VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI-590018, KARNATAKA

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

18CSL76

Manual Prepared by

Dr. Manjunath R, Professor & Head of CSE Department
Prof. Veena V, Assistant Professor
Prof. Shruthi S, Assistant Professor

Academic Year-2023-24

P K M Educational Trust ®
R.R. INSTITUTE OF TECHNOLOGY
(Affiliated to VTU, Belagavi I Approved by AICTE , New Delhi & Government of Karnataka)

Department of Computer Science & Engineering
R.R. Institute of Technology

College Vision	"To be a Premier globally recognized Institute with ensuring academic excellence, Innovation and fostering Research in the field of Engineering"
College Mission	• To consistently strive for Academic Excellence • To promote collaborative Research & Innovation • To create holistic teaching learning environment that build ethically sound manpower who contribute to the stake holders operating at Global environment

Department Vision	To arise as an excellent learning center in the field of Computer Science & Engineering by fostering a skilled, innovative professionals to build a strong nation.
Department Mission	• To provide a conceptual foundation that caters the career required to adopt for changing technology in computer science. • To bridge the gap between academics and the latest tools, technologies in the area of hardware and software. • To set out co-curricular open doors for student's participation in advancements and recent trends. . To explore the potential and excel the students towards research to attain Novelty.

Program Educational Objectives (PEOs)	PEO1: Proficient to recognize contemporary issues and provide solutions using broad knowledge of computer science. PEO2: Ability to plan, analyze, design, evolve project implementing capabilities and skills in IT industry. PEO3: Drive to adapt new computing technologies lifelong to acquire professional greatness. PEO4: Possess professional, ethical, social responsibilities, communicational skills and team work needed for a successful professional carrier.

Program Specific Outcomes (PSOs)	PSO1: Apply the software practices, principals to design and analyse the complex computer based system. PSO2: Design, implement and validate system software and application software to the various societal needs.

Program Outcomes (POs)

PO1	**Engineering Knowledge:** Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.
PO2	**Problem Analysis:** Identify, Formulate, review research literature and analyse complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3	**Design/Development of Solutions:** Design solutions for complex engineering problems related to CSE 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.
PO4	**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.
PO5	**Modern Tool Usage:** Create, Select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to computer science related complex engineering activities with an understanding of the limitations.
PO6	**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 CSE professional engineering practice.
PO7	**Environment and Sustainability:** Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply Ethical Principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and Team Work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary Settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as able to comprehend and with write effective reports and design documentation, make effective presentations and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
PO12	**Life-Long Learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning the broadest context of technological change.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY (Effective from the academic year 2018 -2019) SEMESTER – VII			
Course Code	**18CSL76**	**CIE Marks**	40
Number of Contact Hours/Week	0:0:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03
Credits – 2			

Course Learning Objectives: This course (18CSL76) will enable students to:

- Implement and evaluate AI and ML algorithms in and Python programming language.

Descriptions (if any):

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

1.	Implement A* Search algorithm.
2.	Implement AO* Search algorithm.
3.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
4.	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify 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 Regressionalgorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Laboratory Outcomes: The student should be able to:

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

Conduct of Practical Examination:

- Experiment distribution
    - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
    - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
    q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
    r) 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

Table of Contents

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:

- **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.

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

- **Active learning:** The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling.

- **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.

- **Unsupervised learning:** No labels are given to the learning algorithm, leaving it units 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)

<table>
<tr><th>Supervised learning</th><th>Un Supervised learning</th><th>Instance based learning</th></tr>
<tr><td>Find-s algorithm</td><td>EM algorithm</td><td rowspan="6">Locally weighted Regression algorithm</td></tr>
<tr><td>Candidate elimination algorithm</td><td rowspan="5">K means algorithm</td></tr>
<tr><td>Decision tree algorithm</td></tr>
<tr><td>Back propagation Algorithm</td></tr>
<tr><td>Naïve Bayes Algorithm</td></tr>
<tr><td>K nearest neighbor algorithm(lazy learning algorithm)</td></tr>
</table>

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.

Machine learning Approaches

- **Decision tree learning**
  Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.

- **Association rule learning**
  Association rule learning is a method for discovering interesting relations between variables in large databases.

- **Artificial neural networks**
  An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools.

- **Deep learning**
  Falling hardware prices and the development of GPUs for personal use in the last few years have contributed to the development of the concept of deep learning which consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.

- **Inductive logic programming**
  Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

- **Support vector machines**
  Support vector machines (SVMs) are a set of related supervised learning methods used for classification

and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

- **Clustering**
  Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some pre designated criterion or criteria, while observations drawn from different clusters are dissimilar. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

- **Bayesian networks**
  A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

- **Reinforcement learning**
  Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

- **Similarity and metric learning**
  In this problem, the learning machine is given pairs of examples that are considered similar and pairs of less similar objects. It then needs to learn a similarity function (or a distance metric function) that can predict if new objects are similar. It is sometimes used in Recommendation systems.

- **Genetic algorithms**
  A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s. Conversely, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms.

- **Rule-based machine learning**
  Rule-based machine learning is a general term for any machine learning method that identifies, learns, or evolves "rules" to store, manipulate or apply, knowledge. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

PROGRAM-1

Implement A* Searchalgorithm.

```
class Graph:
   def__init_(self,adjac_lis):
self.adjac_lis = adjac_lis
   def get_neighbours(self,v):
      return self.adjac_lis[v]
   def h(self,n):
      H={'A':1,'B':1, 'C':1,'D':1}
      return H[n]
   def a_star_algorithm(self,start,stop):
open_lst = set([start])
closed_lst = set([])
dist={}
dist[start] = 0
prenode={}
prenode[start] =start
      while len(open_lst)>0:
         n = None
         for v in open_lst:
            if n==None or dist[v]+self.h(v)<dist[n]+self.h(n):
               n=v;
         if n==None:
print("path doesnot exist")
            return None
         if n==stop:
reconst_path=[]
             while prenode[n]!=n:
reconst_path.append(n)
                n = prenode[n]
reconst_path.append(start)
reconst_path.reverse()
print("path found:{}".format(reconst_path))
             return reconst_path
         for (m,weight) in self.get_neighbours(n):
            if m not in open_lst and m not in closed_lst:
open_lst.add(m)
prenode[m] = n
dist[m] = dist[n]+weight
            else:
               if dist[m]>dist[n]+weight:
dist[m] = dist[n]+weight
prenode[m]=n
                  if m in closed_lst:
closed_lst.remove(m)
open_lst.add(m)
```

```
open_lst.remove(n)
closed_lst.add(n)
print("Path doesnot exist")
        return None
adjac_lis ={'A':[('B',1),('C',3),('D',7)],'B':[('D',5)],'C':[('D',12)]}
graph1=Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```

OUTPUT

Path found : [‘A’, ‘B’, ‘D’]
OUT[2]: [‘A’, ‘B’, ‘D’]

PROGRAM-2

Implement AO* Algorithm.

```
def recAOStar(n):
   global finalPath
print("Expanding Node:",n)
and_nodes = []
or_nodes=[]
if(n in allNodes):

       if 'AND' in allNodes[n]:
and_nodes = allNodes[n]['AND']
       if 'OR' in allNodes[n]:
or_nodes = allNodes[n]['OR']
   if len(and_nodes)==0 and len(or_nodes)==0:
       return


   solvable = False
   marked ={}


   while not solvable:

       if len(marked)==len(and_nodes)+len(or_nodes):

min_cost_least,min_cost_group_least =
least_cost_group(and_nodes,or_nodes,{})
          solvable = True
change_heuristic(n,min_cost_least)
optimal_child_group[n] = min_cost_group_least
          continue
```

```
min_cost,min_cost_group = least_cost_group(and_nodes,or_nodes,marked)

is_expanded = False

    if len(min_cost_group)>1:

      if(min_cost_group[0] in allNodes):

is_expanded = True

recAOStar(min_cost_group[0])

      if(min_cost_group[1] in allNodes):

is_expanded = True

recAOStar(min_cost_group[1])

    else:

if(min_cost_group in allNodes):

is_expanded = True

recAOStar(min_cost_group)

    if is_expanded:

min_cost_verify, min_cost_group_verify = least_cost_group(and_nodes,
or_nodes, {})

      if min_cost_group == min_cost_group_verify:

        solvable = True

change_heuristic(n, min_cost_verify)

optimal_child_group[n] = min_cost_group

    else:

      solvable = True

change_heuristic(n, min_cost)

optimal_child_group[n] = min_cost_group

    marked[min_cost_group]=1

  return heuristic(n)


def least_cost_group(and_nodes, or_nodes, marked):
```

```
node_wise_cost = {}
  for node_pair in and_nodes:
    if not node_pair[0] + node_pair[1] in marked:
      cost = 0
      cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
node_wise_cost[node_pair[0] + node_pair[1]] = cost
  for node in or_nodes:
    if not node in marked:
      cost = 0
      cost = cost + heuristic(node) + 1
node_wise_cost[node] = cost
min_cost = 999999
min_cost_group = None
  for costKey in node_wise_cost:
    if node_wise_cost[costKey] <min_cost:
min_cost = node_wise_cost[costKey]
min_cost_group = costKey
  return [min_cost, min_cost_group]


def heuristic(n):
  return H_dist[n]


def change_heuristic(n, cost):
H_dist[n] = cost
  return


def print_path(node):
```

```
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]
    if len(node) > 1:
        if node[0] in optimal_child_group:
print("->",  end="")
print_path(node[0])
        if node[1] in optimal_child_group:
print("->",  end="")
print_path(node[1])
    else:
        if node in optimal_child_group:
print("->",  end="")
print_path(node)
H_dist = {
 'A': -1,
 'B': 4,
 'C': 2,
 'D': 3,
 'E': 6,
 'F': 8,
 'G': 2,
 'H': 0,
 'I': 0,
 'J': 0
}
allNodes = {
 'A': {'AND': [('C', 'D')], 'OR': ['B']},
```

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

OUTPUT

Expanding Node: A

Expanding Node: B

Expanding Node: C

Expanding Node: D

Nodes which gives optimal cost are

CD->HI->J

Optimal cost is :: 5

PROGRAM-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.DataFrame(data=pd.read_csv('finds.csv'))
concepts=np.array(data.iloc[:,0:-1])
target=np.array(data.iloc[:,-1])
def learn(concepts,target):
specific_h=concepts[0].copy()
general_h=[["?"for i in range(len(specific_h))]
for i in range(len(specific_h))]
    for i,h in enumerate(concepts):
        if target[i]=="Yes":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
specific_h[x]='?'
general_h[x][x]='?'
        if target[i]=="No":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
general_h[x][x]=specific_h[x]
                else:
general_h[x][x]='?'
    indices=[i for i,val in enumerate (general_h) if val==['?','?','?','?','?','?']]
    for i in indices:
general_h.remove(['?','?','?','?','?','?'])
    return specific_h,general_h
s_final,g_final=learn(concepts,target)
print("Final S:",s_final,sep="\n")
print("Final G:",g_final,sep="\n")
```

OUTPUT

```
[Sky,Airtemp,Humidity,Wind,Water,Forecast,WaterSport
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Cloudy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes]

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

PROGRAM-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.

```
print("/nThe Resultant Decesion Tree is :\n")
pprint(tree)import pandas as pd
from pandas import DataFrame
df_tennis=DataFrame.from_csv('PlayTennis.csv')
df_tennis
def entropy(probs):
    import math
    return sum([-prob*math.log(prob,2)for prob in probs])
def entropy_of_list(a_list):
        from collections import Counter
cnt = Counter(x for x in a_list)
print("No and Yes Classes:",a_list.name,cnt)
num_instances=len(a_list)*1.0
        probs= [x/num_instances for x in cnt.values()]
        return entropy(probs)
total_entropy = entropy_of_list(df_tennis['PlayTennis'])
print("Entropy of given PlayTennis Data Set:",total_entropy)
def information_gain(df,split_attribute_name, target_attribute_name,
trace=0):
print("Information Gain Calculation of",split_attribute_name)
df_split = df.groupby(split_attribute_name)
    for name,group in df_split:
        print(name)
        nobs = len(df.index)*1.0
df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list,lambda
x:len(x)/nobs]})[target_attribute_name]
df_agg_ent.columns = ['Entropy','PropObservations']
new_entropy = sum(df_agg_ent['Entropy']*df_agg_ent['PropObservations'])
old_entropy = entropy_of_list(df[target_attribute_name])
    return old_entropy-new_entropy
print('Info-gain for Outlook is :'+str(information_gain(df_tennis,'Outlook',
'PlayTennis')),"\n")
print('\n Info-gain for Humidity is
:'+str(information_gain(df_tennis,'Humidity', 'PlayTennis')),"\n")
print('\n Info-gain for Wind is :'+str(information_gain(df_tennis,'Wind',
'PlayTennis')),"\n")
print('\n Info-gain for Temperature is
:'+str(information_gain(df_tennis,'Temperature', 'PlayTennis')),"\n")
def id3(df,target_attribute_name,atrribute_names,default_class=None):

