**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 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).

|  |  |  |
| --- | --- | --- |
| **Supervised learning** | **Un Supervised learning** | **Instance based** |
| Find-s algorithm | EM algorithm | Locally weighted  Regression algorithm |
| Candidate elimination algorithm | K means algorithm |
| Decision tree algorithm |
| Back propagation Algorithm |
| Naïve Bayes Algorithm |
| K nearest neighbor  algorithm(lazy learning  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.

**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. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

**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. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Other methods are based on estimated density and graph connectivity. Clustering is a method of unsupervised learning, and a Common technique for statistical data analysis.

**Bayesian networks**

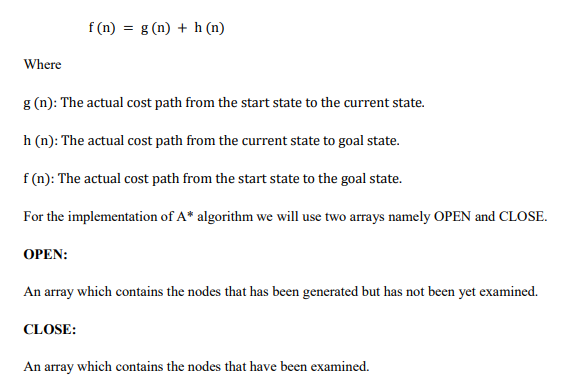
A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships 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**

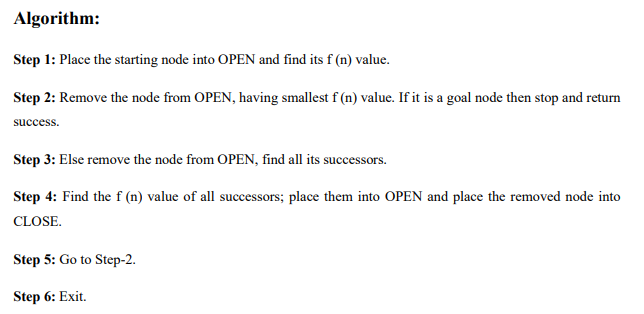
Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement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

1. **Implement A\* Algorithm.**

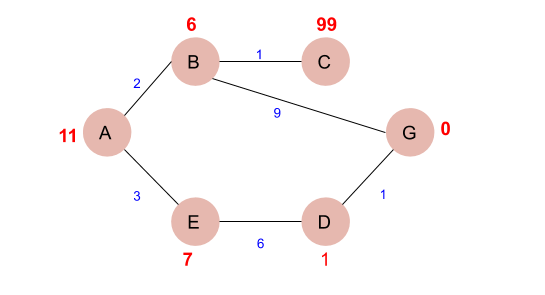
* A\* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael.
* It is the combination of Dijkstra’s algorithm and Best first search.
* It can be used to solve many kinds of problems.
* A\* search finds the shortest path through a search space to goal state using heuristic function.
* This technique finds minimal cost solutions and is directed to a goal state called A\* search.
* In A\*, the \* is written for optimality purpose.
* The A\* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost.
* A\* requires heuristic function to evaluate the cost of path that passes through the particular state.
* This algorithm is complete if the branching factor is finite and every action has fixed cost.
* A\* requires heuristic function to evaluate the cost of path that passes through the particular state.

****

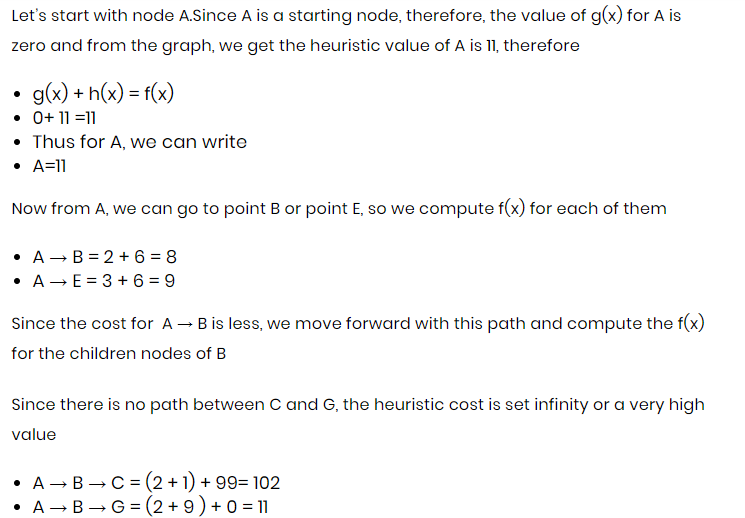
It can be defined by following formula.

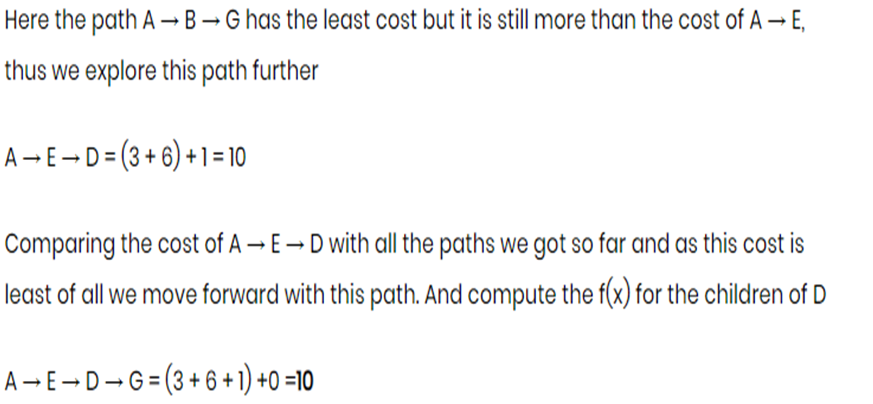


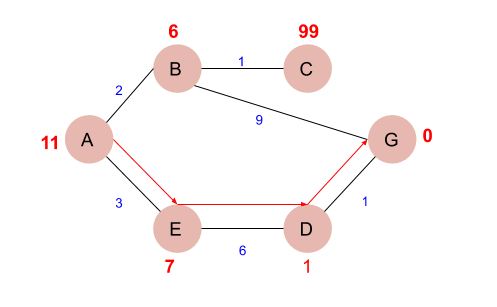
**Consider the following graph below**

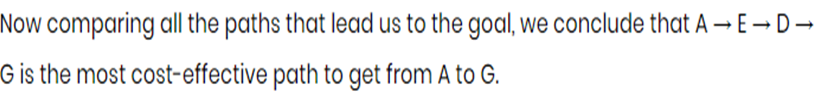
****

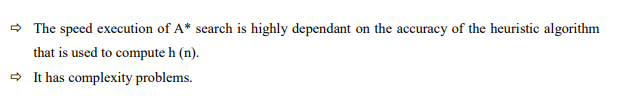
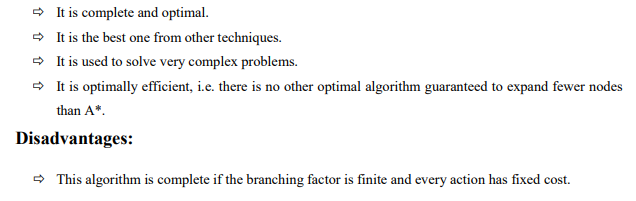
* The numbers written on edges represent the distance between the nodes while the numbers written on nodes represent the heuristic values.
* Let us find the most cost-effective path to reach from start state A to final state G using A\* Algorithm.





****

****

****

**Advantages:**

**Source Code:**

def aStarAlgo(start\_node, stop\_node):

        open\_set = set(start\_node)

        closed\_set = set()

        g = {}

        parents = {}

        g[start\_node] = 0

        parents[start\_node] = start\_node

        while len(open\_set) > 0:

            n = None

            for v in open\_set:

                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

                    n = v

            if n == stop\_node or Graph\_nodes[n] == None:

                pass

            else:

                for (m, weight) in get\_neighbors(n):

                    if m not in open\_set and m not in closed\_set:

                        open\_set.add(m)

                        parents[m] = n

                        g[m] = g[n] + weight

                    else:

                        if g[m] > g[n] + weight:

                            g[m] = g[n] + weight

                            parents[m] = n

                            if m in closed\_set:

                                closed\_set.remove(m)

                                open\_set.add(m)

            if n == None:

                print('Path does not exist!')

                return None

            if n == stop\_node:

                path = []

                while parents[n] != n:

                    path.append(n)

                    n = parents[n]

                path.append(start\_node)

                path.reverse()

                print('Path found: {}'.format(path))

                return path

            open\_set.remove(n)

            closed\_set.add(n)

        print('Path does not exist!')

        return None

def get\_neighbors(v):

    if v in Graph\_nodes:

        return Graph\_nodes[v]

    else:

        return None

        H\_dist = {

            'A': 11,

            'B': 6,

            'C': 99,

            'D': 1,

            'E': 7,

            'G': 0,

        }

        return H\_dist[n]

Graph\_nodes = {

    'A': [('B', 2), ('E', 3)],

    'B': [('C', 1),('G', 9)],

    'C': None,

    'E': [('D', 6)],

    'D': [('G', 1)],

}

aStarAlgo('A', 'G')

**Output:**

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

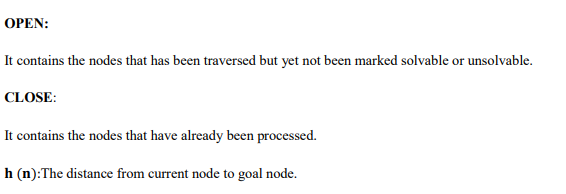
Out[3]:

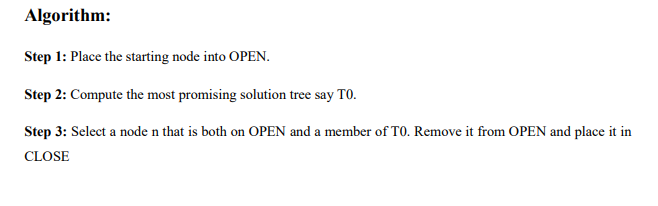
['A', 'E', 'D', 'G']

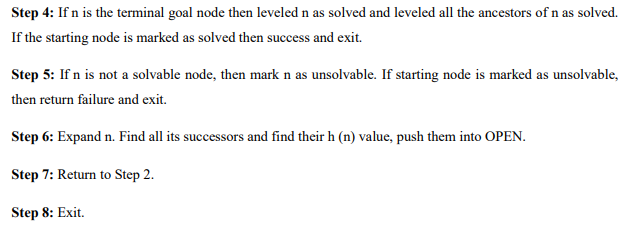
1. **Implement AO\* Algorithm.**

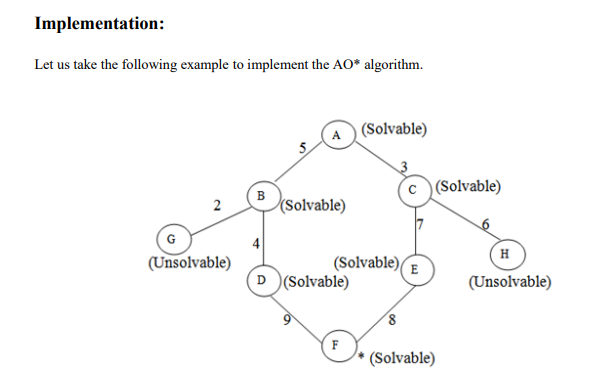
**AO\* Search: (And-Or) Graph**

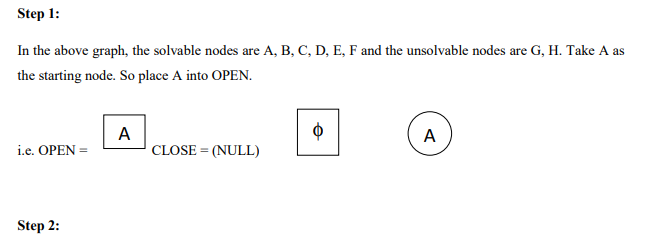
* The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph.
* The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO\* algorithm.

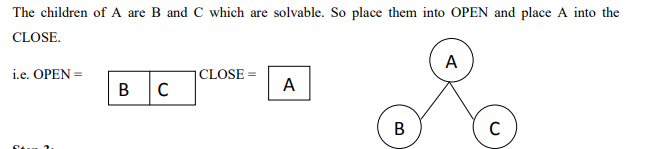
****

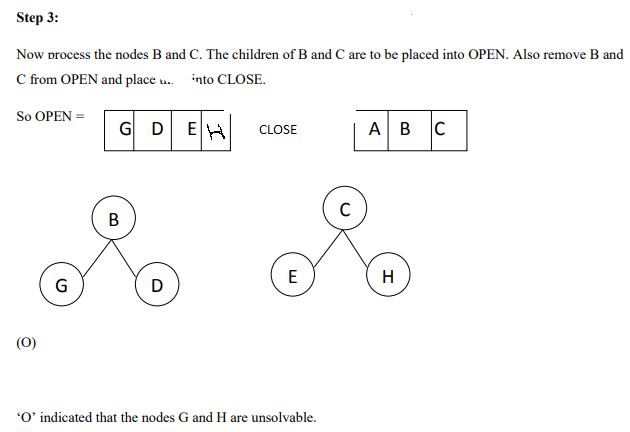
****

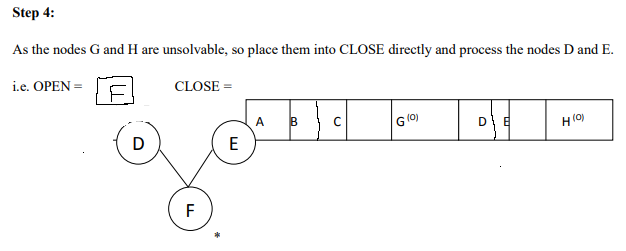
****

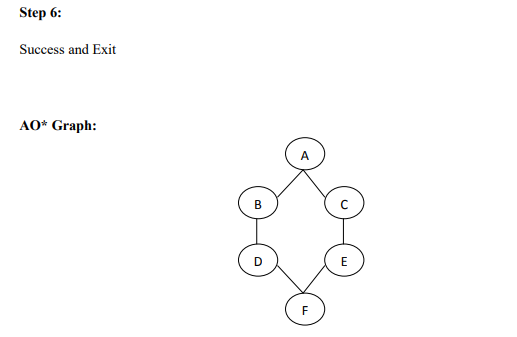


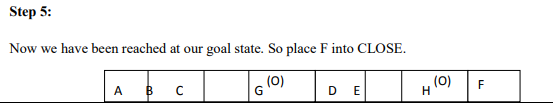
****

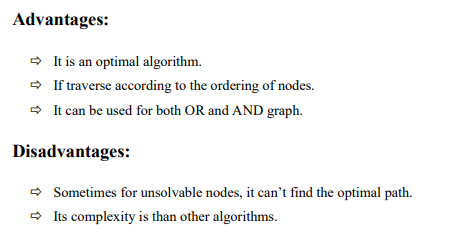
****





****

****



**Source code:**

class Graph:

def \_\_init\_\_(self, graph, hlist, startNode):

self.graph = graph

self.H=hlist

self.start=startNode

self.parent={}

self.status={}

self.solutionGraph={}

def applyAOStar(self):

self.aoStar(self.start, False)

def getNeighbors(self, v):

return self.graph.get(v,'')

def getStatus(self,v):

return self.status.get(v,0)

def setStatus(self,v, val):

self.status[v]=val

def getHvalue(self, n):

return self.H.get(n,0)

def setHvalue(self, n, value):

self.H[n]=value

def printsol(self):

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

print("------------------------------------------------------------")

print(self.solutionGraph)

print("------------------------------------------------------------")

def cmincost(self, v):

min=0

child={}

child[min]=[]

flag=True

for node in self.getNeighbors(v):

cost=0

nodeList=[]

for c, weight in node:

cost=cost+self.getHvalue(c)+weight

nodeList.append(c)

if flag==True:

min=cost

child[min]=nodeList

flag=False

else:

if min>cost:

min=cost

child[min]=nodeList

return min, child[min]

def aoStar(self, v, backTracking):

print("HEURISTIC VALUES :", self.H)

print("SOLUTION GRAPH :", self.solutionGraph)

print("PROCESSING NODE :", v)

print("-----------------------------------------------------------------------------------------")

if self.getStatus(v) >= 0:

mincost, chlist = self.cmincost(v)

print(mincost, chlist)

self.setHvalue(v, mincost)

self.setStatus(v,len(chlist))

solved=True

for i in chlist:

self.parent[i]=v

if self.getStatus(i)!=-1:

solved=solved & False

if solved==True:

self.setStatus(v,-1)

self.solutionGraph[v]=chlist

if v!=self.start:

self.aoStar(self.parent[v], True)

if backTracking==False:

for i in chlist:

self.setStatus(i,0)

self.aoStar(i, False)

print ("Graph - 1")

h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

graph1 = {

'A': [[('B', 1), ('C', 1)], [('D', 1)]],

'B': [[('G', 1)], [('H', 1)]],

'C': [[('J', 1)]],

'D': [[('E', 1), ('F', 1)]],

'G': [[('I', 1)]]

}

G1= Graph(graph1, h1, 'A')

G1.applyAOStar()

G1.printsol()

**Output**

Graph - 1

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

10 ['B', 'C']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

6 ['G']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

10 ['B', 'C']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

8 ['I']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

8 ['H']

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

12 ['B', 'C']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : I

-----------------------------------------------------------------------------------------

0 []

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

-----------------------------------------------------------------------------------------

1 ['I']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

-----------------------------------------------------------------------------------------

2 ['G']

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

2 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

-----------------------------------------------------------------------------------------

0 []

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

-----------------------------------------------------------------------------------------

1 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

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

PROCESSING NODE : A

-----------------------------------------------------------------------------------------

5 ['B', 'C']

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

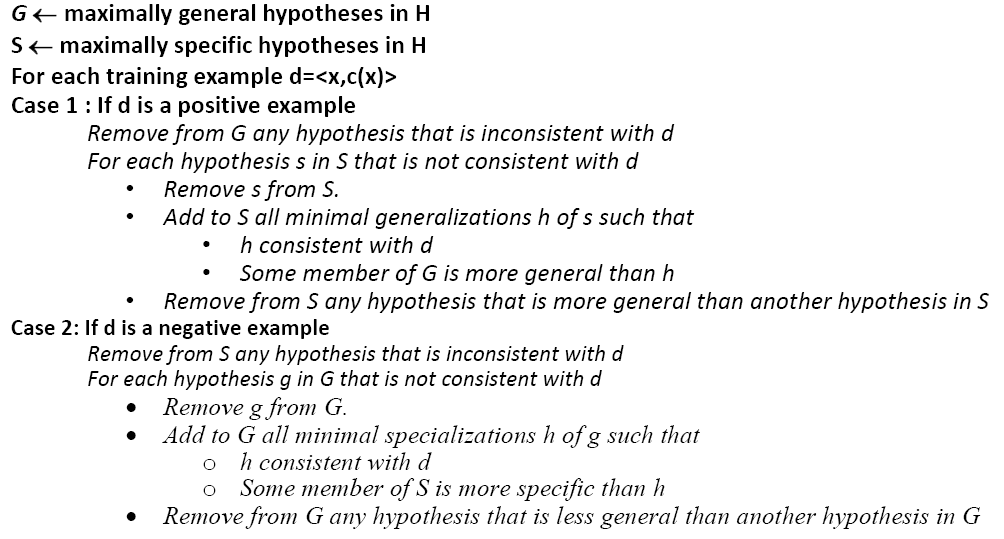
------------------------------------------------------------

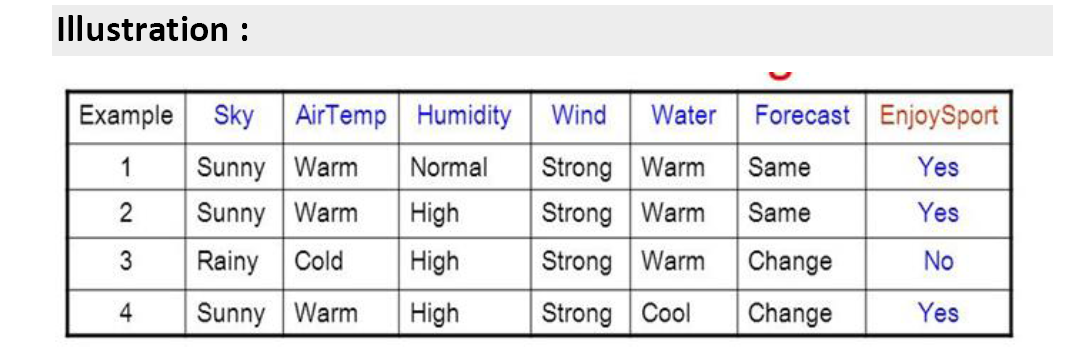
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

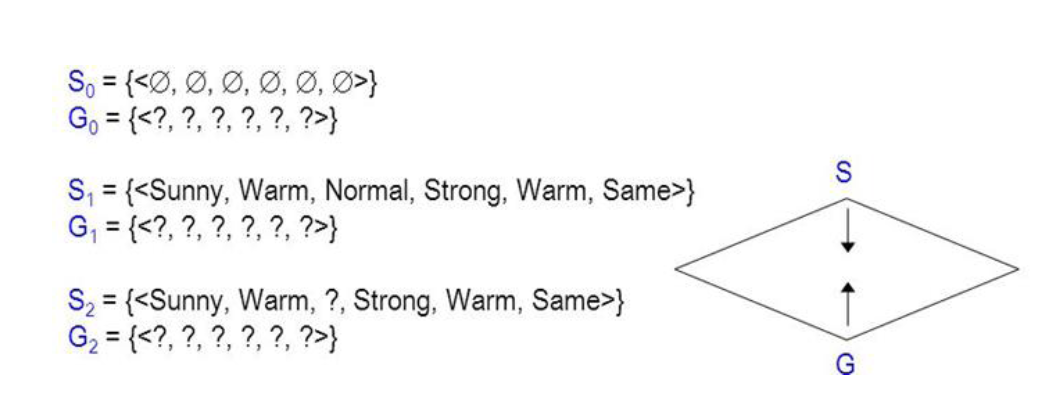
------------------------------------------------------------

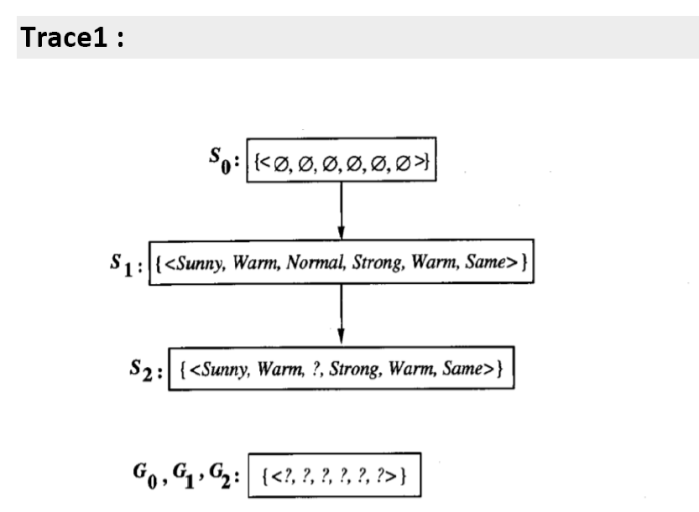
1. For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.

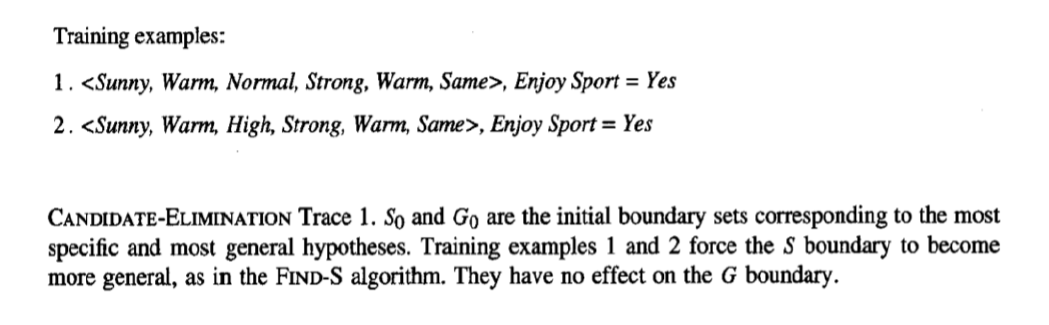
**Algorithm**

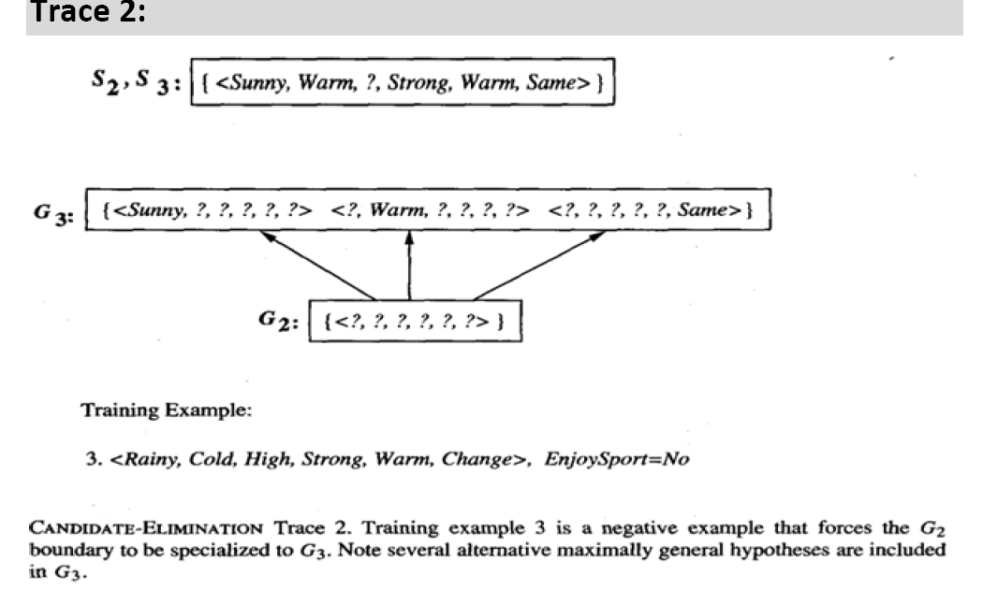
****

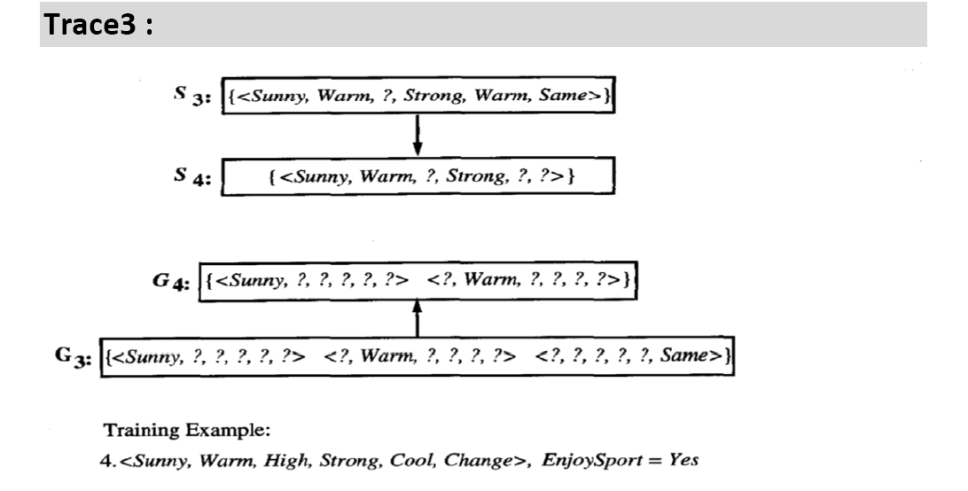
****

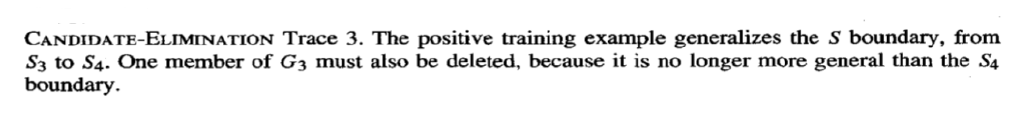
****

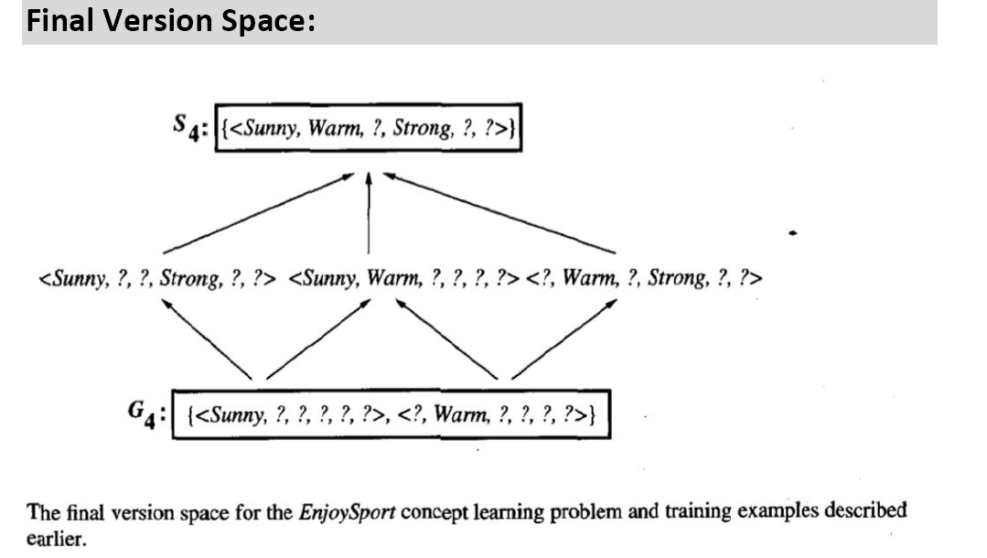
****

****

****

****

****

****

**Source Program**

import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read\_csv("candidatedata.csv",header=None))

print(data)

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":

print(h)

print(target[i])

for x in range(len(specific\_h)):

if h[x] != specific\_h[x]:

specific\_h[x] = "?"

general\_h[x][x] = "?"

if target[i] == "no":

print(h)

print(target[i])

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")

**Input data set sdata.csv**

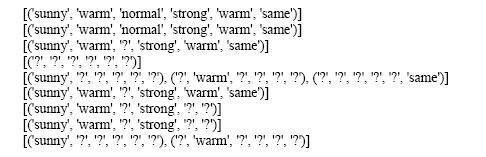
'Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', True

'Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', True

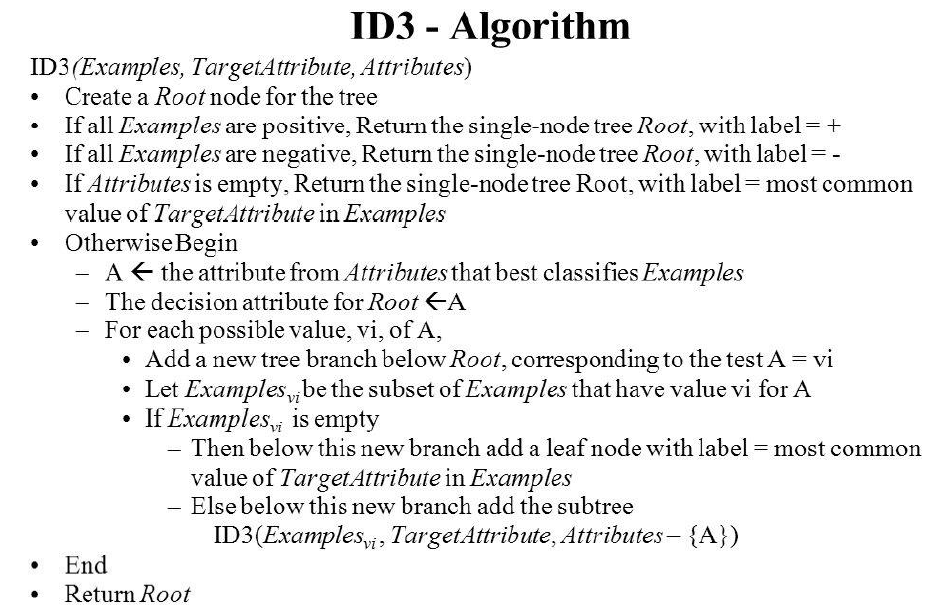
'Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', False

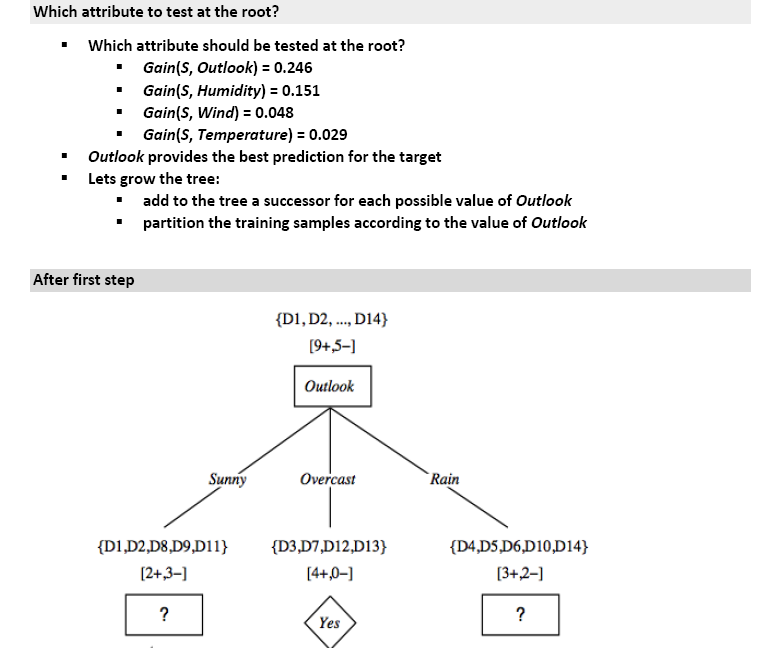
'Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', True

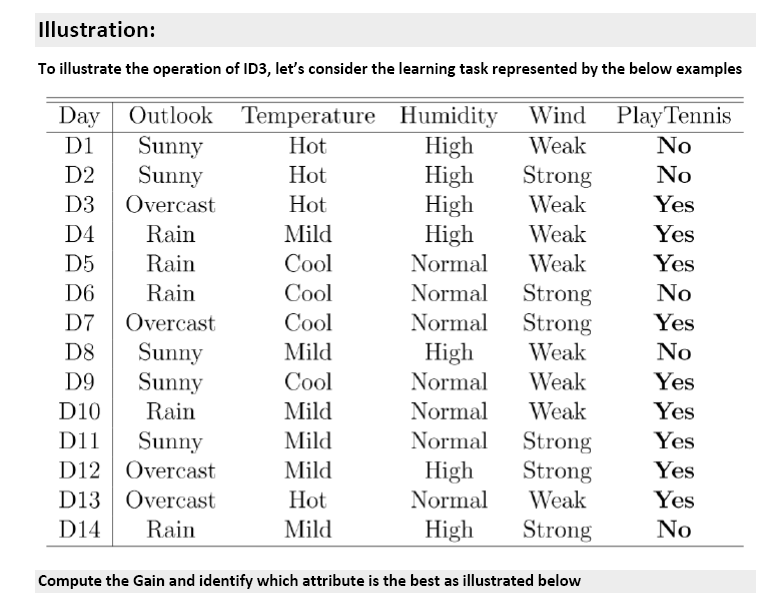
**Output**

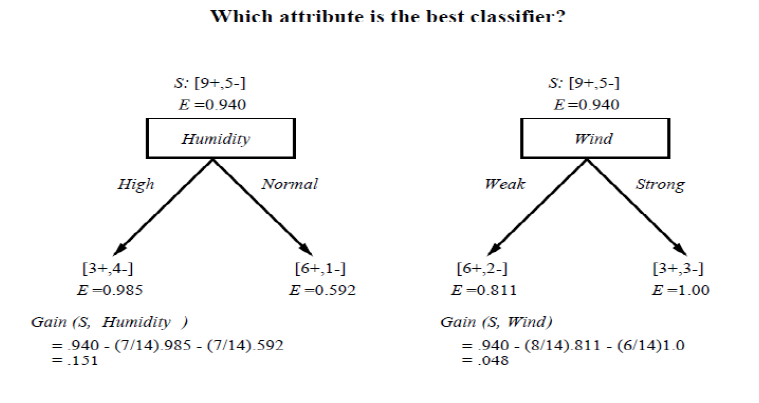


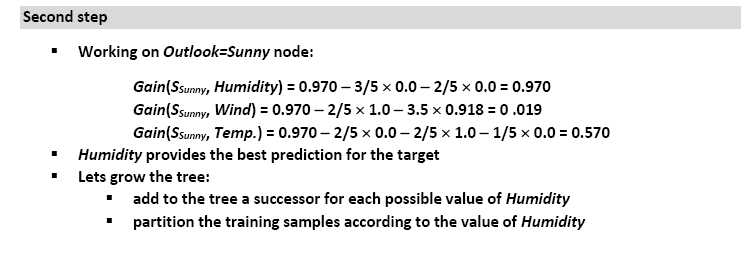
1. Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

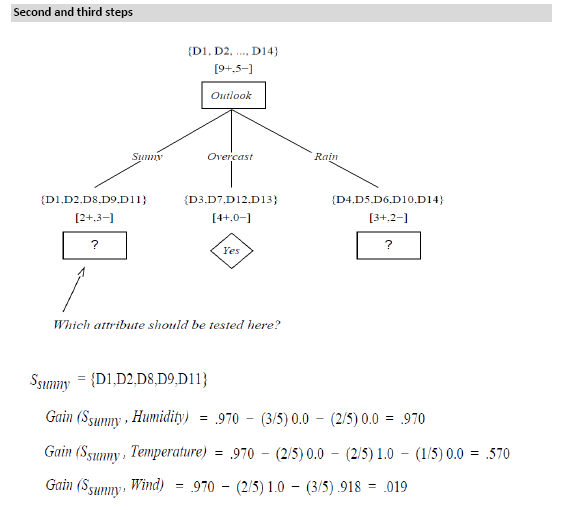












**Source Program**

import numpy as np

import math

import csv

class Node:

def \_\_init\_\_(self, attribute):

self.attribute = attribute

self.children = []

self.answer = ""

#def \_\_str\_\_(self):

# return self.attribute

def subtables(data, col, delete):

dict = {}

items = np.unique(data[:, col])

count = np.zeros((items.shape[0], 1), dtype=np.int32)

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]:

count[x] += 1

for x in range(items.shape[0]):

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

pos = 0

for y in range(data.shape[0]):

if data[y, col] == items[x]:

dict[items[x]][pos] = data[y]

pos += 1

if delete:

dict[items[x]] = np.delete(dict[items[x]], col, 1)

return items, dict

def entropy(S):

items = np.unique(S)

if items.size == 1:

return 0

counts = np.zeros((items.shape[0], 1))

sums = 0

for x in range(items.shape[0]):

counts[x] = sum(S == items[x]) / (S.size \* 1.0)

for count in counts:

sums += -1 \* count \* math.log(count, 2)

return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

total\_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1))

intrinsic = np.zeros((items.shape[0], 1))

for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)

entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

intrinsic[x] = ratio \* math.log(ratio, 2)

total\_entropy = entropy(data[:, -1])

iv = -1 \* sum(intrinsic)

for x in range(entropies.shape[0]):

total\_entropy -= entropies[x]

return total\_entropy / iv

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1:

node = Node("")

node.answer = np.unique(data[:, -1])[0]

return node

gains = np.zeros((data.shape[1] - 1, 1))

for col in range(data.shape[1] - 1):

gains[col] = gain\_ratio(data, col)

split = np.argmax(gains)

node = Node(metadata[split])

metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):

child = create\_node(dict[items[x]], metadata)

node.children.append((items[x], child))

return node

def empty(size):

s = ""

for x in range(size):

s += " "

return s

def print\_tree(node, level):

if node.answer != "":

print(empty(level), node.answer)

return

print(empty(level), node.attribute)

for value, n in node.children:

print(empty(level + 1), value)

print\_tree(n, level + 2)

with open ('id3dataset.csv','r') as f:

datareader = csv.reader(f)

headers = next(datareader)

metadata = []

traindata = []

for name in headers:

metadata.append(name)

for row in datareader:

traindata.append(row)

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

**Input Data Set (id3dataset.csv)**

outlook,temperature,humidity,wind,answer

sunny,hot,high,weak,no

sunny,hot,high,strong,no

overcast,hot,high,weak,yes

rain,mild,high,weak,yes

rain,cool,normal,weak,yes

rain,cool,normal,strong,no

overcast,cool,normal,strong,yes

sunny,mild,high,weak,no

sunny,cool,normal,weak,yes

rain,mild,normal,weak,yes

sunny,mild,normal,strong,yes

overcast,mild,high,strong,yes

overcast,hot,normal,weak,yes

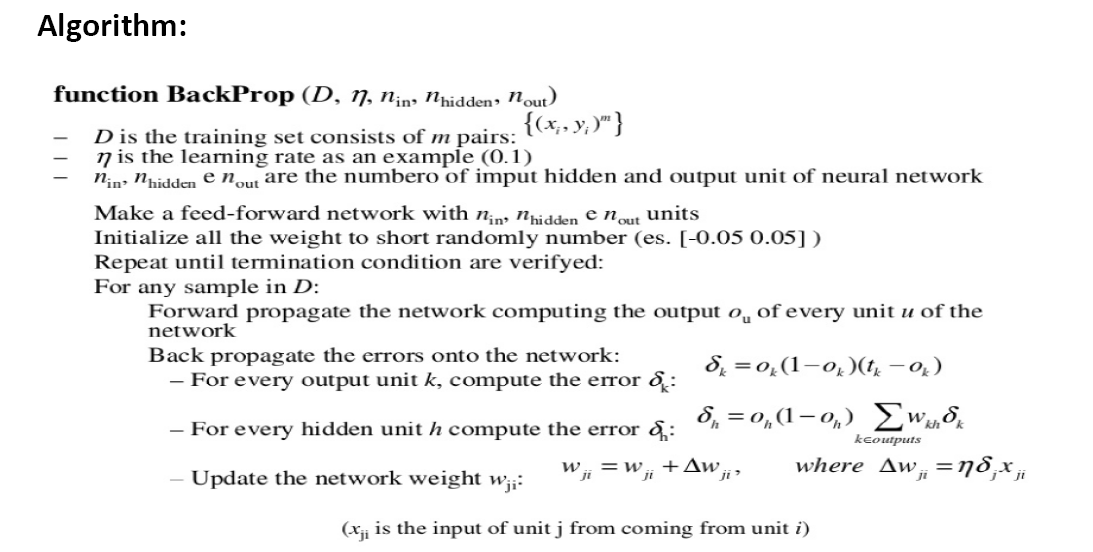
rain,mild,high,strong,no

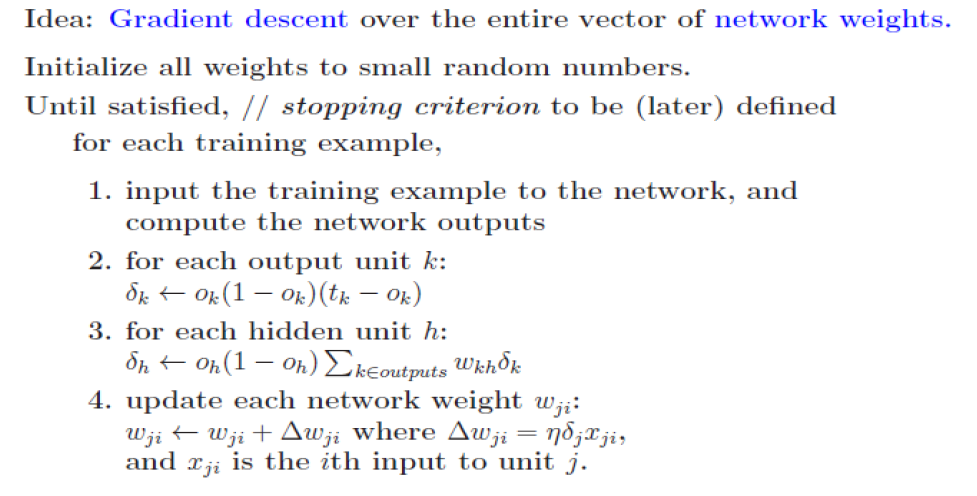
**output**

Display Tree {'outlook': {'outlook': 'answer', 'overcast': 'yes', 'rain': {'wind': {'strong': 'no', 'weak': 'yes'}}, 'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}

The prediction accuracy is: 100.0 %

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





**Source Program**

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

for i in range(epoch):

#Forward Propogation

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:

[[0.89491454]

[0.88236279]

[0.89289415]]

1. 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.

**Bayesian Theorem:**

****

The Naive Bayes algorithm is an intuitive method that uses the probabilities of each attribute belonging to each class to make a prediction. It is the supervised learning approach you would come up with if you wanted to model a predictive modeling problem probabilistically.

Naive bayes simplifies the calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method.

The probability of a class value given a value of an attribute is called the conditional probability. By multiplying the conditional probabilities together for each attribute for a given class value, we have a probability of a data instance belonging to that class

.

To make a prediction we can calculate probabilities of the instance belonging to each class and select the class value with the highest probability.

Naive bases is often described using categorical data because it is easy to describe and calculate using ratios. A more useful version of the algorithm for our purposes supports numeric attributes and assumes the values of each numerical attribute are normally distributed (fall somewhere on a bell curve). Again, this is a strong assumption, but still gives robust results.

**Naïve Bayesian algorithm is broken down into the following steps:**

1. **Handle Data**: Load the data from CSV file and split it into training and test datasets.
2. **Summarize Data**: summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
3. **Make a Prediction**: Use the summaries of the dataset to generate a single prediction.
4. **Make Predictions**: Generate predictions given a test dataset and a summarized training dataset.
5. **Evaluate Accuracy**: Evaluate the accuracy of predictions made for a test dataset as the percentage correct out of all predictions made.
6. **Handle Data:**

The first thing we need to do is load our data file. The data is in CSV format without a header line or any quotes. We can open the file with the open function and read the data lines using the reader function in the csv module.

Next we need to split the data into a training dataset that Naive Bayes can use to make predictions and a test dataset that we can use to evaluate the accuracy of the model. We need to split the data set randomly into train and datasets with a ratio of 67% train and 33% test (this is a common ratio for testing an algorithm on a dataset).

The splitDataset() function that will split a given dataset into a given split ratio.

1. **Summarize Data:**

The naive bayes model is comprised of a summary of the data in the training dataset. This summary is then used when making predictions.

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value. For example, if there are two class values and 7 numerical attributes, then we need a mean and standard deviation for each attribute (7) and class value (2) combination that is 14 attribute summaries.

These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.

**We can break the preparation of this summary data down into the following sub-tasks:**

1. Separate Data By Class
2. Calculate Mean
3. Calculate Standard Deviation
4. Summarize Dataset
5. Summarize Attributes By Class

**Separate Data By Class**

The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.

**Calculate Mean**

We need to calculate the mean of each attribute for a class value. The mean is the central middle or central tendency of the data, and we will use it as the middle of our gaussian distribution when calculating probabilities.

We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.

The standard deviation is calculated as the square root of the variance. The variance is calculated as the average of the squared differences for each attribute value from the mean. Note we are using the N-1 method, which subtracts 1 from the number of attribute values when calculating the variance.

**Summarize Dataset**

Now we have the tools to summarize a dataset. For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.

The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.

**Summarize Attributes By Class**

We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

**3. Make Prediction**

We are now ready to make predictions using the summaries prepared from our training data. Making predictions involves calculating the probability that a given data instance belongs to each class, then selecting the class with the largest probability as the prediction.

**We can divide this part into the following tasks:**

1. Calculate Gaussian Probability Density Function
2. Calculate Class Probabilities
3. Make a Prediction
4. Estimate Accuracy
   1. **Calculate Gaussian Probability Density Function**

We can use a Gaussian function to estimate the probability of a given attribute value, given the known mean and standard deviation for the attribute estimated from the training data. Given that the attribute summaries where prepared for each attribute and class value, the result is the conditional probability of a given attribute value given a class value.

* 1. **Calculate Class Probabilities**

Now that we can calculate the probability of an attribute belonging to a class, we can combine the probabilities of all of the attribute values for a data instance and come up with a probability of the entire data instance belonging to the class.

* 1. **Make a Prediction**

Now that can calculate the probability of a data instance belonging to each class value, we can look for the largest probability and return the associated class.

* 1. **Make Predictions**

Finally, we can estimate the accuracy of the model by making predictions for each data instance in our test dataset. The **getPredictions()** will do this and return a list of predictions for each test instance.

* 1. **Get Accuracy**

The predictions can be compared to the class values in the test dataset and a classification accuracy can be calculated as an accuracy ratio between 0& and 100%. The **getAccuracy()**will calculate this accuracy ratio.

**Source Program**

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 instacnes 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 = {}

for classValue, instances in separated.items():

#summaries is a dic of tuples(mean,std) for each class value

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 = {}

for classValue, classSummaries in summaries.items():#class and attribute information as mean and sd

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 = '5data.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()

**Input dataset 5data.csv:**

6,148,72,35,0,33.6,0.627,50,1

1,85,66,29,0,26.6,0.351,31,0

8,183,64,0,0,23.3,0.672,32,1

1,89,66,23,94,28.1,0.167,21,0

0,137,40,35,168,43.1,2.288,33,1

5,116,74,0,0,25.6,0.201,30,0

3,78,50,32,88,31.0,0.248,26,1

10,115,0,0,0,35.3,0.134,29,0

2,197,70,45,543,30.5,0.158,53,1

8,125,96,0,0,0.0,0.232,54,1

4,110,92,0,0,37.6,0.191,30,0

10,168,74,0,0,38.0,0.537,34,1

10,139,80,0,0,27.1,1.441,57,0

1,189,60,23,846,30.1,0.398,59,1

5,166,72,19,175,25.8,0.587,51,1

7,100,0,0,0,30.0,0.484,32,1

0,118,84,47,230,45.8,0.551,31,1

7,107,74,0,0,29.6,0.254,31,1

1,103,30,38,83,43.3,0.183,33,0

1,115,70,30,96,34.6,0.529,32,1

3,126,88,41,235,39.3,0.704,27,0

8,99,84,0,0,35.4,0.388,50,0

7,196,90,0,0,39.8,0.451,41,1

9,119,80,35,0,29.0,0.263,29,1

11,143,94,33,146,36.6,0.254,51,1

10,125,70,26,115,31.1,0.205,41,1

7,147,76,0,0,39.4,0.257,43,1

1,97,66,15,140,23.2,0.487,22,0

13,145,82,19,110,22.2,0.245,57,0

5,117,92,0,0,34.1,0.337,38,0

5,109,75,26,0,36.0,0.546,60,0

3,158,76,36,245,31.6,0.851,28,1

3,88,58,11,54,24.8,0.267,22,0

6,92,92,0,0,19.9,0.188,28,0

10,122,78,31,0,27.6,0.512,45,0

4,103,60,33,192,24.0,0.966,33,0

11,138,76,0,0,33.2,0.420,35,0

9,102,76,37,0,32.9,0.665,46,1

2,90,68,42,0,38.2,0.503,27,1

4,111,72,47,207,37.1,1.390,56,1

3,180,64,25,70,34.0,0.271,26,0

7,133,84,0,0,40.2,0.696,37,0

7,106,92,18,0,22.7,0.235,48,0

9,171,110,24,240,45.4,0.721,54,1

7,159,64,0,0,27.4,0.294,40,0

0,180,66,39,0,42.0,1.893,25,1

1,146,56,0,0,29.7,0.564,29,0

2,71,70,27,0,28.0,0.586,22,0

7,103,66,32,0,39.1,0.344,31,1

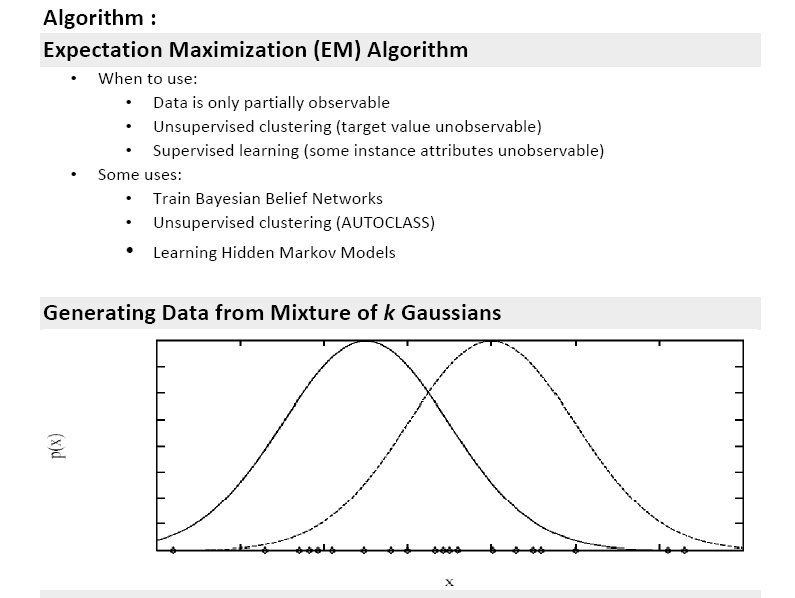
7,105,0,0,0,0.0,0.305,24,0

**Output:**

Split 768 rows into train=514 and test=254 rows

Accuracy of the classifier is : 76.37795275590551%

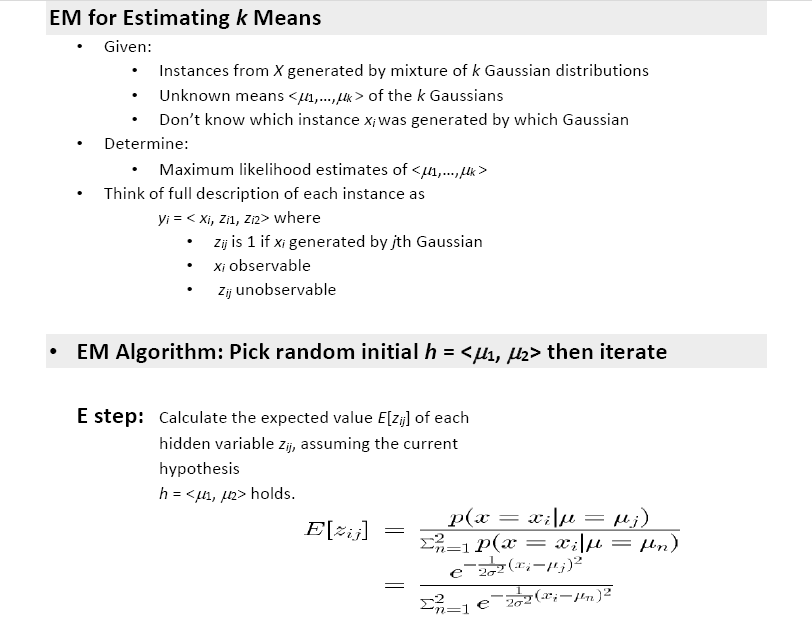
1. Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

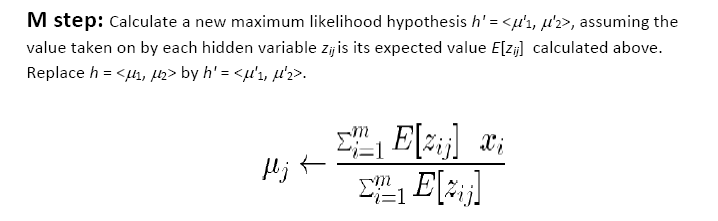


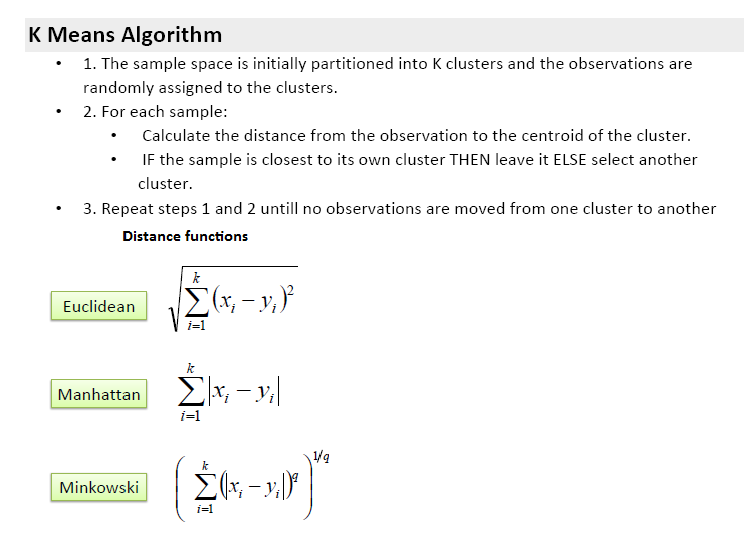
**Each instance x generated by**

1. Choosing one of the *k* Gaussians with uniform probability

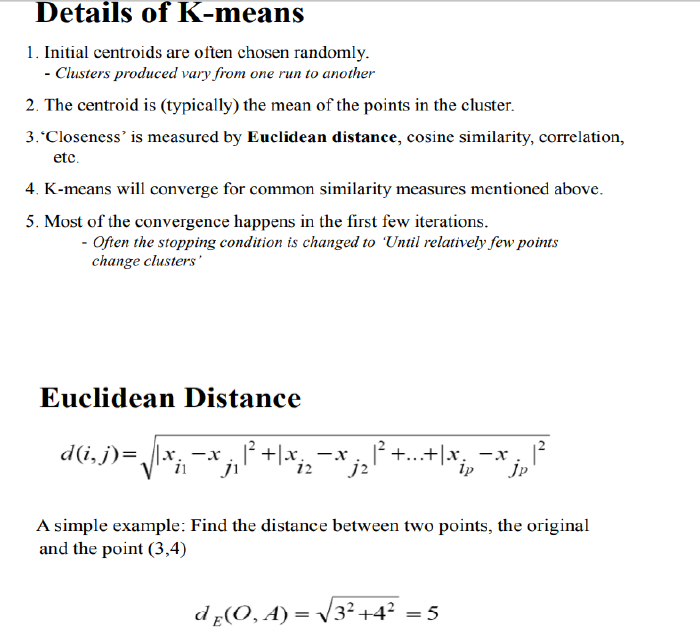
2. Generating an instance at random according to that Gaussian

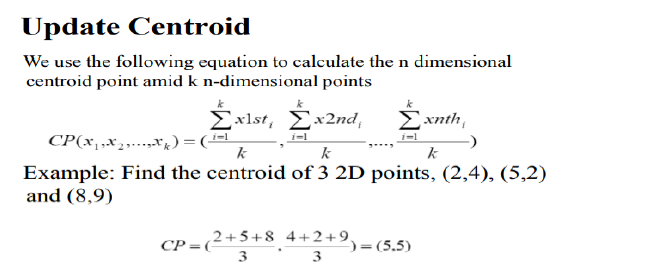


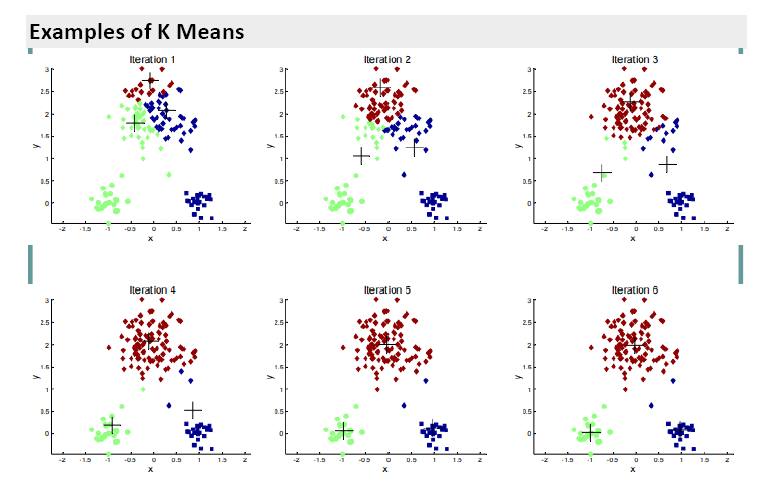












**Source Program**

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.mixture import GaussianMixture

from sklearn.cluster import KMeans

# Importing the dataset

data = pd.read\_csv('kmeansdata.csv')

data.head()

# Getting the values and plotting it

f1 = data[' Distance\_Feature '].values

f2 = data[' Speeding\_Feature '].values

X = np.array(list(zip(f1, f2)))

kmeans = KMeans(3, random\_state=0)

labels = kmeans.fit(X).predict(X)

centroids = kmeans.cluster\_centers\_

plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis');

print('Graph using Kmeans Algorithm')

plt.scatter(centroids[:, 0], centroids[:, 1], marker='\*', s=200,c='#050505')

plt.show()

#gmm

gmm = GaussianMixture(n\_components=3).fit(X)

labels = gmm.predict(X)

# plot

probs = gmm.predict\_proba(X)

size = 10 \* probs.max(1) \*\* 3

print('Graph using EM Algorithm')

#print(probs[:300].round(4))

plt.scatter(X[:, 0], X[:, 1], c=labels, s=size, cmap='viridis');

plt.show()

**Input data set kmeansdata.csv**

Driver\_ID,Distance\_Feature,Speeding\_Feature

3423311935,71.24,28

3423313212,52.53,25

3423313724,64.54,27

3423311373,55.69,22

3423310999,54.58,25

3423313857,41.91,10

3423312432,58.64,20

3423311434,52.02,8

3423311328,31.25,34

3423312488,44.31,19

3423311254,49.35,40

3423312943,58.07,45

3423312536,44.22,22

3423311542,55.73,19

3423312176,46.63,43

3423314176,52.97,32

3423314202,46.25,35

3423311346,51.55,27

3423310666,57.05,26

3423313527,58.45,30

3423312182,43.42,23

3423313590,55.68,37

3423312268,55.15,18

**Output:**

Driver\_ID Distance\_Feature Speeding\_Feature

0 3423311935 71.24 28

1 3423313212 52.53 25

2 3423313724 64.54 27

3 3423311373 55.69 22

4 3423310999 54.58 25

5 3423313857 41.91 10

6 3423312432 58.64 20

7 3423311434 52.02 8

8 3423311328 31.25 34

9 3423312488 44.31 19

10 3423311254 49.35 40

11 3423312943 58.07 45

12 3423312536 44.22 22

13 3423311542 55.73 19

14 3423312176 46.63 43

15 3423314176 52.97 32

16 3423314202 46.25 35

17 3423311346 51.55 27

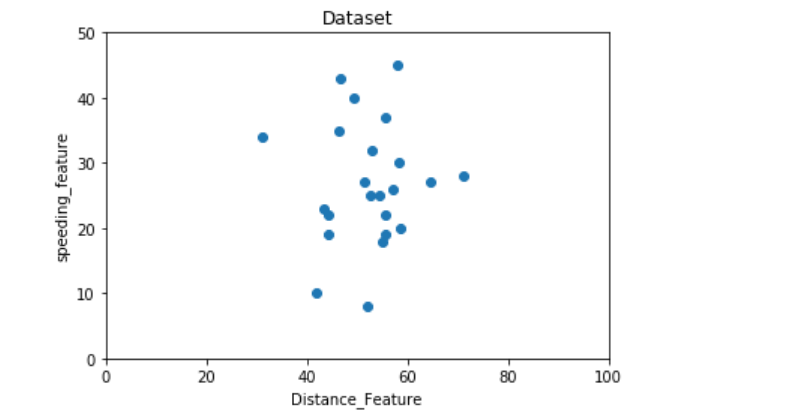
18 3423310666 57.05 26

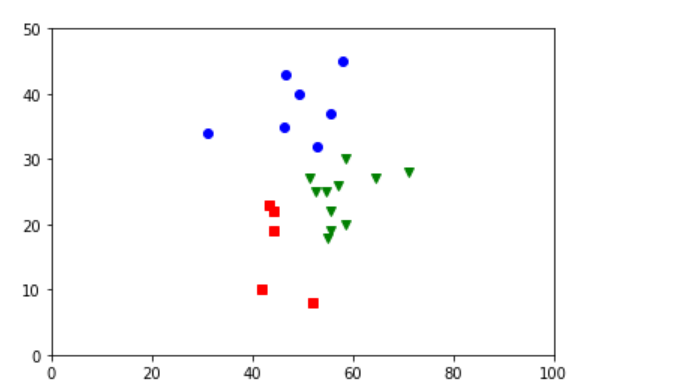
19 3423313527 58.45 30

20 3423312182 43.42 23

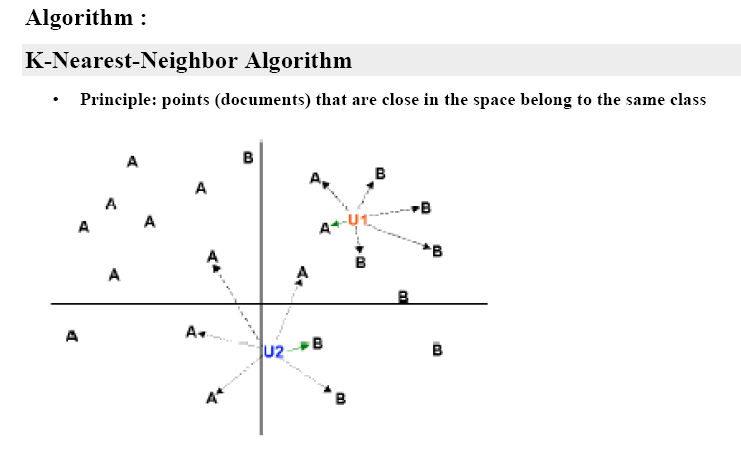
21 3423313590 55.68 37

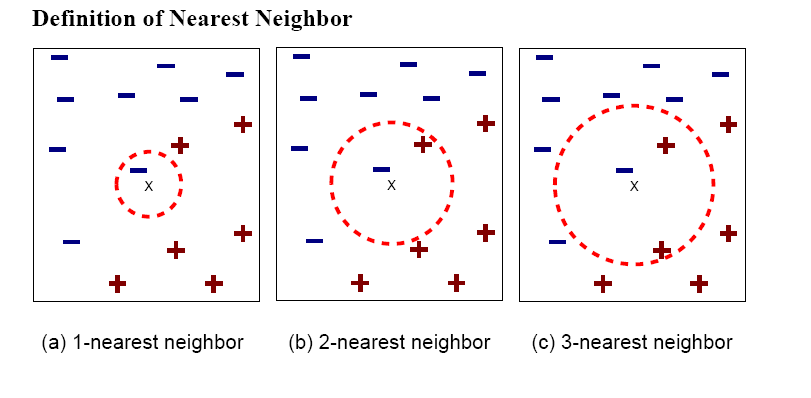
22 3423312268 55.15 18

****



1. 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.





Euclidian Distance



**Algorithm**

**Step 1** − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

**Step 2** − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

**Step 3** − For each point in the test data do the following −

* **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
* **3.2** − Now, based on the distance value, sort them in ascending order.
* **3.3** − Next, it will choose the top K rows from the sorted array.
* **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

**Step 4** – End

**Pros and Cons of KNN**

**Pros**

* It is very simple algorithm to understand and interpret.
* It is very useful for nonlinear data because there is no assumption about data in this algorithm.
* It is a versatile algorithm as we can use it for classification as well as regression.
* It has relatively high accuracy but there are much better supervised learning models than KNN.

**Cons**

* It is computationally a bit expensive algorithm because it stores all the training data.
* High memory storage required as compared to other supervised learning algorithms.
* Prediction is slow in case of big N.
* It is very sensitive to the scale of data as well as irrelevant features.

## Applications of KNN

The following are some of the areas in which KNN can be applied successfully −

### Banking System

KNN can be used in banking system to predict weather an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

### Calculating Credit Ratings

KNN algorithms can be used to find an individual’s credit rating by comparing with the persons having similar traits.

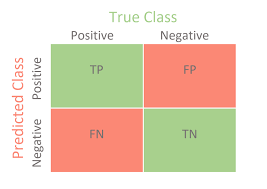
### Politics

With the help of KNN algorithms, we can classify a potential voter into various classes like “Will Vote”, “Will not Vote”, “Will Vote to Party ‘Congress’, “Will Vote to Party ‘BJP’.

Other areas in which KNN algorithm can be used are Speech Recognition, Handwriting Detection, Image Recognition and Video Recognition.

**Confusion Matrix**

A confusion matrix is **a table that** is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

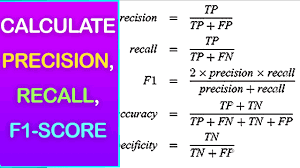


The **True Positive cell,** number 1, Means that the model predicted a positive, and the actual was a positive

The **False Negative** cell, number 3, means that the model predicted a negative, and the actual was a positive.

The **False Positive cell**, number 2, means that the model predicted a positive, but the actual was a negative.

The **True Negative cell,** number 1, Means that the model predicted a negative, and the actual was a negative.



**Source Program**

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn import datasets

iris=datasets.load\_iris()

iris\_data=iris.data

iris\_labels=iris.target

print(iris\_labels)

x\_train, x\_test, y\_train, y\_test=train\_test\_split(iris\_data,iris\_labels,test\_size=0.30)

classifier=KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

print('confusion matrix is as follows')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy metrics')

print(classification\_report(y\_test,y\_pred))

**Input dataset Iris.csv**

5.1,3.5,1.4,0.2,Iris-setosa

4.9,3.0,1.4,0.2,Iris-setosa

4.7,3.2,1.3,0.2,Iris-setosa

4.6,3.1,1.5,0.2,Iris-setosa

5.0,3.6,1.4,0.2,Iris-setosa

5.4,3.9,1.7,0.4,Iris-setosa

4.6,3.4,1.4,0.3,Iris-setosa

5.0,3.4,1.5,0.2,Iris-setosa

4.4,2.9,1.4,0.2,Iris-setosa

4.9,3.1,1.5,0.1,Iris-setosa

7.0,3.2,4.7,1.4,Iris-versicolor

6.4,3.2,4.5,1.5,Iris-versicolor

6.9,3.1,4.9,1.5,Iris-versicolor

5.5,2.3,4.0,1.3,Iris-versicolor

6.5,2.8,4.6,1.5,Iris-versicolor

5.7,2.8,4.5,1.3,Iris-versicolor

6.3,3.3,4.7,1.6,Iris-versicolor

4.9,2.4,3.3,1.0,Iris-versicolor

6.6,2.9,4.6,1.3,Iris-versicolor

5.2,2.7,3.9,1.4,Iris-versicolor

6.4,3.1,5.5,1.8,Iris-virginica

6.0,3.0,4.8,1.8,Iris-virginica

6.9,3.1,5.4,2.1,Iris-virginica

6.7,3.1,5.6,2.4,Iris-virginica

6.9,3.1,5.1,2.3,Iris-virginica

5.8,2.7,5.1,1.9,Iris-virginica

6.8,3.2,5.9,2.3,Iris-virginica

6.7,3.3,5.7,2.5,Iris-virginica

6.7,3.0,5.2,2.3,Iris-virginica

6.3,2.5,5.0,1.9,Iris-virginica

6.5,3.0,5.2,2.0,Iris-virginica

6.2,3.4,5.4,2.3,Iris-virginica

5.9,3.0,5.1,1.8,Iris-virginica

**Output**

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.1 1.5 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1. ]

[6.6 2.9 4.6 1.3]

[5.2 2.7 3.9 1.4]

[5. 2. 3.5 1. ]

[5.9 3. 4.2 1.5]

[6. 2.2 4. 1. ]

[6.1 2.9 4.7 1.4]

[5.6 2.9 3.6 1.3]

[6.7 3.1 4.4 1.4]

[5.6 3. 4.5 1.5]

[5.8 2.7 4.1 1. ]

[6.2 2.2 4.5 1.5]

[5.6 2.5 3.9 1.1]

[5.9 3.2 4.8 1.8]

[6.1 2.8 4. 1.3]

[6.3 2.5 4.9 1.5]

[6.1 2.8 4.7 1.2]

[6.4 2.9 4.3 1.3]

[6.6 3. 4.4 1.4]

[6.8 2.8 4.8 1.4]

[6.7 3. 5. 1.7]

[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1. ]

[5.5 2.4 3.8 1.1]

[5.5 2.4 3.7 1. ]

[5.8 2.7 3.9 1.2]

[6. 2.7 5.1 1.6]

[5.4 3. 4.5 1.5]

[6. 3.4 4.5 1.6]

[6.7 3.1 4.7 1.5]

[6.3 2.3 4.4 1.3]

[5.6 3. 4.1 1.3]

[5.5 2.5 4. 1.3]

[5.5 2.6 4.4 1.2]

[6.1 3. 4.6 1.4]

[5.8 2.6 4. 1.2]

[5. 2.3 3.3 1. ]

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

[5.8 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2]

[7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2. ]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

[5.7 2.5 5. 2. ]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2. ]

[7.7 2.8 6.7 2. ]

[6.3 2.7 4.9 1.8]

[6.7 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

[6.2 2.8 4.8 1.8]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.1]

[7.2 3. 5.8 1.6]

[7.4 2.8 6.1 1.9]

[7.9 3.8 6.4 2. ]

[6.4 2.8 5.6 2.2]

[6.3 2.8 5.1 1.5]

[6.1 2.6 5.6 1.4]

[7.7 3. 6.1 2.3]

[6.3 3.4 5.6 2.4]

[6.4 3.1 5.5 1.8]

[6. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

[6.7 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

[6.8 3.2 5.9 2.3]

[6.7 3.3 5.7 2.5]

[6.7 3. 5.2 2.3]

[6.3 2.5 5. 1.9]

[6.5 3. 5.2 2. ]

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

confusion matrix is as follows

[[14 0 0]

[ 0 11 0]

[ 0 2 18]]

Accuracy metrics

precision recall f1-score support

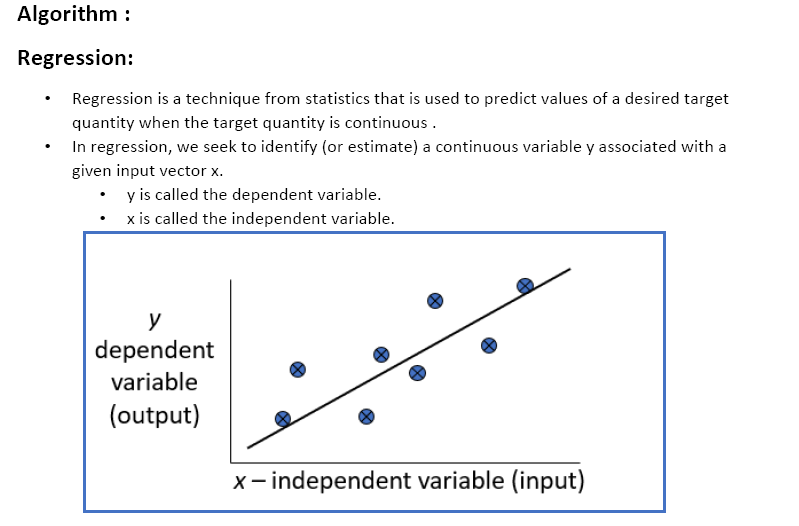
0 1.00 1.00 1.00 14

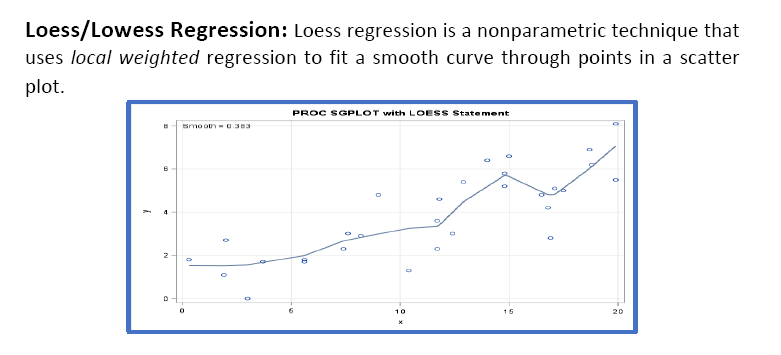
1 0.85 1.00 0.92 11

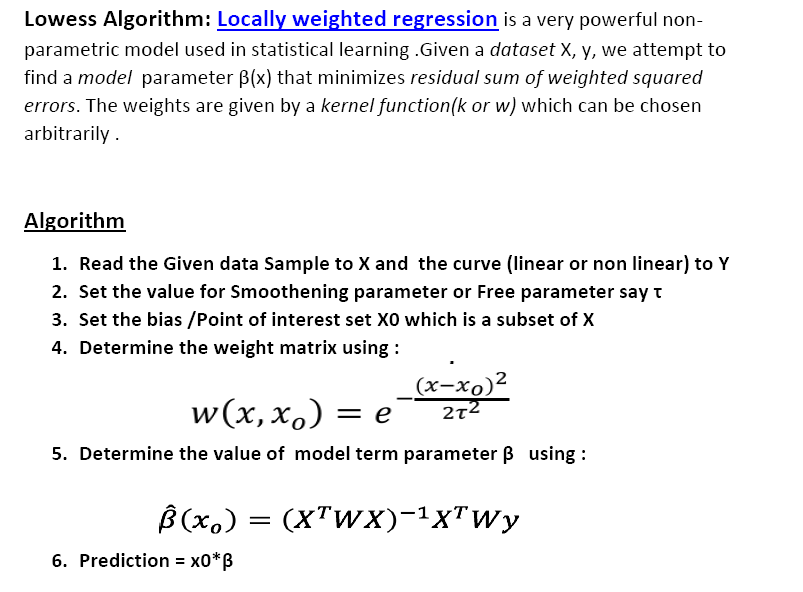
2 1.00 0.90 0.95 20

avg / total 0.96 0.96 0.96 45

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







**Source program**

from numpy import \*

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np1

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')

print(data)

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]

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();

**Input data Set data10.csv**

total\_bill,tip,sex,smoker,day,time,size

16.99,1.01,Female,No,Sun,Dinner,2

10.34,1.66,Male,No,Sun,Dinner,3

21.01,3.5,Male,No,Sun,Dinner,3

23.68,3.31,Male,No,Sun,Dinner,2

24.59,3.61,Female,No,Sun,Dinner,4

25.29,4.71,Male,No,Sun,Dinner,4

8.77,2,Male,No,Sun,Dinner,2

26.88,3.12,Male,No,Sun,Dinner,4

15.04,1.96,Male,No,Sun,Dinner,2

14.78,3.23,Male,No,Sun,Dinner,2

10.27,1.71,Male,No,Sun,Dinner,2

35.26,5,Female,No,Sun,Dinner,4

15.42,1.57,Male,No,Sun,Dinner,2

18.43,3,Male,No,Sun,Dinner,4

14.83,3.02,Female,No,Sun,Dinner,2

21.58,3.92,Male,No,Sun,Dinner,2

10.33,1.67,Female,No,Sun,Dinner,3

16.29,3.71,Male,No,Sun,Dinner,3

16.97,3.5,Female,No,Sun,Dinner,3

20.65,3.35,Male,No,Sat,Dinner,3

17.92,4.08,Male,No,Sat,Dinner,2

20.29,2.75,Female,No,Sat,Dinner,2

15.77,2.23,Female,No,Sat,Dinner,2

39.42,7.58,Male,No,Sat,Dinner,4

19.82,3.18,Male,No,Sat,Dinner,2

17.81,2.34,Male,No,Sat,Dinner,4

13.37,2,Male,No,Sat,Dinner,2

12.69,2,Male,No,Sat,Dinner,2

21.7,4.3,Male,No,Sat,Dinner,2

19.65,3,Female,No,Sat,Dinner,2

9.55,1.45,Male,No,Sat,Dinner,2

18.35,2.5,Male,No,Sat,Dinner,4

15.06,3,Female,No,Sat,Dinner,2

20.69,2.45,Female,No,Sat,Dinner,4

17.78,3.27,Male,No,Sat,Dinner,2

24.06,3.6,Male,No,Sat,Dinner,3

16.31,2,Male,No,Sat,Dinner,3

16.93,3.07,Female,No,Sat,Dinner,3

18.69,2.31,Male,No,Sat,Dinner,3

31.27,5,Male,No,Sat,Dinner,3

16.04,2.24,Male,No,Sat,Dinner,3

17.46,2.54,Male,No,Sun,Dinner,2

13.94,3.06,Male,No,Sun,Dinner,2

9.68,1.32,Male,No,Sun,Dinner,2

30.4,5.6,Male,No,Sun,Dinner,4

18.29,3,Male,No,Sun,Dinner,2

22.23,5,Male,No,Sun,Dinner,2

32.4,6,Male,No,Sun,Dinner,4

28.55,2.05,Male,No,Sun,Dinner,3

18.04,3,Male,No,Sun,Dinner,2

12.54,2.5,Male,No,Sun,Dinner,2

10.29,2.6,Female,No,Sun,Dinner,2

34.81,5.2,Female,No,Sun,Dinner,4

9.94,1.56,Male,No,Sun,Dinner,2

25.56,4.34,Male,No,Sun,Dinner,4

19.49,3.51,Male,No,Sun,Dinner,2

38.01,3,Male,Yes,Sat,Dinner,4

26.41,1.5,Female,No,Sat,Dinner,2

11.24,1.76,Male,Yes,Sat,Dinner,2

**Output:**

total\_bill tip sex smoker day time size

0 16.99 1.01 Female No Sun Dinner 2

1 10.34 1.66 Male No Sun Dinner 3

2 21.01 3.50 Male No Sun Dinner 3

3 23.68 3.31 Male No Sun Dinner 2

4 24.59 3.61 Female No Sun Dinner 4

5 25.29 4.71 Male No Sun Dinner 4

6 8.77 2.00 Male No Sun Dinner 2

7 26.88 3.12 Male No Sun Dinner 4

8 15.04 1.96 Male No Sun Dinner 2

9 14.78 3.23 Male No Sun Dinner 2

10 10.27 1.71 Male No Sun Dinner 2

11 35.26 5.00 Female No Sun Dinner 4

12 15.42 1.57 Male No Sun Dinner 2

13 18.43 3.00 Male No Sun Dinner 4

14 14.83 3.02 Female No Sun Dinner 2

15 21.58 3.92 Male No Sun Dinner 2

16 10.33 1.67 Female No Sun Dinner 3

17 16.29 3.71 Male No Sun Dinner 3

18 16.97 3.50 Female No Sun Dinner 3

19 20.65 3.35 Male No Sat Dinner 3

20 17.92 4.08 Male No Sat Dinner 2

21 20.29 2.75 Female No Sat Dinner 2

22 15.77 2.23 Female No Sat Dinner 2

23 39.42 7.58 Male No Sat Dinner 4

24 19.82 3.18 Male No Sat Dinner 2

25 17.81 2.34 Male No Sat Dinner 4

26 13.37 2.00 Male No Sat Dinner 2

27 12.69 2.00 Male No Sat Dinner 2

28 21.70 4.30 Male No Sat Dinner 2

29 19.65 3.00 Female No Sat Dinner 2

.. ... ... ... ... ... ... ...

214 28.17 6.50 Female Yes Sat Dinner 3

215 12.90 1.10 Female Yes Sat Dinner 2

216 28.15 3.00 Male Yes Sat Dinner 5

217 11.59 1.50 Male Yes Sat Dinner 2

218 7.74 1.44 Male Yes Sat Dinner 2

219 30.14 3.09 Female Yes Sat Dinner 4

220 12.16 2.20 Male Yes Fri Lunch 2

221 13.42 3.48 Female Yes Fri Lunch 2

222 8.58 1.92 Male Yes Fri Lunch 1

223 15.98 3.00 Female No Fri Lunch 3

224 13.42 1.58 Male Yes Fri Lunch 2

225 16.27 2.50 Female Yes Fri Lunch 2

226 10.09 2.00 Female Yes Fri Lunch 2

227 20.45 3.00 Male No Sat Dinner 4

228 13.28 2.72 Male No Sat Dinner 2

229 22.12 2.88 Female Yes Sat Dinner 2

230 24.01 2.00 Male Yes Sat Dinner 4

231 15.69 3.00 Male Yes Sat Dinner 3

232 11.61 3.39 Male No Sat Dinner 2

233 10.77 1.47 Male No Sat Dinner 2

234 15.53 3.00 Male Yes Sat Dinner 2

235 10.07 1.25 Male No Sat Dinner 2

236 12.60 1.00 Male Yes Sat Dinner 2

237 32.83 1.17 Male Yes Sat Dinner 2

238 35.83 4.67 Female No Sat Dinner 3

239 29.03 5.92 Male No Sat Dinner 3

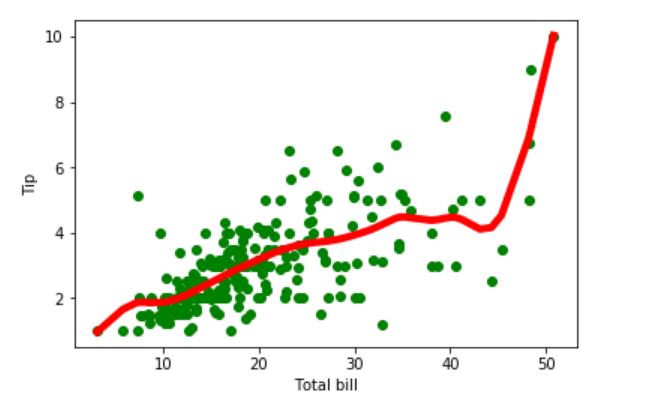
240 27.18 2.00 Female Yes Sat Dinner 2

241 22.67 2.00 Male Yes Sat Dinner 2

242 17.82 1.75 Male No Sat Dinner 2

243 18.78 3.00 Female No Thur Dinner 2

[244 rows x 7 columns]

****