HKBK COLLEGE OF ENGINEERING
(Affiliated to VTU, Belgaum and Approved by AICTE)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NBA Accredited Programme

LABORATORY MANUAL
Machine Learning Laboratory
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2015 -2016)
15CSL76

PREPARED BY

Prof. Smitha Kurian

Prof. Priya Rathod

HKBK COLLEGE OF ENGINEERING
Nagawara, Bangaluru -560 045
www.hkbkeducation.org

Mission and Vision of the Institution

Mission

- To achieve academic excellence in science, engineering and technology through dedication to duty, innovation in teaching and faith in human values.
- To enable our students to develop into outstanding professional with high ethical standards to face the challenges of 21st century.
- To provide educational opportunities to the deprived and weaker section of the society to uplift their socio-economic status.

Vision

To empower students through wholesome education and enable the students to develop into highly qualified and trained professionals with ethics and emerge as responsible citizen with broad outlook to build a vibrant nation.

Mission and Vision of the CSE Department

Mission

- To provide excellent technical knowledge and computing skills to make the graduates globally competitive with professional ethics.
- To involve in research activities and be committed to lifelong learning to make positive contributions to the society.

Vision

To advance the intellectual capacity of the nation and the international community by imparting knowledge to graduates who are globally recognized as innovators, entrepreneur and competent professionals.

Programme Educational Objectives

PEO-1	To provide students with a strong foundation in engineering fundamentals and in the computer science and engineering to work in the global scenario.
PEO-2	To provide sound knowledge of programming and computing techniques and good communication and interpersonal skills so that they will be capable of analyzing, designing and building innovative software systems.
PEO-3	To equip students in the chosen field of engineering and related fields to enable him to work in multidisciplinary teams.
PEO-4	To inculcate in students professional, personal and ethical attitude to relate engineering issues to broader social context and become responsible citizen.
PEO-5	To provide students with an environment for life-long learning which allow them to successfully adapt to the evolving technologies throughout their professional carrier and face the global challenges.

Programme Outcomes	
a.	**Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
b.	**Problem Analysis: Identify,** formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
c.	**Design/ Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
d.	**Conduct investigations of complex problems** using research-based knowledge and research methods including design of experiments, analysis and

	interpretation of data and synthesis of information to provide valid conclusions.
e.	**Modern Tool Usage**: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an under- standing of the limitations.
f.	**The Engineer and Society**: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.
g.	**Environment and Sustainability**: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.
h.	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
i.	**Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams and in multi disciplinary settings.
j.	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
k.	**Life-long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life- long learning in the broadest context of technological change.
l.	**Project Management and Finance**: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
Programme Specific Outcomes	
m.	**Problem-Solving Skills:** An ability to investigate and solve a problem by analysis, interpretation of data, design and implementation through appropriate techniques,tools and skills.
n.	**Professional Skills**: An ability to apply algorithmic principles, computing skills and computer science theory in the modelling and design of computer-based systems.
o.	**Entrepreneurial Ability:** An ability to apply design, development principles and management skills in the construction of software product of varying complexity to become an entrepreneur

HKBK College of Engineering

Department of Computer Sciences and Engineering
Bangalore-560045
Data Structures with C laboratory (15CSL38)

Course objectives:

This course will enable students to

1. Make use of Data sets in implementing the machine learning algorithms
2. Implement the machine learning concepts and algorithms in any suitable language of choice.

S#	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 to classify 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.

Course outcomes:

The students should be able to:

1. Understand the implementation procedures for the machine learning algorithms.
2. Design Java/Python programs for various Learning algorithms.
3. Apply appropriate data sets to the Machine Learning algorithms.
4. Identify and apply Machine Learning algorithms to solve real world problems

Mapping of Course outcome to Programme Outcomes

PO / CO	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
1	2	2	2	2	2	-	-	-	-	-	3	-	3	3	-
2	2	2	2	2	2	-	-	-	-	-	3	-	3	3	-
3	2	2	2	2	2	-	-	-	-	-	3	-	3	3	-

3	High
2	Moderate
1	Low
-	Nil

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.

