DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi)

BACHELOR OF ENGINEERING

DEPARTMENT
OF
INFORMATION SCIENCE AND ENGINEERING

MACHINE LEARNING LABORATORY

[18IS6DLMLL]

VI SEMESTER

2020-21

MACHINE LEARNING LAB

Course code: 18IS6DLMLL **Credits: 02**
L: P: T: S: 0: 2: 1: 0 **CIE Marks: 50**
Exam Hours: 03 **SEE Marks:50**

Course Objectives:

1. Implement supervised and unsupervised machine learning algorithms
2. Perform classification on the preprocessed dataset.
3. Implement the machine learning concepts and algorithms in Python Programming

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

CO1	Understand and implement supervised and unsupervised machine learning algorithms
CO2	Analyze and Implement Machine Learning algorithms on a given dataset
CO3	Construct the linear regression model as a method for prediction
CO4	Develop Bayesian concepts and clustering algorithms using Python program
CO5	Design and implement decision tree using information gain and entropy calculations
CO6	Analyze and build Artificial neural network.

Mapping of Course outcomes to Program outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	2	-	-	3	-	-	-	-	-	-	2	2	-	-
CO2	3	3	-	-	3	-	-	-	-	-	-	2	2	-	-
CO3	3	2	2	-	3	-	-	-	-	-	-	2	2	-	-
CO4	3	2	2	-	3	-	-	-	-	-	-	2	2	-	-
CO5	3	2	2	-	3	-	-	-	-	-	-	2	2	-	-
CO6	3	2	2	-	3	-	-	-	-	-	-	2	2	-	-

Unit	Course Content	Hours	COs
1.	Implement simple linear regression using python program and estimate statistical quantities from training data	3	**CO1, CO2, CO3**
2.	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.	3	**CO1, CO2**
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.	3	**CO1, CO2**
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.	3	**CO1, CO2**

MACHINE LEARNING LAB -18IS6DLMLL

<table>
<tr><td>5.</td><td>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.</td><td>3</td><td>CO2, CO5</td></tr>
<tr><td>6.</td><td>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.</td><td>3</td><td>CO2, CO4</td></tr>
<tr><td>7.</td><td>For the given table, write a python program to perform K-Means Clustering.
<table>
<tr><td>X1</td><td>3</td><td>1</td><td>1</td><td>2</td><td>1</td><td>6</td><td>6</td><td>6</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>8</td><td>9</td><td>9</td><td>8</td></tr>
<tr><td>X2</td><td>5</td><td>4</td><td>6</td><td>6</td><td>5</td><td>8</td><td>6</td><td>7</td><td>6</td><td>7</td><td>1</td><td>2</td><td>1</td><td>2</td><td>3</td><td>2</td><td>3</td></tr>
</table></td><td>3</td><td>CO2, CO4</td></tr>
<tr><td>8.</td><td>Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.</td><td>3</td><td>CO2, CO4</td></tr>
<tr><td>9.</td><td>For the given customer dataset, using the dendogram to find the optimal number of clusters and finding Hierarchical Clustering to the dataset</td><td>3</td><td>CO2, CO4</td></tr>
<tr><td>10.</td><td>Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.</td><td>3</td><td>CO2, CO6</td></tr>
</table>

TEXT BOOKS

1. Ethem Alpaydın: Introduction to Machine Learning, 2nd Edition, The MIT Press Cambridge, Massachusetts London, England, 2010.
2. Tom M. Mitchell, “Machine Learning”, McGraw-Hill Education (INDIAN EDITION), 2018

Assessment Pattern:

CIE –Continuous Internal Evaluation Lab (50 Marks)

Continual Internal Evaluation Marks (25)	IA Test Marks (25)	Final Marks (50)

SEE –Semester End Examination Theory (50 Marks)

MACHINE LEARNING LAB -18IS6DLMLL
Program 1

Implement simple linear regression using python program and estimate statistical quantities from training data

Linear regression models provide a simple approach towards supervised learning. They are simple yet effective. Linear implies the following: arranged in or extending along a straight or nearly straight line. Linear suggests that the relationship between dependent and independent variable can be expressed in a straight line.

y = mx + c

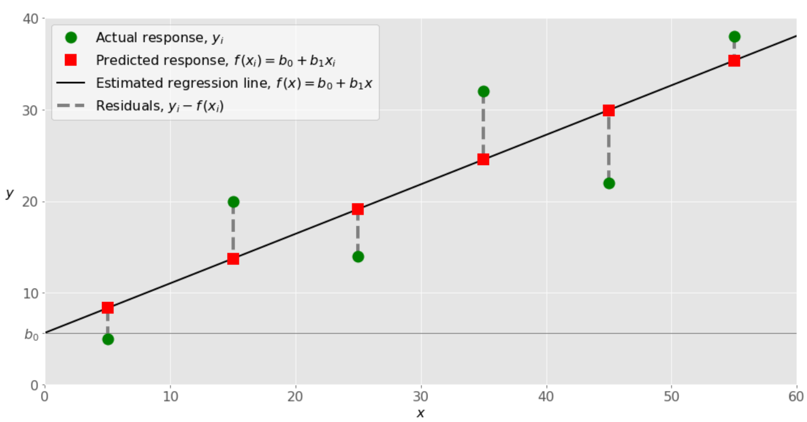
Linear regression is nothing but a manifestation of this simple equation.

- **y** is the dependent variable i.e. the variable that needs to be estimated and predicted.
- **x** is the independent variable i.e. the variable that is controllable. It is the input.
- **m** is the slope. It determines what will be the angle of the line. It is the parameter denoted as β.
- **c** is the intercept. A constant that determines the value of y when x is 0.

$$Y = \beta_0 + \beta_1 X + \epsilon$$

When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and $\mathbf{x}$: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$. This equation is the regression equation. $\beta_0, \beta_1, \ldots, \beta_r$ are the regression coefficients, and ε is the random error.

Linear regression calculates the **estimators** of the regression coefficients or simply the **predicted weights**, denoted with $b_0, b_1, \ldots, b_r$. They define the **estimated regression function** $(\mathbf{x}) = b_0 + b_1 x_1 + \cdots + b_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.



When implementing simple linear regression, you typically start with a given set of input-output (x-y) pairs (green circles). These pairs are your observations. For example, the leftmost observation (green circle) has the input x = 5 and the actual output (response) y = 5. The next one has x = 15 and y = 20, and so on. Linear regression assumes a linear or straight line relationship between the input variables (X) and the single output variable (y).

