 **BELLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT**

**Department of Computer Science & Engineering**

**Academic Year: 2021 - 2022**

**LAB MANUAL**

Semester : VII

Branch **:** CSE

Subject : : Artifical Intelligence and Machine Learning Lab

Subject Code : 18CSL76

Faculty Name : Dr. Yeresime Suresh/Azar Baig M/Shenaz Begum S/Lakshmi Sharma K M

**Syllabus**

|  |
| --- |
| 1. Implement A\* Search algorithm. 2. Implement AO\* Search algorithm. 3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples. 4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. 5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets. 6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. 7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program. 8. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem. 9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs. |

**Course Outcomes (COs):**

|  |  |
| --- | --- |
| ***Course Outcomes: The students will be able to:*** | |
| ***CO*** | ***Description*** |
| ***CO1*** | Explain the underlying basic theory/foundations of concept learning, applications and challenges of machine learning. |
| ***CO2*** | Apply decision tree learning to visualize the real world data. |
| ***CO3*** | Analyze neural network to train linear and non-linear models. |
| ***CO4*** | Apply conditional probability and Bayes' theorem in analysing high dimensional data set. |
| ***CO5*** | Apply various classification methods to analyze real time data. |

Program 1: Implement A \* algorithm

A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

A\*: When you expand a node n, take each successor n' and place it on PriQueue with priority

(Cost of getting to n') + h(n') (1)

Let g(n) = Cost of getting to n (2)

and then define…

f(n) = g(n) + h(n) (3)

**Advantages:**

• It is complete and optimal.

• It is the best one from other techniques. It is used to solve very complex problems.

• It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A\*.

**Disadvantages:**

• This algorithm is complete if the branching factor is finite and every action has fixed cost.

• The speed execution of A\* search is highly dependent on the accuracy of the heuristic algorithm that is used to

compute h (n).

**Source Code:**

import sys

inf = 99999

g = [

[0,1,inf,inf,inf,10],

[1,0,2,1,inf,inf],

[inf,2,0,inf,5,inf],

[inf,1,inf,0,3,4],

[inf,inf,3,0,2],

[10,inf,inf,4,2,0],

]

h = [5,3,4,2,6,0]

src = 0

goal = 5

class obj:

def \_\_init\_\_(self,cost,path):

self.cost = cost

self.path = path

arr = []

new\_item = obj(h[src],[src])

arr.append(new\_item)

# a\* algorithm

while arr:

cur\_item = arr[0]

cur\_node = cur\_item.path[-1]

cur\_cost = cur\_item.cost

cur\_path = cur\_item.path

for i in range(0,len(h)):

if g[cur\_node][i]!=inf and g[cur\_node][i]!=0:

new\_cost = cur\_cost - h[cur\_node] + h[i] + g[cur\_node][i]

new\_path = cur\_path.copy()

new\_path.append(i)

if i==goal:

print(new\_cost)

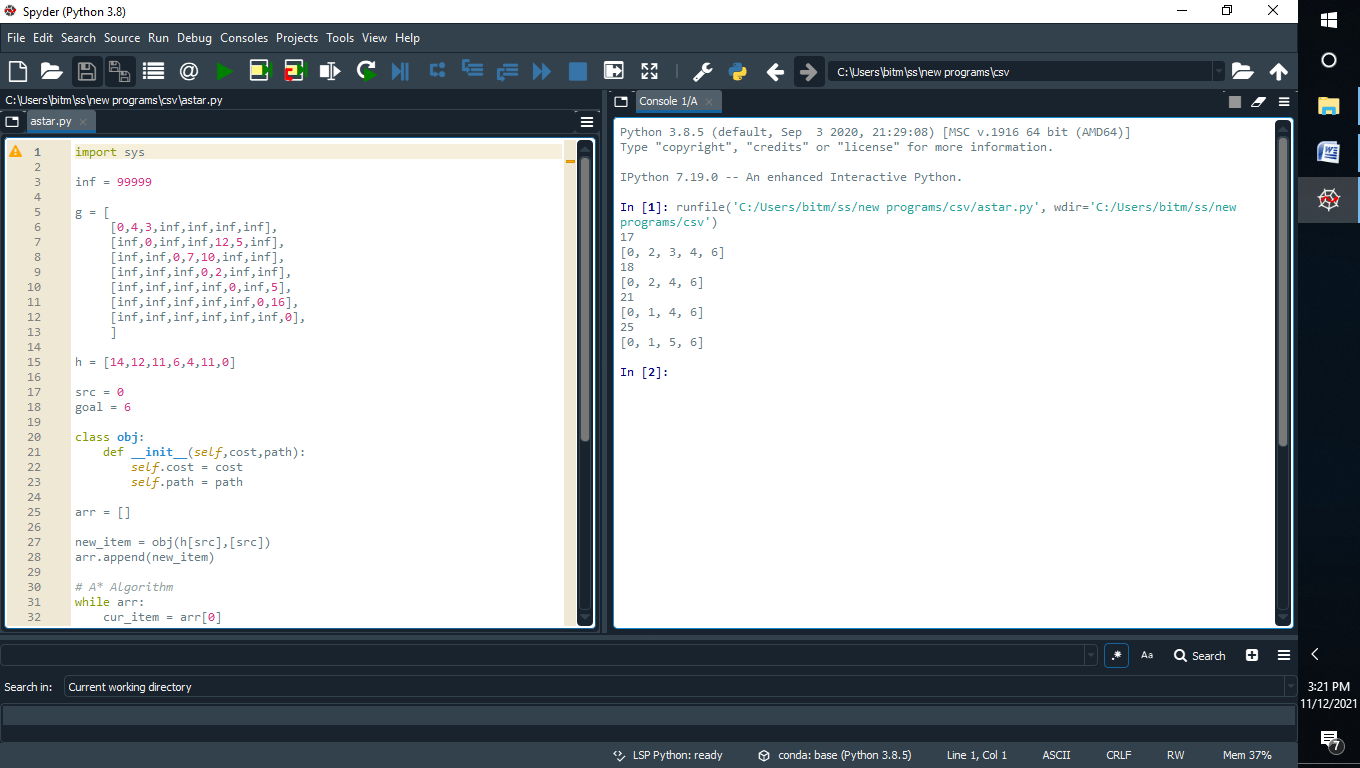
print(new\_path)

# sys.exit()

new\_item = obj(new\_cost,new\_path)

arr.append(new\_item)

arr.pop(0)

arr = sorted(arr,key=lambda item:item.cost) 

Program 2: Implement AO \* algorithm

### ****AO\* Algorithm****

AO\* Algorithm basically based on  problem decompositon (Breakdown problem into small pieces)

When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, **AND-OR graphs**or **AND - OR trees**are used for representing the solution.

The decomposition of the problem or problem reduction generates AND arcs.

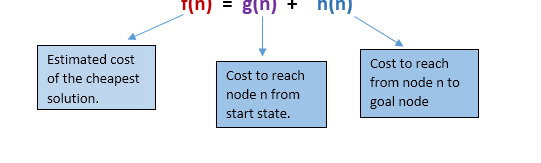
**AND-OR Graph**

**The figure shows an AND-OR graph**

1. To pass any exam, we have two options, either cheating or hard work.
2. In this graph we are given two choices, first do cheating **or (The red line)**work hard and **(The arc)**pass.
3. When we have more than one choice and we have to pick one, we apply **OR condition**to choose one.(That's what we did here).
4. Basically the **ARC**here denote**AND condition**.
5. Here we have replicated the arc between the work hard and the pass because by doing the hard work possibility of passing an exam is more than cheating.

### ****A\* Vs AO\*****

1. Both are part of informed search technique and use heuristic values to solve the problem.
2. The solution is guaranteed in both algorithm.
3. A\*  **always** gives an **optimal solution** (shortest path with low cost) But It is not guaranteed to that**AO\*** always provide **an optimal solutions**.
4. **Reason:** Because AO\* does not explore all the solution path once it got solution.

**Advantages:**

• It is an optimal algorithm.

• If traverse according to the ordering of nodes.

It can be used for both OR and AND graph.

**Disadvantages:**

• Sometimes for unsolvable nodes, it can’t find the optimal p

**Source Code:**

import sys

inf = 99999

g = [

[0,1,inf,inf,inf,10],

[1,0,2,1,inf,inf],

[inf,2,0,inf,5,inf],

[inf,1,inf,0,3,4],

[inf,inf,3,0,2],

[10,inf,inf,4,2,0],

]

h = [5,3,4,2,6,0]

src = 0

goal = 5

class obj:

def \_\_init\_\_(self,cost,path):

self.cost = cost

self.path = path

arr = []

new\_item = obj(h[src],[src])

arr.append(new\_item)

# a\* algorithm

while arr:

cur\_item = arr[0]

cur\_node = cur\_item.path[-1]

cur\_cost = cur\_item.cost

cur\_path = cur\_item.path

for i in range(0,len(h)):

if g[cur\_node][i]!=inf and g[cur\_node][i]!=0:

new\_cost = cur\_cost - h[cur\_node] + h[i] + g[cur\_node][i]

new\_path = cur\_path.copy()

new\_path.append(i)

if i==goal:

print(new\_cost)

print(new\_path)

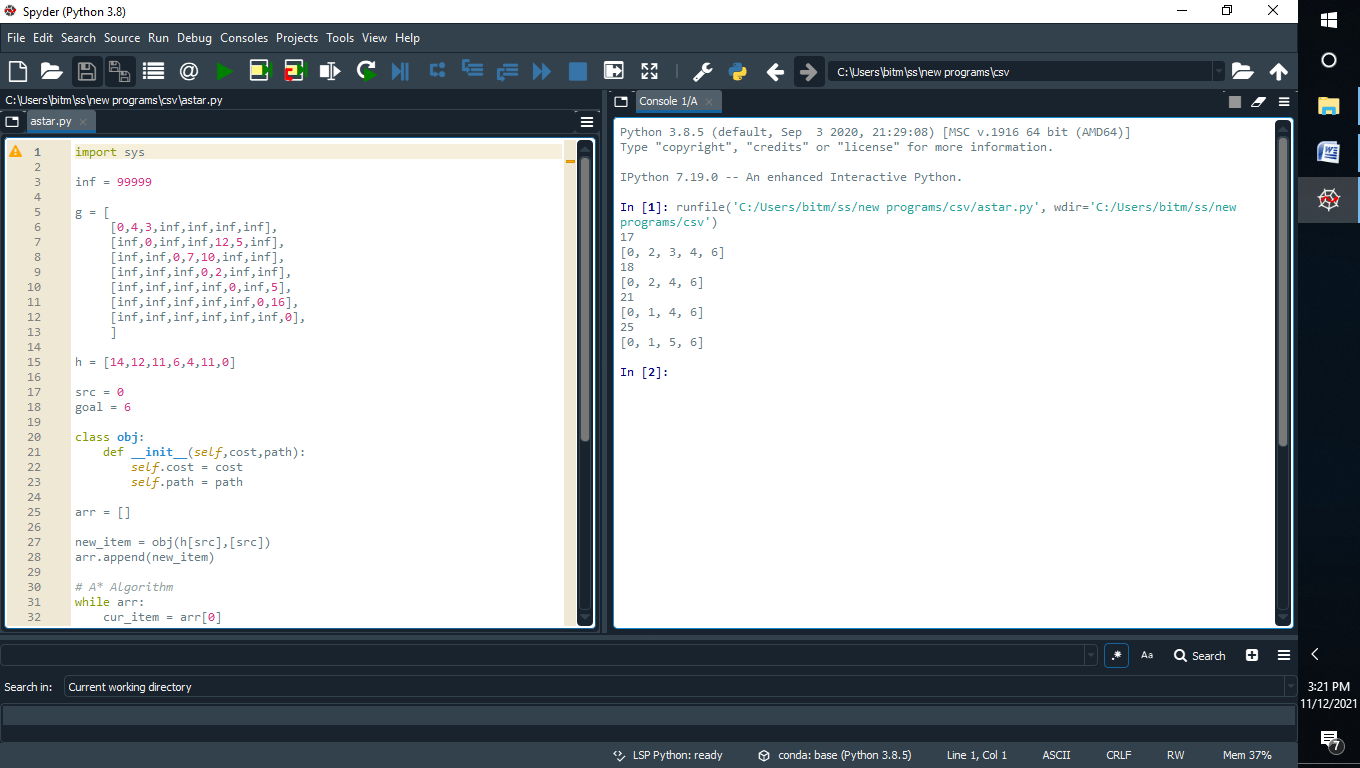
# sys.exit()

new\_item = obj(new\_cost,new\_path)

arr.append(new\_item)

arr.pop(0)

arr = sorted(arr,key=lambda item:item.cost)



**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.

**Algorithm:**

***G*maxim all general hypotheses in H Smaxim all specific hypotheses in H Foreachtrainingexampled=<x,c(x)>Case1:If d is a positive example**

*RemovefromGanyhypothesisthatisinconsistentwithd ForeachhypothesissinSthatisnotconsistentwithd*

* *RemovesfromS.*
* *AddtoSallminimalgeneralizationshofssuchthat*
  + *hconsistentwithd*
  + *SomememberofGismoregeneralthanh*
* *RemovefromSanyhypothesisthatismoregeneralthananotherhypothesisinS*

**Case 2: If d is a negative example**

*RemovefromSanyhypothesisthatisinconsistentwithd ForeachhypothesisginGthatisnotconsistentwithd*

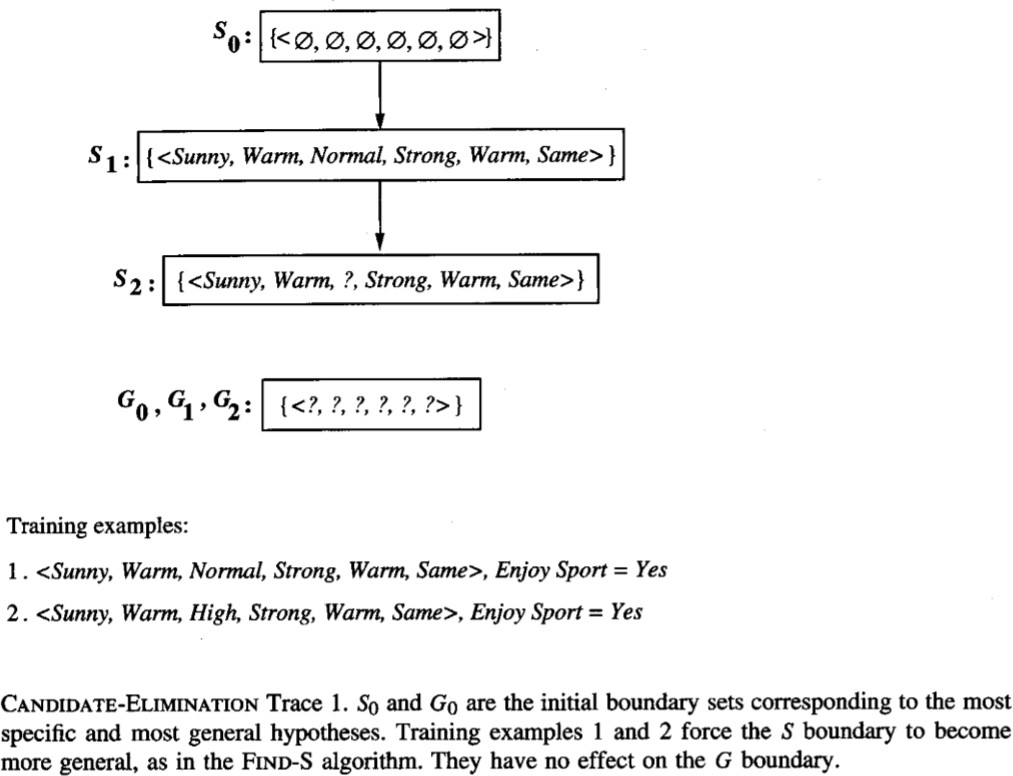
* *Remove g fromG.*
* *Add to G all minimal specializations h of g suchthat*
  + *h consistent withd*
  + *Some member of S is more specific thanh*
* *Remove from G any hypothesis that is less general than another hypothesis inG*

**Illustration:**



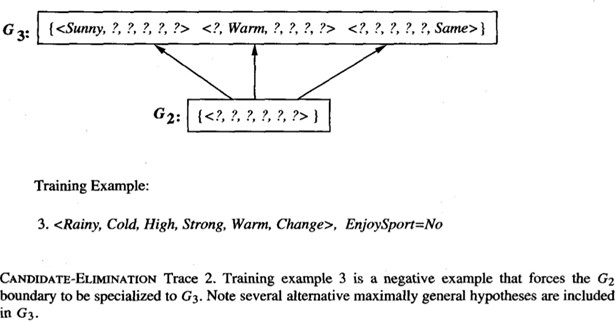
# 

**Trace1:**



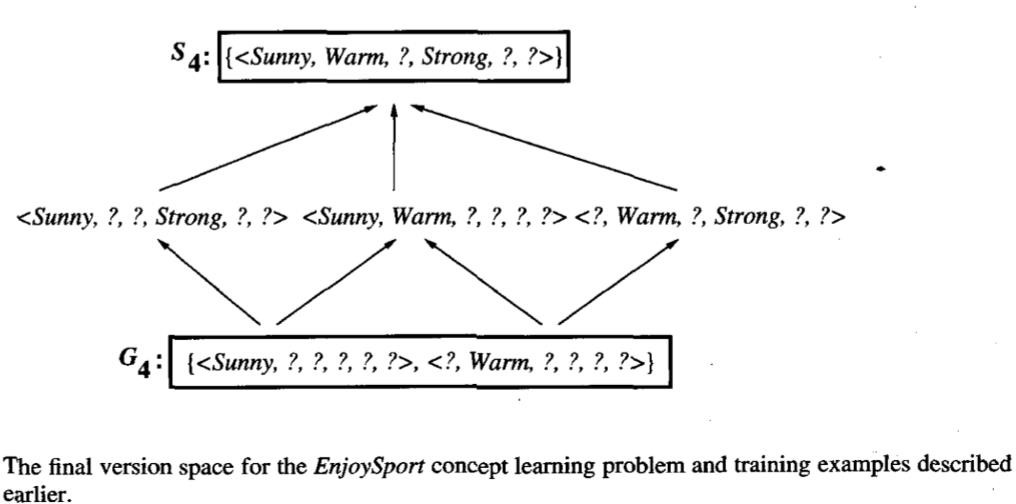
**Trace2:**





**Trace3:**

**FinalVersionSpace:**



**Source Code:**

import csv

a=[]

csvfile=open('1.csv','r') reader=csv.reader(csvfile) for row in reader:

a.append(row) print(row)

num\_attributes=len(a[0])-1 print("Initial hypothesis is ") S=['0']\*num\_attributes G=['?']\*num\_attributes print("The most specific : ",S) print("The most general : ",G)

for j in range(0,num\_attributes): S[j]=a[0][j]

print("The candidate algorithm \n") temp=[]

for i in range(0,len(a)): if(a[i][num\_attributes]=='Yes'):

for j in range(0,num\_attributes): if(a[i][j]!=S[j]):

S[j]='?'

for j in range(0,num\_attributes): for k in range(1,len(temp)):

if temp[k][j]!='?' and temp[k][j]!=S[j]: del temp[k]

print("For instance {0} the hypothesis is S{0}".format(i+1),S) if(len(temp)==0):

print("For instance {0} the hypothesis is G{0}".format(i+1),G) else:

print("For instance {0} the hypothesis is S{0}".format(i+1),temp)

if(a[i][num\_attributes]=='No'):

for j in range(0,num\_attributes): if(S[j]!=a[i][j] and S[j]!='?'):

G[j]=S[j]

temp.append(G)

G=['?']\*num\_attributes

print("For instance {0} the hypothesis is S{0}".format(i+1),S) print("For instance {0} the hypothesis is G{0}".format(i+1),temp)

# output:

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change ', 'No']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change ', 'Yes'] Initial hypothesis is

The most specific : ['0', '0', '0', '0', '0', '0']

The most general : ['?', '?', '?', '?', '?', '?'] The candidate algorithm

For instance 1 the hypothesis is S1 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

For instance 1 the hypothesis is G1 ['?', '?', '?', '?', '?', '?']

For instance 2 the hypothesis is S2 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same'] For instance 2 the hypothesis is G2 ['?', '?', '?', '?', '?', '?']

For instance 3 the hypothesis is S3 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

For instance 3 the hypothesis is G3 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

For instance 4 the hypothesis is S4 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

For instance 4 the hypothesis is S4 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

# 3Data set: playtennis.csv

PlayTennis Outlook Temperature Humidity Wind

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Sunny | Hot | High | Weak |
| No | Sunny | Hot | High | Strong |
| Yes | Overcast | Hot | High | Weak |
| Yes | Rain | Mild | High | Weak |
| Yes | Rain | Cool | Normal | Weak |
| No | Rain | Cool | Normal | Strong |
| Yes | Overcast | Cool | Normal | Strong |
| No | Sunny | Mild | High | Weak |
| Yes | Sunny | Cool | Normal | Weak |
| Yes | Rain | Mild | Normal | Weak |
| Yes | Sunny | Mild | Normal | Strong |
| Yes | Overcast | Mild | High | Strong |
| Yes | Overcast | Hot | Normal | Weak |
| No | Rain | Mild | High | Strong |

# 3output:

Given Play Tennis Data Set:

PlayTennis Outlook Temperature Humidity Wind

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | No | Sunny | Hot | High | Weak |
| 1 | No | Sunny | Hot | High | Strong |
| 2 | Yes | Overcast | Hot | High | Weak |
| 3 | Yes | Rain | Mild | High | Weak |
| 4 | Yes | Rain | Cool | Normal | Weak |
| 5 | No | Rain | Cool | Normal | Strong |
| 6 | Yes | Overcast | Cool | Normal | Strong |
| 7 | No | Sunny | Mild | High | Weak |
| 8 | Yes | Sunny | Cool | Normal | Weak |
| 9 | Yes | Rain | Mild | Normal | Weak |
| 10 | Yes | Sunny | Mild | Normal | Strong |
| 11 | Yes | Overcast | Mild | High | Strong |
| 12 | Yes | Overcast | Hot | Normal | Weak |
| 13 | No | Rain | Mild | High | Strong |

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind'] Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']

Gain= [0.2467498197744391, 0.029222565658954647, 0.15183550136234136,

0.04812703040826927]

Best Attribute: Outlook

Gain= [0.01997309402197489, 0.01997309402197489, 0.9709505944546686]

Best Attribute: Wind

Gain= [0.5709505944546686, 0.9709505944546686, 0.01997309402197489]

Best Attribute: Humidity

The Resultant Decision Tree is :

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

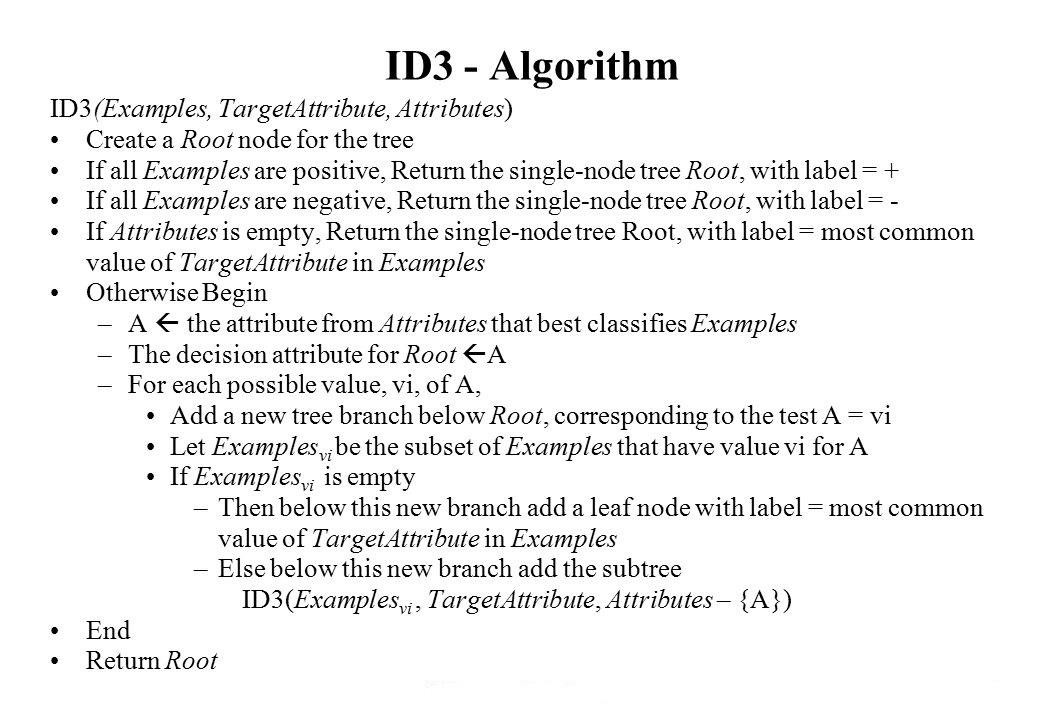
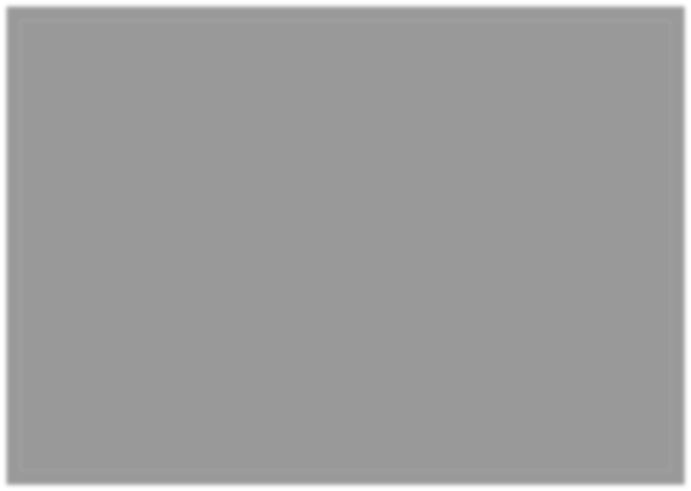
{'High': 'No', 'Normal': 'Yes'}}}}

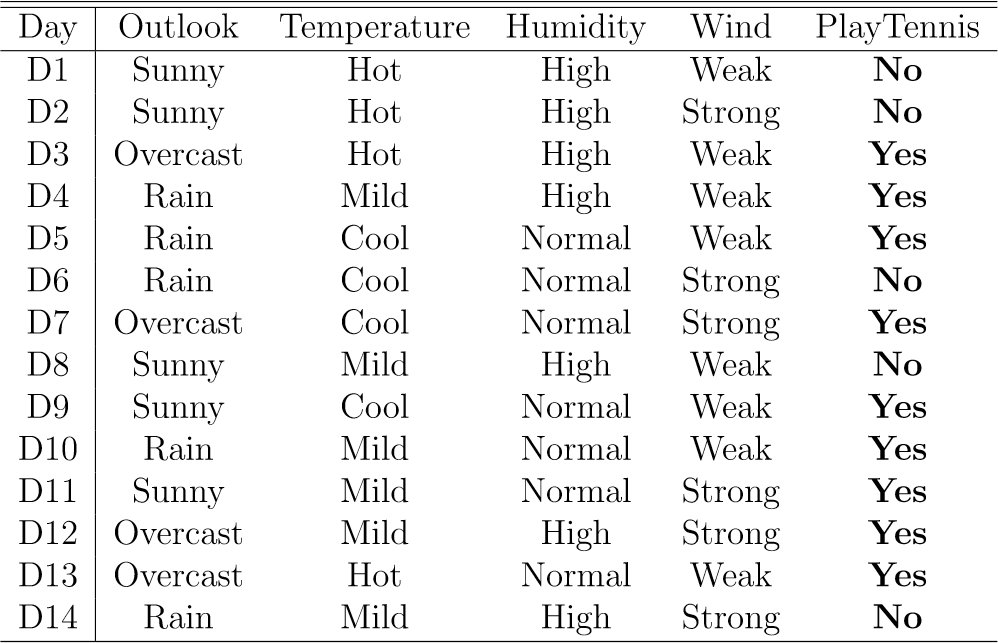
Program: 4.Write a program to demonstrate the working of the decision tree based ID3 algorithm.

Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

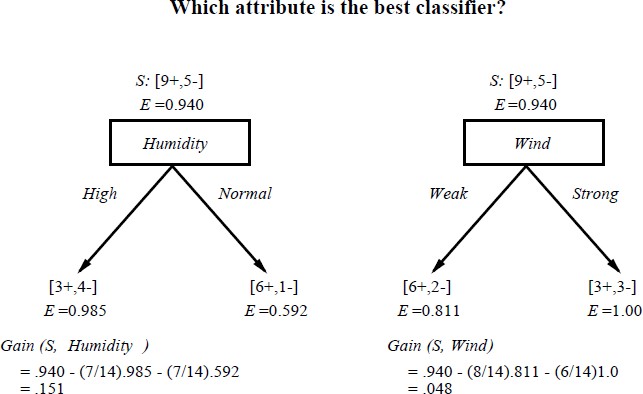
**ALGORITHM:**

ILLUSTRATION:





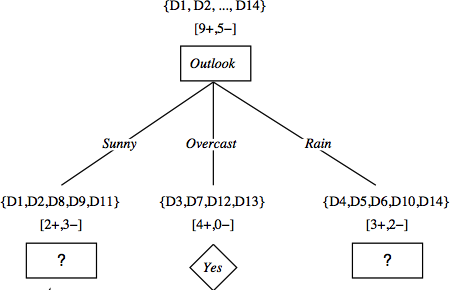
**ComputetheGainandidentifywhichattributeisthebestasillustratedbelow**



**Whichattributetotestattheroot?**

* **Whichattributeshouldbetestedattheroot?**
  + ***Gain*(*S*, *Outlook*) =0.246**
  + ***Gain*(*S*, *Humidity*) =0.151**
  + ***Gain*(*S*, *Wind*) =0.048**
  + ***Gain*(*S*, *Temperature*) =0.029**
* ***Outlook*providesthebestpredictionforthetarget**
* **Lets grow thetree:**
  + **addtothetreeasuccessorforeachpossiblevalueof*Outlook***
  + **partitionthetrainingsamplesaccordingtothevalueof*Outlook***

**After firststep**



**Secondstep**

* **Workingon*Outlook=Sunny*node:**

***Gain*(*SSunny*,*Humidity*)=0.9703/50.02/50.0=0.970 *Gain*(*SSunny*,*Wind*)=0.9702/51.03.50.918=0.019 *Gain*(*SSunny*,*Temp.*)=0.9702/50.02/51.01/50.0=0.570**

* ***Humidity*providesthebestpredictionforthetarget**
* **Lets grow thetree:**
  + **addtothetreeasuccessorforeachpossiblevalueof*Humidity***

**partitionthetrainingsamplesaccordingtothevalueof*Humidity***

**Second and thirdsteps**

### SOFTWARE, PACKAGES AND LIBRARY REQUIREMENT:

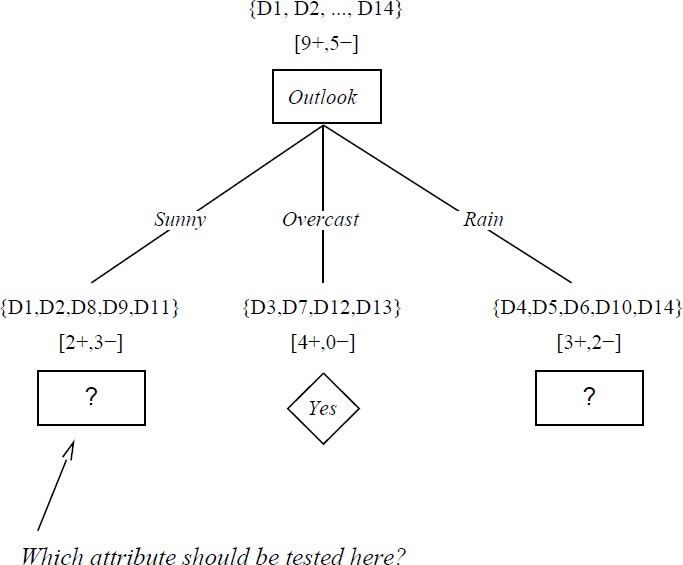
Software packages- python2.7

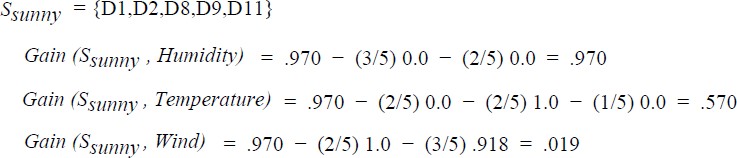
Library – numpy, math

**Numpy**- Stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. You can import this library as

Import numpy

**Math**- It provides access to the mathematical functions defined.you can import this libraryas

Importmath



INPUTS ANDOUTPUTS

**Input-** Input to the decision algorithm is a dataset stored in .csv file which consists of attributes, examples, target concept.

**Output**- For the given dataset decision tree algorithm produces the decision tree starting with rootnode which has highest information gain.

**Source Code:**

import pandas as pd

from collections import Counter import math

tennis = pd.read\_csv('playtennis.csv')

print("\n Given Play Tennis Data Set:\n\n", tennis)

def entropy(alist):

c = Counter(x for x in alist) instances = len(alist)

prob = [x / instances for x in c.values()] return sum( [-p\*math.log(p, 2) for p in prob] )

def information\_gain(d, split, target): splitting = d.groupby(split)

n = len(d.index)

agent = splitting.agg({target : [entropy, lambda x: len(x)/n] })[target] #aggregating agent.columns = ['Entropy', 'observations']

newentropy = sum( agent['Entropy'] \* agent['observations'] ) oldentropy = entropy(d[target])

return oldentropy - newentropy

def id3(sub, target, a):

count = Counter(x for x in sub[target])# class of YES /NO if len(count) == 1:

return next(iter(count)) # next input data set, or raises StopIteration when EOF is hit

else:

gain = [information\_gain(sub, attr, target) for attr in a] print("Gain=",gain)

maximum = gain.index(max(gain)) best = a[maximum]

print("Best Attribute:",best) tree = {best:{}}

remaining = [i for i in a if i != best]

for val, subset in sub.groupby(best): subtree = id3(subset,target,remaining) tree[best][val] = subtree

return tree

names = list(tennis.columns) print("List of Attributes:", names) names.remove('PlayTennis') print("Predicting Attributes:", names)

tree = id3(tennis,'PlayTennis',names) print("\n\nThe Resultant Decision Tree is :\n") print(tree)

# output:

Given Play Tennis Data Set:

PlayTennis Outlook Temperature Humidity Wind

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | No | Sunny | Hot | High | Weak |
| 1 | No | Sunny | Hot | High | Strong |
| 2 | Yes | Overcast | Hot | High | Weak |
| 3 | Yes | Rain | Mild | High | Weak |
| 4 | Yes | Rain | Cool | Normal | Weak |
| 5 | No | Rain | Cool | Normal | Strong |
| 6 | Yes | Overcast | Cool | Normal | Strong |
| 7 | No | Sunny | Mild | High | Weak |
| 8 | Yes | Sunny | Cool | Normal | Weak |
| 9 | Yes | Rain | Mild | Normal | Weak |
| 10 | Yes | Sunny | Mild | Normal | Strong |
| 11 | Yes | Overcast | Mild | High | Strong |
| 12 | Yes | Overcast | Hot | Normal | Weak |
| 13 | No | Rain | Mild | High | Strong |

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind'] Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']

Gain= [0.2467498197744391, 0.029222565658954647, 0.15183550136234136,

0.04812703040826927]

Best Attribute: Outlook

Gain= [0.01997309402197489, 0.01997309402197489, 0.9709505944546686]

Best Attribute: Wind

Gain= [0.5709505944546686, 0.9709505944546686, 0.01997309402197489]

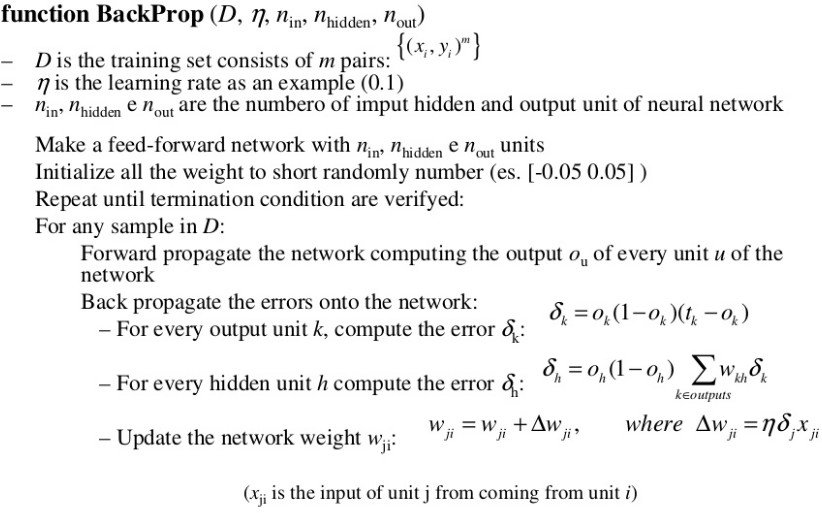
Best Attribute: Humidity

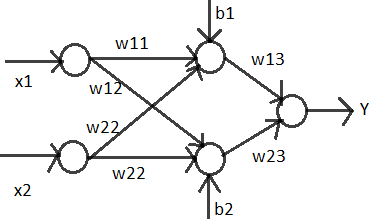
The Resultant Decision Tree is :

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

{'High': 'No', 'Normal': 'Yes'}}}}

**Program5**: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate datasets.





**Source Code:**

import math def sigmoid(x):

y=1/(1+math.exp(-x)) return y

x1=[0,0,1,1]

x2=[0,1,0,1]

t=[0,1,1,0] b1=-0.3 w11=0.21 w21=0.15 b2=0.25 w12=-0.4 w22=0.1 b3=-0.4 w13=-0.2 w23=0.3

error=0 iteration=0 train=True

print("weigth are: ")

print("w11 :%4.2f w12:%4.2f w21:%4.2f w22:%4.2f w13:%4.2f w23:%4.2f \n"

%(w11,w12,w21,w22,w13,w23))

while(train):

for i in range(len(x1)):

z\_in1=b1+x1[i]\*w11+x2[i]\*w21 z\_in2=b2+x1[i]\*w12+x2[i]\*w22 z1=round(sigmoid(z\_in1),4) z2=round(sigmoid(z\_in2),4)

y\_in=b3+z1\*w13+z2\*w23 y=round(sigmoid(y\_in),4)

del\_k=round((t[i]-y)\*y\*(1-y),4) error=del\_k

w13=round(w13+del\_k\*z1,4) w23=round(w23+del\_k\*z2,4) b3=round(b3+del\_k,4)

del\_1=del\_k\*w13\*z1\*(1-z1) del\_2=del\_k\*w23\*z2\*(1-z2)

b1=round(b1+del\_1,4) w11=round(w11+del\_1\*x1[i],4) w12=round(w12+del\_1\*x1[i],4)

b2=round(b2+del\_2,4) w21=round(w21+del\_2\*x2[i],4) w22=round(w22+del\_2\*x2[i],4)

print("iteration: ",iteration)

print("w11:%5.4f w12:%5.4f w21:%5.4fw22:%5.4f w13:%5.4f w23:%5.4f "% (w11,w12,w21,w22,w13,w23))

print("Error:%5.3f"%del\_k) iteration=iteration+1

if(iteration==1000): train=False

**output:(**it will display all iterations from 1-999) iteration: 997

w11:0.8530 w12:0.2430 w21:0.2374 w22:0.1874 w13:-0.2086 w23:0.3359

Error:0.140 iteration: 998

w11:0.8513 w12:0.2413 w21:0.2374 w22:0.1874 w13:-0.1030 w23:0.4420

Error:0.125 iteration: 999

w11:0.8548 w12:0.2448 w21:0.2325 w22:0.1825 w13:-0.2265 w23:0.3187

Error:-0.141

# Dataset:5.csv

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

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

8,183,64,0,0,23.3,0.627,32,1

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

3,78,50,32,88,31,0.284,26,1

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

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

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

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

10,168,74,0,0,38,0.537,34,1

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

7,100,0,0,0,30,0.484,32,1

# 5output:

Size of dataset is: 768 537

{0: [[1.0, 107.0, 68.0, 19.0, 0.0, 26.5, 0.165, 24.0, 0.0], [1.0, 144.0, 82.0, 40.0, 0.0, 41.3, 0.607, 28.0,

0.0], [1.0, 105.0, 58.0, 0.0, 0.0, 24.3, 0.187, 21.0, 0.0]

{0: [(3.454022988505747, 3.1284989024698904), (110.01724137931035, 26.938498454745453),

(67.92528735632185, 18.368785190361336), (19.612068965517242, 15.312369913377424),

(68.95689655172414, 105.42637942980888), (30.54080459770115, 7.710567727617014),

(0.4458764367816092, 0.31886309966940785), (31.74712643678161, 12.079437732209673)], 1:

[(4.64021164021164, 3.7823318201241096), (143.07407407407408, 32.13758346670748),

(72.03174603174604, 19.92883742963596), (22.49206349206349, 18.234179691371473),

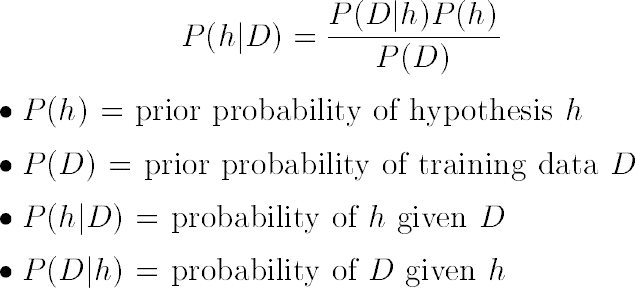
(99.04232804232804, 127.80927573836007), (35.351851851851855, 7.308750166698269),

(0.5427301587301587, 0.3832947121639522), (36.43386243386244, 10.813315097901606)]}

Accuracy: 78.78787878787878

**Program6:** Write a program to implement the naïve Bayesian classifier for a sample training dataset store data CSV file. Compute the accuracy of the classifier, considering few test datasets.

### Bayesian Theorem:



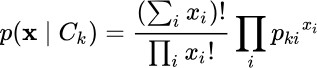
|  |
| --- |
| **NaiveBayes:**FortheBayesianRuleabove,wehavetoextenditsothat  we have |

**Types**

* [**Gaussian:**](http://scikit-learn.org/stable/modules/naive_bayes.html)**Itisusedinclassificationanditassumesthatfeaturesfollowanormaldistribution. GaussianNaiveBayesisusedincaseswhenallourfeaturesarecontinuous.ForexampleinIris datasetfeaturesaresepalwidth,petalwidth,sepallength,petallength.**



* **MultinomialNaiveBayes:Itsisusedwhenwehavediscretedata(e.g.movieratingsranging1 and5aseachratingwillhavecertainfrequencytorepresent).Intextlearningwehavethe countofeachwordtopredicttheclassorlabel**



* **BernoulliNaiveBayes:**It assumesthatallourfeaturesarebinarysuchthattheytakeonly two values. Means 0s can represent “word does not occur in the document” and 1s as "word occurs in thedocument"



**Source Code:**

import csv

import math

import random

import statistics

def cal\_probability(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

dataset=[] dataset\_size=0

with open('lab5.csv') as csvfile: lines=csv.reader(csvfile)

for row in lines:

dataset.append([float(attr) for attr in row]) dataset\_size=len(dataset)

print("Size of dataset is: ",dataset\_size)

train\_size=int(0.7\*dataset\_size) print(train\_size)

X\_train=[] X\_test=dataset.copy()

training\_indexes=random.sample(range(dataset\_size),train\_size)

for i in training\_indexes: X\_train.append(dataset[i]) X\_test.remove(dataset[i])

classes={}

for samples in X\_train: last=int(samples[-1]) if last not in classes:

classes[last]=[] classes[last].append(samples)

print(classes) summaries={}

for classValue,training\_data in classes.items(): summary=[(statistics.mean(attribute),statistics.stdev(attribute)) for attribute in

zip(\*training\_data)] del summary[-1]

summaries[classValue]=summary

print(summaries) X\_prediction=[]

for i in X\_test: probabilities={}

for classValue,classSummary in summaries.items(): probabilities[classValue]=1

for index,attr in enumerate(classSummary): probabilities[classValue]\*=cal\_probability(i[index],attr[0],attr[1])

best\_label,best\_prob=None,-1

for classValue,probability in probabilities.items(): if best\_label is None or probability>best\_prob:

best\_prob=probability best\_label=classValue

X\_prediction.append(best\_label)

correct=0

for index,key in enumerate(X\_test):

if X\_test[index][-1]==X\_prediction[index]: correct+=1

print("Accuracy: ",correct/(float(len(X\_test)))\*100)

# output:

# 

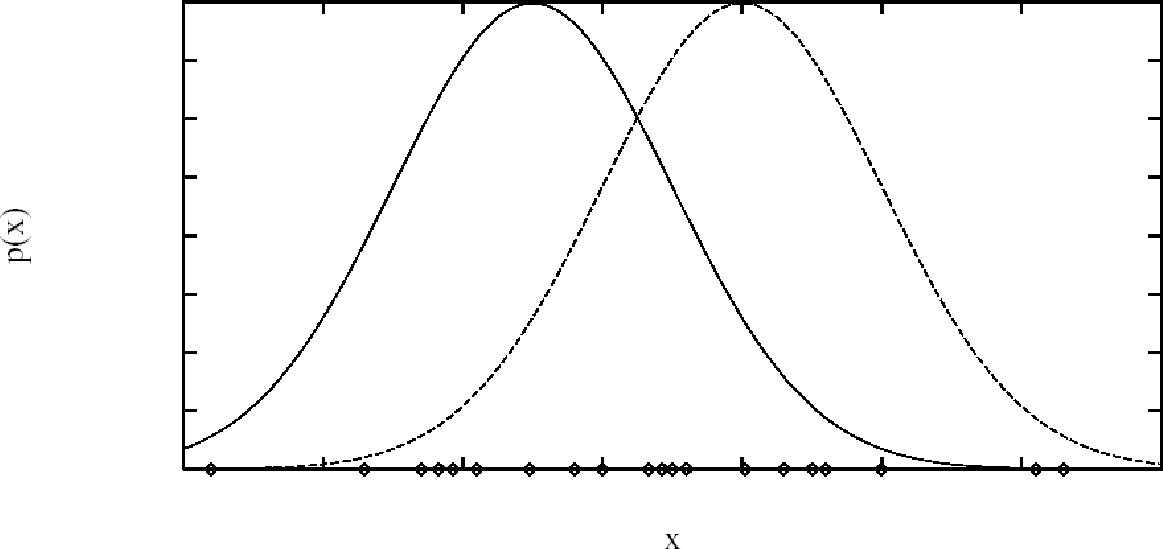
# Program: 7: Apply EM algorithm to clusterasetofdatastoredina.CSV file. Use the same data set for clustering using *k*-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program

### Algorithm :

**Expectation Maximization (EM) Algorithm**

* + - * When touse:
        + Dataisonlypartiallyobservable
        + Unsupervisedclustering(targetvalueunobservable)
        + Supervisedlearning(someinstanceattributesunobservable)
      * Someuses:
        + Train Bayesian BeliefNetworks
        + Unsupervised clustering(AUTOCLASS)
        + Learning Hidden MarkovModels

**GeneratingDatafromMixtureof*k*Gaussians**



• **Eachinstancexgeneratedby**

1. Choosingoneofthe*k*Gaussianswithuniformprobability
2. GeneratinganinstanceatrandomaccordingtothatGaussian

**EMforEstimating*k*Means**

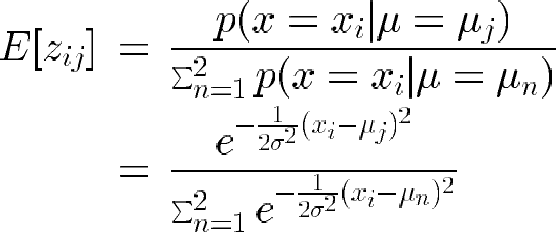
* + - * Given:
        + Instancesfrom*X*generatedbymixtureof*k*Gaussiandistributions
        + Unknownmeans<**1,…,*k*>ofthe*k*Gaussians
        + Don’tknowwhichinstance*xi*wasgeneratedbywhichGaussian
      * Determine:
        + Maximumlikelihoodestimatesof<**1,…,*k*>
      * Thinkoffulldescriptionofeachinstanceas

*yi*= <*xi*, *zi*1, *zi*2> where

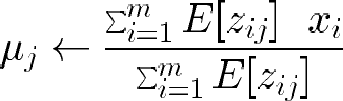
* + - * + *zij*is1if*xi*generatedby*j*thGaussian
        + *xi*observable
        + *zij*unobservable

### • EMAlgorithm:Pickrandominitial*h*=<**1,**2>theniterate

**Estep:**Calculatetheexpectedvalue*E*[*zij*]ofeach hiddenvariable*zij*,assumingthecurrent hypothesis

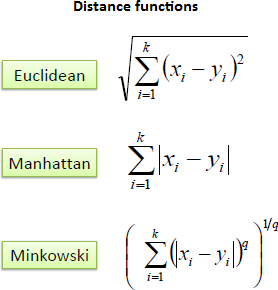
*h* = <**1, **2> holds.

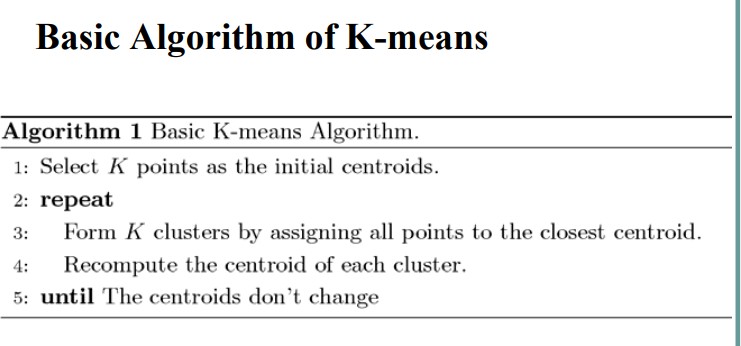
**Mstep:**Calculateanewmaximumlikelihoodhypothesis*h'*=<*'*1,*'*2>,assumingthe valuetakenonbyeachhiddenvariable*zij*isitsexpectedvalue*E*[*zij*]calculatedabove.

Replace *h* = <**1, **2> by *h'* = <*'*1, *'*2>.

### KMeansAlgorithm

* + - * 1.ThesamplespaceisinitiallypartitionedintoKclustersandtheobservationsare randomlyassignedtotheclusters.
      * 2. For eachsample:
        + Calculatethedistancefromtheobservationtothecentroidofthecluster.
        + IFthesampleisclosesttoitsownclusterTHENleaveitELSEselectanother cluster.
      * 3.Repeatsteps1and2untillnoobservationsaremovedfromoneclustertoanother





**Source Code:**

import numpy as np import pandas as pd

from matplotlib import pyplot as plt

from sklearn.mixture import GaussianMixture from sklearn.cluster import KMeans

data = pd.read\_csv('lab8.csv') print("Input Data and Shape") print(data.shape)

data.head()

f1 = data['V1'].values f2 = data['V2'].values

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

print("X ", X)

print('Graph for whole dataset') plt.scatter(f1, f2, c='black', s=7) plt.show()

kmeans = KMeans(20, random\_state=0) labels = kmeans.fit(X).predict(X) print("labels ",labels)

centroids = kmeans.cluster\_centers\_ print("centroids ",centroids)

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

probs = gmm.predict\_proba(X) size = 10 \* probs.max(1) \*\* 3 print('Graph using EM Algorithm')

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

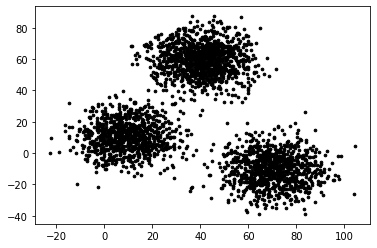
**OUTPUT:**

Input Data and Shape (3000, 3)

X [[ 2.072345 -3.241693][ 17.93671 15.78481 ][ 1.083576 7.319176]...

[ 64.46532 -10.50136 ][ 90.72282 -12.25584 ][ 64.87976 -24.87731 ]]

Graph for whole dataset



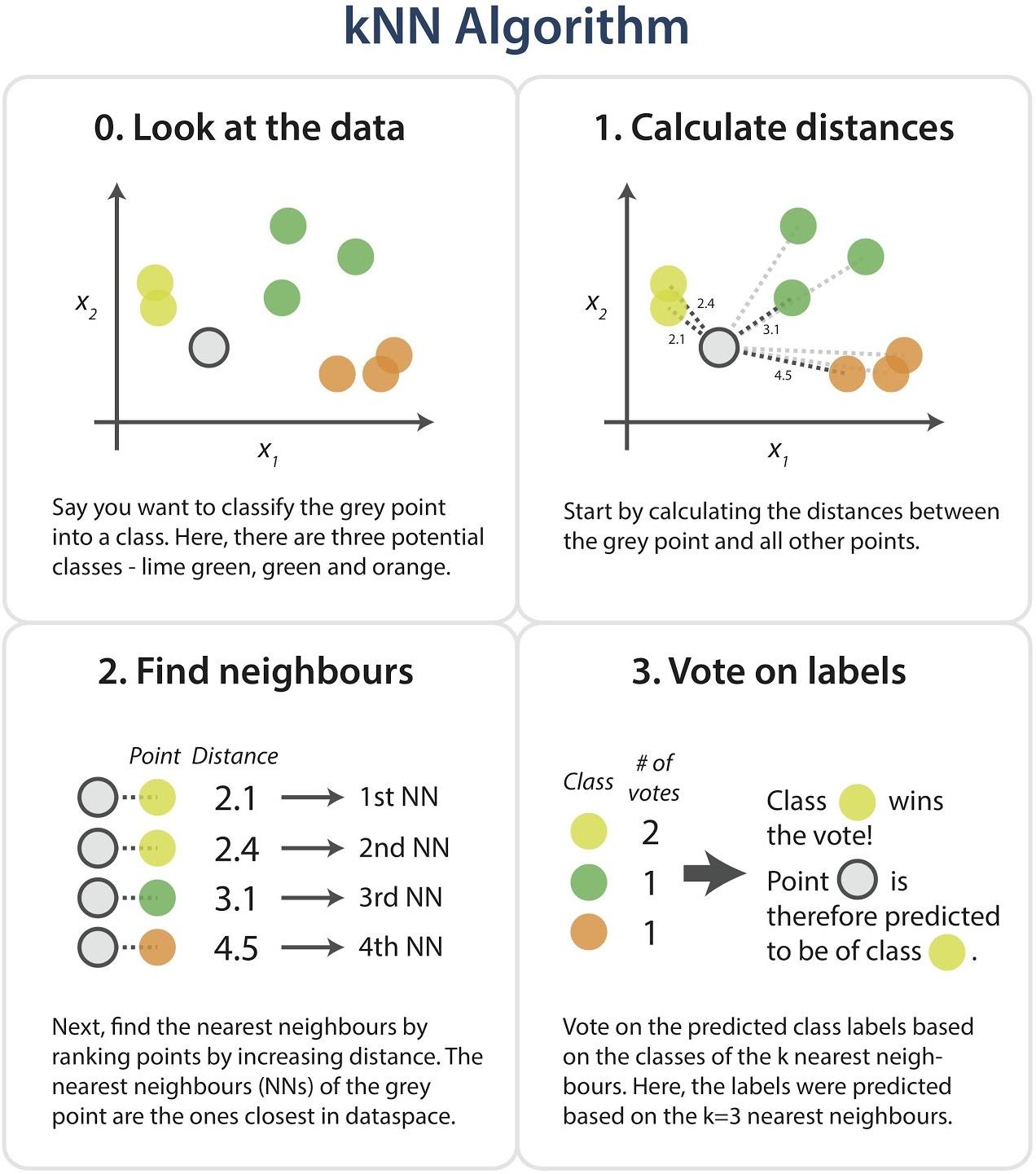
Program 8: Write a program to implement K-nearest neighbor algorithm to classify iris dataset. Print both correct and wrong predication using python machine learning.

k-nearest neighbors algorithm (k-NN) is a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics)method used for[classification](https://en.wikipedia.org/wiki/Statistical_classification)and[regression.](https://en.wikipedia.org/wiki/Regression_analysis)[[1]](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#cite_note-1)In both cases, the input consists of the k closest training examples in the[feature space.](https://en.wikipedia.org/wiki/Feature_space) The output depends on whether k-NN is used for classification orregression.

k-NN is a type of [instance-based learning,](https://en.wikipedia.org/wiki/Instance-based_learning) or[lazy learning,](https://en.wikipedia.org/wiki/Lazy_learning) where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all[machine learning](https://en.wikipedia.org/wiki/Machine_learning)algorithms.

The kNN task can be broken down into writing 3 primary functions:

1. Calculate the distance between any two points
2. Find the nearest neighbours based on these pair wise distances
3. Majority vote on a class labels based on the nearest neighbourlist



### Dataset

Iris dataset, consists of flower measurements for three species of iris flower. Our task is to predict the species labels of a set of flowers based on their flower measurements. Since you‟llbebuildingapredictorbasedonasetofknowncorrectclassifications

The data set contains 3 classes of 151 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly

separable from each other.

**Predicted attribute:** class of iris plant

### Attribute Information:

* + 1. sepal length incm
    2. sepal width incm
    3. petal length incm
    4. petal width incm

### Class:

### Algorithm

1. Load the data and split into train and testsets.
2. Need a measure ofsimilarity.
3. Compute the distance function d(a,b), where a,b are the scenarios composed of N features, such that a={a1,…..,an},b={b1,….,bn}.
4. Euclidean distance measuring: where distance d between two points (a1, a2) and (b1, b2) is given by d = sqrt((a1-b1)^2 + (a2-b2)^2). Each flower in the iris dataset has 4 dimensions, need to find the distance between eachflower.

D=sqrt((a1-b1)^2+(a2-b2)^2+(a3-b3)^2+(a4-b4)^2).

1. The zip function aggregates elements from lists to return a list of tuples. List comprehensions are a powerful Pythonic construct that facilitate quick computations on lists.
2. Iterating over values from the corresponding dimensions in the two datapoints, calculating the differences squared, and storing each dimension's. These are then summed andreturned.
3. This pair wise calculation is done for every train instance and the given testinstance.
4. Next, the distances are sorted in order to find the k closest neighbours to the test instance.
5. Then the training instances ranked from closest to furthest from our test instance, as desired. The function takes the k parameter, which controls how many nearest neighbours arereturned.
6. Finally, using the nearest neighbours just identified, to get a prediction for the class of the test instance by majority voting - simply tally up which class comes up the most often among xthe nearestneighbours.

### Expected Outcome

Iris dataset is a classified dataset . Using K-NN algorithm we need to predict the correct or wrong predictions, and need to get

**Source Code:**

import numpy as np

from sklearn.datasets import load\_iris iris=load\_iris()

x=iris.data y=iris.target print(x[:5],y[:5])

from sklearn.model\_selection import train\_test\_split xtrain,xtest,ytrain,ytest =train\_test\_split(x,y,test\_size=0.4,random\_state=1) print(iris.data.shape)

print(len(xtrain)) print(len(ytest))

from sklearn.neighbors import KNeighborsClassifier knn=KNeighborsClassifier(n\_neighbors=1) knn.fit(xtrain,ytrain)

pred=knn.predict(xtest)

from sklearn import metrics print("Accuracy",metrics.accuracy\_score(ytest,pred)) print(iris.target\_names[2]) ytestn=[iris.target\_names[i] for i in ytest] predn=[iris.target\_names[i] for i in pred]

print(" predicted Actual") for i in range(len(pred)):

print(i," ",predn[i]," ",ytestn[i])

**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]] [0 0 0 0 0]

(150, 4)

90 60

Accuracy 0.9666666666666667

virginica predicted Actual 0 setosa setosa

1. versicolor versicolor
2. versicolor versicolor
3. setosa setosa
4. virginica virginica
5. virginica versicolor
6. virginica virginica
7. setosa setosa
8. setosa setosa
9. virginica virginica
10. versicolor versicolor

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

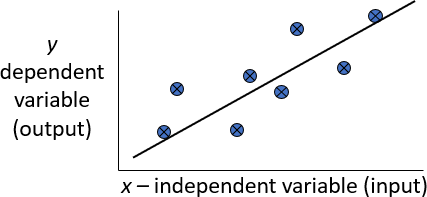
**Algorithm :**Regression:

Regressionisatechniquefromstatisticsthatisusedtopredictvaluesofadesiredtarget quantitywhenthetargetquantityiscontinuous.

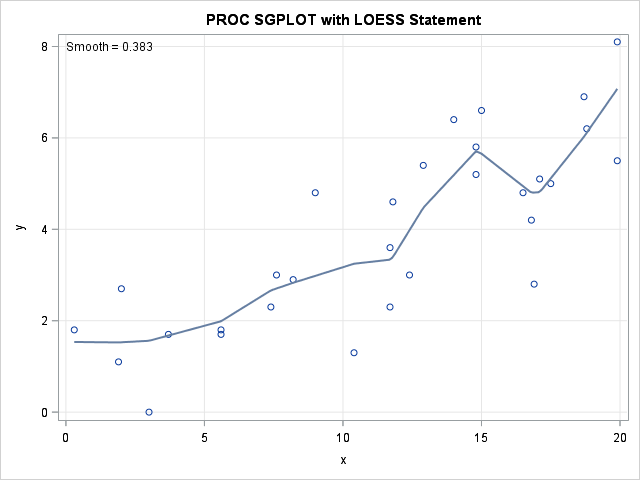
Inregression,weseektoidentify(orestimate)acontinuousvariableyassociatedwitha given input vectorx.

yiscalledthedependentvariable.

xiscalledtheindependentvariable.



**Loess/LowessRegression:**Loessregressionisanonparametrictechniquethat uses*localweighted*regressiontofitasmoothcurvethroughpointsinascatter plot.



**LowessAlgorithm:**

It isa very powerful non- parametric model used in statistical learning.Given a dataset X,y,we attempt to find a model parameterβ(x)that minimizes residual sum of weighted squared errors.The weights are given by a kernel function(korw) which can be chosen arbitrarily.

Algorithm:

Read the Given data Sample to X and the curve(linearornonlinear) toY

**Et the value for Smoothening parameter or Free parameters ay τ**

**Setthebias/PointofinterestsetX 0 which is a subse to fX**

**:**

**Determinethevalueofmodeltermparameterβusing:**



**Prediction =x0\*β**

**Source Code:**

# import numpy as np

import matplotlib.pyplot as plt import pandas as pd

tou = 0.5 data=pd.read\_csv("lab10.csv") X\_train = np.array(data.total\_bill) print(X\_train)

X\_train = X\_train[:, np.newaxis] print(len(X\_train))

y\_train = np.array(data.tip)

X\_test = np.array([i /10 for i in range(500)]) X\_test = X\_test[:, np.newaxis]

y\_test = [] count = 0

for r in range(len(X\_test)):

wts = np.exp(-np.sum((X\_train - X\_test[r]) \*\* 2, axis=1) / (2 \* tou \*\* 2)) W = np.diag(wts)

factor1 = np.linalg.inv(X\_train.T.dot(W).dot(X\_train)) parameters = factor1.dot(X\_train.T).dot(W).dot(y\_train) prediction = X\_test[r].dot(parameters) y\_test.append(prediction)

count += 1 print(len(y\_test))

y\_test = np.array(y\_test) plt.plot(X\_train.squeeze(), y\_train, 'o')

plt.plot(X\_test.squeeze(), y\_test, 'o') plt.show()

**DATASET:[245 rows]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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  **Output** | 1.96 | Male | No | Sun | Dinner | 2 |

