')

# K-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:

Text(0.5, 1.0, 'GMM Classification')



Result:

Thus the program for Expectation Maximization Algorithm was executed and verified.

Ex.No:11	Simple NN Models
Date:	

Aim :

To implement the neural network model for the given numpy array.

Algorithm:

Step 1: Use numpy arrays to store inputs (x) and outputs (y)
Step 2: Define the network model and its arguments.
Step 3: Set the number of neurons/nodes for each layer
Step 4: Compile the model and calculate its accuracy
Step 5: Print a summary of the Keras model

Program:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
x = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
model.summary()
```

Output:

```
>>> # Print a summary of the Keras model:
>>> model.summary()
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 2)                 6
_________________________________________________________________
activation_7 (Activation)    (None, 2)                 0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 3
_________________________________________________________________
activation_8 (Activation)    (None, 1)                 0
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
```

Result:

Thus the program to implement the neural network model for the given dataset was executed and verified.

Ex.No:12	Deep Learning NN Models
Date:	

Aim:

To implement and build a Convolutional neural network model which predictsthe age and gender of a person using the given pre-trained models.

Algorithm:

Step-1: Import the required packages
Step-2: Load the MNIST dataset
Step-3: Normalize the training and test data
Step-4: Visualize the normalized firstimage in the training dataset
Step-5: Define the model architecture
Step-6: Compile the model
Step-7: Train the model
Step-8: Evaluate the model on the test data Model.
Step-9: Save the model
Step -10: Load the saved model
Step- 11: Make predictions on the test data usingthe loaded model
Step -12: Visualize the first test image and itspredicted label

Program:

```
import tensorflow.keras as keras
import tensorflow as tf
print(tf._version__)
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
import matplotlib.pyplot as plt

plt.imshow(x_train[0],cmap=plt.cm.binary)
plt.show()
print(y_train[0])
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
print(x_train[0])

plt.imshow(x_train[0],cmap=plt.cm.binary)
plt.show()
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

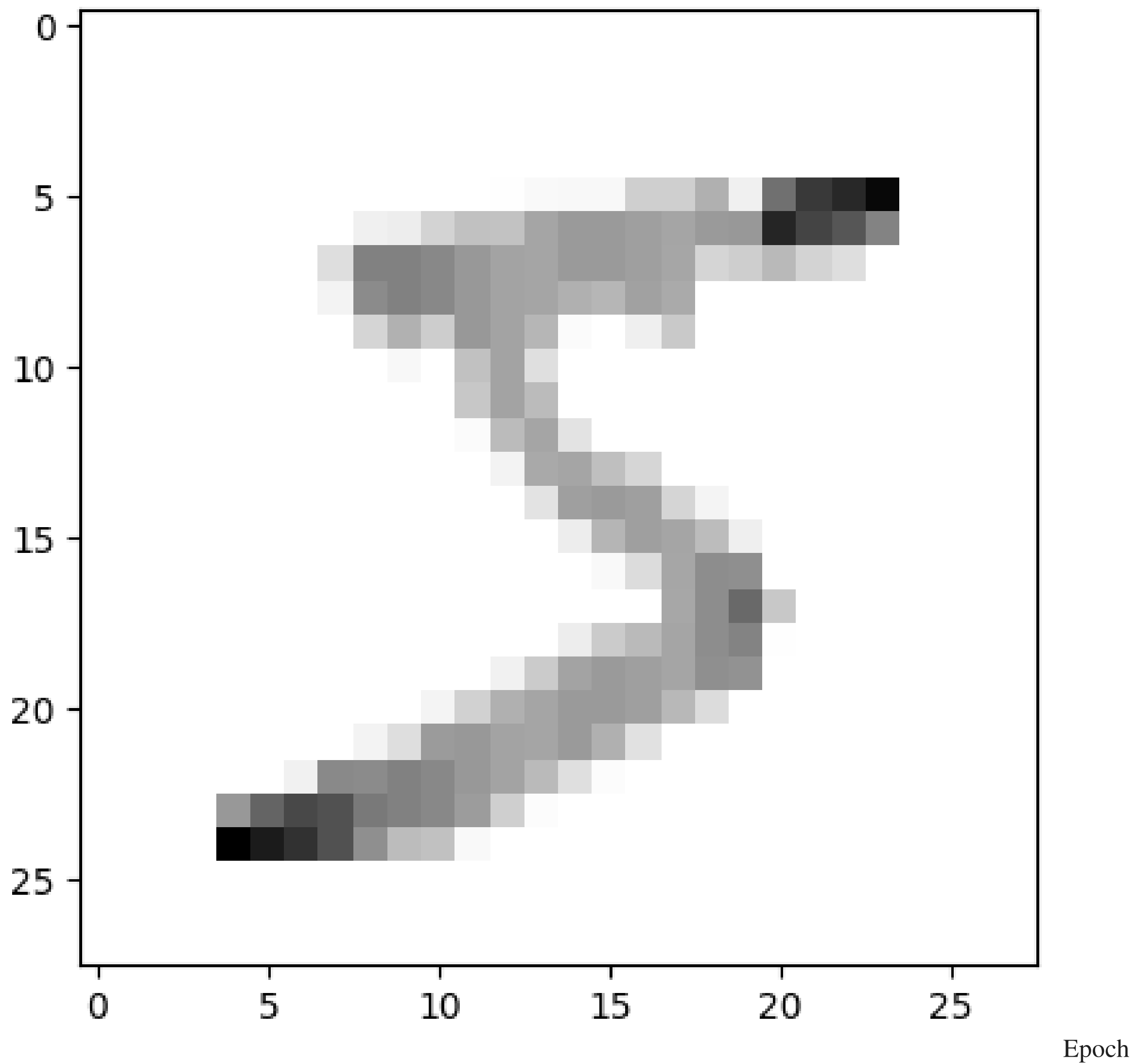
```
model.compile(optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']) model.fit(x_train, y_train, epochs=3)
val_loss, val_acc = model.evaluate(x_test, y_test)
print(val_loss)
print(val_acc)
model.save('epic_num_reader.model')
new_model = tf.keras.models.load_model('epic_num_reader.model')predictions
= new_model.predict(x_test)
import numpy as np

print(np.argmax(predictions[0]))
plt.imshow(x_test[0],cmap=plt.cm.binary)
plt.show()
```

Output:

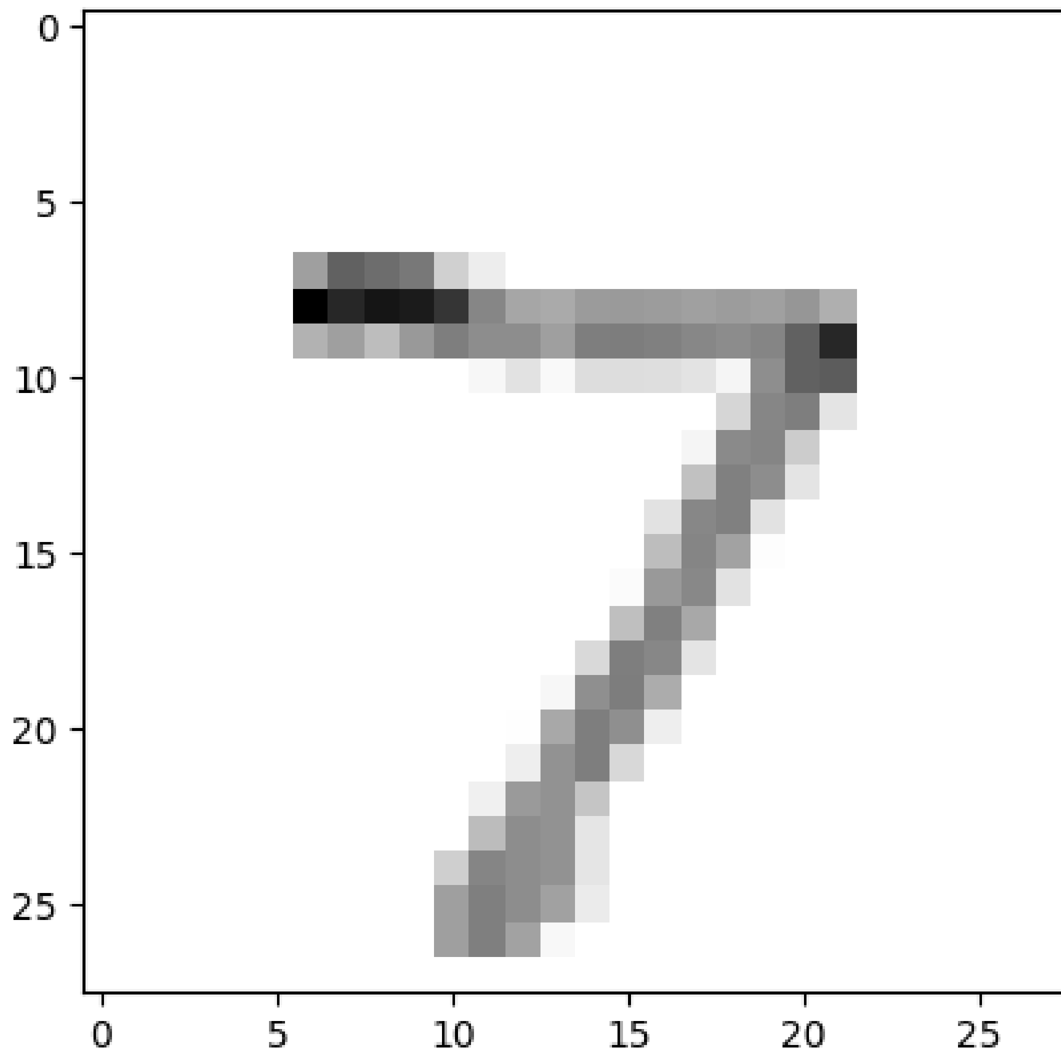
```
5
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
    0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
    0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
    0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
  0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
    0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
    0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.04500225 0.4219755  0.45852825 0.43408872 0.37314701
```

```
 0.33153488 0.32790981 0.28826244 0.26543758 0.34149427 0.31128482
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.1541463  0.28272888 0.18358693 0.37314701
 0.33153488 0.26569767 0.01601458 0.         0.05945042 0.19891229
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.0253731  0.00171577 0.22713296
 0.33153488 0.11664776 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.20500962
 0.33153488 0.24625638 0.00291174 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.01622378
 0.24897876 0.32790981 0.10191096 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.04586451 0.31235677 0.32757096 0.23335172 0.14931733 0.00129164
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.10498298 0.34940902 0.3689874  0.34978968 0.15370495
 0.04089933 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.06551419 0.27127137 0.34978968 0.32678448
 0.245396   0.05882702 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.02333517 0.12857881 0.32549285
 0.41390126 0.40743158 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.32161793
 0.41390126 0.54251585 0.20001074 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.06697006 0.18959827 0.25300993 0.32678448
 0.41390126 0.45100715 0.00625034 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.05110617 0.19182076 0.33339444 0.3689874  0.34978968 0.32678448