MACHINE LEARNING LAB -18IS6DLMLL

Python Code

```
import matplotlib.pyplot as plt
import numpy as np
from math import sqrt

# Calculate root mean squared error
def rmse_metric(actual, predicted):
	sum_error = 0.0
	for i in range(len(actual)):
		prediction_error = predicted[i] - actual[i]
		sum_error += (prediction_error ** 2)
	mean_error = sum_error / float(len(actual))
	return sqrt(mean_error)

# Evaluate regression algorithm on training dataset
def evaluate_algorithm(dataset, algorithm):
	test_set = list()
	for row in dataset:
		row_copy = list(row)
		row_copy[-1] = None
		test_set.append(row_copy)
	predicted = algorithm(dataset, test_set)
	print(predicted)
	actual = [row[-1] for row in dataset]
	rmse = rmse_metric(actual, predicted)
	return rmse

# Calculate the mean value of a list of numbers
def mean(values):
	return sum(values) / float(len(values))

# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
	covar = 0.0
	for i in range(len(x)):
		covar += (x[i] - mean_x) * (y[i] - mean_y)
	return covar

# Calculate the variance of a list of numbers
def variance(values, mean):
	return sum([(x-mean)**2 for x in values])

# Calculate coefficients
def coefficients(dataset):
	x = [row[0] for row in dataset]
	y = [row[1] for row in dataset]
	x_mean, y_mean = mean(x), mean(y)
	b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
	b0 = y_mean - b1 * x_mean
	return [b0, b1]
```

```
# Simple linear regression algorithm
def simple_linear_regression(train, test):
	predictions = list()
	b0, b1 = coefficients(train)
	for row in test:
		yhat = b0 + b1 * row[0]
		predictions.append(yhat)
	return predictions

# Test simple linear regression
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean(y)
var_x, var_y = variance(x, mean_x), variance(y, mean_y)
print('x stats: mean=%.3f variance=%.3f' % (mean_x, var_x))
print('y stats: mean=%.3f variance=%.3f' % (mean_y, var_y))
covar = covariance(x, mean_x, y, mean_y)
print('Covariance: %.3f' % (covar))
rmse = evaluate_algorithm(dataset, simple_linear_regression)
print('RMSE: %.3f' % (rmse))
# calculate coefficients
b0, b1 = coefficients(dataset)
print('Coefficients: B0=%.3f, B1=%.3f' % (b0, b1))
```

OUTPUT:

```
x stats: mean=3.000 variance=10.000
y stats: mean=2.800 variance=8.800
Covariance: 8.000
[1.1999999999999995, 1.9999999999999996, 3.5999999999999996, 2.8, 4.3999999999999995]
RMSE: 0.693
Coefficients: B0=0.400, B1=0.800
```

MACHINE LEARNING LAB -18IS6DLMLL
Program 2

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.

In order to understand Find-S algorithm, following concepts are necessary:

• Concept Learning

• General Hypothesis

• Specific Hypothesis

Most of human learning is based on past instances or experiences. For example, we are able to identify any type of vehicle based on a certain set of features like make, model, etc., that are defined over a large set of features. These special features differentiate the set of cars, trucks, etc from the larger set of vehicles. These features that define the set of cars, trucks, etc are known as concepts. Similar to this, machines can also learn from concepts to identify whether an object belongs to a specific category or not. Any algorithm that supports concept learning requires the following:

– Training Data

– Target Concept

– Actual Data Objects

Hypothesis, in general, is an explanation for something. The general hypothesis basically states the general relationship between the major variables. For example, a general hypothesis for ordering food would be I want a burger.

G = { '?', '?', '?', ...'?'}

The specific hypothesis fills in all the important details about the variables given in the general hypothesis. The more specific details into the example considered would be I want a cheese burger with pepperoni filling with a lot of lettuce.

S = {'Φ','Φ','Φ', ...,'Φ'}

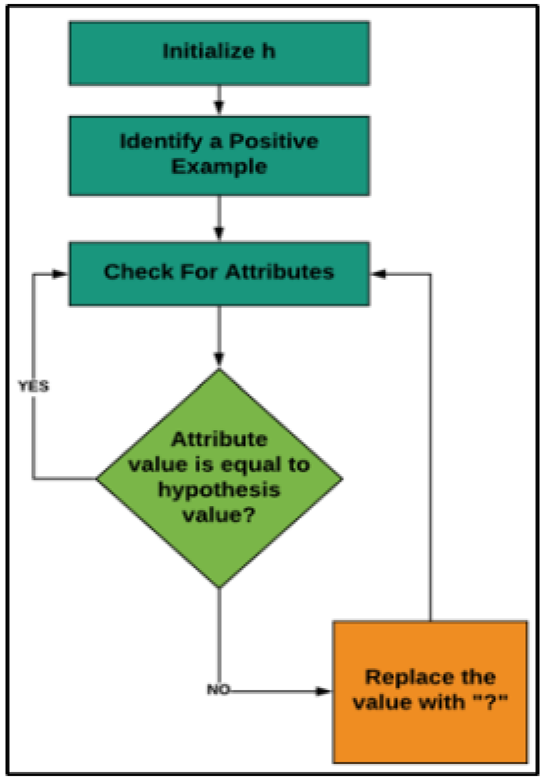
The Find-S algorithm follows the steps below:

– Initialize 'h' to the most specific hypothesis.

– The Find-S algorithm only considers the positive examples and eliminates negative examples.

• For each positive example, the algorithm checks for each attribute in the example. If the attribute value is the same as the hypothesis value, the algorithm moves on without any changes. But if the attribute value is different than the hypothesis value, the algorithm changes it to '?'.

How Does It Work?



- The process starts with initializing 'h' with the most specific hypothesis; generally, it is the first positive example in the data set.
- Then check for each positive example. If the example is negative, move on to the next example but if it is a positive example consider it for the next step.
- Check if each attribute in the example is equal to the hypothesis value.
    - If the value matches, then no changes are made.
    - If the value does not match, the value is changed to '?'.
- Do this until the last positive example in the data set is reached.

Python code

```
import pandas as pd
import numpy as np
#to read the data in the csv file
data = pd.read_csv("C:/Users/Sudipta/Data.csv.csv")
print(data)

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
    return specific_hypothesis

#obtaining the final hypothesis
print("The final hypothesis is:",train(d,target))
```

```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes

The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

The target is:  ['Yes' 'No' 'Yes' 'Yes']

The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

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.

Candidate Elimination Algorithm Concept:

– Will use Version Space. Version Space: It is the intermediate space between Specific hypothesis and general hypothesis. It denotes not just one hypothesis but a set of all possible hypothesis based on training data-set.

– Considers both positive and negative result.

– For positive example: tends to generalize specific hypothesis.

– For Negative example: tends to make general hypothesis more specific.

– Hypothesis space 'h' is described by a conjunction of constraints on the attribute:

– the constraints may be General hypothesis "?" (any value is acceptable),

– Specific hypothesis "φ" (a specific value or no value is accepted).

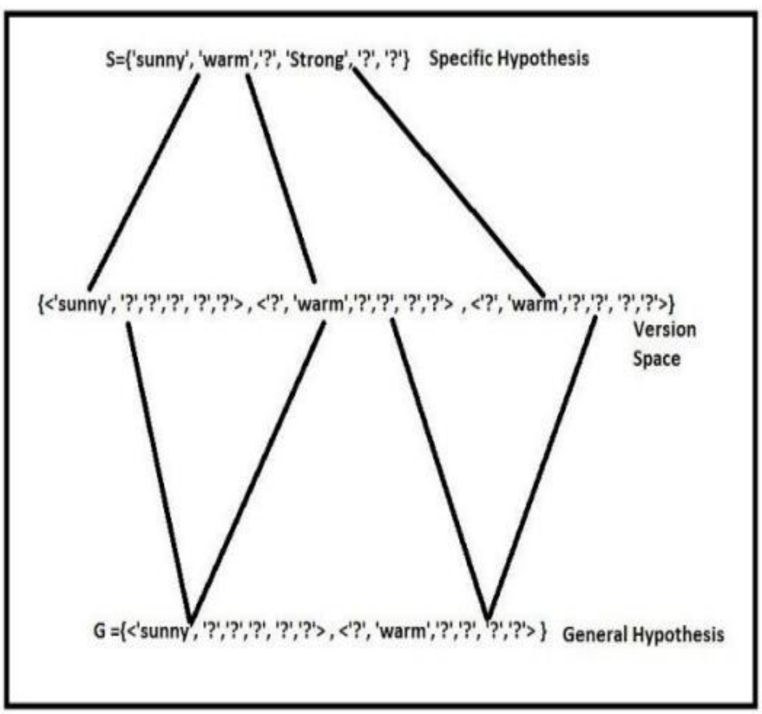
Algorithm

Initialize G & S as most General and specific hypothesis.

• For each example e:

if e is +ve: make specific hypothesis more general.

Else if e is -ve: make a general hypothesis more specific.



MACHINE LEARNING LAB -18IS6DLMLL

Python code

