DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082
(Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Dell
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi
Accredited by NAAC, A+)

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

VII Semester
Course Code: 18CSL76

[As per the Choice Based Credit System Scheme]
Scheme: 2018

Version 3.1
w.e.f 11th September 2023

Editorial Committee
Artificial Intelligence and Machine Learning Lab Faculty, Dept. of CSE

Approved by
H.O.D, Department of CSE

Document Log

Name of the document	Artificial Intelligence and Machine Learning Lab Manual
Scheme	2018
Current version number and date	V 3.1 / 11.09.2023
Subject code	18CSL76
Editorial Committee	Prof. Manasa Sandeep Prof. Chethana V Prof. Shylaja B
Computer Programmer	Mr. Dijin Ramesh K P
Approved by	HOD, Dept. of CSE

Table of Contents

Vision of the Department

Epitomize CSE graduate to carve a niche globally in the field of computer science to excel in the world of information technology and automation by imparting knowledge to sustain skills for the changing trends in the society and industry.

Mission of the Department

M1: To educate students to become excellent engineers in a confident and creative environment through world-class pedagogy.

M2: Enhancing the knowledge in the changing technology trends by giving hands-on experience through continuous education and by making them to organize & participate in various events.

M3: Impart skills in the field of IT and its related areas with a focus on developing the required competencies and virtues to meet the industry expectations.

M4: Ensure quality research and innovations to fulfill industry, government & social needs.

M5: Impart entrepreneurship and consultancy skills to students to develop self-sustaining life skills in multi-disciplinary areas.

Program Educational Objectives (PEOs)

After the course completion, CSE graduates will be able to:

PEO1: Engage in professional practice to promote the development of innovative systems and optimized solutions for Computer Science and Engineering.

PEO2: Adapt to different roles and responsibilities in multidisciplinary working environment by respecting professionalism and ethical practices within organization and society at national and international level.

PEO3: Graduates will engage in life-long learning and professional development to acclimate the rapidly changing work environment and develop entrepreneurship skills.

Program Specific Outcomes (PSO)

PSO1: Foundation of Mathematical Concepts: Ability to use mathematical methodologies to solve the problem using suitable mathematical analysis, data structure and suitable algorithm.

PSO2: Foundation of Computer System: Ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.

PSO3: Foundations of Software Development: Ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research.

PSO4: Foundations of Multi-Disciplinary Work: Ability to acquire leadership skills to perform professional activities with social responsibilities, through excellent flexibility to function in multi-disciplinary work environment with self-learning skills.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Details

Subject Code	**: 18CSL76**	**IA Marks**	**: 40**
No. of Practical Hrs/ Week	**: 0:0:2**	**Exam Hours**	**: 03**
Total No. of Practical Hrs	**: 36**	**Exam Marks**	**: 60**

Course Outcomes: At the end of the Course, the Student will be able to:

CO 1	Apply appropriate Machine Learning Algorithms for given Datasets
CO 2	Evaluate Machine Learning Algorithms using different parameters
CO 3	Implement Machine learning programs for Supervised and Unsupervised learning models

Syllabus

Laboratory Experiments:

Implement the following programs using Python Programming Language

1. Implement A* Search algorithm.
2. Implement AO* Search algorithm.
3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a CSV file. Compute the accuracy of the classifier, considering few test data sets.

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

8. Write a program to implement k-Nearest 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.

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

Conduction of Practical Examination:

Experiment distribution

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

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

For laboratories having PART A and PART B

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

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

CHAPTER 1
INTRODUCTION

Artificial Intelligence (AI)

Artificial Intelligence (AI) is a branch of science which deals with helping machines find solutions to complex problems in a more human-like fashion. Artificial is defined in different approaches by various researchers during its evolution, such as “Artificial Intelligence is the study of how to make computers do things which at the moment, people do better.” There are other possible definitions “AI is a collection of hard problems which can be solved by humans and other living things, but for which we don’t have good algorithms for solving.”

Machine learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.

Machine learning tasks Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

1. Supervised Learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback:

2. Semi-supervised Learning: the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

3. Unsupervised Learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

4. Reinforcement Learning: training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

AI Algorithms		**Un Supervised Learning**	
1	A* Search algorithm	1	EM Algorithm
2	AO* Search algorithm.	2	k-Means Algorithm
Supervised Learning		**Instance Based Learning**	
1	Candidate-Elimination Algorithm	1	Locally Weighted Regression Algorithm
2	Decision tree - ID3 Algorithm		
3	Backpropagation Algorithm		
4	Naïve Bayesian Algorithm		
5	K-Nearest Neighbor Algorithm		

Machine Learning Applications

- In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised manner. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
- In regression, also a supervised problem, the outputs are continuous rather than discrete.
- In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- Density estimation finds the distribution of inputs in some space.
- Dimensionality reduction simplifies inputs by mapping them into a lower dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked with finding out which documents cover similar topics.

CHAPTER 2

INSTALLATION PROCEDURE OF THE REQUIRED SOFTWARE

Software Requirements:

- **Anaconda:** An Integrated Development Environment (IDE) for Python Programming

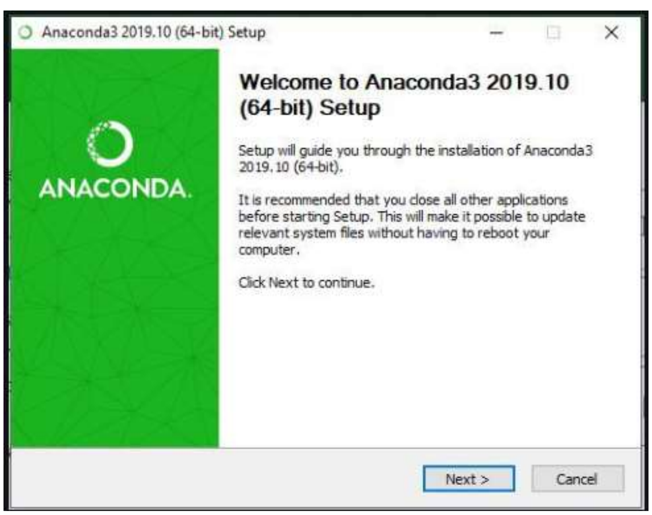
  Anaconda is an open-source software that contains Jupyter, spyder, etc that are used for large data processing, data analytics, heavy scientific computing. It contains a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions. Anaconda is a new distribution of the Python and R data science package. It was formerly known as Continuum Analytics. Anaconda has more than 100 new packages. Anaconda is used for scientific computing, data science, statistical analysis, and machine learning. Package versions are managed by the package management system called conda.

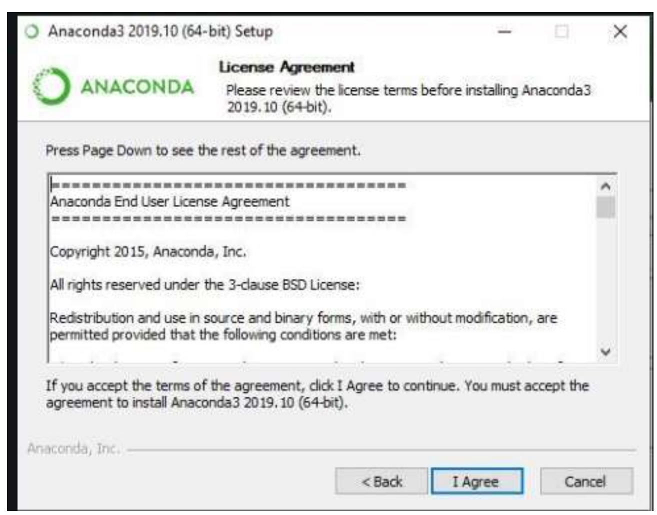
  Download and install Anaconda:

- Go to anaconda.com and install the latest version of Anaconda. Make sure to download the “Python 3.7 Version” for the appropriate architecture.

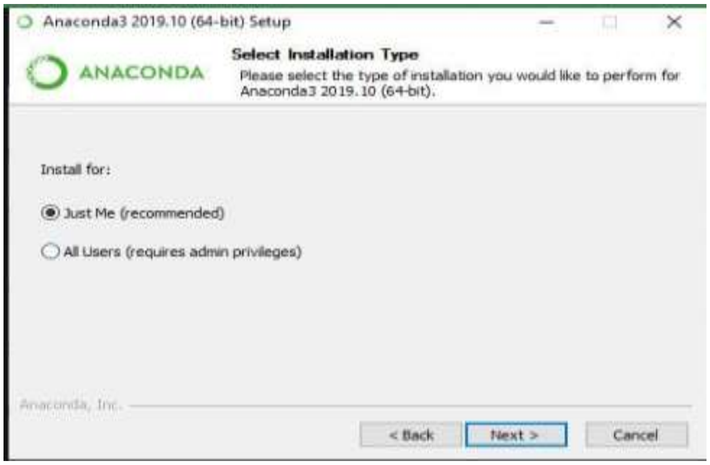
  Begin with the installation process:

