VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

B. L. D. E. Association's

V. P. Dr. P. G. HALAKATTI COLLEGE OF ENGINEERING & TECHNOLOGY, BIJAPUR - 586 103.

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

Machine learning Laboratory Journal
(15CSL76)

Name :______________________________

USN No :_____________________________

Class with Division : ______________________________

Roll No :____________

Year :_____________

B. L. D. E. Association's

V. P. Dr. P. G. HALAKATTI COLLEGE OF ENGINEERING & TECHNOLOGY, BIJAPUR - 586 103.

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that Mr./Ms……………………………………………… has satisfactorily completed the course of experiments in Machine Learning Laboratory prescribed by Visvevaraya Technological University for the VII semester class in laboratory of this department during the year 2019-20

Staff Incharge: **HoD**

1.

2.

Examiners: 1.

2.

INDEX PAGE

Program No 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

****

```
import csv
with open('tennis1.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)
```

****

Dataset:

sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
cloudy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

```
cmd : python p1.py
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['cloudy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change ', 'yes']
Most Specific Hypothesis is:
[['sunny',          'warm',          '?',          'strong',          '?',          '?']]
```

Program No. 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.

****

```
class Holder:
    factors={}
    attributes = ()

    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={}
    Negative={}
    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):
        G = self.initializeG()
        S = self.initializeS()
        count=0
        for trial_set in self.dataset:
            if self.is_positive(trial_set):
                G = self.remove_inconsistent_G(G,trial_set[0])
                S_new = S[:]
                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:
                S = self.remove_inconsistent_S(S,trial_set[0])
```

```
            G_new = G[:]
            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):

        S = tuple(['-' for factor in range(self.num_factors)])
        return [S]
    def initializeG(self):

        G = tuple(['?' for factor in range(self.num_factors)])
        return [G]
    def is_positive(self,trial_set):

        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):

        if value1 == '?' or value2 == '?':
            return True
        elif value1 == value2 :
            return True
        return False
    def consistent(self,hypothesis,instance):

        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):
```

```
        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):

        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):
        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):
        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):
         hypo = list(hypothesis)
        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)
        return generalization
    def specialize_inconsistent_G(self,hypothesis,instance):
         specializations = []
        hypo = list(hypothesis)
        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)
                specializations.append(hyp)
        return specializations
    def get_general(self,generalization,G):
        for g in G:
            if self.more_general(g,generalization):
                return generalization
        return None
    def get_specific(self,specializations,S):
        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):
        for g in G:
            if self.more_general(g,hypothesis):
                return True
        return False
    def exists_specific(self,hypothesis,S):

        for s in S:
            if self.more_specific(s,hypothesis):
                return True
        return False
    def more_general(self,hyp1,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):

        return self.more_general(hyp2,hyp1)
```

```
dataset=[(('sunny','warm','normal','strong','warm','same'),'Y'),(('sunny','warm','high','strong','w
arm','same'),'Y'),(('rainy','cold','high','strong','warm','change'),'N'),(('sunny','warm','high','stron
g','cool','change'),'Y')]
attributes =('Sky','Temp','Humidity','Wind','Water','Forecast')
f = Holder(attributes)
f.add_values('Sky',('sunny','rainy','cloudy'))
f.add_values('Temp',('cold','warm'))
f.add_values('Humidity',('normal','high'))
f.add_values('Wind',('weak','strong'))
f.add_values('Water',('warm','cold'))
f.add_values('Forecast',('same','change'))
a = CandidateElimination(dataset,f)
a.run_algorithm()
```

****

Output:

```
cmd : python p2.py
[('-', '-', '-', '-', '-', '-')]
[('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', '?', '?', '?', '?')]
```

Program No. 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.

****

```
import numpy as np
import math
import csv

class Node:
	def __init__(self,attribute):
		self.attribute = attribute
		self.children = []
		self.answer = ""
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)
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.childern:
        print(empty(level +1),value)
        print_tree(n,level +2)
metadata,traindata=read_data("sample1.csv")
data=np.array(traindata)
node=create_node(data,metadata)
print_tree(node,0)
```

****

DataSet:

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: cmd : python p3.py

```
('', 'sunny')
('  ', 'overcast')
('    ', 'yes')
('  ', 'rain')
('    ', 'weak')
('      ', 'strong')
('        ', 'no')
('      ', 'weak')
('        ', 'yes')
('  ', 'sunny')
('    ', 'high')
('      ', 'high')
('        ', 'no')
('      ', 'normal')
('        ', 'yes')
```

Program No 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)
y=y/100
def sigmoid(x):
	return 1/(1+np.exp(-x))
def derivatives_sigmoid(x):
	return x*(1-x)
epoch=7000
lr=0.25
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1
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))
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)
	EO=y-output
	outgrad=derivatives_sigmoid(output)
	d_output=EO*outgrad
	EH=d_output.dot(wout.T)
	hiddengrad=derivatives_sigmoid(hlayer_act)
	d_hiddenlayer=EH*hiddengrad
	wout+=hlayer_act.T.dot(d_output)*lr
	wh+=X.T.dot(d_hiddenlayer)*lr
print("Input=\n"+str(X))
print("Actual output:\n"+str(y))
print("predicated output:",output)
```

****

Output:

```
cmd : python p4.py
Input=
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output:
[[0.92]
 [0.86]
 [0.89]]
('predicated output:', array([[0.8967796 ],
       [0.8778105 ],
       [0.89453124]]))
```

Program No 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)):
                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));
                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="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)))
        summaries = summarizeByClass(trainingSet);
        predictions=getPredictions(summaries,testSet)
        accuracy=getAccuracy(testSet,predictions) print('accuracy
        of the classifier is:{0}%'.format(accuracy))
main()
```

DataSet:

6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0
5	109	75	26	0	36	0.546	60	0
3	158	76	36	245	31.6	0.851	28	1
3	88	58	11	54	24.8	0.267	22	0
6	92	92	0	0	19.9	0.188	28	0
10	122	78	31	0	27.6	0.512	45	0
4	103	60	33	192	24	0.966	33	0
11	138	76	0	0	33.2	0.42	35	0
9	102	76	37	0	32.9	0.665	46	1
2	90	68	42	0	38.2	0.503	27	1
4	111	72	47	207	37.1	1.39	56	1

Output:

1: cmd : python p5.py
Split40 rows into train26 and test=14 rows
accuracy of the classifier is:50.0%

2: cmd : python p5.py
Split40 rows into train26 and test=14 rows
accuracy of the classifier is:57.1428571429%

Program No 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.txt',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)

#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]))'''
```

****

Input Text Data:

```
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:

```
cmd : python p6.py
('The dimensions of the dataset', (18, 2))
0                     I love this sandwich
1                 This is an amazing place
2       I feel very good about these beers
3                     This is my best work
4                     What an awesome view
5            I do not like this restaurant
6                 I am tired of this stuff
7                   I can't deal with this
8                     He is my sworn enemy
9                      My boss is horrible
10                This is an awesome place
11   I do not like the taste of this juice
```

```
  12                          I love to dance
  13      I am sick and tired of this place
  14                     What a great holiday
15            That is a bad locality to stay
16            We will have good fun tomorrow
17          I went to my enemy's house
today Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype:
int64 (5,)
(13,)
(5,)
(13,)
[u'about', u'am', u'amazing', u'an', u'and', u'awesome', u'bad', u'beers', u'best', u'boss', u'can',
u'deal', u'do', u'enemy', u'feel', u'fun', u'good', u'great', u'have', u'he', u'holiday', u'horrible',
u'is', u'juice', u'like', u'locality', u'my', u'not', u'of', u'place', u'sick', u'stay', u'stuff', u'sworn',
u'taste', u'that', u'the', u'these', u'this', u'tired', u'to', u'tomorrow', u'very', u'view', u'we',
u'what', u'will', u'with', u'work']
    about am amazing an and awesome bad beers best boss can ... this tired to
tomorrow very view we what will with work
0        0   0      0   0    0      0    0     0     0     0     0 ...     0     0   0       1     0     0  1     0
1      0     0
1        0   0      0   0    0      0    0     0     0     0     0 ...     0     0   0       0     0     0  0     0
0      0     0
2        0   0      0   0    0      0    1     0     0     0     0 ...     0     0   1       0     0     0  0     0
0      0     0
```

```
3        1  0       0  0   0       0   0     1    0    0    0 ...    0     0  0        0    1    0  0      0
0     0    0
4        0  0       0  0   0       0   0     0    0    1    0 ...    0     0  0        0    0    0  0      0
0     0    0
5        0  0       1  1   0       0   0     0    0    0    0 ...    1     0  0        0    0    0  0      0
0     0    0
6        0  0       0  0   0       0   0     0    0    0    0 ...    0     0  0        0    0    0  0      1
0     0    0
7        0  1       0  0   1       0   0     0    0    0    0 ...    1     1  0        0    0    0  0      0
0     0    0
8        0  0       0  0   0       0   0     0    1    0    0 ...    1     0  0        0    0    0  0      0
0     0    1
9        0  0       0  0   0       0   0     0    0    0    1 ...    1     0  0        0    0    0  0      0
0     1    0
10       0  0       0  0   0       0   0     0    0    0    0 ...    1     0  0        0    0    0  0      0
0     0    0
11       0  1       0  0   0       0   0     0    0    0    0 ...    1     1  0        0    0    0  0      0
0     0    0
12       0  0       0  1   0       1   0     0    0    0    0 ...    0     0  0        0    0    1  0      1
0     0    0

 [13 rows x 49 columns]
 (0, 41)          1
 (0, 15)          1
 (0, 16)          1
 (0, 18)          1
 (0, 46)          1
 (0, 44)          1
 (1, 13)          1
 (1, 33)          1
 (1, 26)          1
 (1, 22)          1
 (1, 19)          1
 (2, 31)          1
 (2, 40)          1
 (2, 25)          1
 (2, 6) 1
 (2, 35)          1
 (2, 22)          1
 (3, 7) 1
 (3, 37)          1
 (3, 0) 1
 (3, 42)          1
 (3, 14)          1
```

```
  (3, 16)        1
  (4, 21)        1
  (4, 9) 1
  :          :
  (8, 8) 1
  (8, 38)        1
  (8, 26)        1
  (8, 22)        1
  (9, 47)        1
  (9, 11)        1
  (9, 10)        1
  (9, 38)        1
  (10, 23)       1
  (10, 34)       1
  (10, 36)       1
  (10, 24)       1
  (10, 27)       1
  (10, 12)       1
  (10, 28)       1
  (10, 38)       1
  (11, 32)       1
  (11, 28)       1
  (11, 39)       1
  (11, 1)        1
  (11, 38)       1
  (12, 43)       1
  (12, 5)        1
  (12, 45)       1
  (12, 3)        1
Accuracy metrics
('Accuracy of the classifer is', 0.6)
Confusion matrix
[[2 0]
 [2 1]]
Recall and Precison
0.3333333333333333
1.0
```

Program No. 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.

****

```
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", "False", 0.02],
[ "False", "False", 0.98]], [ smoking])
Bronchitis = ConditionalProbabilityTable(
[[ "True", "True", 0.92],
["True", "False",0.08],
[ "False", "False",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_proba({"Tuberculosis": "False"}))
s0 = State(Asia, name="Asia")
s1 = State(Lung, name="Lung")
s2 = State(dyspnea, name="Cancer")
network = BayesianNetwork("Asia")
network.add_nodes(s0,s1,s2)
network.add_edge(s0,s1)
network.add_edge(s1,s2)
network.bake()
print(network.predict_proba({"Lung": "True"}))
```

****

Output:

```
[{
   "frozen" :false,
   "dtype" :"str",
   "class" :"Distribution",
   "parameters" :[
      {
         "False" :0.550561797752809,
         "True" :0.4494382022471911
      }
   ],
   "name" :"DiscreteDistribution"
}
 'False'
 {
   "frozen" :false,
   "dtype" :"str",
   "class" :"Distribution",
   "parameters" :[
      {
         "False" :0.5,
         "True" :0.5
      }
   ],
   "name" :"DiscreteDistribution"
}]
```

Program No 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.

****

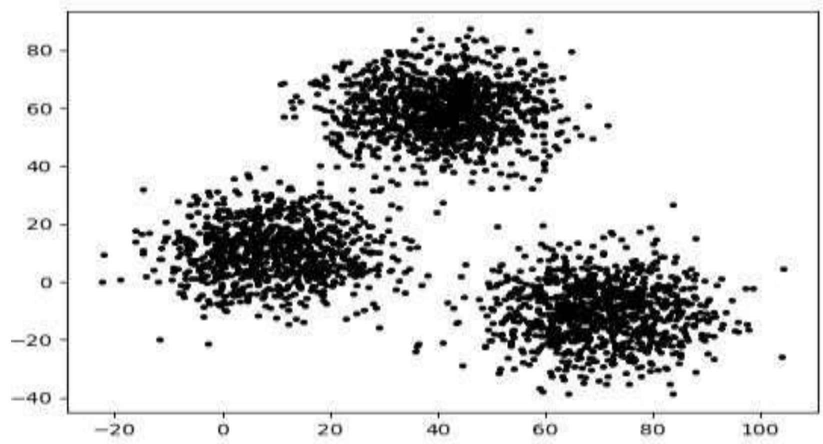
```
from sklearn.cluster import KMeans

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()
plt.plot()
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
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()
```

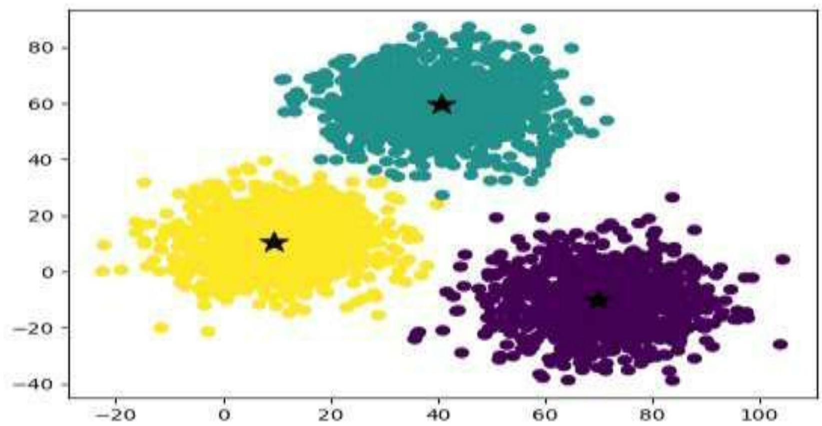
****

Output:
Input Data and Shape
(3000, 3)

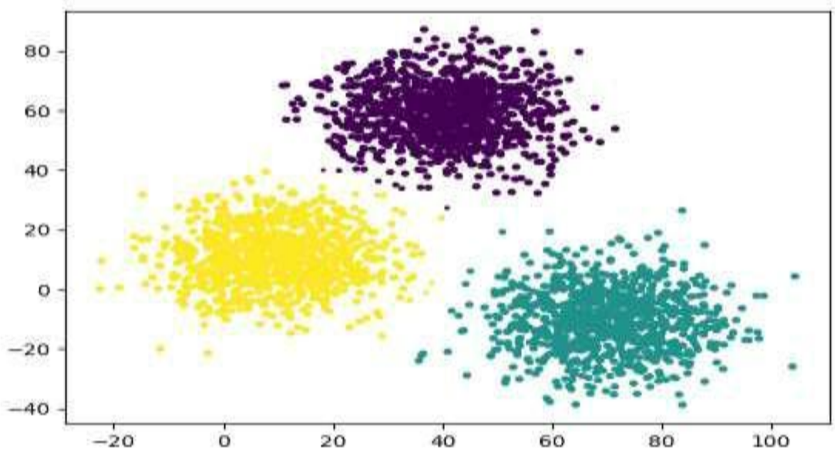
Graph for whole dataset



Graph using Kmeans Algorithm



Graph using EM Algorithm



Program No. 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
print(iris_data)
print(iris_labels)
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 metrics')
print(classification_report(y_test,y_pred))
```

****

Output:

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1. 0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5. 3. 1.6 0.2]
[5. 3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5. 3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3. 1.3 0.2]
[5.1 3.4 1.5 0.2]
[5. 3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5. 3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3. 1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5. 3.3 1.4 0.2]
[7. 3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4. 1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1.]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5. 2. 3.5 1.]
[5.9 3. 4.2 1.5]
[6. 2.2 4. 1.]

```
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3. 4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4. 1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3. 4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3. 5. 1.7]
[6. 2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6. 2.7 5.1 1.6]
[5.4 3. 4.5 1.5]
[6. 3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3. 4.1 1.3]
[5.5 2.5 4. 1.3]
[5.5 2.6 4.4 1.2]
[6.1 3. 4.6 1.4]
[5.8 2.6 4. 1.2]
[5. 2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3. 4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3. 1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6. 2.5]
[5.8 2.7 5.1 1.9]
[7.1 3. 5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3. 5.8 2.2]
[7.6 3. 6.6 2.1]
[4.9 2.5 4.5 1.7]
```

```
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3. 5.5 2.1]
[5.7 2.5 5. 2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3. 5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6. 2.2 5. 1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6. 1.8]
[6.2 2.8 4.8 1.8]
[6.1 3. 4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3. 5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3. 6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6. 3. 4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3. 5.2 2.3]
[6.3 2.5 5. 1.9]
[6.5 3. 5.2 2. ]
[6.2 3.4 5.4 2.3]
[5.9 3. 5.1 1.8]]
[0000000000000000000000000000000000000
```

```
 000000000000011111111111111111111111
 111111111111111111111111122222222222
 222222222222222222222222222222222222
 2 2]
confusion matrix is as follows
[[19  0   0]
 [013     3]
 [ 0   0 10]]
Accuracy metrics
              precision  recall   f1-score  support

          0       1.00    1.00       1.00       19
          1       1.00    0.81       0.90       16
          2       0.77    1.00       0.87       10

   micro avg       0.93      0.93      0.93       45
   macro avg       0.92      0.94      0.92       45
 weighted avg       0.95      0.93      0.93      45
```

Program No. 10: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

****

```
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('10data.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]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

****

Output: