BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
Avalahalli, Doddaballapur Main Road, Bengaluru - 560064
Department of Computer Science and Engineering.

MACHINE LEARNING LABORATORY
18CSL76

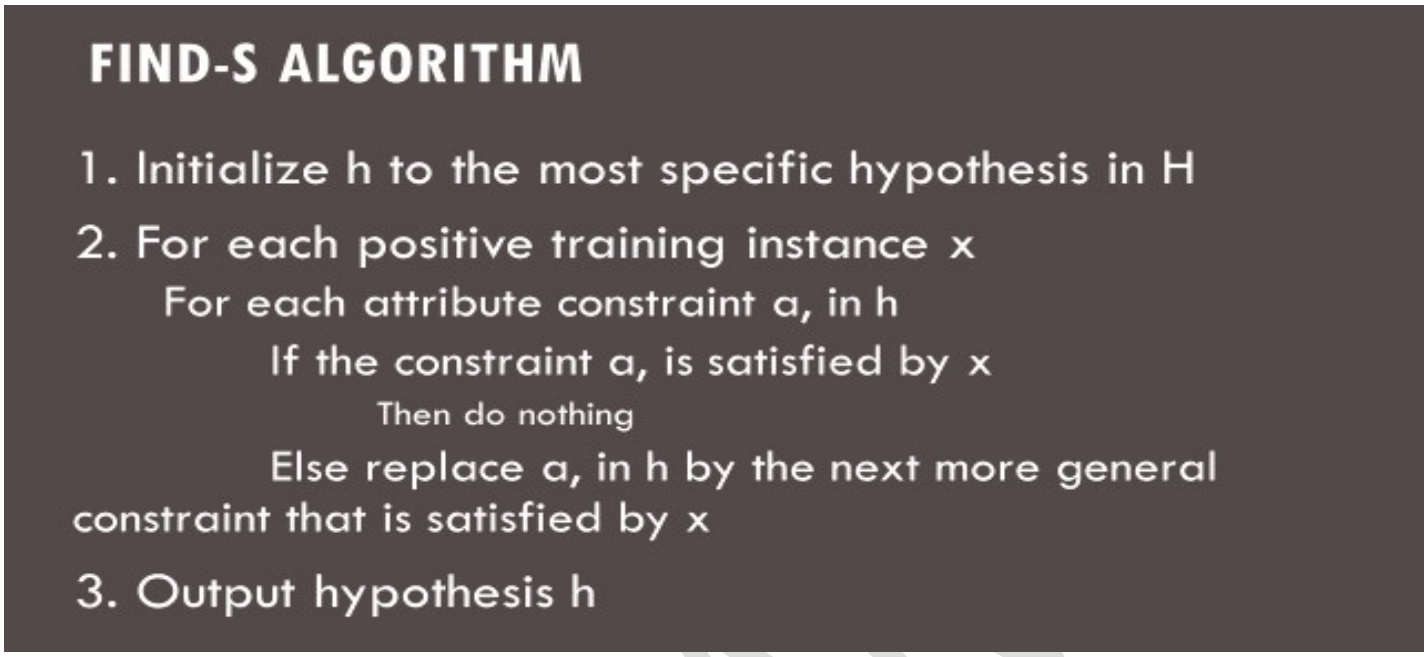
BY
Prof. Chethana C
Department of CSE
BMSIT&M

Programs Implemented By
Bhavya Sheth
1BY15CS110

LAB Experiments

1. Implement and demonstratethe **FIND-Salgorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. 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.
3. Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.
4. Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.
5. 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.
6. Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
7. Write a program to construct a**Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.
8. 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.
9. 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.
10. Implement the non-parametric **Locally Weighted Regressionalgorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

PROGRAM 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.



FIND-S Algorithm starts from the most specific hypothesis and generalize it by considering only positive examples.

• FIND-S algorithm ignores negative examples.

– As long as the hypothesis space contains a hypothesis that describes the true target concept, and the training data contains no errors, ignoring negative examples does not cause to any problem.

• FIND-S algorithm finds the most specific hypothesis within H that is consistent with the positive training examples.

– The final hypothesis will also be consistent with negative examples if the correct target concept is in H, and the training examples are correct.

Program

```
import csv
num_attributes = 6
a = []
print("\n The Given Training Data Set \n")
csvfile = open('enjoysport.csv', 'r')
```

```
reader = csv.reader(csvfile)
for row in reader:
 a.append (row)
 print(row)
 print("\n The initial value of hypothesis: ")
 hypothesis = ['0'] * num_attributes
 print(hypothesis)
 for j in range(0,num_attributes):
 hypothesis[j] = a[0][j];
 print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
 if a[i][num_attributes]=='Yes':
  for j in range(0,num_attributes):
   if a[i][j]!=hypothesis[j]:
    hypothesis[j]='?'
   else :
    hypothesis[j]= a[i][j]
print(" For Training instance No:{0} the hypothesis is ".format(i),hypothesis)
print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

INPUT: enjoysport.csv

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

OUTPUT

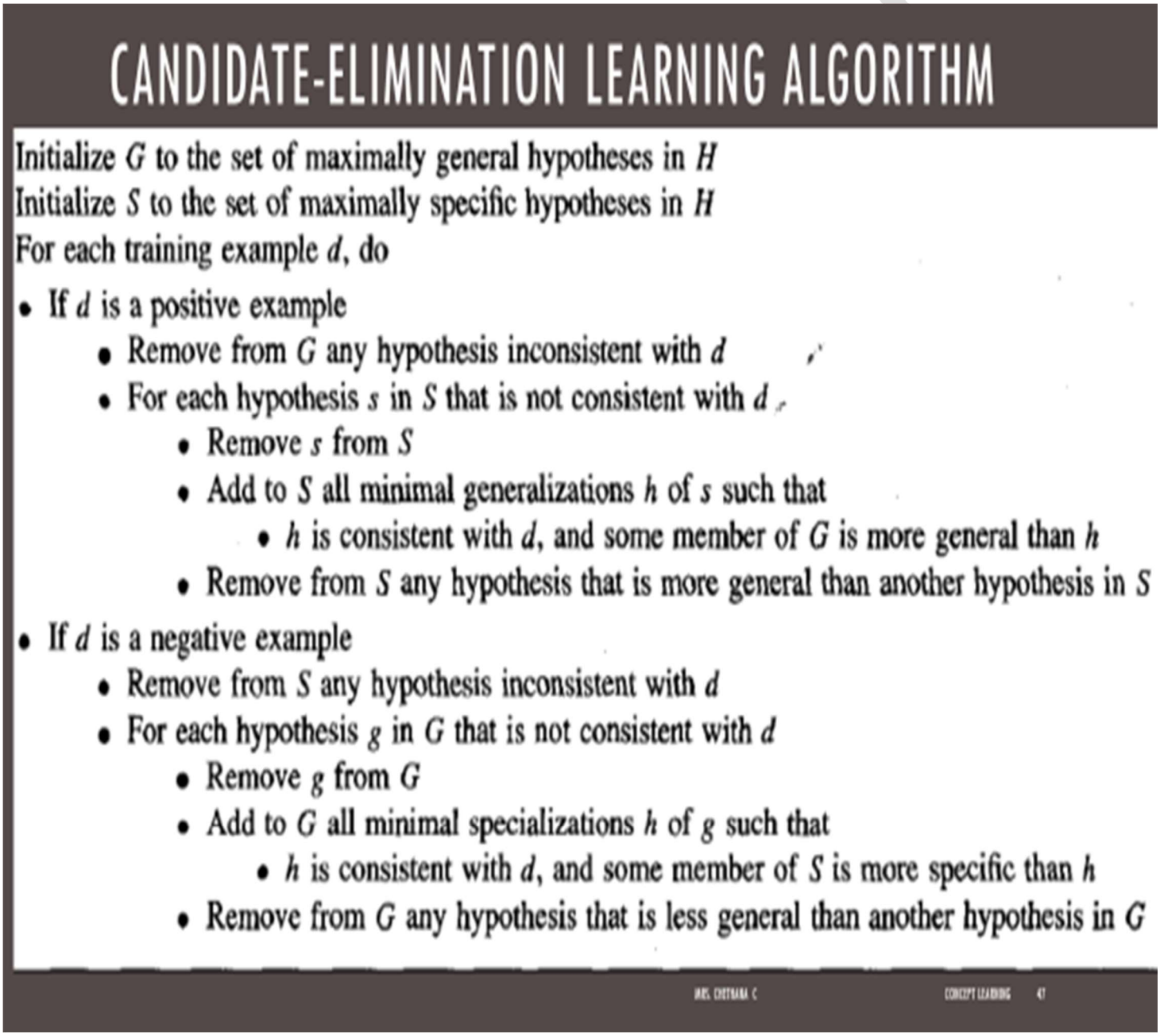
The initial value of hypothesis: ['0', '0', '0', '0', '0', '0']

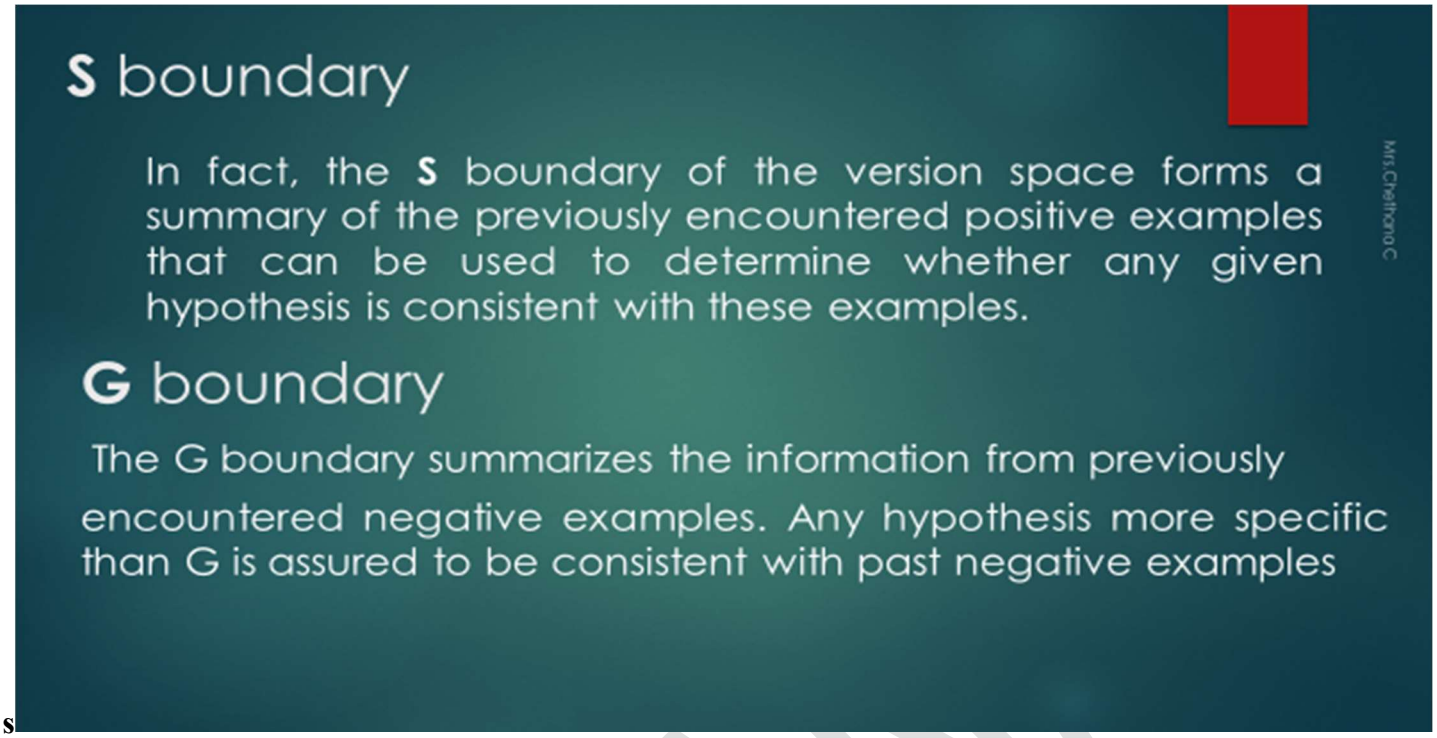
Find S: Finding a Maximally Specific Hypothesis

For Training instance No: 3 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Program 2: 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.





s

Summary

For all Positive Instances

1. **Delete all members of G that do not match +ve.(Inconsistent)**
2. **For Each Member s, if S does not match +ve replace s with its minimal generalization that match +ve (Let it be hypothesis h)**
3. **Delete all inconsistent hypothesis from S. (check h is consistent with d)**
4. **Delete any member of s , if more general than another member in s**
5. **Delete any member of s , if more general than another member in g**

For all Negative Instances

1. **Delete all members of S that match -ve. (Inconsistent)**
2. **For Each Member g, that match negative, replace g with its most General Specialization h ,that do not match Negative(Let it be hypothesis h)**
3. **Delete all inconsistent hypothesis from G. (check h is consistent with s)**
4. **Delete any member of g, if more specific than another member in g**
5. **Delete any member of g, if more specific than another member in s**

Program

```
import csv
a = []
print("\n The Given Training Data Set \n")

with open('enjoysport.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
```

```
        print(row)
num_attributes = len(a[0])-1

print("\n The initial value of hypothesis: ")


S = ['0'] * num_attributes
G = ['?'] * num_attributes
print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]\n")
print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]\n")

# Comparing with First Training Example
for j in range(0,num_attributes):
       S[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

print("\n Candidate Elimination algorithm  Hypotheses Version Space Computation\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 Training Example No :{0} the hypothesis is S{0}  ".format(i+1),S)
        if (len(temp)==0):
            print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),G)
        else:
            print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)

    if a[i][num_attributes]=='No':
        for j in range(0,num_attributes):
            # print("S[j]  ",S[j])
            # print("a[i][j]  ",a[i][j])
             if S[j] != a[i][j] and S[j]!= '?':
                G[j]=S[j]
                temp.append(G)
                print("Temp  ",temp)
                G = ['?'] * num_attributes

        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)
```

```
print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

INPUT: enjoysport.csv

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

OUTPUT

The Given Training Data Set

['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']

The initial value of hypothesis:
The most specific hypothesis S0 : [0,0,0,0,0,0]
The most general hypothesis G0 : [?,?,?,?,?,?]

Candidate Elimination algorithm Hypotheses Version Space Computation

For Training Example No :1 the hypothesis is S1 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
For Training Example No :1 the hypothesis is G1 ['?', '?', '?', '?', '?', '?']

For Training Example No :2 the hypothesis is S2 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For Training Example No :2 the hypothesis is G2 ['?', '?', '?', '?', '?', '?']

For Training Example No :3 the hypothesis is S3 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For Training Example No :3 the hypothesis is G3 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

For Training Example No :4 the hypothesis is S4 ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For Training Example No :4 the hypothesis is G4 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Program 3: 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

Steps in ID3 algorithm

1. Calculate entropy for dataset.
2. For each attribute/feature
   - Calculate entropy for all its categorical values.
   - Calculate information gain for the feature.
3. Find the feature with maximum information gain.
4. Repeat it until we get the desired tree.

THE BASIC DECISION TREE LEARNING ALGORITHM

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

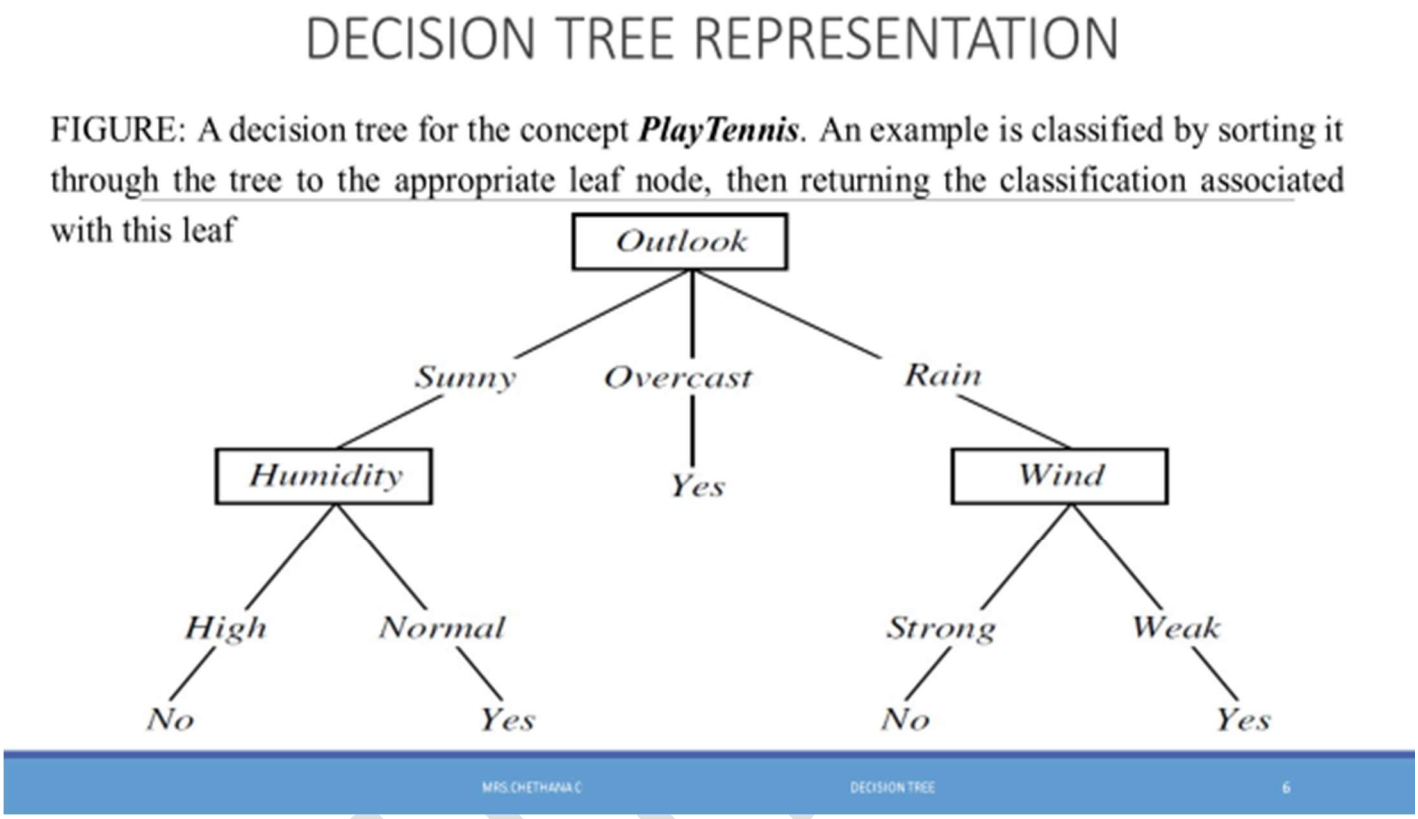
- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = –
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
  - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
  - The decision attribute for *Root* $\leftarrow A$
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
    - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
    - If $Examples_{v_i}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* – {*A*}))
- End
- Return *Root*

* The best attribute is the one with highest *information gain*, as defined in Equation (3.4).

Program

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

    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)
```



INPUT: 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

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'}}}}

Program 5: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

A DIFFERENTIABLE THRESHOLD UNIT (SIGMOID UNIT)

Sigmoid unit-a unit very much like a perceptron, but based on a smoothed, differentiable threshold function.

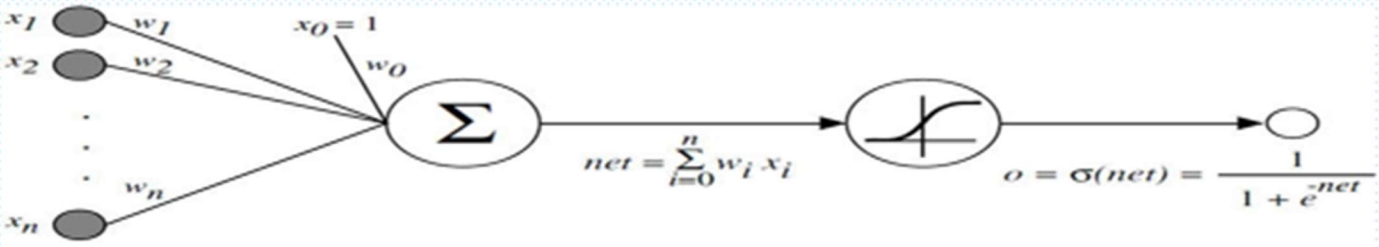


Figure: A Sigmoid Threshold Unit

$$o = \sigma(\vec{w} \cdot \vec{x})$$

Where,

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

- The sigmoid unit first computes a linear combination of its inputs, then applies a threshold to the result and the threshold output is a continuous function of its input.
- More precisely, the sigmoid unit computes its output O as

σ is the sigmoid function

THE BACKPROPAGATION ALGORITHM

- The BACKPROPAGATION Algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.
- In BACKPROPAGATION algorithm, we consider networks with multiple output units rather than single units as before, so we redefine E to sum the errors over all of the network output units.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \quad \text{.....equ. (1)}$$

where,

- ***outputs*** - is the set of output units in the network
- t_{kd} and O_{kd} - the target and output values associated with the k_{th} output unit
- ***d*** - training example

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $(\vec{x}, \vec{t})$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i *into unit* j *is denoted* x_{ji}, *and the weight from unit* i *to unit* j *is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between −.05 and .05).
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \delta_j x_{ji} \tag{T4.5}$$

TABLE 4.2
The stochastic gradient descent version of the BACKPROPAGATION algorithm for feedforward networks containing two layers of sigmoid units.

Program

```
import math
def sigmoid(x):
        y= 1/(1+math.exp(-x))
        return y
##define inputs and target for xor gate
x1=[0,0,1,1]        #input1
x2=[0,1,0,1]        #input2
t=[0,1,1,0]         #target

## Initialize random weights and biases

# Hidden layer first Perceptron
b1=-0.3
w11=0.21
w21= 0.15
# Hidden Layer Second Perceptron
b2=0.25
w12=-0.4
w22=0.1
# Output layer Perceptron
b3=-0.4
w13=-0.2
w23=0.3
error=0
iteration=0
train=True
print("weight 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))
```

```
## Training Starts
while(train):
    for i in range(len(x1)):
        ##input for each perceptron of hidden layer
        z_in1=b1+x1[i]*w11+x2[i]*w21
        z_in2=b2+x1[i]*w12+x2[i]*w22
        ##computing activation function output
        z1=round(sigmoid(z_in1),4)
        z2=round(sigmoid(z_in2),4)

    # Output layer forward pass
        y_in=b3+z1*w13+z2*w23
        y=round(sigmoid(y_in),4)
    #error computation
        del_k=round((t[i]-y)*y*(1-y),4)
        error=del_k
                                        ##Back pass
    # weight update for output layer
        w13=round(w13+del_k*z1,4)
        w23=round(w23+del_k*z2,4)
        b3=round(b3+del_k,4)

        ##error computation for hidden layer
        del_1=del_k*w13*z1*(1-z1)
        del_2=del_k*w23*z2*(1-z2)

        ## update weight and biases
        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: %5.4f w22: %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

Sample

```
weight are:
w11 : 0.21 w12: -0.40 w21: 0.15 w22: 0.10 w13: -0.20  w23: 0.30

Iteration:  0
w11 : 0.2100 w12: -0.4000 w21: 0.1500 w22: 0.1000 w13: -0.2438  w23: 0.2422
Error: -0.103
Iteration:  1
w11 : 0.2100 w12: -0.4000 w21: 0.1616 w22: 0.1116 w13: -0.1762  w23: 0.3274
Error: 0.146
Iteration:  2
w11 : 0.2062 w12: -0.4038 w21: 0.1616 w22: 0.1116 w13: -0.1094  w23: 0.3923
Error: 0.140
Iteration:  3
w11 : 0.2113 w12: -0.3987 w21: 0.1516 w22: 0.1016 w13: -0.1712  w23: 0.3331
Error: -0.120
Iteration:  4
w11 : 0.2113 w12: -0.3987 w21: 0.1516 w22: 0.1016 w13: -0.2180  w23: 0.2711
Error: -0.110
Iteration:  5
w11 : 0.2113 w12: -0.3987 w21: 0.1640 w22: 0.1140 w13: -0.1513  w23: 0.3554
```

```
w11 : 0.8512 w12: 0.2412 w21: 0.2361 w22: 0.1861 w13: -0.2090  w23: 0.3361
Error: 0.140
Iteration:  994
w11 : 0.8495 w12: 0.2395 w21: 0.2361 w22: 0.1861 w13: -0.1035  w23: 0.4421
Error: 0.125
Iteration:  995
w11 : 0.8530 w12: 0.2430 w21: 0.2311 w22: 0.1811 w13: -0.2269  w23: 0.3189
Error: -0.141
Iteration:  996
w11 : 0.8530 w12: 0.2430 w21: 0.2311 w22: 0.1811 w13: -0.3139  w23: 0.2177
Error: -0.124
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
```

Program 5: 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.

Naïve Bayes Classifier

Along with decision trees, neural networks, nearest neighor, one of the most practical learning methods.

When to use

- Moderate or large training set available
- Attributes that describe instances are conditionally independent given classification

Successful applications:

- Diagnosis
- Classifying text documents

10

Bayesian Methods

Mrs. Chethana C

The Bayesian approach to classifying the new instance is to assign the most probable target value, V_{MAP}, given the attribute values (a_1, a_2.. .a_m) that describe the instance

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

Use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \qquad \textbf{equ (1)}$$

- The naive Bayes classifier is based on the assumption that the attribute values are conditionally independent given the target value. Means, the assumption is that given the target value of the instance, the probability of observing the conjunction (a_1, a_2.. .a_m), is just the product of the probabilities for the individual attributes:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

Substituting this into Equation (1),

Bayesian Methods

Mrs. Chethana C

s

- Naive Bayes classifier:

$$V_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j) \qquad \text{equ (2)}$$

- Where, V_{NB} denotes the target value output by the naive Bayes classifier

Program

```
import csv
import math
import random
import statistics

#Gaussian formula
def calculate_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)  # 70 % as test data

print(train_size)

X_train = []
X_test = dataset.copy()
training_indexes = random.sample(range(dataset_size), train_size)

# Split Data
for i in training_indexes:
    X_train.append(dataset[i]) # 70% of data   traning phase
    X_test.remove(dataset[i])   # 30% of data   testing phase

# Separate Data based on class value
```

```
classes = {}
for samples in X_train:
    last = int(samples[-1])
    if last not in classes:
        #print("Concept value", last)
        classes[last] = []
    classes[last].append(samples)
    #print(classes)


# Find mean and variance of each attribute by adding all attributes
summaries = {}
for classValue, training_data in classes.items():
    summary = [(statistics.mean(attribute), statistics.stdev(attribute)) for attribute in zip(*training_data)]
    #print("summary   classValue,",summary,classValue )
    del summary[-1]
    summaries[classValue] = summary
    #print("summaries  next",summaries)

X_prediction = []

# Predict the output of test data
for i in X_test:
    probabilities = {}
    for classValue, classSummary in summaries.items():
      #print("classValue",classValue,"classSummary",classSummary)
       probabilities[classValue] = 1
       #print("probabilities1",probabilities)
       for index, attr in enumerate(classSummary):
          #print("index",index,"attr",attr)

          # call  calculate_probability method to find value for each class value
          probabilities[classValue] *= calculate_probability(i[index], attr[0], attr[1])
          #print("probabilities2",probabilities)

    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)       #best Concept
    #print("X_prediction",X_prediction)

# Find Accuracy
correct = 0
for index, key in enumerate(X_test):

    #print("X_test[index][-1]",X_test[index][-1] )
    #print( "X_prediction[index]",X_prediction[index])
```

```
    if X_test[index][-1] == X_prediction[index]:
        correct += 1     #if actual concept equals predicted concept values
print("Accuracy : ", correct / (float(len(X_test))) * 100)
```

INPUT : lab5.csv (Given Below Sample Data)

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	0.248	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	0.232	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
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

OUTPUT

```
Size of dataset is :  768
train_size 537
Accuracy :  68.83116883116884
```

Program 6: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

LEARN_NAIVE_BAYES_TEXT(*Examples*, *V*)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. collect all words, punctuation, and other tokens that occur in Examples

- *Vocabulary* ← the set of all distinct words and other tokens occurring in any text document from *Examples*

2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms

- For each target value v_j in V do
  - $docs_j$ ← the subset of documents from *Examples* for which the target value is v_j
  - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
  - $Text_j$ ← a single document created by concatenating all members of $docs_j$
  - n ← total number of distinct word positions in $Text_j$
  - for each word w_k in *Vocabulary*
    - n_k ← number of times word w_k occurs in $Text_j$
    - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT(*Doc*)

Return the estimated target value for the document Doc. a_i denotes the word found in the ith position within Doc.

- *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

TABLE 6.2
Naive Bayes algorithms for learning and classifying text. In addition to the usual naive Bayes assumptions, these algorithms assume the probability of a word occurring is independent of its position within the text.

Program

```
import pandas as pd
msg=pd.read_csv('lab6.txt',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)


from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print("Uniqu Labels")
print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df)


from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
print("ytest")
print(ytest)
print("predicted")
print(predicted)
from sklearn import metrics

print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))

print('Confusion matrix')

import numpy as np
labels = np.unique(ytest) #Assign concept labels
a =  metrics.confusion_matrix(ytest, predicted, labels=labels)
cm = pd.DataFrame(a, index=labels, columns=labels)
```

```
print(cm)


print('Recall and Precison ')
print('Recall',   metrics.recall_score(ytest,predicted))
print('Precison', metrics.precision_score(ytest,predicted))
```

INPUT:lab6.txt

```
I love this sandwich,pos
This is an amazing place,pos
I feel very good about these beers,pos
This is my best work,pos
What an awesome view,pos
I do not like this restaurant,neg
I am tired of this stuff,neg
I can't deal with this,neg
He is my sworn enemy,neg
My boss is horrible,neg
This is an awesome place,pos
I do not like the taste of this juice,neg
I love to dance,pos
I am sick and tired of this place,neg
What a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,pos
I went to my enemy's house today,neg
```

OUTPUT

```
The dimensions of the dataset (18, 2)
0                    I love this sandwich
1                 This is an amazing place
2       I feel very good about these beers
3                    This is my best work
4                    What an awesome view
5             I do not like this restaurant
6                  I am tired of this stuff
7                  I can't deal with this
8                    He is my sworn enemy
9                    My boss is horrible
10                This is an awesome place
11   I do not like the taste of this juice
12                      I love to dance
13        I am sick and tired of this place
14                   What a great holiday
15           That is a bad locality to stay
```

```
16          We will have good fun tomorrow
17        I went to my enemy's house today
Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64
(5,)
(13,)
(5,)
(13,)
```

Uniqu Labels

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'dance', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'juice', 'like', 'love', 'my', 'not', 'of', 'place', 'sick', 'sworn', 'taste', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']

	about	am	amazing	an	and	awesome	...	we	went	what	will	with	work
0	0	0	0	1	0	1	...	0	0	0	0	0	0
1	0	1	0	0	1	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	...	1	0	0	1	0	0
3	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	...	0	0	0	0	0	0
5	1	0	0	0	0	0	...	0	0	0	0	0	0
6	0	0	0	0	0	0	...	0	0	0	0	1	0
7	0	0	0	0	0	0	...	0	0	1	0	0	0
8	0	0	0	0	0	0	...	0	0	0	0	0	1
9	0	0	0	0	0	0	...	0	0	0	0	0	0
10	0	0	1	1	0	0	...	0	0	0	0	0	0
11	0	0	0	1	0	1	...	0	0	1	0	0	0
12	0	0	0	0	0	0	...	0	1	0	0	0	0

[13 rows x 46 columns]

```
ytest
13    0
4     1
5     0
17    0
7     0
Name: labelnum, dtype: int64
predicted
[0 1 0 0 1]
Accuracy metrics
Accuracy of the classifer is 0.8
Confusion matrix
   0  1
0  3  1
1  0  1
Recall and Precison
Recall 1.0
Precison 0.5
```

Program 7: Write a program to construct aBayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

Math Behind Bayesian Networks

As mentioned earlier, Bayesian models are based on the simple concept of probability. So let's understand what conditional probability and Joint probability distribution mean.

What Is Joint Probability?

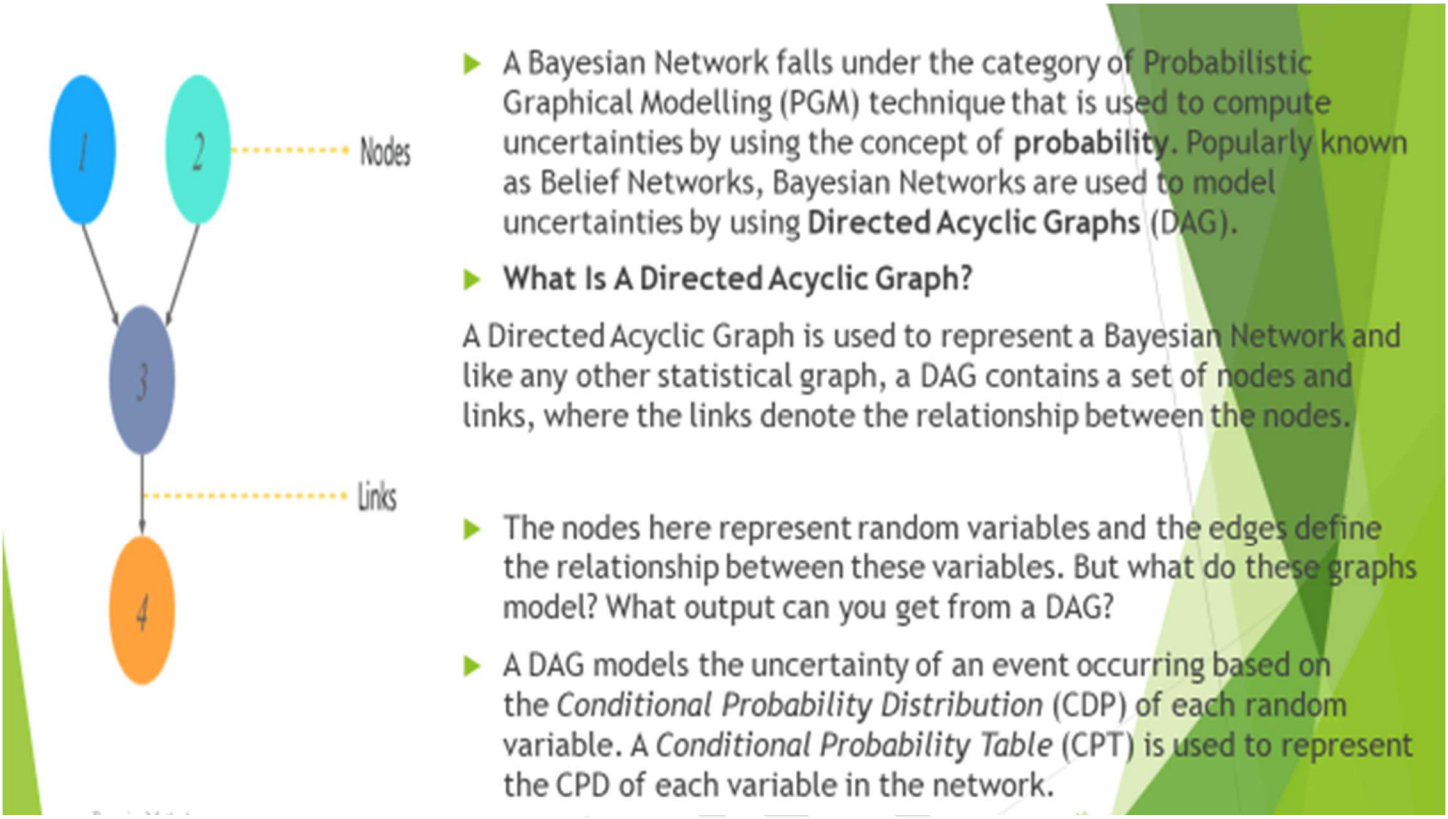
Joint Probability is a statistical measure of two or more events happening at the same time, i.e., P(A, B, C), The probability of event A, B and C occurring. It can be represented as the probability of the intersection two or more events occurring.

What Is Conditional Probability?

Conditional Probability of an event X is the probability that the event will occur given that an event Y has already occurred.

p(X| Y) is the probability of event X occurring, given that event, Y occurs.

- If X and Y are dependent events then the expression for conditional probability is given by:
  P (X| Y) = P (X and Y) / P (Y)
- If A and B are independent events then the expression for conditional probability is given by:
  P(X| Y) = P (X)



The DAG clearly shows how each variable (node) depends on its parent node, i.e., the marks of the student depends on the exam level (parent node) and IQ level (parent node). Similarly, the aptitude score depends on the IQ level (parent node) and finally, his admission into a university depends on his marks (parent node). This relationship is represented by the edges of the DAG.

If you notice carefully, we can see a pattern here. The probability of a random variable depends on his parents. Therefore, we can formulate Bayesian Networks as:

$$P(X_1, \ldots, X_n) = \Pi_{i=1}^{n}\, p(X_i \mid Parents(X_i))$$

Where, X_i denotes a random variable, whose probability depends on the probability of the parent nodes, *Parents(X_i)*.

Progrm

```
imort pandas as pd
col =['Ag','Gender','FamilyHist','Diet','LifeStyle','Cholesterol','HeartDisease']
data = pd.read_csv('lab7.csv',names =col )
print(dat)

#encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
for i in range(len(col)):
    data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])

#spliting data
X = data.iloc[:,0:6]
y = data.iloc[:,-1]
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print("X_test")
print(X_test)
print("y_test")
print(y_test)
#prediction
from sklearn.naive_bayes import GaussianNB
#Create naive bayes classifier
clf = GaussianNB()
#Fit the dataset on classifier
clf.fit(X_train,y_train)

#Perform prediction
y_pred = clf.predict(X_test)
print("y_pred")
print(y_pred)

#confusion mtx output
import numpy as np
from sklearn.metrics import confusion_matrix

labels = np.unique(y_test)
a = confusion_matrix(y_test, y_pred, labels=labels)

print('Confusion matrix')
cm =pd.DataFrame(a, index=labels, columns=labels)

print(cm)
```

INPUT: lab7.csv

SuperSeniorCitizen	Male	Yes	Medium	Sedetary	High	Yes
SuperSeniorCitizen	Female	Yes	Medium	Sedetary	High	Yes
SeniorCitizen	Male	No	High	Moderate	BorderLine	Yes
Teen	Male	Yes	Medium	Sedetary	Normal	No
Youth	Female	Yes	High	Athlete	Normal	No
MiddleAged	Male	Yes	Medium	Active	High	Yes
Teen	Male	Yes	High	Moderate	High	Yes
SuperSeniorCitizen	Male	Yes	Medium	Sedetary	High	Yes
Youth	Female	Yes	High	Athlete	Normal	No
SeniorCitizen	Female	No	High	Athlete	Normal	Yes
Teen	Female	No	Medium	Moderate	High	Yes
Teen	Male	Yes	Medium	Sedetary	Normal	No
MiddleAged	Female	No	High	Athlete	High	No
MiddleAged	Male	Yes	Medium	Active	High	Yes
Youth	Female	Yes	High	Athlete	BorderLine	No
SuperSeniorCitizen	Male	Yes	High	Athlete	Normal	Yes
SeniorCitizen	Female	No	Medium	Moderate	BorderLine	Yes
Youth	Female	Yes	Medium	Athlete	BorderLine	No
Teen	Male	Yes	Medium	Sedetary	Normal	No

OUTPUT

	Age	Gender	FamilyHist	...	LifeStyle	Cholesterol	HeartDisease
0	SuperSeniorCitizen	Male	Yes	...	Sedetary	High	Yes
1	SuperSeniorCitizen	Female	Yes	...	Sedetary	High	Yes
2	SeniorCitizen	Male	No	...	Moderate	BorderLine	Yes
3	Teen	Male	Yes	...	Sedetary	Normal	No
4	Youth	Female	Yes	...	Athlete	Normal	No
5	MiddleAged	Male	Yes	...	Active	High	Yes
6	Teen	Male	Yes	...	Moderate	High	Yes
7	SuperSeniorCitizen	Male	Yes	...	Sedetary	High	Yes
8	Youth	Female	Yes	...	Athlete	Normal	No
9	SeniorCitizen	Female	No	...	Athlete	Normal	Yes
10	Teen	Female	No	...	Moderate	High	Yes
11	Teen	Male	Yes	...	Sedetary	Normal	No
12	MiddleAged	Female	No	...	Athlete	High	No
13	MiddleAged	Male	Yes	...	Active	High	Yes
14	Youth	Female	Yes	...	Athlete	BorderLine	No
15	SuperSeniorCitizen	Male	Yes	...	Athlete	Normal	Yes
16	SeniorCitizen	Female	No	...	Moderate	BorderLine	Yes
17	Youth	Female	Yes	...	Athlete	BorderLine	No
18	Teen	Male	Yes	...	Sedetary	Normal	No

[19 rows x 7 columns]

	Age	Gender	FamilyHist	Diet	LifeStyle	Cholesterol
0	2	1	1	1	3	1
1	2	0	1	1	3	1
2	1	1	0	0	2	0
3	3	1	1	1	3	2
4	4	0	1	0	1	2
5	0	1	1	1	0	1
6	3	1	1	0	2	1
7	2	1	1	1	3	1
8	4	0	1	0	1	2
9	1	0	0	0	1	2
10	3	0	0	1	2	1
11	3	1	1	1	3	2
12	0	0	0	0	1	1
13	0	1	1	1	0	1
14	4	0	1	0	1	0
15	2	1	1	0	1	2
16	1	0	0	1	2	0
17	4	0	1	1	1	0
18	3	1	1	1	3	2

X_test

	Age	Gender	FamilyHist	Diet	LifeStyle	Cholesterol
0	2	1	1	1	3	1
3	3	1	1	1	3	2
5	0	1	1	1	0	1
10	3	0	0	1	2	1

y_test

0	1
3	0
5	1
10	1

Name: HeartDisease, dtype: int32

y_pred

[1 0 1 1]

Confusion matrix

	0	1
0	1	0
1	0	3

Program 8: Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
import 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()
```

INPUT: lab8.csv

Sample data set

	V1	V2
1	2.072345	-3.24169
2	17.93671	15.78481
3	1.083576	7.319176
4	11.12067	14.40678
5	23.71155	2.557729
6	24.16993	32.02478
7	21.66578	4.892855
8	4.693684	12.34217
9	19.21191	-1.12137
10	4.230391	-4.44154
11	9.12713	23.60572
12	0.407503	15.29705
13	7.314846	3.309312
14	-3.4384	-12.0253

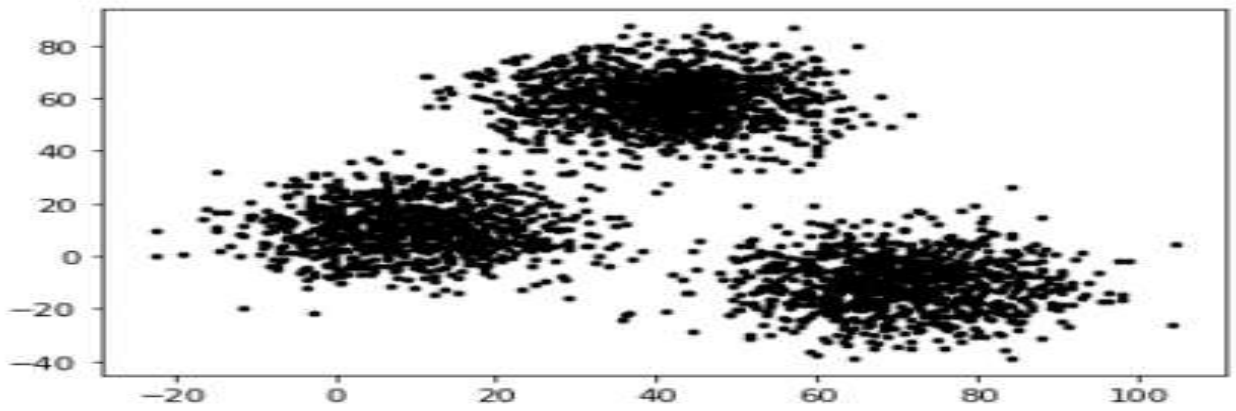
15	17.63935	-3.21235
16	4.415292	22.81555
17	11.94122	8.122487
18	0.725853	1.806819
19	8.185273	28.1326
20	-5.77359	1.0248

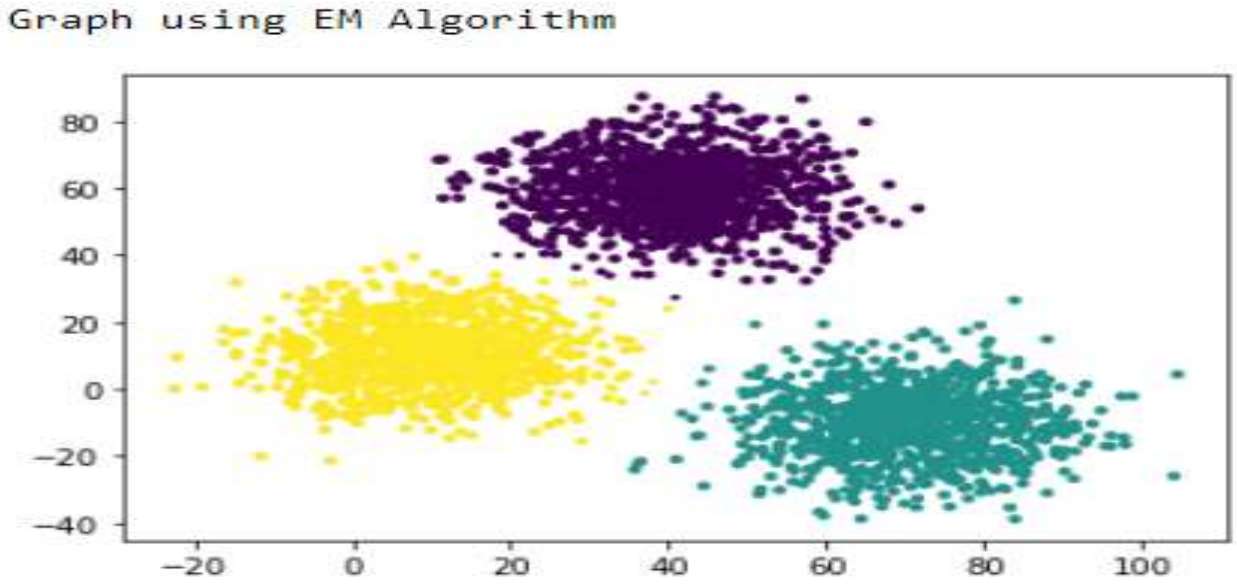
OUTPUT

```
Input Data and Shape
(3000, 3)
```

	Unnamed: 0	V1	V2
0	1	2.072345	-3.241693
1	2	17.936710	15.784810
2	3	1.083576	7.319176
3	4	11.120670	14.406780
4	5	23.711550	2.557729

```
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
```





<u>Program 9:</u> 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.

```
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

```
x sample [[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]]

y sample [0 0 0 0 0]

iris size (150, 4)
xtrain size 90
ytest size 60
iris size (150, 4)
xtrain size 90
ytest size 60

Prediced [0 1 1 0 2 2 2 0 0 2 1 0 2 1 1 0 1 1 0 0 1 1 1 0 2 1 0 0 1 2 1 2 1 2 2 0 1
 0 1 2 2 0 1 2 1 2 0 0 0 1 0 0 2 2 2 2 2 1 2 1]

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
11     setosa     setosa
12     virginica     virginica
13     versicolor     versicolor
14     versicolor     versicolor
15     setosa     setosa
16     versicolor     versicolor
17     versicolor     versicolor
18     setosa     setosa
19     setosa     setosa
20     versicolor     versicolor
```

```
21     versicolor     versicolor
22     versicolor     versicolor
23     setosa     setosa
24     virginica     virginica
25     versicolor     versicolor
26     setosa     setosa
27     setosa     setosa
28     versicolor     versicolor
29     virginica     virginica
30     versicolor     versicolor
31     virginica     virginica
32     versicolor     versicolor
33     virginica     virginica
34     virginica     virginica
35     setosa     setosa
36     versicolor     versicolor
37     setosa     setosa
38     versicolor     versicolor
39     virginica     virginica
40     virginica     virginica
41     setosa     setosa
42     versicolor     virginica
43     virginica     virginica
44     versicolor     versicolor
45     virginica     virginica
46     setosa     setosa
47     setosa     setosa
48     setosa     setosa
49     versicolor     versicolor
50     setosa     setosa
51     setosa     setosa
52     virginica     virginica
53     virginica     virginica
54     virginica     virginica
55     virginica     virginica
56     virginica     virginica
57     versicolor     versicolor
58     virginica     virginica
59     versicolor     versicolor
```

Program 10: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import 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()
```

INPUT

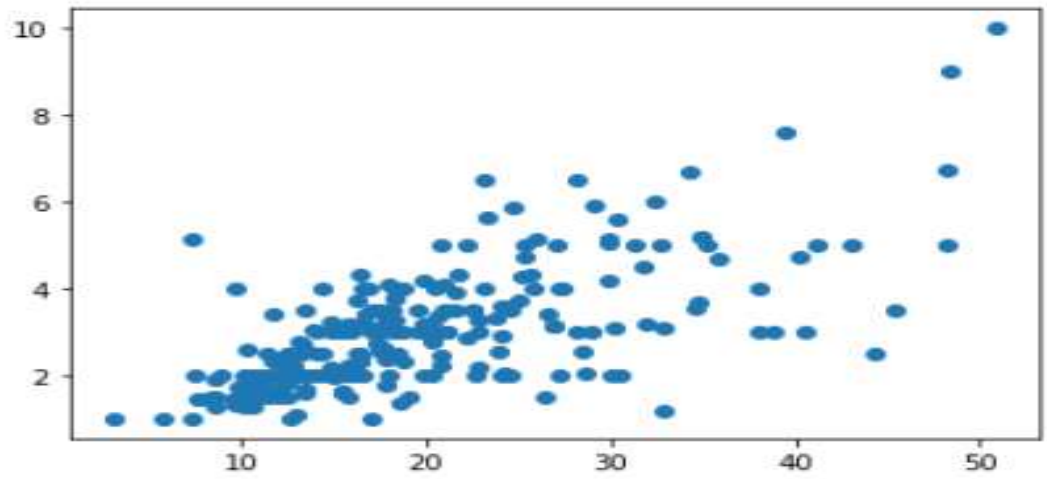
1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2
11	14.78	3.23	Male	No	Sun	Dinner	2
12	10.27	1.71	Male	No	Sun	Dinner	2
13	35.26	5	Female	No	Sun	Dinner	4
14	15.42	1.57	Male	No	Sun	Dinner	2
15	18.43	3	Male	No	Sun	Dinner	4
16	14.83	3.02	Female	No	Sun	Dinner	2
17	21.58	3.92	Male	No	Sun	Dinner	2
18	10.33	1.67	Female	No	Sun	Dinner	3
19	16.29	3.71	Male	No	Sun	Dinner	3
20	16.97	3.5	Female	No	Sun	Dinner	3
21	20.65	3.35	Male	No	Sat	Dinner	3
22	17.92	4.08	Male	No	Sat	Dinner	2
23	20.29	2.75	Female	No	Sat	Dinner	2

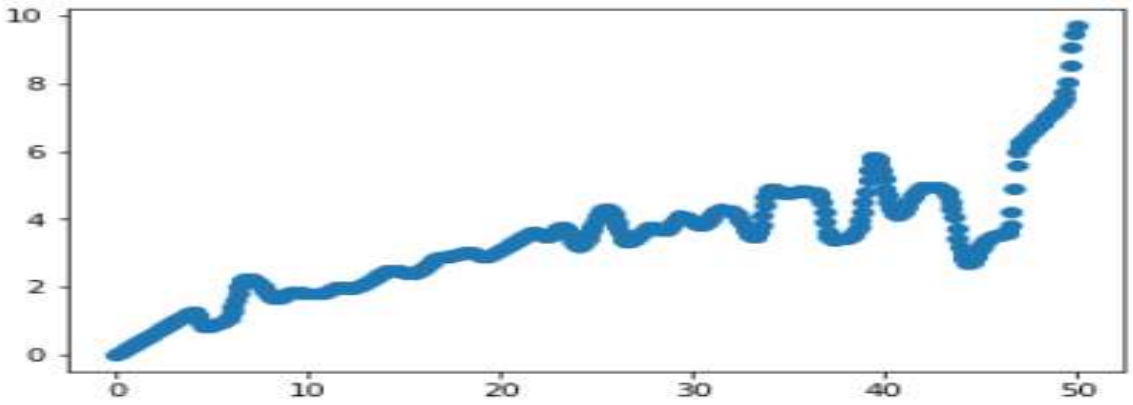
OUTPUT

```
X_train [16.99 10.34 21.01 23.68 24.59 25.29  8.77 26.88 15.04 14.78 10.27 35.26
 15.42 18.43 14.83 21.58 10.33 16.29 16.97 20.65 17.92 20.29 15.77 39.42
 19.82 17.81 13.37 12.69 21.7  19.65  9.55 18.35 15.06 20.69 17.78 24.06
 16.31 16.93 18.69 31.27 16.04 17.46 13.94  9.68 30.4  18.29 22.23 32.4
 28.55 18.04 12.54 10.29 34.81  9.94 25.56 19.49 38.01 26.41 11.24 48.27
 20.29 13.81 11.02 18.29 17.59 20.08 16.45  3.07 20.23 15.01 12.02 17.07
 26.86 25.28 14.73 10.51 17.92 27.2  22.76 17.29 19.44 16.66 10.07 32.68
 15.98 34.83 13.03 18.28 24.71 21.16 28.97 22.49  5.75 16.32 22.75 40.17
 27.28 12.03 21.01 12.46 11.35 15.38 44.3  22.42 20.92 15.36 20.49 25.21
 18.24 14.31 14.    7.25 38.07 23.95 25.71 17.31 29.93 10.65 12.43 24.08
 11.69 13.42 14.26 15.95 12.48 29.8   8.52 14.52 11.38 22.82 19.08 20.27
 11.17 12.26 18.26  8.51 10.33 14.15 16.   13.16 17.47 34.3  41.19 27.05
 16.43  8.35 18.64 11.87  9.78  7.51 14.07 13.13 17.26 24.55 19.77 29.85
 48.17 25.   13.39 16.49 21.5  12.66 16.21 13.81 17.51 24.52 20.76 31.71
 10.59 10.63 50.81 15.81  7.25 31.85 16.82 32.9  17.89 14.48  9.6  34.63
 34.65 23.33 45.35 23.17 40.55 20.69 20.9  30.46 18.15 23.1  15.69 19.81
 28.44 15.48 16.58  7.56 10.34 43.11 13.   13.51 18.71 12.74 13.   16.4
 20.53 16.47 26.59 38.73 24.27 12.76 30.06 25.89 48.33 13.27 28.17 12.9
 28.15 11.59  7.74 30.14 12.16 13.42  8.58 15.98 13.42 16.27 10.09 20.45
 13.28 22.12 24.01 15.69 11.61 10.77 15.53 10.07 12.6  32.83 35.83 29.03
 27.18 22.67 17.82 18.78]
X_train size 244
```

```
y_test size 500
```

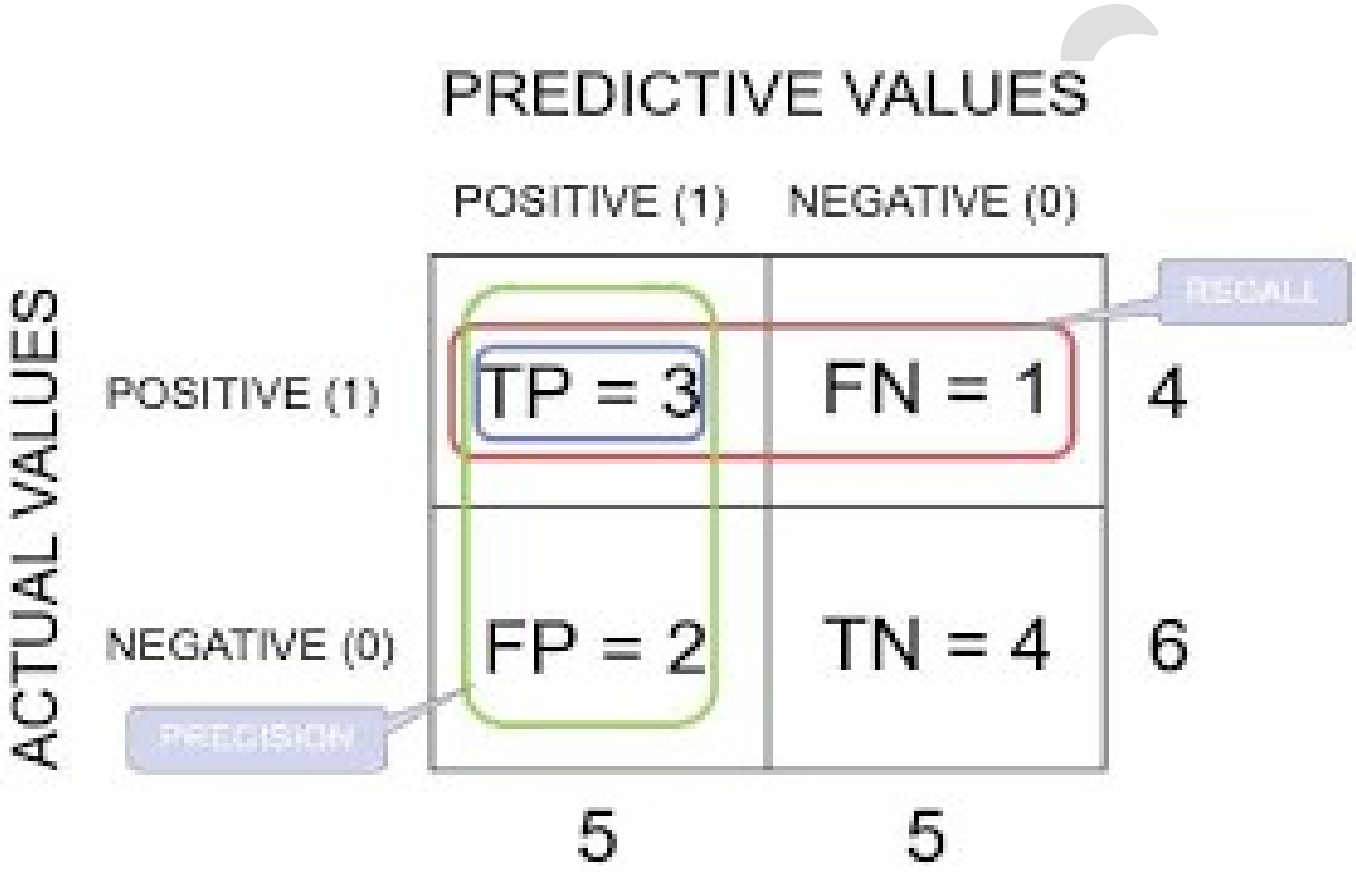
Out[21]: [<matplotlib.lines.Line2D at 0x1fabf67eb20>]





What is a Confusion Matrix?

- A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.
- For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:



Let's decipher the matrix:

- The target variable has two values: **Positive** or **Negative**
- The **columns** represent the **actual values** of the target variable
- The **rows** represent the **predicted values** of the target variable

True Positive (TP)

- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value

- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted
- The actual value was Negative but the model predicted a positive value
- Also known as the **Type 1 error**

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted
- The actual value was positive but the model predicted a Negative value
- Also known as the **Type 2 error**
- A confusion matrix is a table which is used for summarizing the performance of a classification algorithm. It is also known as the **error matrix**.

Actual	Predicted	outcome
0	0	TN
0	1	FP
1	0	FN
1	1	TP

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision is a useful metric in cases where False Positive is a higher concern than False Negatives.

$$Precision = \frac{TP}{TP + FP}$$

Recall is a useful metric in cases where False Negative trumps False Positive.

$$Recall = \frac{TP}{TP + FN}$$

Assuming you have a multi-class confusion matrix of the form,

$$C = \begin{array}{c} \\ \text{Actual} \end{array} \begin{array}{c} \text{Classifed} \\ \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \\ c_{n1} & & c_{nn} \end{bmatrix} \end{array}$$

The confusion elements for each class are given by:

$tp_i = c_{ii}$

$fp_i = \sum_{l=1}^{n} c_{li} - tp_i$

$fn_i = \sum_{l=1}^{n} c_{il} - tp_i$

$tn_i = \sum_{l=1}^{n} \sum_{k=1}^{n} c_{lk} - tp_i - fp_i - fn_i$

APPENDIX A

Program 3

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Where,

p_+ is the proportion of positive examples in S

p_- is the proportion of negative examples in S

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Program 6

Tokenizing text with scikit-learn

- Text preprocessing, tokenizing and filtering of stopwords are all included in **CountVectorizer**, which builds a dictionary of features and transforms documents to feature vectors:
- The **CountVectorizer** provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

- Is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation.

fit(raw_documents[, y])	Learn a vocabulary dictionary of all tokens in the raw documents.
fit_transform(raw_documents[, y])	Learn the vocabulary dictionary and return document-term matrix.

This is equivalent to fit followed by transform, but more efficiently implemented.

get_feature_names()[source]	Array mapping from feature integer indices to feature name.

- **Multinomial Naïve Bayes** uses term frequency i.e. the number of times a given term appears in a document. ... After normalization, term frequency can be used to compute maximum likelihood estimates based on the training data to estimate the conditional probability.

Naive Bayes classifier for multinomial models

- The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification).
- The multinomial distribution normally requires integer feature counts.

Program 7

- The sklearn.preprocessing.LabelEncoder().fit_transform(y) fits the label encoder, or assigns labels, for a single column and transforms the values in that column to the correct labels.

- **Label Encoding** refers to converting the labels into numeric form so as to convert it into the machine-readable form.
  - Machine learning algorithms can then decide in a better way on how those labels must be operated.
  - It is an important pre-processing step for the structured dataset in supervised learning.
  - It assigns a unique number(starting from 0) to each class of data.

Height
Tall
Medium
Short

Height
0
1
2

where 0 is the label for tall, 1 is the label for medium and 2 is label for short height.

`fit`(y)	Fit label encoder
`fit_transform`(y)	Fit label encoder and return encoded labels

Program 8

What Is Clustering?

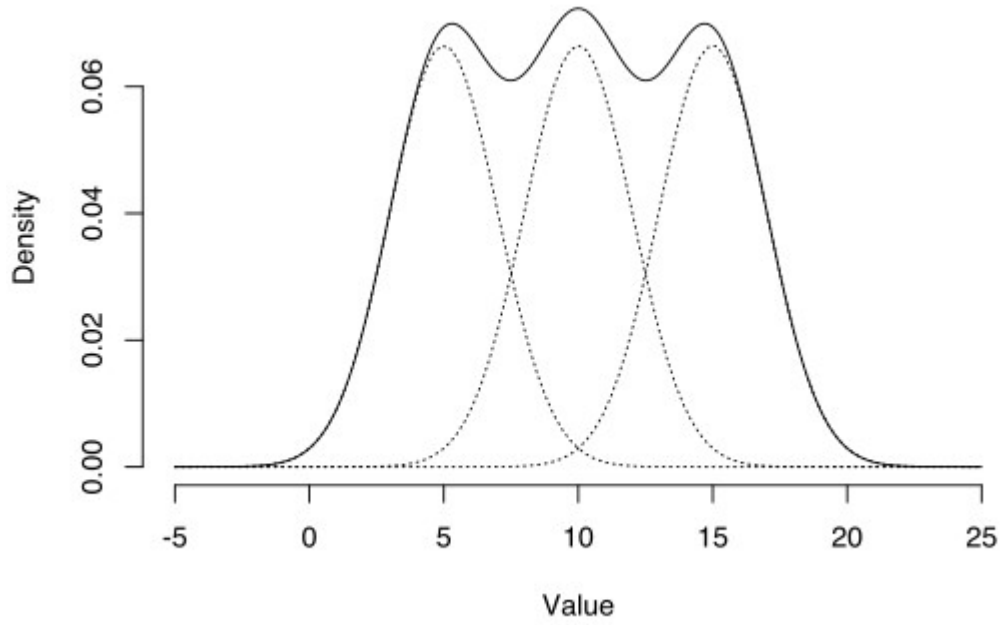
- Clustering is a set of techniques used to partition data into groups, or clusters.
- **Clusters** are loosely defined as groups of data objects that are more similar to other objects in their cluster than they are to data objects in other clusters.

K-Means Algorithm

- Conventional *k*-means requires only a few steps. The first step is to randomly select *k* centroids, where *k* is equal to the number of clusters we choose.
- **Centroids** are data points representing the center of a cluster.
- The main element of the algorithm works by a two-step process called **expectation-maximization**.
- The **expectation** step assigns each data point to its nearest centroid.
- Then, the **maximization** step computes the mean of all the points for each cluster and sets the new centroid.

What is the EM Algorithm?

- The **Expectation-Maximization (EM) algorithm** is a way to find maximum-likelihood estimates for model parameters when your data is
  - incomplete,
  - has missing data points, or
  - has unobserved (hidden) latent variables.
- It is an iterative way to approximate the maximum likelihood function.
- While maximum likelihood estimation can find the "best fit" model for a set of data, it doesn't work particularly well for incomplete data sets.



Applications

The EM algorithm has many applications, including:

- Dis-entangling superimposed signals,
- Estimating Gaussian mixture models (GMMs),
- Estimating hidden Markov models (HMMs),
- Estimating parameters for compound Dirichlet distributions,
- Finding optimal mixtures of fixed models.

Limitations

- The EM algorithm can be **very slow**, even on the fastest computer.
- It works best when you only have a small percentage of missing data and the dimensionality of the data isn't too big.
- The higher the dimensionality, the slower the E-step; for data with larger dimensionality, we may find the E-step runs extremely slow as the procedure approaches a local maximum.

Program 9

KNeighborsClassifier

- K nearest neighbors is a simple **algorithm** that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

- The K in the name of this classifier represents the k nearest neighbors, where k is an integer value specified by the user.
- Hence as the name suggests, this classifier implements learning based on the k nearest neighbors. The choice of the value of k is dependent on data.

Program 10

- We use **numpy.linalg.inv()** function to calculate the inverse of a matrix.
- The inverse of a matrix is such that if it is multiplied by the original matrix, it results in identity matrix.

BASICS

Strings

```
1. str1 = 'Hello Python'
2. print(str1)
3. #Using double quotes
4. str2 = "Hello Python"
5. print(str2)
```

```
1. # Given String
2. str = "BMS Institue of Technology"
3. # Start Oth index to end
4. print(str[0:])
5. # Starts 1th index to 4th index
6. print(str[1:5])
7. # Starts 2nd index to 3rd index
8. print(str[2:4])
9. # Starts 0th to 2nd index
10. print(str[:3])
11. #Starts 4th to 6th index
12. print(str[4:7])
```

LIST

```
1.  L1 = ["John", 102, "USA"]
2.  L2 = [1, 2, 3, 4, 5, 6]
```

IIf we try to print the type of L1, L2, and L3 using type() function then it will come out to be a list.

```
1. print(type(L1))
2. print(type(L2))
```

Output:

```
<class 'list'>
<class 'list'>
```

Characteristics of Lists

The list has the following characteristics:

- The lists are ordered.
- The element of the list can access by index.
- The lists are the mutable type.
- The lists are mutable types.
- A list can store the number of various elements.

Let's check the first statement that lists are the ordered.

```
1.  a = [1,2,"Peter",4.50,"Ricky",5,6]
2.  b = [1,2,5,"Peter",4.50,"Ricky",6]
3.  a ==b
```

Output:

```
False
```

Both lists have consisted of the same elements, but the second list changed the index position of the 5th element that violates the order of lists. When compare both lists it returns the false.

Lists maintain the order of the element for the lifetime. That's why it is the ordered collection of objects.

```
1.  a = [1, 2,"Peter", 4.50,"Ricky",5, 6]
2.  b = [1, 2,"Peter", 4.50,"Ricky",5, 6]
3.  a == b
```

Output:

```
True
```

```
emp = ["John", 102, "USA"]
Dep1 = ["CS",10]
Dep2 = ["IT",11]
HOD_CS = [10,"Mr. Holding"]
HOD_IT = [11, "Mr. Bewon"]
print("printing employee data...")
print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))
print("printing departments...")
print("Department 1:\nName: %s, ID: %d\nDepartment 2:\nName: %s, ID: %s"%(Dep1[0],Dep2[1],Dep2[0],Dep2[1]))
print("HOD Details ....")
print("CS HOD Name: %s, Id: %d"%(HOD_CS[1],HOD_CS[0]))
print("IT HOD Name: %s, Id: %d"%(HOD_IT[1],HOD_IT[0]))
print(type(emp),type(Dep1),type(Dep2),type(HOD_CS),type(HOD_IT))
```

```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
# Slicing the elements
print(list[0:6])
# By default the index value is 0 so its starts from the 0th element and go for index -1.
print(list[:])
print(list[2:5])
print(list[1:6:2])
list = [1,2,3,4,5]
print(list[-1])
print(list[-3:])
print(list[:-1])

print(list[-3:-1])
```

```
#updation
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to the second index
list[2] = 10
print(list)
# Adding multiple-element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)


Student = {"Name": "John", "Age": 10, "Result":"Passed","Rollno":"009001"}
print(type(Student))
print("printing Student data .... ")
print(Student)
```

How to create a dictionary in Python

A Python dictionary is stored the data in the pair of key-value. It organizes the data in a unique manner where some specific value exists for some particular key. It is a mutable data-structure; its element can be modified after creation. Before creating a dictionary, we should remember the following points.

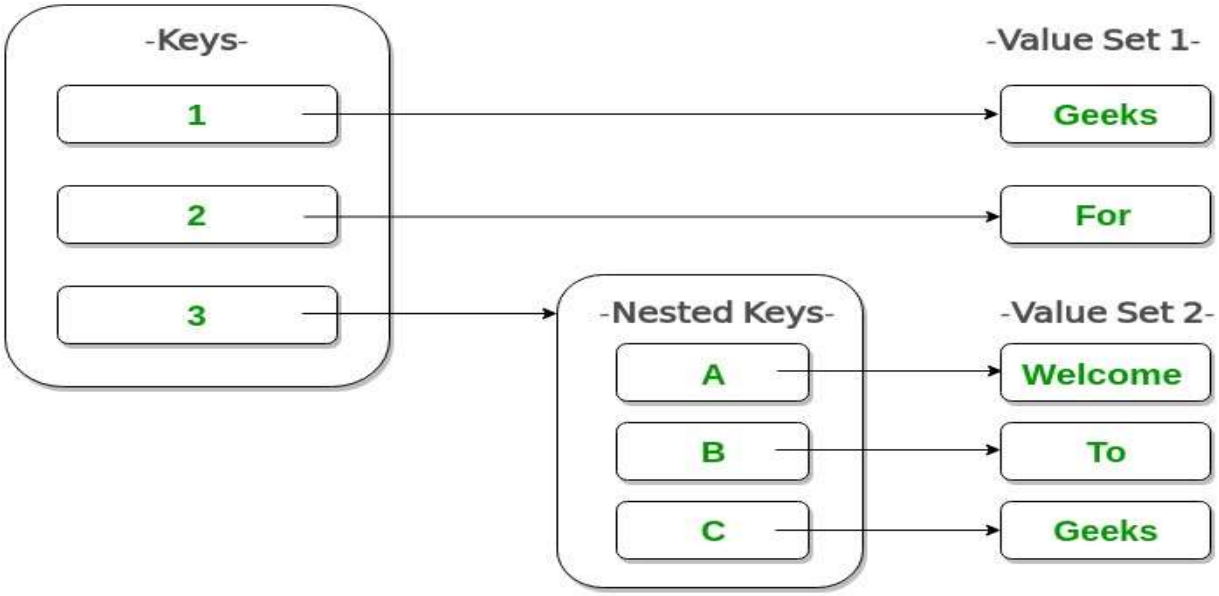
- Keys must be unique and must contain a single value.
- Values can be any type, such as integer, list, tuple, string, etc.
- Keys must be immutable.

Creating a Dictionary

The dictionary is created using the multiple key-value pair, which enclosed within the curly brackets {}, and each key is separated from its value by the colon (:). The syntax is given below.

```
dict = {}
```

```
print("Empty Dictionary is: ")
print(dict)

# Creating a Dictionary
# using the dict() method
dict1 = dict({1: 'Hello', 2: 'Hi', 3: 'Hey'})
print("\nCreate Dictionary by using the dict() method : ")
print(dict1)
```



References for the points

1. **Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education.**
2. **google.com**