Algorithm:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   For each attribute constraint a, in h
   If the constraint a, is satisfied by x Then do nothing
   Else replace a, in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Code:

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

with open('ws.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: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

# Comparing with First Training Example

for j in range(0,num_attributes):
      hypothesis[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

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]='?'

   print("For training example No",(i+1),"hypothesis is", hypothesis)
print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
```

```
print(hypothesis)
```

CSV file- Tennis 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
```

Output:

```
The most general hypothesis : ['?','?','?','?','?','?']
 The most specific hypothesis : ['0','0','0','0','0','0']

 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:
['0', '0', '0', '0', '0', '0']

 Find S: Finding a Maximally Specific Hypothesis

 For Training Example No :0 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
 For Training Example No :1 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
 For Training Example No :2 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
 For Training Example No :3 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

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

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.

Algorithm:

Algorithm Candidate elimination

1) Initialize G to the set of maximally general hypotheses in H
2) Initialize S to the set of maximally specific hypotheses in H
3) For each training example d, do
4) If d is a positive example
    a. Remove from G any hypothesis inconsistent with d ,
        i. For each hypothesis **s** in S that is not consistent with d
            1. Remove **s** from S
            2. Add to S all minimal generalizations h of **s** such that
                a. h is consistent with d, and some member of G is more general than h
        ii. Remove from S any hypothesis that is more general than another hypothesis in S
5) If d is a negative example
    a. Remove from S any hypothesis inconsistent with d
    b. For each hypothesis g in G that is not consistent with d
        i. Remove g from G
        ii. Add to G all minimal specializations h of g such that
            1. h is consistent with d, and some member of **S** is more specific than h
        iii. Remove from G any hypothesis that is less general than another hypothesis in G

Code:

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

with open('ws.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)):
   print("-------------------------------------------------------------------------------")
   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):
          if S[j] != a[i][j] and S[j]!= '?':
             G[j]=S[j]
             temp.append(G)
             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)
```

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', '?', '?', '?', '?']]

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

Algorithm:

ID3(Examples, Targetattribute, 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 learneddecision tree. Returns a decision tree that correctly classifies the given Examples**.**

1) Create a **Root** node for the tree
2) **If** all **Examples** are positive, Return the single-node tree **Root,** with label = +
3) **If** all **Examples** are negative, Return the single-node tree **Root,** with label = -
4) **If Attributes** is empty, Return the single-node tree **Root,** with label = most common value of **Target_attribute** in **Examples**
5) Otherwise Begin
   a. **A** ←the attribute from **Attributes** that best classifies **Examples(**The best attribute is the one with highest ***information gain,*** as defined in Equation below)

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

   i.

b. The decision attribute for **Root ←A**
c. For each possible value, **vi,** of **A,**
i. Add a new tree branch below **Root,** corresponding to the test **A** = **vi**
ii. Let **Examples$_{Vi}$**be the subset of **Examples** that have value **vi** for **A**
iii. If **Examples$_{Vi}$**is empty
1. Then below this new branch add a leaf node with label = most common value of **Target attribute** in **Examples**
2. Else below this new branch add the subtree
a. ID3(Examples$_{Vi}$ Targetattribute, Attributes - (A)))

6) End
7) Return **Root**

Code:

```
import numpy as np
import math
import csv

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute

def read_data(filename):

    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col]) # get unique values in particular column

    count = np.zeros((items.shape[0], 1), dtype=np.int32)   #number of row = number of values

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

```
            if data[y, col] == items[x]:
                count[x] += 1
    #count has the data of number of times each value is present in

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1

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

def entropy(S):
    items = np.unique(S)
    if items.size == 1:
        return 0

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

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

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



    for count in counts:
        sums += -1 * count * math.log(count, 2)

    return sums

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)
    #item is the unique value and dict is the data corresponding to it
    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
```

```
    total_entropy = entropy(data[:, -1])


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

    return total_entropy

def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])
        return node

    gains = np.zeros((data.shape[1] - 1, 1))
    #size of gains= number of attribute to calculate gain


    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)


    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)


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

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node

def empty(size):
    s = ""
    for x in range(size):
        s += "   "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
```

```
    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)

metadata, traindata = read_data("tennis.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

Output:

```
outlook
   overcast
      [b'yes']
   rainy
      windy
         b'Strong'
            [b'no']
         b'Weak'
            [b'yes']
   sunny
      humidity
         b'high'
            [b'no']
         b'normal'
            [b'yes']
```

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

Algorithm:

BACKPROPOGATION (training_examples,η,n_{in},n_{out},n_{hidden})

Each training example is a pair of the form (x,t), where x is the vector of network input values, and t is the vector of target network output values.

η is the learning rate (e.g., .O5). 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 xji, and the weight from unit i to unit j is denoted wji.

1. Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
2. Initialize all network weights to small random numbers
3. Until the termination condition is met, Do
    - For each ***(x,t***) in *trainingaxamples,* **Do**
        - *Propagate the input forward through the network:*
            1. Input the instance x to the network and compute the output *O,* 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 *δk*

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

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

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

            4. Update each network weight ***wji***

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

            Where

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji}$$

Code:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
y = y/100
#Sigmoid Function
def sigmoid (x):
   return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
   return x * (1 - x)
#Variable initialization
```

```
epoch=10000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and currentlayerop
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

Input:

```
[[2. 9.]
 [1. 5.]
 [3. 6.]]
```

Actual Output:

```
[[0.92]
 [0.86]
 [0.89]]
```

Predicted Output:

```
 [[0.89345619]
 [0.87813113]
 [0.89718075]]
```

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

Algorithm:
NaiveBaiseClassifier(training_examples, New_Instance)

Each instance ***x*** is described by a conjunction of attribute values(a_i) and the target V can take j finite set of values.

a. For each value j in target estimate the P(Vj)
b. For each attribute in the training example estimate Estimate the P(ai|Vj)
c. Classify each instance as per the rule in equation

$$v_{NB} = \underset{v_j \in V}{\text{argmax}}\ P(v_j) \prod_i P(a_i|v_j)$$

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

d. Output V_{NB}

Code:

```
import numpy as np
import math
import csv

def read_data(filename):

    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

def splitDataset(dataset, splitRatio):        #splits dataset to training set and test set based on split ratio
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) < trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]

def classify(data,test):

    total_size = data.shape[0]
    print("training data size=",total_size)
```

```
print("test data sixe=",test.shape[0])
target=np.unique(data[:,-1])
count = np.zeros((target.shape[0]), dtype=np.int32)
prob = np.zeros((target.shape[0]), dtype=np.float32)
print("target   count  probability")
for y in range(target.shape[0]):
    for x in range(data.shape[0]):
        if data[x,data.shape[1]-1] == target[y]:
            count[y] += 1
    prob[y]=count[y]/total_size                                  # comptes the probability of target
    print(target[y],"\t",count[y],"\t",prob[y])

prob0 = np.zeros((test.shape[1]-1), dtype=np.float32)
prob1 = np.zeros((test.shape[1]-1), dtype=np.float32)
accuracy=0
print("Instance prediction taget")
for t in range(test.shape[0]):
    for k in range(test.shape[1]-1):       # for each attribute in column
        count1=count0=0
        for j in range(data.shape[0]):
            if test[t,k]== data[j,k] and data[j,data.shape[1]-1]== target[0]:
                count0+=1
            elif test[t,k]== data[j,k] and data[j,data.shape[1]-1]== target[1]:
                count1+=1
        prob0[k]= count0/count[0]                                #Find no probability of each attribute
        prob1[k]= count1/count[1]                                #Find yes probability of each attribute


    probno=prob[0]
    probyes=prob[1]
    for i in range(test.shape[1]-1):
         probno=probno*prob0[i]
         probyes=probyes*prob1[i]


    if probno>probyes:     # prediction
        predict='no'
    else:
        predict='yes'
    print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])


    if predict== test[t,test.shape[1]-1]:                        # computing accuracy
       accuracy+=1

final_accuracy=(accuracy/test.shape[0])*100
print("accuracy",final_accuracy,"%")
```

```
    return

metadata, traindata = read_data("tennis.csv")
splitRatio = 0.6
trainingset, testset = splitDataset(traindata, splitRatio)
training=np.array(trainingset)
testing=np.array(testset)
print("------------------Training Data--------------------")
print(trainingset)
print("-------------------Test Data--------------------")
print(testset)

classify(training,testing)
```

Output:

```
------------------Training Data--------------------
[['sunny', 'hot', 'high', 'Weak', 'no'], ['sunny', 'hot', 'high', 'Strong', 'no'], ['overcast', 'hot', 'high', 'Weak',
'yes'], ['rainy', 'mild', 'high', 'Weak', 'yes'], ['rainy', 'cool', 'normal', 'Weak', 'yes'], ['rainy', 'cool',
'normal', 'Strong', 'no'], ['overcast', 'cool', 'normal', 'Strong', 'yes'], ['sunny', 'mild', 'high', 'Weak', 'no']]
-------------------Test Data--------------------
[['sunny', 'cool', 'normal', 'Weak', 'yes'], ['rainy', 'mild', 'normal', 'Weak', 'yes'], ['sunny', 'mild',
'normal', 'Strong', 'yes'], ['overcast', 'mild', 'high', 'Strong', 'yes'], ['overcast', 'hot', 'normal', 'Weak',
'yes'], ['rainy', 'mild', 'high', 'Strong', 'no']]
training data size= 8
test data sixe= 6
['no' 'yes']
target   count  probability
no          4        0.5
yes         4        0.5
Instance prediction taget
1           no          yes
2           yes         yes
3           no          yes
4           yes         yes
5           yes         yes
6           no          no
accuracy 66.66666666666666 %
```

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.

```
import pandas as pd
msg=pd.read_csv('naivetext1.csv',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
#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)

#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print("the vocabulary")
print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy metrics
from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

Output:

```
The dimensions of the dataset (18, 2)
the vocabulary
['about', 'am', 'amazing', 'an', 'and', 'beers', 'boss', 'can', 'dance', 'deal', 'do', 'enemy', 'feel', 'fun', 'good',
'great', 'have', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant',
'sandwich', 'sick', 'stuff', 'taste', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'we', 'went',
'what', 'will', 'with']
Accuracy metrics
Accuracy of the classifer is 0.6
Confusion matrix
[[1 1]
 [1 2]]
Recall and Precison
0.6666666666666666
0.6666666666666666
```

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.

```
import pandas as pd
import bayespy as bp
#import csv

data=pd.read_csv("heart_disease_data1.csv")
heart_disease=pd.DataFrame(data)
print(heart_disease)

from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator
model=BayesianModel([('age','Lifestyle'),('Gender','Lifestyle'),('Family','heartdisease'),('diet','cholestrol'),
('Lifestyle','diet'),('cholestrol','heartdisease'),('diet','cholestrol')])
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)
print('For age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4')
print('For Gender Enter Male:0, Female:1')
print('For Family History Enter yes:1, No:0')
print('For diet Enter High:0, Medium:1')
print('for lifeStyle Enter Athlete:0, Active:1, Moderate:2, Sedetary:3')
print('for cholesterol Enter High:0, BorderLine:1, Normal:2')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age':int(input('enter
age')),'Gender':int(input('enter Gender')),'Family':int(input('enter Family history')),'diet':int(input('enter
diet')),'Lifestyle':int(input('enter Lifestyle')),'cholestrol':int(input('enter cholestrol'))})
print(q['heartdisease'])
```

Output