            from collections import Counter
cnt = Counter(x for x in df[target_attribute_name])
```

```
        if len(cnt) == 1:
            return next(iter(cnt))

elifdf.empty or (not attribute_names):
            return default_class
        else:

default_class = max(cnt.keys())
gainz=[information_gain(df, attr, target_attribute_name) for attr in
attribute_names]
index_of_max = gainz.index(max(gainz))
best_attr = attribute_names[index_of_max]

                tree = {best_attr:{}}
remaining_attribute_names = [i for i in attribute_names if i != best_attr]

                for attr_val, data_subset in df.groupby(best_attr):
                    subtree =
id3(data_subset,target_attribute_name,remaining_attribute_names,default_c
lass)
                    tree[best_attr][attr_val] = subtree
        return tree
attribute_names = list(df_tennis.columns)
print("List of Attribute:", attribute_names)
attribute_names.remove('PlayTennis')
print("Predicting Attributes:", attribute_names)

from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
```

OUTPUT

No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5}) Entropy of given PlayTennis Data Set: 0.9402859586706309 Information Gain Calculation ofOutlook
Overcast Rain Sunny
No and Yes Classes: PlayTennisCounter({'Yes': 4})
No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 2}) No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5}) Info-gain for Outlook is :0.2467498197744391

Information Gain Calculation of Humidity High
Normal
No and Yes Classes: PlayTennisCounter({'No': 4, 'Yes': 3}) No and Yes Classes: PlayTennisCounter({'Yes': 6, 'No': 1}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5})

Info-gain for Humidity is: 0.15183550136234136 Information Gain

Calculation of Wind
Strong
Weak
No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 3}) No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 2}) No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})

Info-gain for Wind is:0.04812703040826927

Information Gain Calculation ofTemperature Cool
Hot Mild
No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 1}) No and Yes Classes: PlayTennisCounter({'No': 2, 'Yes': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 4, 'No': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5})

Info-gain for Temperature is:0.029222565658954647

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind'] Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind'] Information Gain Calculation ofOutlook
Overcast Rain Sunny
No and Yes Classes: PlayTennisCounter({'Yes': 4})
No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 2}) No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5}) Information Gain Calculation ofTemperature
Cool Hot Mild
No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 1}) No and Yes Classes: PlayTennisCounter({'No': 2, 'Yes': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 4, 'No': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5}) Information Gain Calculation ofHumidity
High Normal
No and Yes Classes: PlayTennisCounter({'No': 4, 'Yes': 3}) No and Yes Classes: PlayTennisCounter({'Yes': 6, 'No': 1}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5}) Information Gain Calculation ofWind
Strong Weak
No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 3}) No and Yes Classes: PlayTennisCounter({'Yes': 6, 'No': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 9, 'No': 5}) Information Gain Calculation ofTemperature
Cool Mild
No and Yes Classes: PlayTennisCounter({'Yes': 1, 'No': 1}) No and Yes Classes: PlayTennisCounter({'Yes': 2, 'No': 1}) No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 2}) Information Gain Calculation of Humidity

High Normal
No and Yes Classes: PlayTennisCounter({'Yes': 1, 'No': 1}) No and Yes Classes: PlayTennisCounter({'Yes': 2, 'No': 1}) No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 2}) Information Gain Calculation ofWind
Strong Weak
No and Yes Classes: PlayTennisCounter({'No': 2}) No and Yes Classes: PlayTennisCounter({'Yes': 3})
No and Yes Classes: PlayTennisCounter({'Yes': 3, 'No': 2}) Information Gain Calculation ofTemperature
Cool Hot Mild
No and Yes Classes: PlayTennisCounter({'Yes': 1}) No and Yes Classes: PlayTennisCounter({'No': 2})
No and Yes Classes: PlayTennisCounter({'No': 1, 'Yes': 1}) No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 2}) Information Gain Calculation ofHumidity
High Normal
No and Yes Classes: PlayTennisCounter({'No': 3}) No and Yes Classes: PlayTennisCounter({'Yes': 2})
No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 2}) Information Gain Calculation ofWind
Strong Weak
No and Yes Classes: PlayTennisCounter({'No': 1, 'Yes': 1}) No and Yes Classes: PlayTennisCounter({'No': 2, 'Yes': 1}) No and Yes Classes: PlayTennisCounter({'No': 3, 'Yes': 2})

The Resultant Decision Tree is :

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

PROGRAM-5

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

```
importnumpy as np X=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float) X=X/np.amax(X,axis=0)
y=y/100

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

defderivatives_sigmoid(x):
return x*(1-x)

epoch=7000 lr=0.1
inputlayer_neurons=2 hiddenlayer_neurons=3 output_neurons=1

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

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)

EO=y-output outgrad=derivatives_sigmoid(output) d_output=EO*outgrad
EH=d_output.dot(wout.T) hiddengrad=derivatives_sigmoid(hlayer_act)
d_hiddenlayer=EH*hiddengradwout+=hlayer_act.T.dot(d_output)*lr

wh+=X.T.dot(d_hiddenlayer)*lr

print("Input: \n"+str(X)) print("Actual Output: \n"+str(y)) print("Predicted
Output: \n",output)
```

OUTPUT

```
Input:
[[0.666666671.        ]
[0.33333333 0.55555556]
[1.  0.66666667]]
Actual Output: [[0.92]
[0.86]
[0.89]]
Predicted Output: [[0.78963493]
[0.77489109]
[0.79355268]]
```

PROGRAM-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.

```
importcsv import random import math
defloadCsv(filename):
lines = csv.reader(open(filename, "r")) dataset = list(lines)
for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]] return dataset

defsplitDataset(dataset, splitRatio):
trainSize = int(len(dataset) * splitRatio) trainSet = []
copy = list(dataset) whilelen(trainSet) <trainSize:
index = random.randrange(len(copy)) trainSet.append(copy.pop(index))
return [trainSet, copy]

defseparateByClass(dataset):
separated = {}
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))

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

def summarize(dataset):
summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)] del summaries[-1]
return summaries

defsummarizeByClass(dataset):
separated = separateByClass(dataset) summaries = {}
forclassValue, instances in separated.items(): summaries[classValue] =
summarize(instances)
return summaries

defcalculateProbability(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

defcalculateClassProbabilities(summaries, inputVector): probabilities = {}
forclassValue, classSummaries in summaries.items(): probabilities[classValue] =
            1
for i in range(len(classSummaries)): mean, stdev =classSummaries[i] x
            =inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
            return probabilities

            def predict(summaries, inputVector):
            probabilities = calculateClassProbabilities(summaries, inputVector)
            bestLabel, bestProb = None, -1
forclassValue, probability in probabilities.items(): ifbestLabel is None or
            probability >bestProb:
            bestProb = probability bestLabel = classValue
            returnbestLabel

defgetPredictions(summaries, testSet): predictions =[]
            for i in range(len(testSet)):
            result = predict(summaries, testSet[i]) predictions.append(result)
            return predictions

            defgetAccuracy(testSet, predictions):
            correct = 0
            for i in range(len(testSet)):
            print(testSet[i][-1]," ",predictions[i]) iftestSet[i][-1] == predictions[i]:
            correct += 1
            return (correct/float(len(testSet))) * 100.0

            def main():
            filename = 'pima-indians-diabetes.data.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))) summaries =
            summarizeByClass(trainingSet)
            predictions = getPredictions(summaries, testSet)
            accuracy = getAccuracy(testSet, predictions) print('Accuracy:
            {0}%'.format(accuracy)
```

<u>OUTPUT</u>

Split 768 rows into train=514 and test=254 rows 0.0 0.0
Accuracy: 0.39370078740157477

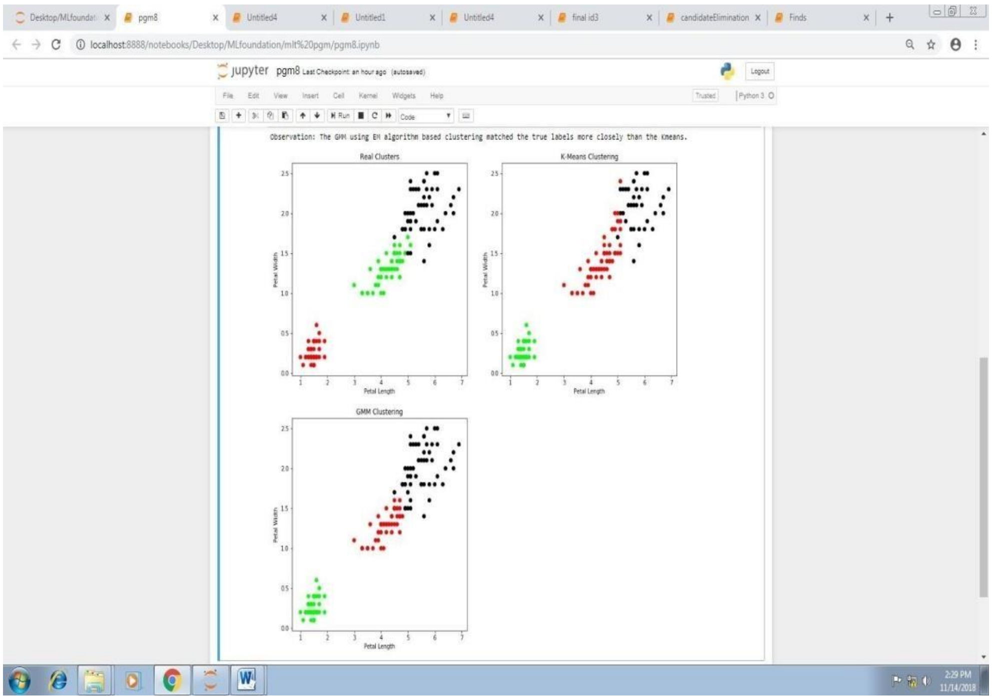
PROGRAM-7

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.

```
importmatplotlib.pyplot as pltfromsklearn import datasets
fromsklearn.cluster import KMeans import pandas as pd
importnumpy 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') fromsklearn import
preprocessing
scaler = preprocessing.StandardScaler() scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns) fromsklearn.mixture
import GaussianMixturegmm = 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 algorithm based clustering
matched the true labels more closely than the Kmeans.')
```

OUTPUT

Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.



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.

```
fromsklearn.model_selectionimport
train_test_splitfromsklearn.neighborsimport
KNeighborsClassifierfromsklearn import datasets
iris=datasets.load_iris() print("Iris Data set loaded...")
x_train, x_test, y_train, y_test =
train_test_split(iris.data,iris.target,test_size=0.1) print("Dataset is split
into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape) for i
in range(len(iris.target_names)):
print("Label", i , "-",str(iris.target_names[i])) classifier =
KNeighborsClassifier(n_neighbors=1) classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print("Results of Classification using K-nn with K=1 ") for r in
range(0,len(x_test)):
print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), "
Predicted-label:", str(y_pred[r])) print("Classification Accuracy :" ,
classifier.score(x_test,y_test));
```

OUTPUT

```
Iris Data set loaded...
Dataset is split into training and testing...
Size of trainng data and its label (135, 4) (135,) Size of trainng data and
its label (15, 4) (15,) Label 0 - setosa
Label 1 - versicolor Label 2 - virginica
Results of Classification using K-nn with K=1
Sample: [6.7 3. 5.2 2.3] Actual-label: 2 Predicted-label: 2
Sample: [6.5 2.8 4.6 1.5] Actual-label: 1 Predicted-label: 1
Sample: [7. 3.2 4.7 1.4] Actual-label: 1 Predicted-label: 1
Sample: [4.8 3.1 1.6 0.2]  Actual-label: 0  Predicted-label:0
Sample: [6.8 3.2 5.9 2.3]  Actual-label: 2  Predicted-label:2
Sample: [6.7 3.1 4.7 1.5]  Actual-label: 1  Predicted-label:1
Sample: [6.6 3. 4.4 1.4] Actual-label: 1 Predicted-label: 1
Sample: [4.6 3.1 1.5 0.2]  Actual-label: 0  Predicted-label:0
Sample: [7.3 2.9 6.3 1.8]  Actual-label: 2  Predicted-label:2
Sample: [5.1 3.7 1.5 0.4]  Actual-label: 0  Predicted-label:0
Sample: [4.9 3.1 1.5 0.1]  Actual-label: 0  Predicted-label:0
Sample: [5. 2. 3.5 1. ] Actual-label: 1 Predicted-label: 1
Sample: [6.6 2.9 4.6 1.3] Actual-label: 1 Predicted-label: 1
Sample: [7.7 3. 6.1 2.3] Actual-label: 2 Predicted-label: 2
 Sample: [5.7 2.6 3.5 1. ] Actual-label: 1 Predicted-label: 1 Classification
Accuracy : 1.0
```

PROGRAM-9

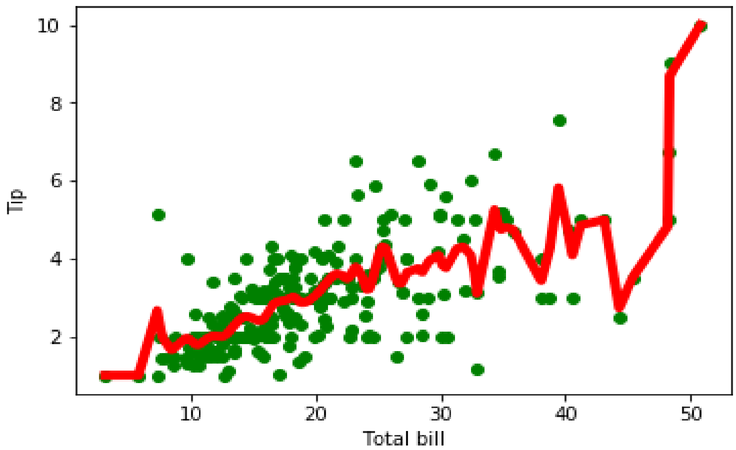
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import matplotlib.pyplot as pltimport pandas as pd
import numpy asnp

 def kernel(point,xmat, k): m,n =np.shape(xmat)
weights = np.mat(np.eye((m))) for j in range(m):
diff = point - X[j]
   weights[j,j] = np.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 = np.shape(xmat)
ypred = np.zeros(m) for i in range(m):
 ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k) return ypred
# load data points
data = pd.read_csv('tips.csv') bill = np.array(data.total_bill) tip =
np.array(data.tip) #preparing and add 1 in bill mbill =np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1] one = np.mat(np.ones(m))
X= np.hstack((one.T,mbill.T)) print(X.shape)
#set k here
ypred = localWeightRegression(X,mtip,0.5) SortIndex =
X[:,1].argsort(0)
xsort = X[SortIndex][:,0] fig=plt.figure() ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip') plt.show();
```

OUTPUT

(224, 2)



EXTRA PROGRAM

1) Load CSV with Python StandardLibrary

First, we need to import the csv module provided by Python standard library as follows:

```
import csv
```

Next, we need to import Numpy module for converting the loaded data into NumPy array.

```
import numpy as np
```

Now, provide the full path of the file, stored on our local directory, having the CSV datafile:

```
path = r"c:\iris.csv"
```

Next, use the csv.reader()function to read data from CSV file:

```
with open(path,'r') as f:
    reader = csv.reader(f,delimiter = ',')headers =
    next(reader)

    data = list(reader)
```

2) LOAD CSV WITH NUMPY

In this example, we are using the Pima Indians Dataset having the data of diabetic patients. This dataset is a numeric dataset with no header. It can also be downloaded intoour local directory. After loading the data file, we can convert it into **NumPy** array and useit for ML projects. The following is the Python script for loading CSV datafile:

```
from numpy import loadtxt
path = r"C:\pima-indians-diabetes.csv"
datapath= open(path, 'r')

data = loadtxt(datapath, delimiter=",")
print(data.shape)
```

We can print the names of the headers with the following line of script:

```
print(headers)
```

The following line of script will print the shape of the data i.e. number of rows &columns in the file:

```
print(data.shape)
```

Next script line will give the first three line of data file:

```
print(data[:3])
```

OUTPUT

```
['sepal_length','sepal_width','petal_length','petal_width']
```

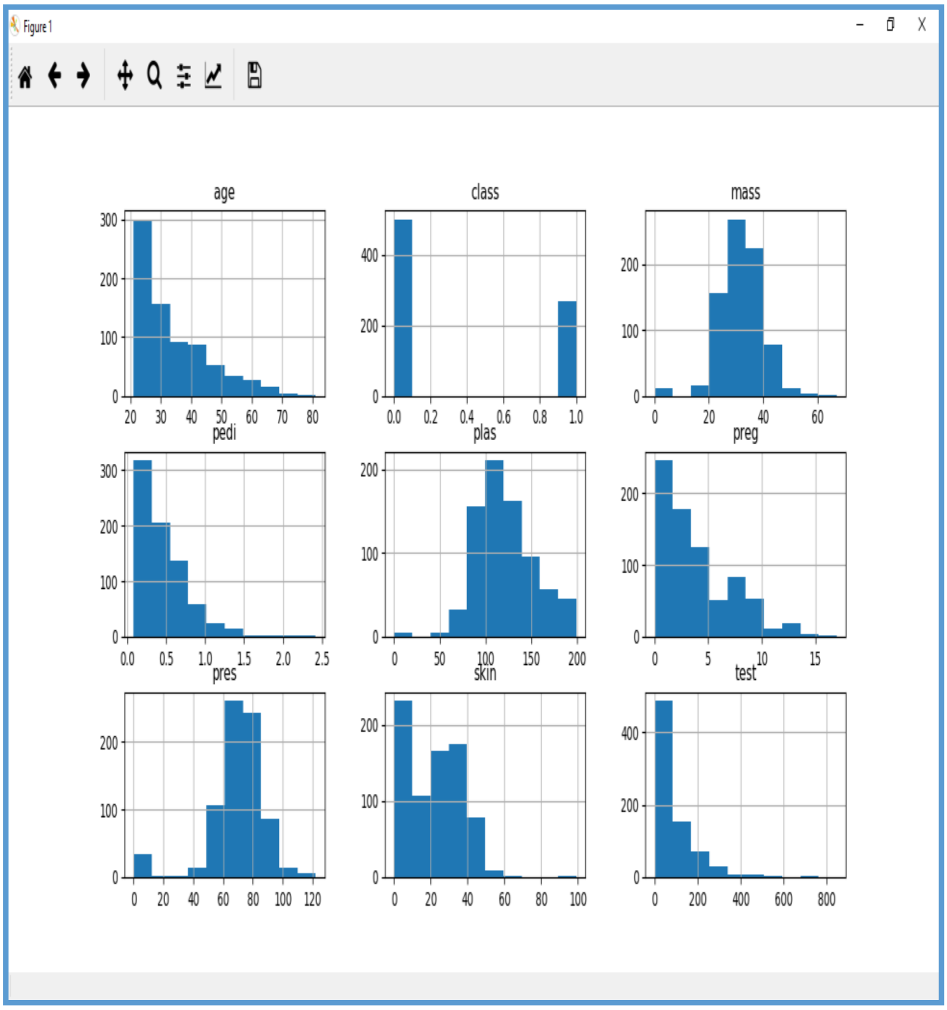
3)

```
from matplotlib import pyplotfrom
pandas import read_csv

path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age','class']
data = read_csv(path, names=names)

data.hist()
```

OUTPUT



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 networks
21. Differentiate hard and soft clustering
22. Define variance
23. What is inductive machine learning?
24. Why K nearest neighbor 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