 0.40899334 0.39653769 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.04117838 0.16813739
 0.28960162 0.32790981 0.36833534 0.3689874  0.34978968 0.25961929
 0.12760592 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.04431706 0.11961607 0.36545809 0.37314701
 0.33153488 0.32790981 0.36833534 0.28877275 0.111988              0.00258328
 0.
 0.

[0.          0.          0.          0.          0.          0.
  0.05298497  0.42752138  0.4219755   0.45852825  0.43408872  0.37314701
  0.33153488  0.25273681  0.11646967  0.01312603  0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]
[0.          0.          0.          0.          0.37491383  0.56222061
  0.66525569  0.63253163  0.48748768  0.45852825  0.43408872  0.359873
  0.17428513  0.01425695  0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]
[0.          0.          0.          0.          0.92705966  0.82698729
  0.74473314  0.63253163  0.4084877   0.24466922  0.22648107  0.02359823
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]
[0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]
[0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]
[0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]]
```



Epoch
1/3
1875/1875 [==============================] - 18s 8ms/step - loss: 0.2588 -
accuracy: 0.9252Epoch
2/3
1875/1875 [==============================] - 16s 9ms/step - loss: 0.1055 -
accuracy: 0.9679Epoch
3/3
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0723 -
accuracy: 0.9773
313/313 [==============================] - 2s 4ms/step - loss: 0.1149 -
accuracy: 0.9651
0.11487378180027008
0.9650999903678894



Result:

Thus the program to implement and build a Convolutional neural network model whichhave been executed successfully and the output got verified.

lOMoARcPSD|21772091