```
# Importing Important Libraries
import numpy as np
import pandas as pd

data=pd.read_csv('ENJOYSPORT.csv')
print(data)
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)

# Candidate Elimination algorithm
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific hypothesis: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric hypothesis: ",general_h)
    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Specific hypothesis after ", i+1, "Instance is ", specific_h)
        print("Generic hypothesis after ", i+1, "Instance is ", general_h)
        print("\n")
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]

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

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

Output:

```
    Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        yes
1  Sunny    Warm     High  Strong  Warm     Same        yes
2  Rainy    Cold     High  Strong  Warm   Change         no
3  Sunny    Warm     High  Strong  Cool   Change        yes
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
['yes' 'yes' 'no' 'yes']
Initialization of specific_h and genearal_h

Specific hypothesis:  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific hypothesis after  1 Instance is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm'
'Same']
Generic hypothesis after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is  ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific hypothesis after  2 Instance is  ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic hypothesis after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is  ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
Instance is Negative
Specific hypothesis after  3 Instance is  ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic hypothesis after  3 Instance is  [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Instance 4 is  ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
Instance is Positive
Specific hypothesis after  4 Instance is  ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Generic hypothesis after  4 Instance is  [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Program 4

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

Task: ID3 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain as the root node of the tree. The information gain values for all four attributes are calculated using the following formula:

$$\textbf{Entropy(S)} = \sum - P(I).\log_2 P(I)$$

$$\textbf{Gain(S,A)} = \textbf{Entropy(S)} - \sum [\, P(S/A).\textbf{Entropy}(S/A) \,]$$

Dataset:

Table: Training examples for the target concept PlayTennis.

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Calculation:

Decision/play column consists of 14 instances and includes two labels: yes and no. There are 9 decisions labeled yes and 5 decisions labeled no.

$$\begin{aligned} \text{Entropy[Decision]} &= -P(yes).\log_2 P(yes) - P(no).\log_2 P(no) \\ &= -(9/14).\log_2(9/14) - (5/14).\log_2(5/14) \\ &= 0.940 \end{aligned}$$

Now, we need to find out the most dominant factor for decision.

1) Wind factor on decision:

Gain(Decision,wind)= Entropy(Decision) - ∑[P(Decision/Wind]. Entropy (Decision/Wind)]

Wind attribute has two labels : Weak and Strong

Gain(Decision,Wind)= Entropy(Decision) –
[P(Decision/Wind=Weak). Entropy(Decision/Wind=Weak)] -
[[P(Decision/Wind=strong).
Entropy(Decision/Wind=strong)]

There are 8 instances for weak. In that decision of 2 items are no and 6 items are yes.

- Entropy[Decision/Wind= Weak] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
  = -[2/8].$\log_2$(2/8) – [6/8].$\log_2$(6/8)
  = 0.811

- Entropy[Decision/Wind= Strong] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
  = -[3/6].$\log_2$(3/6) – [3/6].$\log_2$(3/6)
  = 1

  Note: There are 6 instances for strong. In that decisionof 3 items are yes and 3 items are no.

- Gain(Decision,Wind) = 0.940 – [(8/14).0.811] – [(6/14).1]
  = 0.048

  Similarly calculate gain for other factors:

2) Outlook factor on decision:

1) Gain(Decision,outlook) = Entropy (decision) - ∑ [P (Decision /Outlook). Entropy(Decision/Outlook)]

outlook has three parameters : Sunny, Overcast, and rain

Gain(Decision,outlook) = Entropy(decision)-
[P(Decision/Outlook=Sunny).Entropy(Decision/Outlook=Sunny)]-[
P(Decision/Outlook=overcast).Entropy(Decision/Outlook=overcast)
- [P(P(Decision/Outlook=Rain).Entropy(Decision/Outlook=Rain)]

Sunny	Overcast:	Rain:
Instances: 5	Instances:4	Instances:5
yes:2	yes:4	yes:3
No:3	No: -	No: 2

1) Entropy[Decision/Outlook= Sunny] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
=0.97094

2) Entropy[Decision/Outlook= Overcast] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
= 0

3) Entropy[Decision/Wind= Rain] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
=0.9708

Gain(Decision,Outlook)= 0.940-(5/14)(0.9709)-(4/14)(0)-(5/14)(0.9708)
=0.2473

3) Temperature factor on decision:

Gain(Decision,Temperature) = Entropy(decision)-
∑[P(Decision/Temperature).Entropy(Decision/Temperature)]

Temperature has 3 parameters: hot, mild, cool

Gain(Decision,Temp) = Entropy(decision)-
[P(Decision/Temp=hot).Entropy(Decision/Temp=hot)]-[
P(Decision/Temp=mild).Entropy(Decision/Temp=mild)]-
[P(P(Decision/Temp=cool).Entropy(
Decision/Temp=cool)]

Hot	Mild:	cool:
Instances:4	Instances:6	Instances:4
Yes:2	Yes:4	Yes:3
No:2	No: 2	No:1

Entropy[Decision/Temp= hot] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
= 1
Entropy[Decision/Temp= mild] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
=0.9182
Entropy[Decision/Temp= cool] = -P[no].$\log_2$P(no) – p(yes).$\log_2$P(yes)
=0.8112

Gain(Decision,Temp) = 0.940 - (4/14)(1) - (6/14)(0.9182) - (4/14)(0.8112)
= 0.0291

4) Humidity factor on decision:

Gain(Decision,Humidity) = Entropy(decision)-
∑[P(Decision/Humidity).Entropy(Decision/Humidity)]

Humidity has 2 factors: high and normal
Gain(Decision,humidity) = Entropy(decision)-
[P(Decision/humidity=high).Entropy(Decision/humidity=high]-
[P(Decision/humidity=normal).Entropy(Decision/humidity=nor
mal)]

High	Normal:
Instances:7	Instances:7
Yes:3	Yes:6
No:4	No: 1

Entropy[Decision/ humidity= high] = $-P[no].\log_2 P(no) - p(yes).\log_2 P(yes)$
= 0.9851

Entropy[Decision/humidity= normal] = $-P[no].\log_2 P(no) - p(yes).\log_2 P(yes)$
= 0.5916

Gain(Decision, Humidity)= 0.940 - (7/14)(0.9851) - (7/14)(0.5916)
= 0.1517

Thus the outlook factor on decision produces the highest score. That's why outlook decision will appear in the root node of the tree. Since Outlook has three possible values, the root node has three branches (sunny, overcast, rain). The next question is "what attribute should be tested at the Sunny branch node?" Since we have used Outlook at the root, we only decide on the remaining three attributes: Humidity, Temperature, or Wind.

Now calculate sunny outlook on decision, overcast outlook on decision, and rain outlook on decision to generate the decision tree.

Sunny outlook on decision:

5 instances of sunny : In that 3 instances are NO and 2 instances are YES

Gain(Outlook = Sunny/Temp)= 0.570

Gain (Outlook = Sunny/Humidity) = 0.970

Gain (Outlook = Sunny/ Wind) = 0.019

Since humidity produces the highest score, if outlook were Sunny.

Overcast outlook on decision:

Decision will always be yes, if outlook were overcast.

Rain outlook on decision:

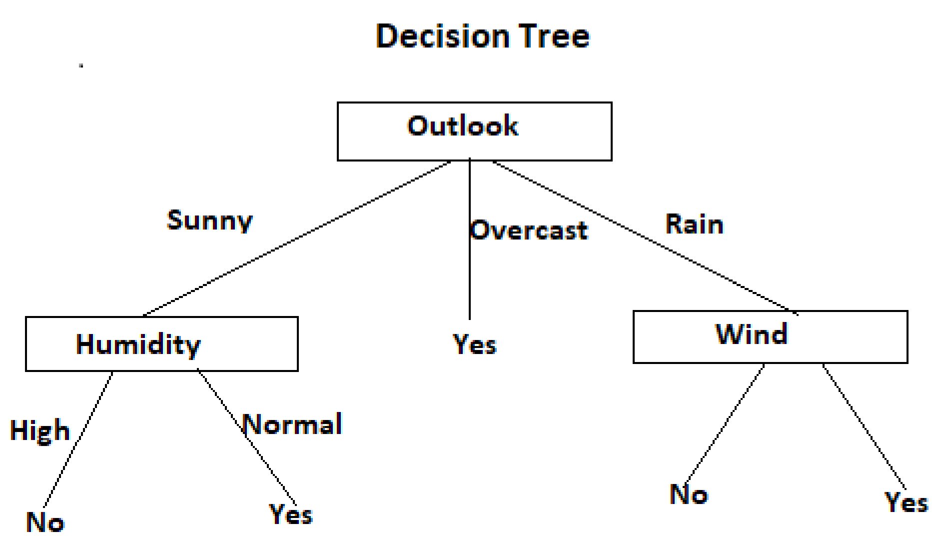
5 instances of rain : In that 3 instances are YES and 2 instances are NO.

Gain(Outlook= Rain/Temp)

Gain(Outlook= Rain/Humidity)

Gain(Outlook= Rain/Wind)

Here, wind produces the highest score . And wind has two attributes namely strong and weak.



ID3 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 ← the attribute from *Attributes* that best* classifies *Examples*
- ❖ The decision attribute for *Root* ←A
- ❖ For each possible value, υ_i, of A,

- Add a new tree branch below *Root*, corresponding to the test A = υ_i
- Let *Examples*$_{\upsilon i}$,be the subset of *Examples* that have value υ_i for A
- If *Examples*$_{\upsilon i}$, 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*$_{\upsilon i}$,*Target_attribute*, *Attributes*–*{A}*))

End

Return Root

ID3 in Python:

```
#Importing important libraries
import pandas as pd
from pandas import DataFrame
#Reading Dataset
df_tennis = pd.read_csv('DS.csv')
print( df_tennis)
```

Output

```
      Outlook Temperature Humidity   Windy   PT
0       Sunny         Hot     High    Weak   No
1       Sunny         Hot     High  Strong   No
2    Overcast         Hot     High    Weak  Yes
3       Rainy        Mild     High    Weak  Yes
4       Rainy        Cool   Normal    Weak  Yes
5       Rainy        Cool   Normal  Strong   No
6    Overcast        Cool   Normal  Strong  Yes
7       Sunny        Mild     High    Weak   No
8       Sunny        Cool   Normal    Weak  Yes
9       Rainy        Mild   Normal    Weak  Yes
10      Sunny        Mild   Normal  Strong  Yes
11   Overcast        Mild     High  Strong  Yes
12   Overcast         Hot   Normal    Weak  Yes
13      Rainy        Mild     High  Strong   No
```

Calculating Entropy of Whole Data-set

```
#Function to calculate final Entropy
def entropy(probs):
import math
return sum( [-prob*math.log(prob, 2) for prob in probs] )
#Function to calculate Probabilities of positive and negative examples
def entropy_of_list(a_list):
from collections import Counter
cnt = Counter(x for x in a_list)
#Count the positive and negative ex
num_instances = len(a_list)
#Calculate the probabilities that we required for our entropy formula
probs = [x / num_instances for x in cnt.values()]
#Calling entropy function for final entropy
return entropy(probs)
total_entropy = entropy_of_list(df_tennis['PT'])
print("\n Total Entropy of PlayTennis Data Set:",total_entropy)
```

Output

```
Total Entropy of PlayTennis Data Set: 0.9402859586706309
```

collections.Counter()

A counter is a container that stores elements as dictionary keys, and their counts are stored as dictionary values.

Calculate Information Gain for each Attribute

```
#Defining Information Gain Function
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
print("Information Gain Calculation of ",split_attribute_name)
print("target_attribute_name",target_attribute_name)
#Grouping features of Current Attribute
 df_split = df.groupby(split_attribute_name)
   for name,group in df_split:
      print("Name: ",name)
      print("Group: ",group)
nobs = len(df.index) * 1.0
   print("NOBS",nobs)
#Calculating Entropy of the Attribute and probability part of formula
   df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs]
})[target_attribute_name]
   print("df_agg_ent",df_agg_ent)
# Calculate Information Gain
   avg_info = sum( df_agg_ent['Entropy'] * df_agg_ent['Prob1'] )
   old_entropy = entropy_of_list(df[target_attribute_name])
   return old_entropy - avg_info
print('Info-gain for Outlook is :'+str(information_gain(df_tennis, 'Outlook', 'PT')),"\n")
```

Output

```
target_attribute_name PT
Name:  Overcast
Group:       Outlook Temperature Humidity   Windy   PT predicted
2   Overcast         Hot     High    Weak  Yes       Yes
6   Overcast        Cool   Normal  Strong  Yes       Yes
11  Overcast        Mild     High  Strong  Yes       Yes
12  Overcast         Hot   Normal    Weak  Yes       Yes
Name:  Rainy
Group:     Outlook Temperature Humidity   Windy   PT predicted
3     Rainy        Mild     High    Weak  Yes       Yes
4     Rainy        Cool   Normal    Weak  Yes       Yes
5     Rainy        Cool   Normal  Strong   No        No
9     Rainy        Mild   Normal    Weak  Yes       Yes
13    Rainy        Mild     High  Strong   No        No
Name:  Sunny
Group:     Outlook Temperature Humidity   Windy   PT predicted
0     Sunny         Hot     High    Weak   No        No
1     Sunny         Hot     High  Strong   No        No
7     Sunny        Mild     High    Weak   No        No
8     Sunny        Cool   Normal    Weak  Yes       Yes
10    Sunny        Mild   Normal  Strong  Yes       Yes
NOBS 14.0
df_agg_ent           entropy_of_list  <lambda_0>
Outlook
Overcast         0.000000    0.285714
Rainy            0.970951    0.357143
Sunny            0.970951    0.357143
df_agg_ent.columns Index(['Entropy', 'Prob1'], dtype='object')
Info-gain for Outlook is :0.2467498197744391
```

Defining ID3 Algorithm

```
#Defining ID3  Algorithm Function
def id3(df, target_attribute_name, attribute_names, default_class=None):
#Counting Total number of yes and no classes (Positive and negative Ex)
from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])
if len(cnt) == 1:
        return next(iter(cnt))
# Return None for Empty Data Set
 elif df.empty or (not attribute_names):
        return default_class
 else:
default_class = max(cnt.keys())
print("attribute_names:",attribute_names)
        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names]
#Separating the maximum information gain attribute after calculating the information gain
        index_of_max = gainz.index(max(gainz)) #Index of Best Attribute
best_attr = attribute_names[index_of_max] #choosing best attribute
#The tree is initially an empty dictionary
tree = {best_attr:{}} # Initiate the tree with best attribute as a node
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]
for attr_val, data_subset in df.groupby(best_attr):
subtree = id3(data_subset,
                        target_attribute_name,
                        remaining_attribute_names,
                        default_class)
            tree[best_attr][attr_val] = subtree
        return tree

# Get Predictor Names (all but 'class')
attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PT')
#Remove the class attribute
print("Predicting Attributes:", attribute_names)
```

Output

```
List of Attributes: ['Outlook', 'Temperature', 'Humidity', 'Windy', 'PT', 'pred
icted']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Windy', 'predict
ed']
```

```
# Run Algorithm (Calling ID3 function)
from pprint import pprint
tree = id3(df_tennis,'PT',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)
```

```
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

```
The Resultant Decision Tree is :

{'Outlook': {'Overcast': 'Yes',
             'Rainy': {'Windy': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Best Attribute :
 Outlook
Tree Keys:
 dict_keys(['Overcast', 'Rainy', 'Sunny'])
```

ACCURACY

```
#Defining a function to calculate accuracy
def classify(instance, tree, default=None):
attribute = next(iter(tree))
print("Key:",tree.keys())
print("Attribute:",attribute)
print("Insance of Attribute :",instance[attribute],attribute)
    if instance[attribute] in tree[attribute].keys():
        result = tree[attribute][instance[attribute]]
        print("Instance Attribute:",instance[attribute],"TreeKeys :",tree[attribute].keys())
        if isinstance(result, dict):
            return classify(instance, result)
        else:
            return result
    else:
        return default

df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree,'No') )
print(df_tennis['predicted'])
print('\n Accuracy is:\n' + str( sum(df_tennis['PT']==df_tennis['predicted'] ) / (1.0*len(df_tennis.index)) ))
df_tennis[['PT', 'predicted']]

training_data = df_tennis.iloc[1:-4]
test_data  = df_tennis.iloc[-4:]
train_tree = id3(training_data, 'PT', attribute_names)
test_data['predicted2'] = test_data.apply(
classify, axis=1, args=(train_tree,'Yes') )
print ('\n\n Accuracy is : ' + str( sum(test_data['PT']==test_data['predicted2'] ) / (1.0*len(test_data.index)) ))
```

Output

```
Accuracy is : 0.75
```

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.

Task: It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where, P(h|D) is the probability of hypothesis h given the data D. This is called the posterior probability. P(D|h) is the probability of data d given that the hypothesis h was true. P(h) is the probability of hypothesis h being true. This is called the prior probability of h. P(D) is the probability of the data. This is called the prior probability of D.

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$
$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$
$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)

Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution.

Sample Examples:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

- The data set used in this program is the Pima Indians Diabetes problem.
- This data set is comprised of 768 observations of medical details for Pima Indians patents.

  The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are **Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome**
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Python code

```
import csv
import random
import math
def loadcsv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    for i in range(len(dataset)):
        # converting the attributes from string to floating point numbers
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
```

```
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy)) # random index
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated
def mean(numbers):
    return sum(numbers)/float(len(numbers))
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)
def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries
def calculateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
```

```
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions
def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0
def main():
    filename = 'C:\\Users\\DELL\\.conda\\envs\\ml_env\\Scripts\\naivedata.csv'
    splitRatio = 0.67
    dataset = loadcsv(filename)

    #print("\n The Data Set :\n",dataset)
    print("\n The length of the Data Set : ",len(dataset))

    print("\n The Data Set Splitting into Training and Testing \n")
```

```
    trainingSet, testSet = splitDataset(dataset, splitRatio)

    print('\n Number of Rows in Training Set:{0} rows'.format(len(trainingSet)))
    print('\n Number of Rows in Testing Set:{0} rows'.format(len(testSet)))

    print("\n First Five Rows of Training Set:\n")
    for i in range(0,5):
        print(trainingSet[i],"\n")

    print("\n First Five Rows of Testing Set:\n")
    for i in range(0,5):
        print(testSet[i],"\n")

    # prepare model
    summaries = summarizeByClass(trainingSet)
    print("\n Model Summaries:\n",summaries)

    # test model
    predictions = getPredictions(summaries, testSet)
    print("\nPredictions:\n",predictions)

    accuracy = getAccuracy(testSet, predictions)
    print('\n Accuracy: {0}%'.format(accuracy))
main()
```

OUTPUT

```
The length of the Data Set :  768
The Data Set Splitting into Training and Testing

Number of Rows in Training Set:514 rows
Number of Rows in Testing Set:254 rows
```

First Five Rows of Training Set:

[11.0, 111.0, 84.0, 40.0, 0.0, 46.8, 0.925, 45.0, 1.0]

[11.0, 138.0, 74.0, 26.0, 144.0, 36.1, 0.557, 50.0, 1.0]

[1.0, 97.0, 66.0, 15.0, 140.0, 23.2, 0.487, 22.0, 0.0]

[5.0, 162.0, 104.0, 0.0, 0.0, 37.7, 0.151, 52.0, 1.0]

[2.0, 134.0, 70.0, 0.0, 0.0, 28.9, 0.542, 23.0, 1.0]

First Five Rows of Testing Set:

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]

[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

[5.0, 116.0, 74.0, 0.0, 0.0, 25.6, 0.201, 30.0, 0.0]

[8.0, 125.0, 96.0, 0.0, 0.0, 0.0, 0.232, 54.0, 1.0]

Model Summaries:
{1.0: [(4.621621621621622, 3.675353808400119),(142.18378378378378, 32.791812948886125), (71.55135135135136, 20.365380119287128),(22.524324324324326, 17.700733916947893), (104.57297297297298, 143.58157931205457),(35.32162162162162, 7.162057905588884), (0.543054054054054, 0.37339656809119126), (36.45945945945946, 10.470441299705367)], 0.0: [(3.2401215805471124, 3.009147838987053), (108.44376899696049, 25.944312415767783),(66.65653495440729, 19.37925314843171), (19.89969604863222, 14.814059938401465), (65.09422492401215, 93.30385842522621), (29.891793313069915, 7.963504376664226), (0.41687537993920976, 0.3016472554815733), (30.93617021276596, 11.493590416134817)]}

Predictions:
[1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0]

Accuracy: 72.83464566929135%

Program 6

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.

A Bayesian belief network describes the probability distribution over a set of variables.

Probability

P(A) is used to denote the probability of A. For example if A is discrete with states {True, False} then P(A) might equal [0.2, 0.8]. I.e. 20% chance of being True, 80% chance of being False.

Joint probability

A joint probability refers to the probability of more than one variable occurring together, such as the probability of A and B, denoted P(A,B).

Conditional probability

Conditional probability is the probability of a variable (or set of variables) given another variable (or set of variables), denoted P(A|B).For example, the probability of Windy being True, given that Raining is True might equal 50%.This would be denoted P(Windy = True | Raining = True) = 50%.

Once the structure has been defined (i.e. nodes and links), a Bayesian network requires a probability distribution to be assigned to each node.Each node X in a Bayesian network requires a probability distribution P(X | pa(X)).Note that if a node X has no parents pa(X) is empty, and the required distribution is just P(X) sometimes referred to as the prior.This is the probability of itself given its parent nodes.

If U = {A_1,...,A_n} is the universe of variables (all the variables) in a Bayesian network, and pa(A_i) are the parents of A_i then the joint probability distribution P(U) is the simply the product of all the probability distributions (prior and conditional) in the network, as shown in the equation below.This equation is known as the chain rule.

$$P(\mathbf{X}, \mathbf{e}) = \sum_{\mathbf{U}\setminus\mathbf{X}} P(\mathbf{U}, \mathbf{e}) = \sum_{\mathbf{U}\setminus\mathbf{X}} \prod_i P(\mathbf{U}_i | pa(\mathbf{U}_i))\mathbf{e}$$

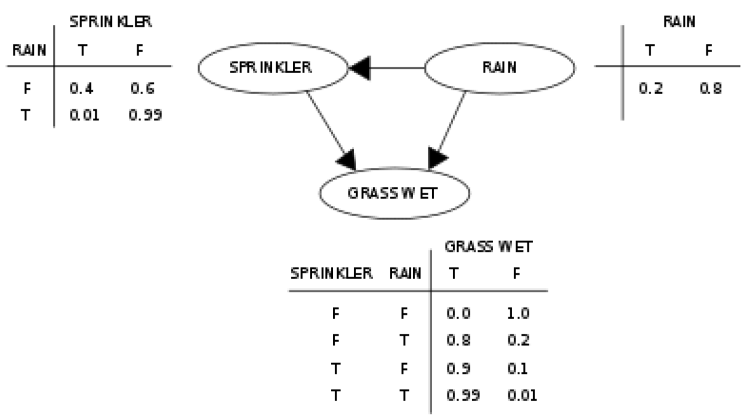
From the joint distribution over U we can in turn calculate any query we are interested in (with or without evidence set).

Suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a Bayesian network (shown to the right). All three variables have two possible values, T (for true) and F (for false).

The joint probability function is:

$$\Pr(G,S,R)=\Pr(G|S,R)\Pr(S|R)\Pr(R)$$

The model can answer questions like "What is the probability that it is raining, given the grass is wet?" by using the conditional probability formula and summing over all nuisance variables:



$$\Pr(R=T|G=T)=\frac{\Pr(G=T,R=T)}{\Pr(G=T)}=\frac{\sum_{S\in\{T,F\}}\Pr(G=T,S,R=T)}{\sum_{S,R\in\{T,F\}}\Pr(G=T,S,R)}$$

$$\begin{aligned}\Pr(G=T,S=T,R=T)&=\Pr(G=T|S=T,R=T)\Pr(S=T|R=T)\Pr(R=T)\\&=0.99\times 0.01\times 0.2\\&=0.00198.\end{aligned}$$

Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database :	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
   - Value 1: typical angina
   - Value 2: atypical angina
   - Value 3: non-anginal pain
   - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
   - Value 0: normal
   - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. ca = number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. Heartdisease: It is integer valued from 0 (no presence) to 4. Diagnosis of heart disease (angiographic disease status)

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Python code

```
import numpy as np
import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv ("C:\\Desktop\\dataset.csv")
heartDisease = heartDisease.replace('?',np.nan)
print('Few examples from the dataset are given below')
print(heartDisease.head())


model=BayesianModel([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),
('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\n Learning CPD using Maximum likelihood estimators')
model.fit (heartDisease,estimator=MaximumLikelihoodEstimator)
print ('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print ('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'age':28})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'chol':100})
print(q2)
```

Output:

```
Few examples from the dataset are given below
     age   sex cp  trestbps   ...slope  ca  thal  heartdisease
0    63    1   1      145       ...  3    0     6             0
1    67    1   4      160       ...  2    3     3             2
2    67    1   4      120       ...  2    2     7             1
3    37    1   3      130       ...  3    0     3             0
4    41    0   2      130       ...  1    0     3             0

[5 rows x 14 columns]

Learning CPD using Maximum likelihood estimators
```

Inferencing with Bayesian Network

1. Probability of HeartDisease given Age=28

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

2. Probability of HeartDisease given cholesterol=100

heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506

<u>MACHINE LEARNING LAB -18IS6DLMLL</u>
Program 7

For the given table, write a python program to perform K-Means Clustering.

X1	3	1	1	2	1	6	6	6	5	6	7	8	9	8	9	9	8
X2	5	4	6	6	5	8	6	7	6	7	1	2	1	2	3	2	3

K-Means clustering intends to partition n objects into k clusters in which each object belongs to the cluster with the nearest mean. This method produces exactly k different clusters of greatest possible distinction. The best number of clusters k leading to the greatest separation (distance) is not known as a priori and must be computed from the data. The objective of K-Means clustering is to minimize total intra-cluster variance, or, the squared error function:

number of clusters   number of cases   centroid for cluster *j*
case *i*

objective function ← $$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \underbrace{\left\| x_i^{(j)} - c_j \right\|^2}_{\text{Distance function}}$$

Algorithm:

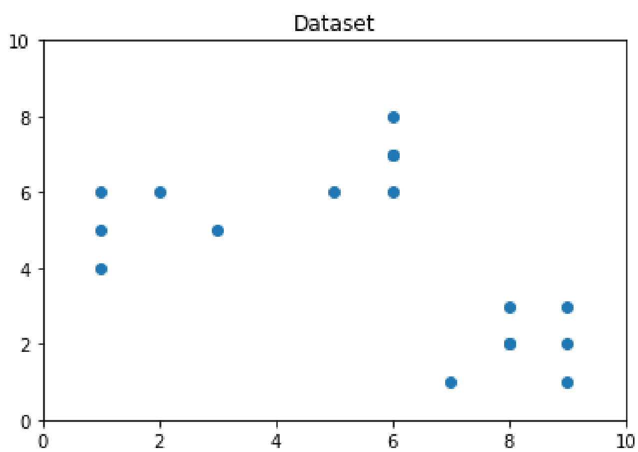
1. Clusters the data into k groups where k is predefined.
2. Select k points at random as cluster centers.
3. Assign objects to their closest cluster center according to the Euclidean distance function.
4. Calculate the centroid or mean of all objects in each cluster.
5. Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds.

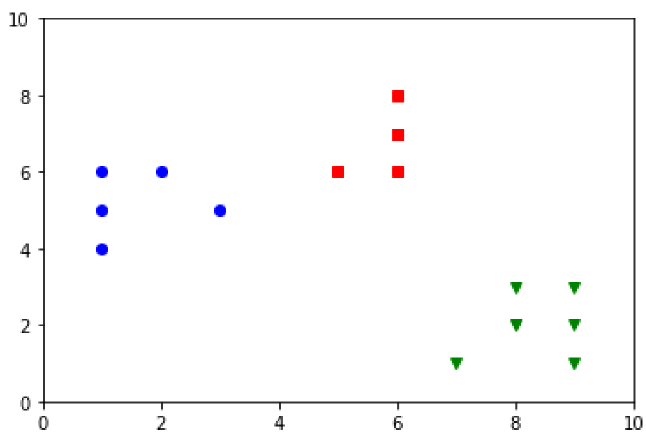
K-Means is relatively an efficient method. However, we need to specify the number of clusters, in advance and the final results are sensitive to initialization and often terminates at a local optimum. Unfortunately, there is no global theoretical method to find the optimal number of clusters. A practical approach is to compare the outcomes of multiple runs with different k and choose the best one based on a predefined criterion. In general, a large k probably decreases the error but increases the risk of over fitting.

Python code

```
# clustering dataset
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
K = 3
kmeans_model = KMeans(n_clusters=K).fit(X)
plt.plot()
for i, l in enumerate(kmeans_model.labels_):
 plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.show()
```

Output:





Program 8

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

In statistics, an expectation–maximization (**EM**) **algorithm** is an iterative method to find (local) maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models. EM algorithms are known for density estimation (maximum likelihood estimation) and EM is also famous for clustering algorithm. The EM algorithm is an approach for performing maximum likelihood estimation in the presence of latent variables.

Probability Density estimation is basically the construction of an estimate based on observed data. It involves selecting a probability distribution function and the parameters of that function that best explains the joint probability of the observed data.

Maximum likelihood estimation

To select the joint probability distribution, what we require is Density estimation. Density estimation needs to find out a probability distribution function and the parameters of that distribution. The most common technique to solve this problem is the Maximum Likelihood Estimation or simply "maximum likelihood". In statistics, maximum likelihood estimation is the method of estimating the parameters of a probability distribution by maximizing the likelihood function in order to make the observed data most probable for the statistical model.

Steps involved in EM Algorithm

1. Consider a set of starting parameters in incomplete data (consider complete data with latent variables or missing values).

2. E-Step (Expectation step): In this step, what we do is that Basically the data in which the missing values and latent variables are present, we estimate them by observe data that we have. (Updating variables and data)

3. M-step (Maximization step): This step is basically used to complete the data we get from E-step. This step updates the hypothesis.

4. If the convergence is not matched then repeat step 2 and 3.

Use of GMM (Gaussian mixture model) in EM

1. Gaussian Mixture Model is used the combination of probability distributions and the estimation of mean and standard deviation parameters.
2. Gaussian mixture model has number of techniques to estimate data but common one is maximum likelihood.

- **K-means** is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

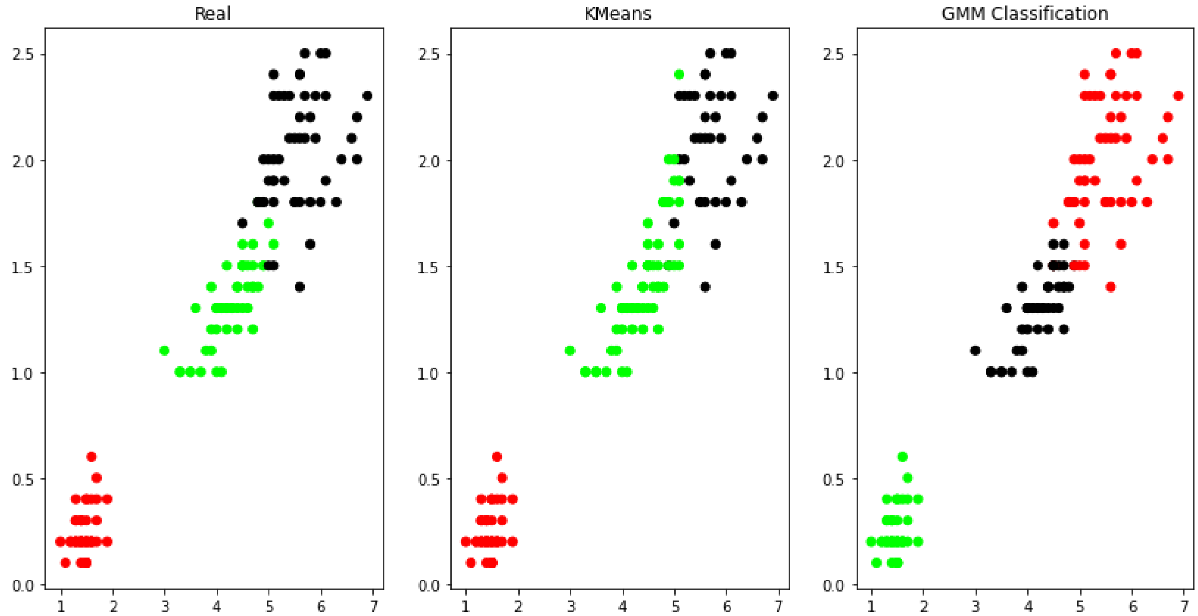
Python code

```
from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("8-dataset.csv", names=names)
```

```
X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT

model=KMeans(n_clusters=3, random_state=3425).fit(X)

plt.subplot(1,3,2)

plt.title('KMeans')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))

print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT

gmm=GaussianMixture(n_components=3, random_state=3425).fit(X)

y_cluster_gmm=gmm.predict(X)

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

Output:

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:
 [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.0
The Confusion matrix of EM:
  [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```


Real
KMeans
GMM Classification
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Program 9

For the given customer dataset, develop dendrogram to find the optimal number of clusters and finding Hierarchical Clustering to the dataset.

Hierarchical clustering involves creating clusters that have a predetermined ordering from top to bottom. For example, all files and folders on the hard disk are organized in a hierarchy. There are two types of hierarchical clustering, Divisive and Agglomerative.

A **dendrogram** is a diagram representing a tree. This diagrammatic representation is frequently used in different contexts:

- in hierarchical clustering, it illustrates the arrangement of the clusters produced by the corresponding analyses.
- in computational biology, it shows the clustering of genes or samples, sometimes in the margins of heatmaps.

Python code

```
# Hierarchical Clustering
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Mall_Customer.csv')
X = dataset.iloc[:, [3, 4]].values
# y = dataset.iloc[:, 3].values
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
fromsklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Data Set : Mall_Customer.CSV

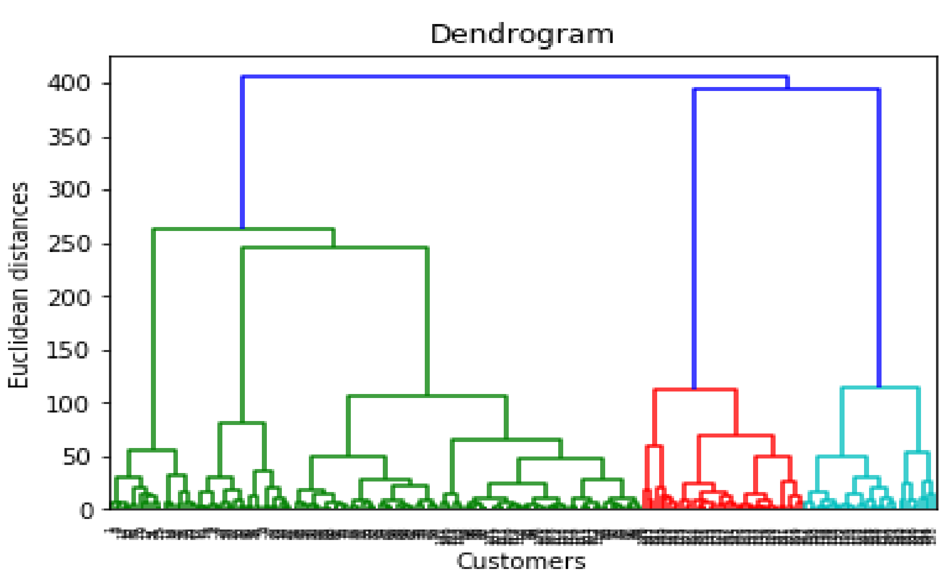
CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35
18	Male	20	21	66
19	Male	52	23	29
20	Female	35	23	98
21	Male	35	24	35
22	Male	25	24	73
23	Female	46	25	5
24	Male	31	25	73
25	Female	54	28	14
26	Male	29	28	82
27	Female	45	28	32
28	Male	35	28	61
29	Female	40	29	31
30	Female	23	29	87
31	Male	60	30	4
32	Female	21	30	73
33	Male	53	33	4
34	Male	18	33	92
35	Female	49	33	14
36	Female	21	33	81
37	Female	42	34	17
38	Female	30	34	73

39	Female	36	37	26
40	Female	20	37	75
41	Female	65	38	35
42	Male	24	38	92
43	Male	48	39	36
44	Female	31	39	61
45	Female	49	39	28
46	Female	24	39	65
47	Female	50	40	55
48	Female	27	40	47
49	Female	29	40	42
50	Female	31	40	42
51	Female	49	42	52
52	Male	33	42	60
53	Female	31	43	54
54	Male	59	43	60
55	Female	50	43	45
56	Male	47	43	41
57	Female	51	44	50
58	Male	69	44	46
59	Female	27	46	51
60	Male	53	46	46
61	Male	70	46	56
62	Male	19	46	55
63	Female	67	47	52
64	Female	54	47	59
65	Male	63	48	51
66	Male	18	48	59
67	Female	43	48	50
68	Female	68	48	48
69	Male	19	48	59
70	Female	32	48	47
71	Male	70	49	55
72	Female	47	49	42
73	Female	60	50	49
74	Female	60	50	56
75	Male	59	54	47
76	Male	26	54	54
77	Female	45	54	53
78	Male	40	54	48
79	Female	23	54	52
80	Female	49	54	42
81	Male	57	54	51
82	Male	38	54	55
83	Male	67	54	41
84	Female	46	54	44

85	Female	21	54	57
86	Male	48	54	46
87	Female	55	57	58
88	Female	22	57	55
89	Female	34	58	60
90	Female	50	58	46
91	Female	68	59	55
92	Male	18	59	41
93	Male	48	60	49
94	Female	40	60	40
95	Female	32	60	42
96	Male	24	60	52
97	Female	47	60	47
98	Female	27	60	50
99	Male	48	61	42
100	Male	20	61	49
101	Female	23	62	41
102	Female	49	62	48
103	Male	67	62	59
104	Male	26	62	55
105	Male	49	62	56
106	Female	21	62	42
107	Female	66	63	50
108	Male	54	63	46
109	Male	68	63	43
110	Male	66	63	48
111	Male	65	63	52
112	Female	19	63	54
113	Female	38	64	42
114	Male	19	64	46
115	Female	18	65	48
116	Female	19	65	50
117	Female	63	65	43
118	Female	49	65	59
119	Female	51	67	43
120	Female	50	67	57
121	Male	27	67	56
122	Female	38	67	40
123	Female	40	69	58
124	Male	39	69	91
125	Female	23	70	29
126	Female	31	70	77
127	Male	43	71	35
128	Male	40	71	95
129	Male	59	71	11
130	Male	38	71	75

131	Male	47	71	9
132	Male	39	71	75
133	Female	25	72	34
134	Female	31	72	71
135	Male	20	73	5
136	Female	29	73	88
137	Female	44	73	7
138	Male	32	73	73
139	Male	19	74	10
140	Female	35	74	72
141	Female	57	75	5
142	Male	32	75	93
143	Female	28	76	40
144	Female	32	76	87
145	Male	25	77	12
146	Male	28	77	97
147	Male	48	77	36
148	Female	32	77	74
149	Female	34	78	22
150	Male	34	78	90
151	Male	43	78	17
152	Male	39	78	88
153	Female	44	78	20
154	Female	38	78	76
155	Female	47	78	16
156	Female	27	78	89
157	Male	37	78	1
158	Female	30	78	78
159	Male	34	78	1
160	Female	30	78	73
161	Female	56	79	35
162	Female	29	79	83
163	Male	19	81	5
164	Female	31	81	93
165	Male	50	85	26
166	Female	36	85	75
167	Male	42	86	20
168	Female	33	86	95
169	Female	36	87	27
170	Male	32	87	63
171	Male	40	87	13
172	Male	28	87	75
173	Male	36	87	10
174	Male	36	87	92
175	Female	52	88	13
176	Female	30	88	86

177	Male	58	88	15
178	Male	27	88	69
179	Male	59	93	14
180	Male	35	93	90
181	Female	37	97	32
182	Female	32	97	86
183	Male	46	98	15
184	Female	29	98	88
185	Female	41	99	39
186	Male	30	99	97
187	Female	54	101	24
188	Male	28	101	68
189	Female	41	103	17
190	Female	36	103	85
191	Female	34	103	23
192	Female	32	103	69
193	Male	33	113	8
194	Female	38	113	91
195	Female	47	120	16
196	Female	35	120	79
197	Female	45	126	28
198	Male	32	126	74
199	Male	32	137	18
200	Male	30	137	83




Clusters of customers
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5

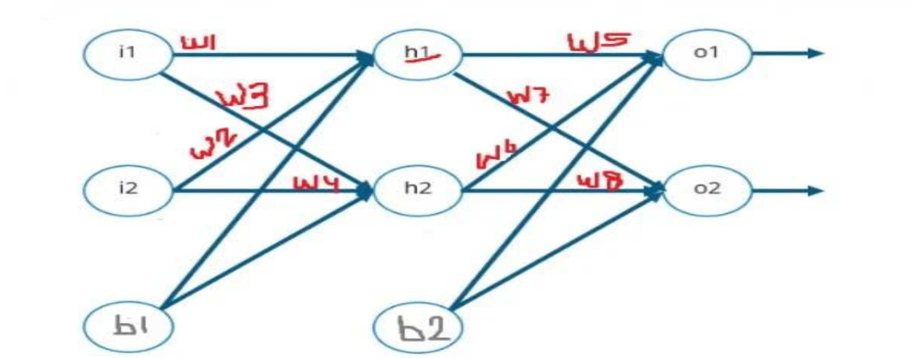
MACHINE LEARNING LAB -18IS6DLMLL

Program 10

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

Backpropagation is supervised learning algorithm , for training Neural Networks. Every node in Neural Network represent a Neuron, so we can say that Neural Network is a circuit of neurons. Neural Network consist an Input layer, an output layer and a hidden layer.

We have to learn our model to change the weights automatically so that we get least error. We first calculated the error of our model, after that we saw that if the error is minimal then our model is ready for prediction. If the error is not minimized, we will update the parameters (weights) and calculate the error again. These processes will run until the error of our model is minimized.



FORWARD PROPAGATION

1. To calculate value of h1

   net h1=w1*i1+w2*i2+b1*1

2. To calculate the output of h1

   out h1=1/1+e pow –neth1

2. To calculate error of output of h1

   Eo1=E1/2(target-output)pow2

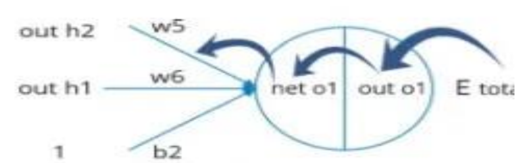
4. To calculate total error of the model

   Etotal=Eo1+Eo2

BACKWARD PROPAGATION

Here we are writing the process and formulas to update our w5 weight.

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



1. Calculating our total total error with respect to output one.

$$\frac{\delta Etotal}{\delta out\ o1} = -(\text{target } o1 - \text{out } o1)$$

2. calculating our total output 1 with respect to net output 1

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out o1 (1 - out o1)}$$

$$\text{out o1} = 1/1+e^{-neto1}$$

$$\text{net o1} = \text{w5} * \text{out h1} + \text{w6} * \text{out h2} + \text{b2} * 1$$

3. Calculate net output1 with respect to weight5

$$\frac{\delta net\ o1}{\delta w5} = 1 * \text{out h1}\ w5^{(1-1)}.$$

4. Calculating updated weight

$$\text{w5}^{+} = \text{w5} - \text{n}\frac{\delta Etotal}{\delta w5}$$

Python code

```
import numpy as np
x = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
print("small x",x)
#original output
y = np.array(([92], [86], [89]), dtype=float)
X = x/np.amax(x,axis=0) #maximum along the first axis
print("Capital X",X)
#Defining Sigmoid Function for output
def sigmoid (x):
    return (1/(1 + np.exp(-x)))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variables initialization
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of input layer neurons
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#Defining weight and biases for hidden and output layer
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#Forward Propagation
for i in range(epoch):
    hinp1=np.dot(X,wh)
hinp=hinp1 + bh
hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
outinp= outinp1+ bout
    output = sigmoid(outinp)
```

```
   #Backpropagation Algorithm
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)
#how much hidden layer wts contributed to error
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop
bout += np.sum(d_output, axis=0,keepdims=True) *lr
#Updating Weights
wh += X.T.dot(d_hiddenlayer) *lr
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Sample Output

```
small x [[2. 9.]
 [1. 5.]
 [3. 6.]]
Capital X [[0.66666667 1. ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.92928201]
 [0.92075172]
 [0.93223878]]
```