```
For age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4
For Gender Enter Male:0, Female:1
For Family History Enter yes:1, No:0
For diet Enter High:0, Medium:1
for lifeStyle Enter Athlete:0, Active:1, Moderate:2, Sedetary:3
for cholesterol Enter High:0, BorderLine:1, Normal:2
enter age1
enter Gender0
enter Family history1
enter diet1
enter Lifestyle2
enter cholestrol1
```

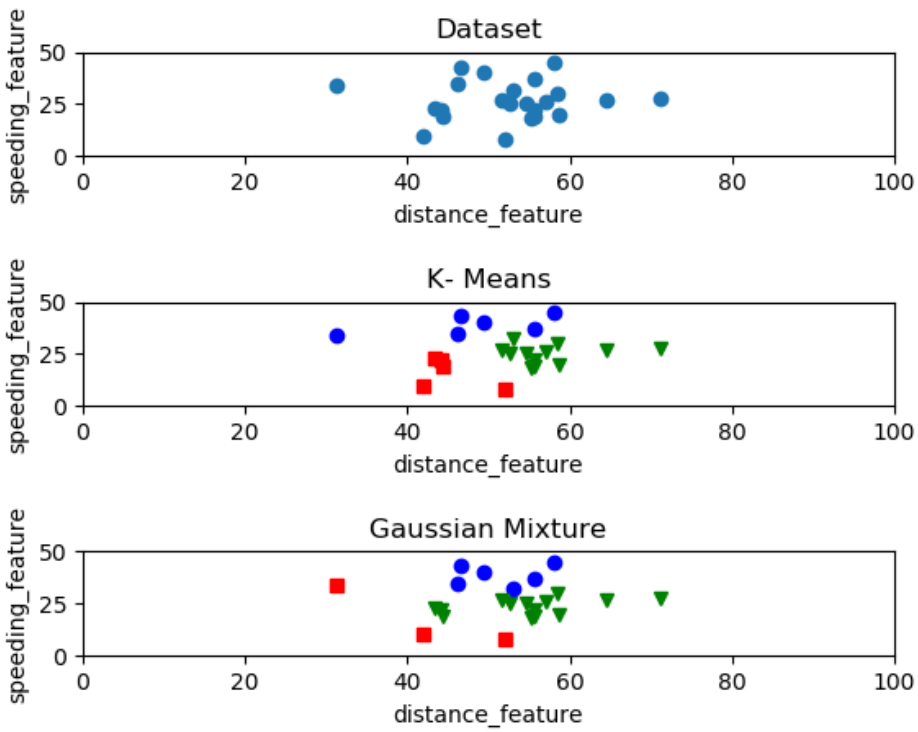
heartdisease	phi(heartdisease)
heartdisease_0	1.0000
heartdisease_1	0.0000

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 matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

data=pd.read_csv("clusterdata.csv")
df1=pd.DataFrame(data)
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values

X=np.matrix(list(zip(f1,f2)))
plt.plot(1)
plt.subplot(511)
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
plt.scatter(f1,f2)

colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# create new plot and data for K- means algorithm
plt.plot(2)
ax=plt.subplot(513)
kmeans_model = KMeans(n_clusters=3).fit(X)

for i, l in enumerate(kmeans_model.labels_):
    plt.plot(f1[i], f2[i], color=colors[l],marker=markers[l])
```

```
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('K- Means')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')


# create new plot and data for gaussian mixture
plt.plot(3)
plt.subplot(515)
gmm=GaussianMixture(n_components=3).fit(X)
labels= gmm.predict(X)
for i, l in enumerate(labels):
    plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l])
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Gaussian Mixture')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')


plt.show()
```

Output



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

```
 from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,confusion_matrix

from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.30)

classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
print('Confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Matrics')
print(classification_report(y_test,y_pred))
```

Output

```
Confusion matrix is as follows
[[18  0  0]
 [ 0 11  1]
 [ 0  0 15]]
Accuracy Matrics
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        18
          1       1.00      0.92      0.96        12
          2       0.94      1.00      0.97        15

avg / total       0.98      0.98      0.98        45
```

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

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1

def kernel(point,xmat, k):
    m,n= np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('tips1.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)  # mat treats array as matrix
mtip = np1.mat(tip)
m= np1.shape(mbill)[1]
print("******",m)
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T))    #Stack arrays in sequence horizontally (column wise).

#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=1)
plt.xlabel('Total bill')
plt.ylabel('Tip')
#plt.show();
```

Output: