Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BE - VII SEMESTER

MACHINE LEARNING LABORATORY MANUAL -15CSL76

ACADEMIC YEAR – 2018-19

PROGRAMME OUTCOMES(PO's)

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

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 understanding of the limitations.

The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

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.

Machine learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.

Machine learning tasks

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback:

Semi-supervised learning: the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

Active learning: the computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling.

Reinforcement learning: training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

MACHINE LEARNING LABORATORY
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2016 -2017)
SEMESTER – VII

Subject Code	15CSL76	IA Marks	20
Number of Lecture Hours/Week	01I + 02P	Exam Marks	80
Total Number of Lecture Hours	40	Exam Hours	03

CREDITS – 02

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.

Description (If any):

1. The programs can be implemented in either JAVA or Python.
2. For Problems 1 to 6 and 10, programs are to be developed without using the built-in classes or APIs of Java/Python.
3. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students.

Lab Experiments:

1. Implement and demonstratethe **FIND-Salgorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm**to output a description of the set of all hypotheses consistent with the training examples.
3. Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.
4. Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.
5. Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
6. Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
7. Write a program to construct a**Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.
8. Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
9. Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
10. Implement the non-parametric **Locally Weighted Regressionalgorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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.

```
import csv

with open('tennis.csv', 'r') as f:
    reader = csv.reader(f)
    your_list = list(reader)

h = [['0', '0', '0', '0', '0', '0']]

for i in your_list:
    print(i)
    if i[-1] == "True":
        j = 0
        for x in i:
            if x != "True":
                if x != h[0][j] and h[0][j] == '0':
                    h[0][j] = x
                elif x != h[0][j] and h[0][j] != '0':
                    h[0][j] = '?'
                else:
                    pass
            j = j + 1
print("Most specific hypothesis is")
print(h)
```

Output

'Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same',True
'Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same',True
'Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change',False
'Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change',True

Maximally Specific set
[['Sunny', 'Warm', '?', 'Strong', '?', '?']]

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

```
class Holder:
    factors={} #Initialize an empty dictionary
    attributes = () #declaration of dictionaries parameters with an arbitrary length

    '''
    Constructor of class Holder holding two parameters,
    self refers to the instance of the class
    '''
    def __init__(self,attr): #
        self.attributes = attr
        for i in attr:
            self.factors[i]=[]

    def add_values(self,factor,values):
        self.factors[factor]=values

class CandidateElimination:
    Positive={} #Initialize positive empty dictionary
    Negative={} #Initialize negative empty dictionary

    def __init__(self,data,fact):
        self.num_factors = len(data[0][0])
        self.factors = fact.factors
        self.attr = fact.attributes
        self.dataset = data

    def run_algorithm(self):
        '''
        Initialize the specific and general boundaries, and loop the dataset against the
algorithm
        '''
        G = self.initializeG()
        S = self.initializeS()

        '''
        Programmatically populate list in the iterating variable trial_set
        '''
        count=0
        for trial_set in self.dataset:
            if self.is_positive(trial_set): #if trial set/example consists of positive examples
                G = self.remove_inconsistent_G(G,trial_set[0]) #remove inconsitent data from
the general boundary
```

```
                S_new = S[:] #initialize the dictionary with no key-value pair
                print (S_new)
                for s in S:
                    if not self.consistent(s,trial_set[0]):
                        S_new.remove(s)
                        generalization = self.generalize_inconsistent_S(s,trial_set[0])
                        generalization = self.get_general(generalization,G)
                        if generalization:
                            S_new.append(generalization)
                    S = S_new[:]
                    S = self.remove_more_general(S)
                print(S)

            else:#if it is negative

                S = self.remove_inconsistent_S(S,trial_set[0]) #remove inconsitent data from
the specific boundary
                G_new = G[:] #initialize the dictionary with no key-value pair (dataset can
take any value)
                print (G_new)
                for g in G:
                    if self.consistent(g,trial_set[0]):
                        G_new.remove(g)
                        specializations = self.specialize_inconsistent_G(g,trial_set[0])
                        specializationss = self.get_specific(specializations,S)
                        if specializations != []:
                            G_new += specializationss
                    G = G_new[:]
                    G = self.remove_more_specific(G)
                print(G)

        print (S)
        print (G)

    def initializeS(self):
        ''' Initialize the specific boundary '''
        S = tuple(['-' for factor in range(self.num_factors)]) #6 constraints in the vector
        return [S]

    def initializeG(self):
        ''' Initialize the general boundary '''
        G = tuple(['?' for factor in range(self.num_factors)]) # 6 constraints in the vector
        return [G]

    def is_positive(self,trial_set):
        ''' Check if a given training trial_set is positive '''
        if trial_set[1] == 'Y':
```

```
            return True
        elif trial_set[1] == 'N':
            return False
        else:
            raise TypeError("invalid target value")

    def match_factor(self,value1,value2):
        ''' Check for the factors values match,
            necessary while checking the consistency of
            training trial_set with the hypothesis '''
        if value1 == '?' or value2 == '?':
            return True
        elif value1 == value2 :
            return True
        return False

    def consistent(self,hypothesis,instance):
        ''' Check whether the instance is part of the hypothesis '''
        for i,factor in enumerate(hypothesis):
            if not self.match_factor(factor,instance[i]):
                return False
        return True

    def remove_inconsistent_G(self,hypotheses,instance):
        ''' For a positive trial_set, the hypotheses in G
            inconsistent with it should be removed '''
        G_new = hypotheses[:]

        for g in hypotheses:
            if not self.consistent(g,instance):
                G_new.remove(g)
        return G_new

    def remove_inconsistent_S(self,hypotheses,instance):
        ''' For a negative trial_set, the hypotheses in S
            inconsistent with it should be removed '''
        S_new = hypotheses[:]
        for s in hypotheses:
            if self.consistent(s,instance):
                S_new.remove(s)
        return S_new

    def remove_more_general(self,hypotheses):
        ''' After generalizing S for a positive trial_set, the hypothesis in S
         general than others in S should be removed '''
        S_new = hypotheses[:]
        for old in hypotheses:
```

```
            for new in S_new:
                if old!=new and self.more_general(new,old):
                    S_new.remove[new]
        return S_new

    def remove_more_specific(self,hypotheses):
        ''' After specializing G for a negative trial_set, the hypothesis in G
        specific than others in G should be removed '''
        G_new = hypotheses[:]
        for old in hypotheses:
            for new in G_new:
                if old!=new and self.more_specific(new,old):
                    G_new.remove[new]
        return G_new


    def generalize_inconsistent_S(self,hypothesis,instance):
        ''' When a inconsistent hypothesis for positive trial_set is seen in the specific
boundary S,
            it should be generalized to be consistent with the trial_set ... we will get one
hypothesis'''
        hypo = list(hypothesis) # convert tuple to list for mutability
        for i,factor in enumerate(hypo):
            if factor == '-':
                hypo[i] = instance[i]
            elif not self.match_factor(factor,instance[i]):
                hypo[i] = '?'
        generalization = tuple(hypo) # convert list back to tuple for immutability
        return generalization

    def specialize_inconsistent_G(self,hypothesis,instance):
        ''' When a inconsistent hypothesis for negative trial_set is seen in the general
boundary G
            should be specialized to be consistent with the trial_set.. we will get a set of
hypotheses '''
        specializations = []
        hypo = list(hypothesis) # convert tuple to list for mutability
        for i,factor in enumerate(hypo):
            if factor == '?':
                values = self.factors[self.attr[i]]
                for j in values:
                    if instance[i] != j:
                        hyp=hypo[:]
                        hyp[i]=j
                        hyp=tuple(hyp) # convert list back to tuple for immutability
                        specializations.append(hyp)
        return specializations
```

```
def get_general(self,generalization,G):
    ''' Checks if there is more general hypothesis in G
        for a generalization of inconsistent hypothesis in S
        in case of positive trial_set and returns valid generalization '''

    for g in G:
        if self.more_general(g,generalization):
            return generalization
    return None

def get_specific(self,specializations,S):
    ''' Checks if there is more specific hypothesis in S
        for each of hypothesis in specializations of an
        inconsistent hypothesis in G in case of negative trial_set
        and return the valid specializations'''
    valid_specializations = []
    for hypo in specializations:
        for s in S:
            if self.more_specific(s,hypo) or s==self.initializeS()[0]:
                valid_specializations.append(hypo)
    return valid_specializations


def exists_general(self,hypothesis,G):
    '''Used to check if there exists a more general hypothesis in
        general boundary for version space'''

    for g in G:
        if self.more_general(g,hypothesis):
            return True
    return False

def exists_specific(self,hypothesis,S):
    '''Used to check if there exists a more specific hypothesis in
        general boundary for version space'''

    for s in S:
        if self.more_specific(s,hypothesis):
            return True
    return False

def more_general(self,hyp1,hyp2):
    ''' Check whether hyp1 is more general than hyp2 '''
    hyp = zip(hyp1,hyp2)
    for i,j in hyp:
        if i == '?':
            continue
```

```
            elif j == '?':
                if i != '?':
                    return False
            elif i != j:
                return False
            else:
                continue
        return True

    def more_specific(self,hyp1,hyp2):
        ''' hyp1 more specific than hyp2 is
            equivalent to hyp2 being more general than hyp1 '''
        return self.more_general(hyp2,hyp1)

dataset=[(('sunny','warm','normal','strong','warm','same'),'Y'),(('sunny','warm','high','strong','warm','same'),'Y'),(('rainy','cold','high','strong','warm','change'),'N'),(('sunny','warm','high','strong','cool','change'),'Y')]
attributes =('Sky','Temp','Humidity','Wind','Water','Forecast')
f = Holder(attributes)
f.add_values('Sky',('sunny','rainy','cloudy')) #sky can be sunny rainy or cloudy
f.add_values('Temp',('cold','warm')) #Temp can be sunny cold or warm
f.add_values('Humidity',('normal','high')) #Humidity can be normal or high
f.add_values('Wind',('weak','strong')) #wind can be weak or strong
f.add_values('Water',('warm','cold')) #water can be warm or cold
f.add_values('Forecast',('same','change')) #Forecast can be same or change
a = CandidateElimination(dataset,f) #pass the dataset to the algorithm class and call the run algoritm method
a.run_algorithm()
```

Output

```
[('sunny', 'warm', 'normal', 'strong', 'warm', 'same')]
[('sunny', 'warm', 'normal', 'strong', 'warm', 'same')]
[('sunny', 'warm', '?', 'strong', 'warm', 'same')]
[('?', '?', '?', '?', '?', '?')]
[('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')]
[('sunny', 'warm', '?', 'strong', 'warm', 'same')]
[('sunny', 'warm', '?', 'strong', '?', '?')]
[('sunny', 'warm', '?', 'strong', '?', '?')]
[('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 toclassify a new sample.

```
import numpy as np
import math
from data_loader import read_data

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

    def __str__(self):
        return self.attribute

def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])

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

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

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


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

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

    return items, dict

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

```
        return 0

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

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

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


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

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))
    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

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

    return total_entropy / iv

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

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

    split = np.argmax(gains)


    node = Node(metadata[split])
```

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

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

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

    return node

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

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

    print(empty(level), node.attribute)

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


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

Data_loader.py

```
import csv
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
```

Tennis.csv

```
outlook,temperature,humidity,wind,answer
sunny,hot,high,weak,no
sunny,hot,high,strong,no
overcast,hot,high,weak,yes
rain,mild,high,weak,yes
rain,cool,normal,weak,yes
rain,cool,normal,strong,no
overcast,cool,normal,strong,yes
sunny,mild,high,weak,no
sunny,cool,normal,weak,yes
rain,mild,normal,weak,yes
sunny,mild,normal,strong,yes
overcast,mild,high,strong,yes
overcast,hot,normal,weak,yes
rain,mild,high,strong,no
```

Output

```
outlook
  overcast
    b'yes'
  rain
    wind
      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.

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):

#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts
contributed to error
```

```
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and
currentlayerop
   # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

output

```
Input:
[[ 0.66666667 1.   ]
 [ 0.33333333 0.55555556]
 [ 1.     0.66666667]]
Actual Output:
[[ 0.92]
 [ 0.86]
 [ 0.89]]
Predicted Output:
 [[ 0.89559591]
 [ 0.88142069]
 [ 0.8928407 ]]
```

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.

```
import csv
import random
import math

def loadCsv(filename):
  lines = csv.reader(open(filename, "r"));
  dataset = list(lines)
  for i in range(len(dataset)):
       #converting strings into numbers for processing
            dataset[i] = [float(x) for x in dataset[i]]

  return dataset

def splitDataset(dataset, splitRatio):
    #67% training size
  trainSize = int(len(dataset) * splitRatio);
  trainSet = []
  copy = list(dataset);
  while len(trainSet) < trainSize:
#generate indices for the dataset list randomly to pick ele for training data
            index = random.randrange(len(copy));
            trainSet.append(copy.pop(index))
  return [trainSet, copy]
def separateByClass(dataset):
  separated = {}
#creates a dictionary of classes 1 and 0 where the values are the instacnes belonging to
each class
  for i in range(len(dataset)):
            vector = dataset[i]
            if (vector[-1] not in separated):
                        separated[vector[-1]] = []
            separated[vector[-1]].append(vector)
  return separated

def mean(numbers):
  return sum(numbers)/float(len(numbers))

def stdev(numbers):
  avg = mean(numbers)
  variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
  return math.sqrt(variance)
```

```
def summarize(dataset):
  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
  del summaries[-1]
  return summaries

def summarizeByClass(dataset):
  separated = separateByClass(dataset);
  summaries = {}
  for classValue, instances in separated.items():
#summaries is a dic of tuples(mean,std) for each class value
          summaries[classValue] = summarize(instances)
  return summaries

def calculateProbability(x, mean, stdev):
  exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
  return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
  probabilities = {}
  for classValue, classSummaries in summaries.items():#class and attribute information
as mean and sd
          probabilities[classValue] = 1
          for i in range(len(classSummaries)):
                  mean, stdev = classSummaries[i] #take mean and sd of every attribute
for class 0 and 1 seperaely
                  x = inputVector[i] #testvector's first attribute
                  probabilities[classValue] *= calculateProbability(x, mean, stdev);#use
normal dist
  return probabilities

def predict(summaries, inputVector):
  probabilities = calculateClassProbabilities(summaries, inputVector)
  bestLabel, bestProb = None, -1
  for classValue, probability in probabilities.items():#assigns that class which has he
highest prob
          if bestLabel is None or probability > bestProb:
                  bestProb = probability
                  bestLabel = classValue
  return bestLabel

def getPredictions(summaries, testSet):
  predictions = []
  for i in range(len(testSet)):
          result = predict(summaries, testSet[i])
          predictions.append(result)
  return predictions
```

```
def getAccuracy(testSet, predictions):
  correct = 0
  for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
              correct += 1
  return (correct/float(len(testSet))) * 100.0

def main():
  filename = '5data.csv'
  splitRatio = 0.67
  dataset = loadCsv(filename);

  trainingSet, testSet = splitDataset(dataset, splitRatio)
  print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingSet), len(testSet)))
  # prepare model
  summaries = summarizeByClass(trainingSet);
  # test model
  predictions = getPredictions(summaries, testSet)
  accuracy = getAccuracy(testSet, predictions)
  print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

Output

```
confusion matrix is as follows
[[17 0 0]
 [ 0 17 0]
 [ 0 0 11]]
Accuracy metrics
             precision recall f1-score support

          0      1.00      1.00      1.00       17
          1      1.00      1.00      1.00       17
          2      1.00      1.00      1.00       11

avg / total      1.00      1.00      1.00       45
```

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
print(X)
print(y)

#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)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)
#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(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df)#tabular representation
print(xtrain_dtm) #sparse matrix representation

# 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))

'''docs_new = ['I like this place', 'My boss is not my saviour']
```

```
X_new_counts = count_vect.transform(docs_new)
predictednew = clf.predict(X_new_counts)
for doc, category in zip(docs_new, predictednew):
   print('%s->%s' % (doc, msg.labelnum[category]))'''
```

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

OUTPUT

```
['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'boss', 'can', 'deal',
'do', 'enemy', 'feel', 'fun', 'good', 'have', 'horrible', 'house', 'is', 'like', 'love', 'my',
'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'stuff', 'these', 'this', 'tired', 'to',
'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']
    about am amazing an and awesome beers best boss can ... today \
0   1 0   0 0 0   0   1  0  0 0 ...   0
1   0 0   0 0 0   0   0  1  0 0 ...   0
2      0 0   1 1 0   0   0  0  0 0 ...   0
3   0 0   0 0 0   0   0  0  0 0 ...   1
4   0 0   0 0 0   0   0  0  0 0 ...   0
5   0 1            0 0 1    0   0  0  0 0 ...   0
6   0 0   0 0 0   0   0  0  0 1 ...   0
7   0 0   0 0 0   0   0  0  0 0 ...   0
8   0 1   0 0 0   0   0  0  0 0 ...   0
9   0 0   0 1 0   1   0  0  0 0 ...   0
10   0 0   0 0 0   0   0  0  0 0 ...   0
11   0 0           0 0 0    0   0  0  1 0 ...   0
12   0 0   0 1 0   1   0  0  0 0 ...   0

    tomorrow very    view we went    what will with work
0         0  1  0        0  0 0           0     0  0
```

1	0 0 0	0 0 0 0	0 1
2	0 0 0	0 0 0 0	0 0
3	0 0 0	0 1 0 0	0 0
4	0 0 0	0 0 0 0	0 0
5	0 0 0	0 0 0 0	0 0
6	0 0 0	0 0 0 0	1 0
7	1 0 0	1 0 0 1	0 0
8	0 0 0	0 0 0 0	0 0

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

```
From pomegranate import*
Asia=DiscreteDistribution({ „True‟:0.5, „False‟:0.5 })
Tuberculosis=ConditionalProbabilityTable(
[[ „True‟, „True‟, 0.2],
[„True‟, „False‟, 0.8],
[ „False‟, „True‟, 0.01],
[ „False‟, „False‟, 0.98]], [asia])

Smoking = DiscreteDistribution({ „True‟:0.5, „False‟:0.5 })
Lung = ConditionalProbabilityTable(
[[ „True‟, „True‟, 0.75],
[„True‟, „False‟,0.25].
[ „False‟, „True‟, 0.02],
[ „False‟, „False‟, 0.98]], [ smoking])

Bronchitis = ConditionalProbabilityTable(
[[ „True‟, „True‟, 0.92],
[„True‟, „False‟,0.08].
[ „False‟, „True‟,0.03],
[ „False‟, „False‟, 0.98]], [ smoking])

Tuberculosis_or_cancer = ConditionalProbabilityTable(
 [[ „True‟, „True‟, „True‟, 1.0],
[„True‟, „True‟, „False‟, 0.0],
[„True‟, „False‟, „True‟, 1.0],
[„True‟, „False‟, „False‟, 0.0],
[„False‟, „True‟, „True‟, 1.0],
[„False‟, „True‟, „False‟, 0.0],
[„False‟, „False‟ „True‟, 1.0],
[„False‟, „False‟, „False‟, 0.0]], [tuberculosis, lung])

Xray = ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.885],
[„True‟, „False‟, 0.115],
[ „False‟, „True‟, 0.04],
```

```
[ „False‟, „False‟, 0.96]], [tuberculosis_or_cancer])
dyspnea = ConditionalProbabilityTable(
[[ „True‟, „True‟, „True‟, 0.96],
[„True‟, „True‟, „False‟, 0.04],
[„True‟, „False‟, „True‟, 0.89],
[„True‟, „False‟, „False‟, 0.11],
[„False‟, „True‟, „True‟, 0.96],
[„False‟, „True‟, „False‟, 0.04],
[„False‟, „False‟ „True‟, 0.89],
[„False‟, „False‟, „False‟, 0.11 ]], [tuberculosis_or_cancer, bronchitis])
s0 = State(asia, name=”asia”)
s1 = State(tuberculosis, name=” tuberculosis”)
s2 = State(smoking, name=” smoker”)

network = BayesianNetwork(“asia”)
network.add_nodes(s0,s1,s2)
network.add_edge(s0,s1)
network.add_edge(s1.s2)
network.bake()
print(network.predict_probal({„tuberculosis‟: „True‟}))
```

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
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=100, centers =
4,Cluster_std=0.60,random_state=0)
X = X[:, ::-1]

#flip axes for better plotting
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture (n_components = 4).fit(X)
lables = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap=‟viridis‟);
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
size = 50 * probs.max(1) ** 2 # square emphasizes differences
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap=‟viridis‟, s=size);


from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs);
        “””Draw an ellipse with a given position and covariance”””
Ax = ax or plt.gca()
# Convert covariance to principal axes
if covariance.shape ==(2,2):

  U, s, Vt = np.linalg.svd(covariance)
  Angle = np.degrees(np.arctan2(U[1, 0], U[0,0]))
  Width, height = 2 * np.sqrt(s)
else:
     angle = 0
     width, height = 2 * np.sqrt(covariance)

#Draw the Ellipse
for nsig in range(1,4):
    ax.add_patch(Ellipse(position, nsig * width, nsig *height,
                       angle, **kwargs))

 def plot_gmm(gmm, X, label=True, ax=None):
   ax = ax or plt.gca()
   labels = gmm.fit(X).predict(X)
   if label:
```

```
        ax.scatter(X[:, 0], x[:, 1], c=labels, s=40, cmap="viridis", zorder=2)
    else:
        ax.scatter(X[:, 0], x[:, 1], s=40, zorder=2)
    ax.axis(„equal")

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
      draw_ellipse(pos, covar, alpha=w * w_factor)

    gmm = GaussianMixture(n_components=4, random_state=42)
    plot_gmm(gmm, X)
    gmm = GaussianMixture(n_components=4, covariance_type="full",
                    random_state=42)
    plot_gmm(gmm, X)
```

Output

```
[[1 ,0, 0, 0]
 [0 ,0, 1, 0]
 [1 ,0, 0, 0]
 [1 ,0, 0, 0]
 [1 ,0, 0, 0]]
```

K-means

```
from sklearn.cluster import KMeans

#from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv("kmeansdata.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()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()

# create new plot and data
plt.plot()
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
#K = 3
kmeans_model = KMeans(n_clusters=3).fit(X)

plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')
    plt.xlim([0, 100])
    plt.ylim([0, 50])

plt.show()
```

Driver_ID,Distance_Feature,Speeding_Feature

```
3423311935,71.24,28
3423313212,52.53,25
3423313724,64.54,27
3423311373,55.69,22
3423310999,54.58,25
```

3423313857,41.91,10
3423312432,58.64,20
3423311434,52.02,8
3423311328,31.25,34
3423312488,44.31,19
3423311254,49.35,40
3423312943,58.07,45
3423312536,44.22,22
3423311542,55.73,19
3423312176,46.63,43
3423314176,52.97,32
3423314202,46.25,35
3423311346,51.55,27
3423310666,57.05,26
3423313527,58.45,30
3423312182,43.42,23
3423313590,55.68,37
3423312268,55.15,18

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.

```
import csv
import random
import math
import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]):
    with open(filename, 'rb') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])


def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
```

```
    sortedVotes = sorted(classVotes.iteritems(),
reverse=True)
    return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
    key=operator.itemgetter(1),
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

def main():
    # prepare data
    trainingSet=[]
    testSet=[]
    split = 0.67
    loadDataset('knndat.data', split, trainingSet, testSet)
    print('Train set: ' + repr(len(trainingSet)))
    print('Test set: ' + repr(len(testSet)))
    # generate predictions
    predictions=[]
    k=3
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: ' + repr(accuracy) + '%')

main()
```

OUTPUT

Confusion matrix is as follows

```
[[11 0 0]
 [0 9 1]
 [0  1 8]]
```

Accuracy metrics

0 1.00 1.00 1.00 11

1 0.90 0.90 0.90 10

2 0.89 0.89 0,89 9
Avg/Total 0.93 0.93 0.93 30

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

```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

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

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

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

# load data points
data = pd.read_csv('data10.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
m= np1.shape(mbill)[1]
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,2)
```

```
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

Output

Viva Questions

1. What is machine learning?
2. Define supervised learning
3. Define unsupervised learning
4. Define semi supervised learning
5. Define reinforcement learning
6. What do you mean by hypotheses
7. What is classification
8. What is clustering
9. Define precision, accuracy and recall
10. Define entropy
11. Define regression
12. How Knn is different from k-means clustering
13. What is concept learning
14. Define specific boundary and general boundary
15. Define target function
16. Define decision tree
17. What is ANN
18. Explain gradient descent approximation
19. State Bayes theorem
20. Define Bayesian belief networks
21. Differentiate hard and soft clustering
22. Define variance
23. What is inductive machine learning
24. Why K nearest neighbour algorithm is lazy learning algorithm
25. Why naïve Bayes is naïve
26. Mention classification algorithms
27. Define pruning

28. Differentiate Clustering and classification
29. Mention clustering algorithms
30. Define Bias