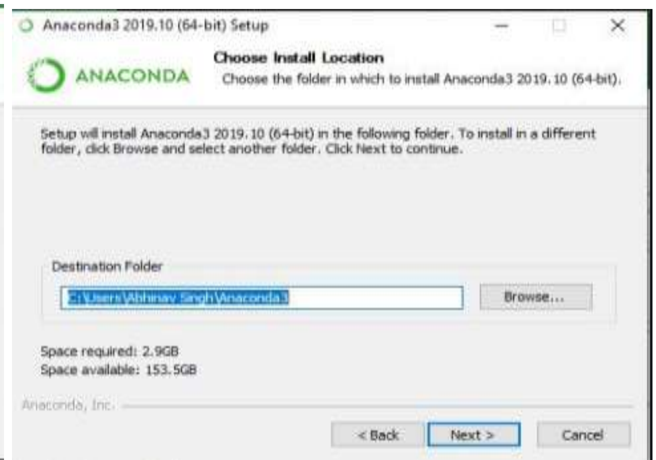
  Getting Started:





Select Installation Type: Select Just Me if you want the software to be used by a single User
Choose Installation Location:

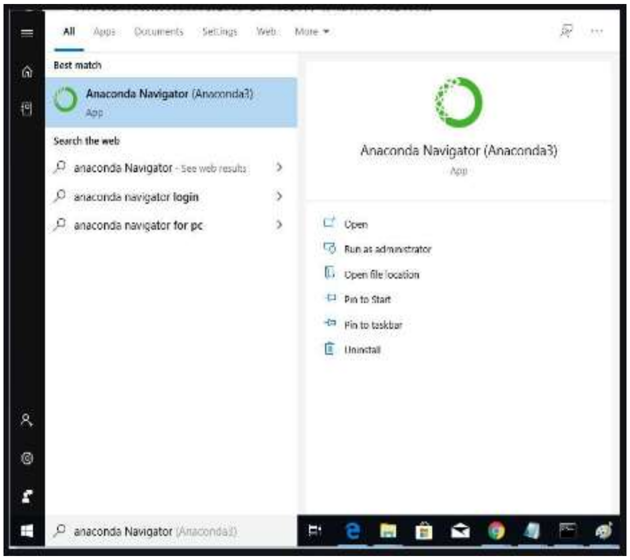


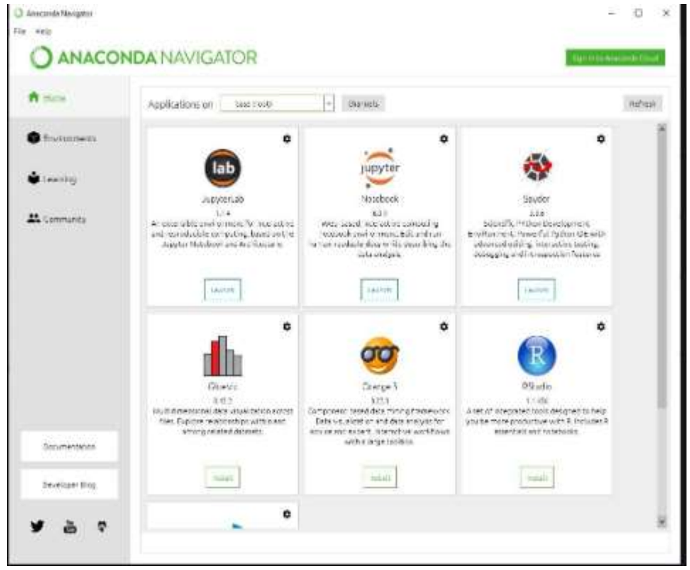


Finishing up the Installation:

Working with Anaconda:

Once the installation process is done, Anaconda can be used to perform multiple operations. To begin using Anaconda, search for Anaconda Navigator from the Start Menu in Windows

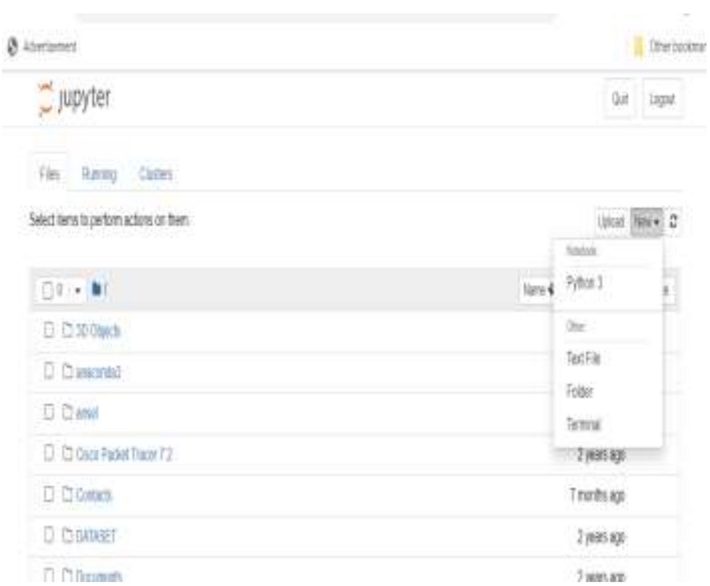


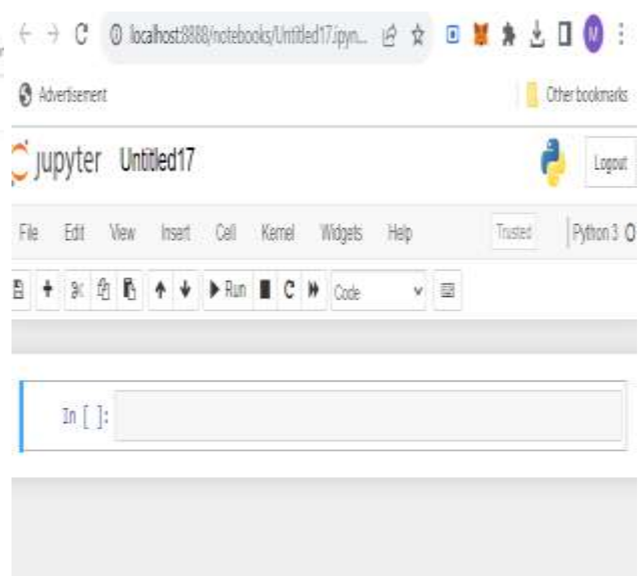


Launch Jupyter NoteBook Home

Create New Notebook by clicking on New → Python 3

Type your Program and Run ()





CHAPTER 3

SAMPLE PROGRAMS

1. Write a python program to calculate the area of square. Print the results. Take input from user.

```
side=int(input(“'enter side value of the square: ”))
sqarea= side*side
print('side value of the square is :'+ str(side))
print('area of the sqaure is :' + str(sqarea))
```

Output:
Enter Side Value of the Square : 5

Side Value of the Square is :5
Area of the sqaure is :25

2. Write a python program to check whether a given number is even or odd.

```
print("Enter the Number: ")
num = int(input())
if num%2==0:
   print("\nIt is an Even Number")
else:
   print("\nIt is an Odd Number")
```

Output:

```
Enter the Number:
5

It is an Odd Number
```

3. Write a python Program to print half Diamond star pattern

```
def halfDiamondStar(N):
   for i in range(N):
      for j in range(0, i + 1):
         print("*", end = " ")
      print()

   for i in range(1, N):
      for j in range(i, N):
```

Output:

```
Enter the value of N: 6
*
**
***
****
*****
******
*****
****
***
**
*
```

```
        print("*", end = "")
    print()

N=int(input("Enter the value of N"))
halfDiamondStar(N)
```

4. **Write a Python program to interchange first and last elements in a list**

```
list1=[]
num=int(input("Enter the length of the list: "))
print('Enter the elements: ')
for i in range(0, num):
    ele= int(input())
    list1.append(ele)
list1[0], list1[-1] = list1[-1], list1[0]
print(list1)
```

Output:

```
Enter the length of the list: 5
Enter the elements:
6
8
4
2
3
[3, 8, 4, 2, 6]
```

5. Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 2000 and 2200 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.

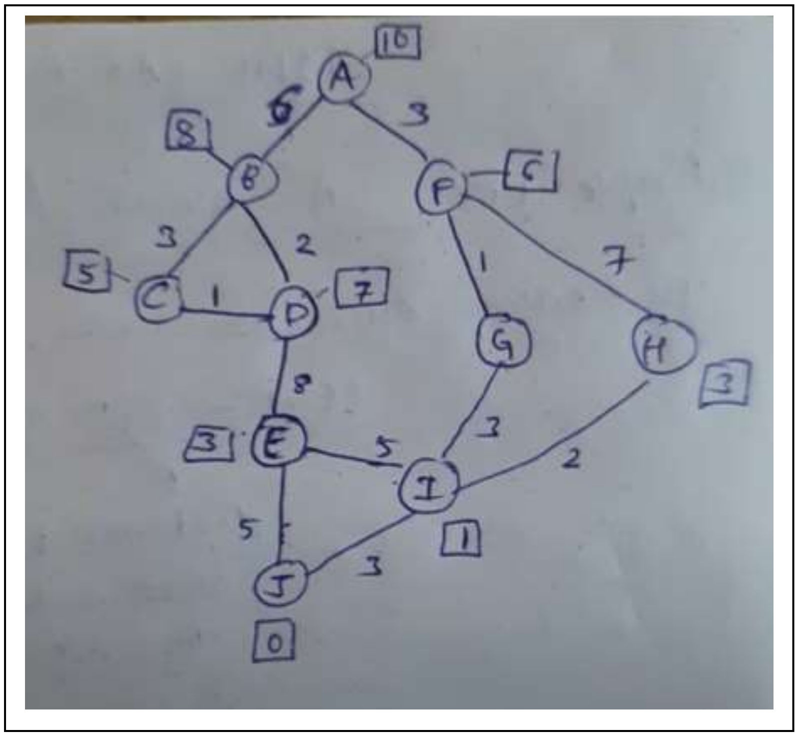
```
for i in range(2000, 2201):
    if (i%7==0) and (i%5!=0):
        print(i, end= ", ")
```

```
2002, 2009, 2016, 2023, 2037, 2044, 2051, 2058, 2072, 2079, 2086, 2093, 2107, 2114, 2121, 2128, 2142, 2149, 2156, 2163, 2177, 2
184, 2191, 2198,
```

CHAPTER 4

LAB SYLLABUS PROGRAMS

1. Implement A* Search algorithm




```
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)],

}

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

def h(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]

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] + h(v) < g[n] + h(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)
```

```python
            closed_set.add(n)

        print('Path does not exist!')
        return None

aStarAlgo('A', 'J')
```

Output:

```
Path found: ['A', 'F', 'G', 'I', 'J']
Out[1]: ['A', 'F', 'G', 'I', 'J']
```

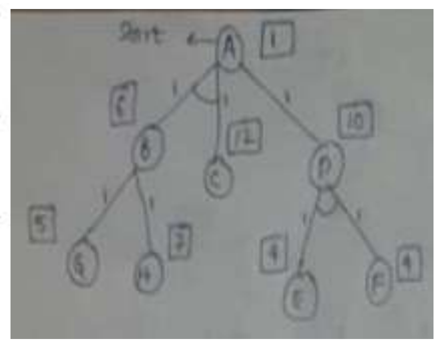
2. Implement AO* Search algorithm.

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):  #instantiate graph object with graph topology, heuristic values, st

        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

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

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

    def getStatus(self,v):          # return the status of a given node
        return self.status.get(v,0)  #GET IS INBUILT,RETURNS VALUE OF THE KEY. IF KEY NOT PRESENT THEN RETURN "SECOND PARAMETER"

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

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

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

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

    def computeMinimumCostChildNodes(self, v):  # Computes the Minimum Cost of child nodes of a given node v
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v):  # iterate over all the set of child node/s
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)

            if flag==True:                      # initialize Minimum Cost with the cost of first set of child node/s
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList      # set the Minimum Cost child node/s
                flag=False
            else:                               # checking the Minimum Cost nodes with the current Minimum Cost
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList  # set the Minimum Cost child node/s


        return minimumCost, costToChildNodeListDict[minimumCost]   # return Minimum Cost and Minimum Cost child node/s
```

```python
    def aoStar(self, v, backTracking):     # AO* algorithm for a start node and backTracking status flag

        print("HEURISTIC VALUES  :", self.H)
        print("SOLUTION GRAPH    :", self.solutionGraph)
        print("PROCESSING NODE   :", v)
        print("-----------------------------------------------------------------------------------------")

        if self.getStatus(v) >= 0:       # if status node v >= 0, compute Minimum Cost nodes of v(FOR START NODE, STATUS WILL BE
            minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList)) #THEN STATUS KEEPS UPDATING (HOW MANY TO VISIT(NO OF CHILDREN))

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

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


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

            if backTracking==False:      # check the current call is not for backtracking
                for childNode in childNodeList:   # for each Minimum Cost child node
                    self.setStatus(childNode,0)   # set the status of child node to 0(needs exploration)
                    self.aoStar(childNode, False) # Minimum Cost child node is further explored with backtracking status as false
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}  # Heuristic values of Nodes
graph2 = {                                        # Graph of Nodes and Edges
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],      # Neighbors of Node 'A', B, C & D with repective weights
    'B': [[('G', 1)], [('H', 1)]],                # Neighbors are included in a list of lists
    'D': [[('E', 1), ('F', 1)]]                   # Each sublist indicate a "OR" node or "AND" nodes
}

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

```
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : E
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : F
-----------------------------------------------------------------------------------------
```

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



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

```
import numpy as np
import pandas as pd

data = pd.read_csv('enjoysport.csv')
print(data)

concepts = np.array(data.iloc[: , 0:-1])
target = np.array(data.iloc[:, -1])

def learn(concepts, target):
    #Trying to find out the first YES row....
    for i, val in enumerate(target):
        if val == 'yes':
            break

    specific_h = concepts[i]. copy()

    generic_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':
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    specific_h[x] = '?'
                    generic_h[x][x] = '?'


        if target[i] == 'no':
            for x in range(len(specific_h)):
```

```
            if h[x]!=specific_h[x]:
                generic_h[x][x] = specific_h[x]
            else:
                generic_h[x][x] = '?'

    indices = [i for i, val in enumerate(generic_h) if val == ['?','?','?','?','?','?']]

    for i in indices:
        generic_h.remove(['?','?','?','?','?','?'])
    return specific_h, generic_h

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

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

Final S:
['sunny' 'warm' '?' 'strong' '?' '?']

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

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

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

data = pd.read_csv("play.csv")
features = [feat for feat in data]
features.remove("classification")
```

```
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["classification"] == "Yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
```

```
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["classification"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
```

printTree(root)

Output:

```
       A1    A2      A3 classification
0    True   Hot    High             No
1    True   Hot    High             No
2   False   Hot    High            Yes
3   False  Cool  Normal            Yes
4   False  Cool  Normal            Yes
5    True  Cool    High             No
6    True   Hot    High             No
7    True   Hot  Normal            Yes
8   False  Cool  Normal            Yes
9   False  Cool    High             No


                        A3
                                High
                                        A1
                                                False
                                                        A2
                                                                Cool ->  ['No']

                                                                Hot ->  ['Yes']

                                                True ->  ['No']

                                Normal ->  ['Yes']
```

5. 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) # maximum of X array longitudinally y = y/100
#Sigmoid Function
def sigmoid (x):
    return (1/(1 + np.exp(-x)))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
                                    #Variable initialization
epoch=7000                          #Setting training iterations
lr=0.1                              #Setting learning rate
inputlayer_neurons = 2              #number of features in data set
hiddenlayer_neurons = 3             #number of hidden layers neurons
output_neurons = 1                  #number of neurons at output layer
                                    #weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))

bout=np.random.uniform(size=(1,output_neurons))
        # draws a random range of numbers uniformly of dim x*y
#Forward Propagation
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
```

```
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)

#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop

    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
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:
 [[ 0.89559591]
 [ 0.88142069]
 [ 0.8928407 ]]
```

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

```
import csv
import random
import math

def loadCsv(filename):
  lines = csv.reader(open(filename, "r"));
  dataset = list(lines)
  for i in range(len(dataset)):
                                        #converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

  return dataset

def splitDataset(dataset, splitRatio):                    #67% training size
  trainSize = int(len(dataset) * splitRatio);
  trainSet = [ ]
  copy = list(dataset);
  while len(trainSet) < trainSize:

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

def separateByClass(dataset):
  separated = {}

#creates a dictionary of classes 1 and 0 where the values are the instances belonging to # each
class
  for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
              separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
  return separated

def mean(numbers):
  return sum(numbers)/float(len(numbers))

def stdev(numbers):
  avg = mean(numbers)
  variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
  return math.sqrt(variance)

def summarize(dataset):
```

```
  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
  del summaries[-1]
  return summaries

def summarizeByClass(dataset):
  separated = separateByClass(dataset);
  summaries = {}

                                                #summaries is a dic of tuples(mean,std) for each class value
  for classValue, instances in separated.items():
          summaries[classValue] = summarize(instances)
  return summaries

def calculateProbability(x, mean, stdev):
  exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
  return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
  probabilities = {}

                                                #class and attribute information as mean and sd
   for classValue, classSummaries in summaries.items():
          probabilities[classValue] = 1
          for i in range(len(classSummaries)):
                  mean, stdev = classSummaries[i]
                                                #take mean and sd of every attribute
          for class 0 and 1 seperaely
                  x = inputVector[i]                    #testvector's first attribute
                  probabilities[classValue]  *=  calculateProbability(x,  mean,  stdev);
                                                                    #use normal dist
      return probabilities

def predict(summaries, inputVector):
  probabilities = calculateClassProbabilities(summaries, inputVector)
  bestLabel, bestProb = None, -1


  for classValue, probability in probabilities.items():
                                                #assigns that class which has he highest prob
          if bestLabel is None or probability > bestProb:
                  bestProb = probability
                  bestLabel = classValue
  return bestLabel

def getPredictions(summaries, testSet):
  predictions = []
  for i in range(len(testSet)):
          result = predict(summaries, testSet[i])
```

```
            predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
            if testSet[i][-1] == predictions[i]:
                    correct += 1
    return (correct/float(len(testSet))) * 100.0


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

main()
```

Output

Split 767 rows into train=513 and test=254 rows
Accuracy of the classifier is : 67.32283464566929%

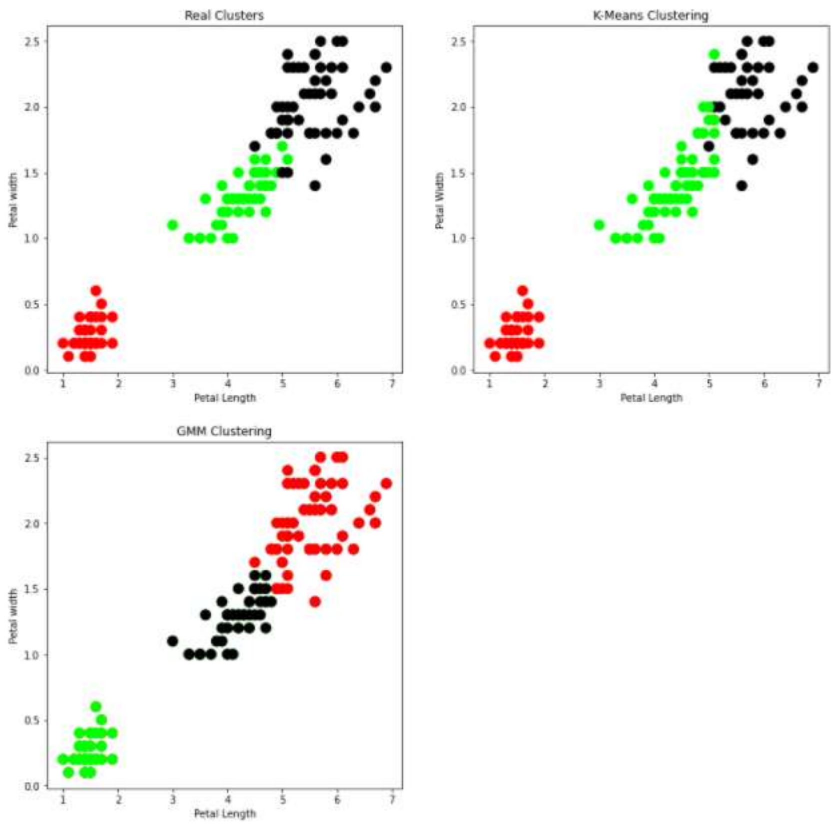
7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using *k*-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

iris=datasets.load_iris()
X=pd.DataFrame(iris.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y=pd.DataFrame(iris.target)
y.columns=['Targets']

model=KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,14))
colormap=np.array(['red','lime','black'])
plt.subplot(2,2,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal width')
plt.subplot(2,2,2)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_],s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.subplot(2,2,2)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_],s=40)
plt.title('K-Means Clustering')
plt.ylabel('Petal Width')

from sklearn import preprocessing
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)

from sklearn.mixture import GaussianMixture
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y=gmm.predict(xs)
plt.subplot(2,2,3)
```

```
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[gmm_y],s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal width')
print('Observation:The GMM using EM algo based clustering matched the true labels more closely than KMeans.')
```

<u>output</u>

```
Observation:The GMM using EM algo based clustering matched the true labels more closely than KMeans.
```



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

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset = load_iris()
#print(iris_dataset)
targets = iris_dataset.target_names
print("Class : number")
for i in range(len(targets)):
   print(targets[i], ':', i)
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"])
kn = KNeighborsClassifier(1)
kn.fit(X_train, y_train)
for i in range(len(X_test)):
   x_new = np.array([X_test[i]])
   prediction = kn.predict(x_new)
   print("Actual:[{0}] [{1}],Predicted:{2} {3}".format(y_test[i], targets[y_test[i]], prediction,
targets[prediction]))
print("\nAccuracy:",kn.score(X_test,y_test))
```

<u>Output:</u>

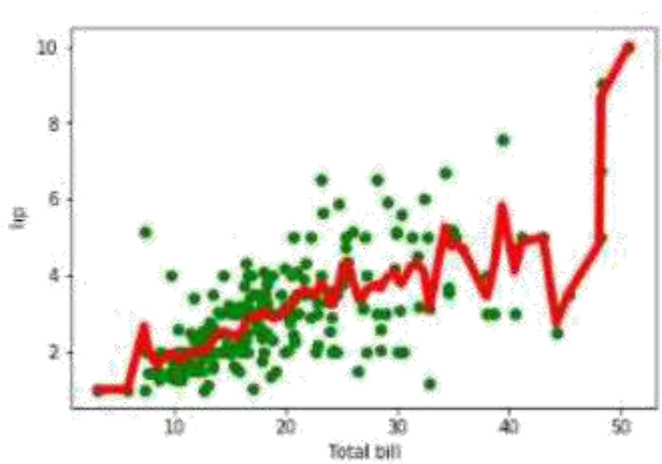
```
Class : number
setosa : 0
versicolor : 1
virginica : 2
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']

Accuracy:  1.0
```

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

```
from numpy import *

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


def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]

        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W=(X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
                                            # load data points
data = pd.read_csv('data10.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
                                       #preparing and add 1 in bill
mbill = np1.mat(bill)
```

```
mtip = np1.mat(tip)

m= np1.shape(mbill)[1]
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T))                                  #set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

Output



CHAPTER 5

VIVA QUESTIONS

1. What is machine learning?
2. Define supervised learning
3. Define unsupervised learning
4. Define semi supervised learning
5. Define reinforcement learning
6. What do you mean by hypotheses?
7. What is classification?
8. What is clustering?
9. Define precision, accuracy and recall
10. Define entropy
11. Define regression
12. How Knn is different from k-means clustering?
13. What is concept learning
14. Define specific boundary and general boundary
15. Define target function
16. Define decision tree
17. What is ANN
18. Explain gradient descent approximation
19. State Bayes theorem
20. Define Bayesian belief network
21. Differentiate hard and soft clustering
22. Define variance
23. What is inductive machine learning?
24. Why K nearest neighbour algorithm is lazy learning algorithm?
25. Why naïve Bayes is naïve?
26. Mention classification algorithms
27. Define pruning
28. Differentiate Clustering and classification
29. Mention clustering algorithms
30. Define Bias
31. What is learning rate? Why it is need.
32. What is the Difference Between Supervised and Unsupervised Machine Learning?
33. How machine learning is different from general programming?
34. What is Instance Based Learning?
35. What is Activation Function?
36. What is Sigmoid?
37. What is Gradient Descent?
38. What is Gibbs Algorithm
39. What is Q-Learning
40. What Heuristic search techniques
41. What is A* algorithm

42. Explain AO* Algorithm
43. Explain Water Jug Probe
44. Explain Hill Climbing
45. Explain BFS
46. Explain Heuristic Technique
47. Explain Generate and Test Method
48. What do you mean by Predicate Logic
49. Give the drawbacks of hill Climbing?
50. Differentiate between Simple Hill Climbing and Steepest-Ascent Hill Climbing